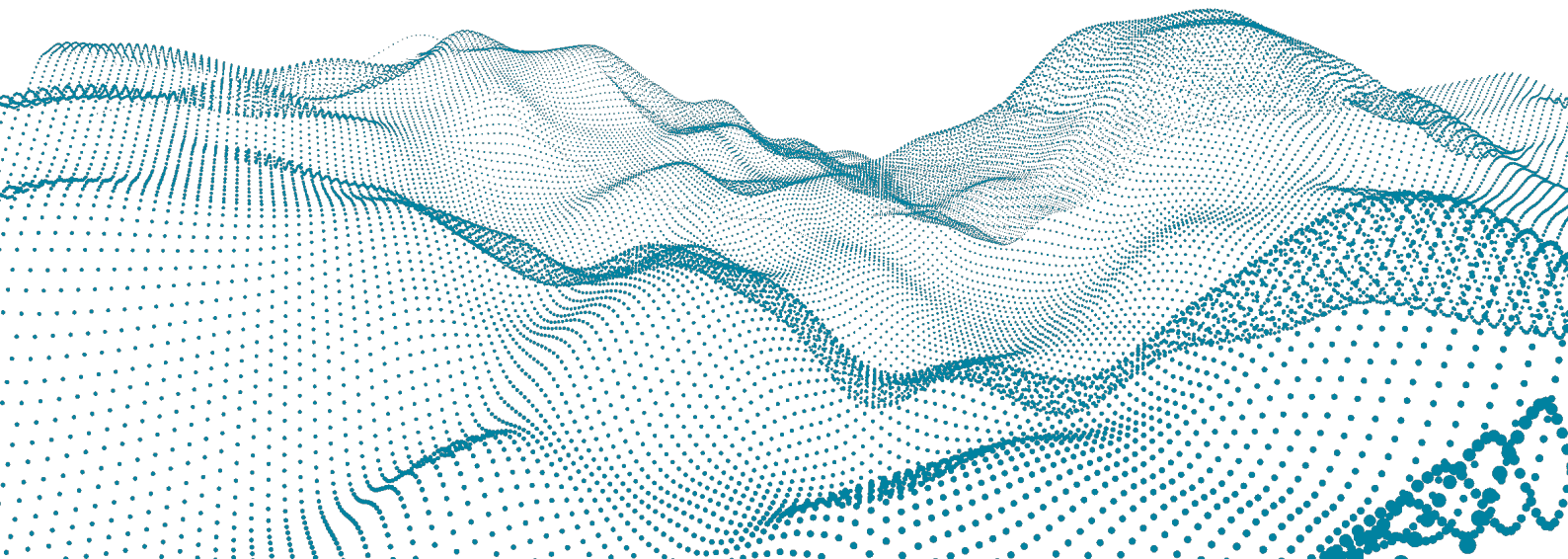
roboception

Roboception GmbH | Oktober 2026

rc_visard 3D Stereosensor

MONTAGE- UND BETRIEBSANLEITUNG



Revisionen

Dieses Produkt kann bei Bedarf jederzeit ohne Vorankündigung geändert werden, um es zu verbessern, zu optimieren oder an eine überarbeitete Spezifikation anzupassen. Werden solche Änderungen vorgenommen, wird auch das vorliegende Handbuch überarbeitet. Beachten Sie die angegebene Versionsnummer.

DOKUMENTATIONSVERSION 26.10.3 27.10.2026

Gültig für *rc_visard* Firmware 26.10.x

HERSTELLER

Roboception GmbH

Kaflerstraße 2

81241 München

Deutschland

KUNDENSUPPORT: support@roboception.de | +49 89 889 50 79-0 (09:00-17:00 CET)

Betriebsanleitung bitte vollständig lesen und produktnah aufbewahren.

COPYRIGHT

Das vorliegende Handbuch und das darin beschriebene Produkt sind durch Urheberrechte geschützt. Sofern das deutsche Urheber- und Leistungsschutzrecht nichts anderes vorschreibt, darf der Inhalt dieses Handbuchs nur mit dem vorherigen Einverständnis von Roboception bzw. des Inhabers des Schutzrechts verwendet und verbreitet werden. Das vorliegende Handbuch und das darin beschriebene Produkt dürfen ohne das vorherige Einverständnis von Roboception weder für Verkaufs- noch für andere Zwecke weder teilweise noch vollständig vervielfältigt werden.

Die in diesem Dokument bereitgestellten Informationen sind nach bestem Wissen und Gewissen zusammengestellt worden. Roboception haftet jedoch nicht für deren Verwendung.

Wurden nach Redaktionsschluss noch Änderungen am Produkt vorgenommen, kann es vorkommen, dass das Produkt vom Handbuch abweicht. Die im vorliegenden Dokument enthaltenen Informationen können sich ohne Vorankündigung ändern.

Inhaltsverzeichnis

1 Einführung **6**
- 1.1 Überblick . . . 6
- 1.2 Garantie . . . 8
- 1.3 Schnittstellen, Zulassungen und Normen . . . 9
  - 1.3.1 Schnittstellen . . . 9
  - 1.3.2 Zulassungen . . . 9
  - 1.3.3 Normen . . . 9
- 1.4 Informationen zur Entsorgung . . . 10
- 1.5 Glossar . . . 12

2 Sicherheit **14**
- 2.1 Allgemeine Sicherheitshinweise . . . 14
- 2.2 Bestimmungsgemäße Verwendung . . . 15

3 Hardware-Spezifikation **16**
- 3.1 Lieferumfang . . . 16
- 3.2 Technische Spezifikation . . . 17
- 3.3 Umwelt- und Betriebsbedingungen . . . 20
- 3.4 Spezifikationen für die Stromversorgung . . . 20
- 3.5 Verkabelung . . . 21
- 3.6 Mechanische Schnittstelle . . . 24
- 3.7 Koordinatensysteme . . . 24

4 Installation **26**
- 4.1 Softwarelizenz . . . 26
- 4.2 Einschalten . . . 26
- 4.3 Aufspüren von *rc_visard*-Geräten . . . 27
  - 4.3.1 Zurücksetzen der Konfiguration . . . 27
- 4.4 Netzwerkkonfiguration . . . 28
  - 4.4.1 Host-Name . . . 29
  - 4.4.2 Automatische Konfiguration (werksseitige Voreinstellung) . . . 29
  - 4.4.3 Manuelle Konfiguration . . . 30

5 Softwaremodule **31**
- 5.1 Kamera Modul . . . 31
  - 5.1.1 Parameter, Statuswerte und Services . . . 31
  - 5.1.2 Rektifizierung . . . 40
  - 5.1.3 Anzeigen und Herunterladen von Bildern . . . 40
- 5.2 Stereo-Matching Modul . . . 40
  - 5.2.1 Parameter . . . 40
  - 5.2.2 Statuswerte . . . 48
  - 5.2.3 Services . . . 48
  - 5.2.4 Berechnung von 3D Daten . . . 50
  - 5.2.5 Anzeigen und Herunterladen von Tiefenbildern und Punktwolken . . . 51
- 5.3 Navigationsmodule . . . 52
  - 5.3.1 Sensordynamik . . . 52

5.3.2 Visuelle Odometrie . . . 61
5.3.3 Stereo-INS . . . 65
5.3.4 SLAM . . . 66
5.4 Detektions- und Messmodule . . . 73
5.4.1 Measure . . . 74
5.4.2 LoadCarrier . . . 81
5.4.3 TagDetect . . . 97
5.4.4 ItemPick . . . 110
5.4.5 BoxPick . . . 134
5.4.6 Pack . . . 168
5.4.7 SilhouetteMatch . . . 180
5.5 Konfigurationsmodule . . . 222
5.5.1 Hand-Auge-Kalibrierung . . . 222
5.5.2 CollisionCheck . . . 245
5.5.3 Kamerakalibrierung . . . 254
5.5.4 IOControl und Projektor-Kontrolle . . . 261
5.6 Datenbankmodule . . . 264
5.6.1 LoadCarrierDB . . . 265
5.6.2 RoiDB . . . 273
5.6.3 PrimitiveDB . . . 280
5.6.4 GripperDB . . . 284

6 Schnittstellen 299
6.1 Web GUI . . . 299
6.1.1 Zugriff auf die Web GUI . . . 300
6.1.2 Kennenlernen der Web GUI . . . 300
6.1.3 Web GUI Zugriffskontrolle . . . 301
6.1.4 Herunterladen von Kamerabildern . . . 301
6.1.5 Herunterladen von Tiefenbildern und Punktwolken . . . 302
6.2 REST-API-Schnittstelle . . . 302
6.2.1 Allgemeine Struktur der Programmierschnittstelle (API) . . . 303
6.2.2 Migration von API Version 1 . . . 304
6.2.3 Verfügbare Ressourcen und Anfragen . . . 305
6.2.4 Datentyp-Definitionen . . . 338
6.2.5 Swagger UI . . . 351
6.3 Die rc_dynamics-Schnittstelle . . . 355
6.3.1 Starten/Stoppen der Dynamik-Zustandsschätzungen . . . 355
6.3.2 Konfiguration von Datenströmen . . . 356
6.3.3 Datenstromprotokoll . . . 357
6.3.4 Die rc_dynamics-Programmierschnittstelle . . . 359
6.4 Generic Robot Interface . . . 359
6.4.1 Überblick . . . 359
6.4.2 Job Definition . . . 359
6.4.3 Hand-Auge-Kalibrierung . . . 369
6.4.4 Spezifikation des Binären GRI Protokolls . . . 369
6.4.5 Integration mit einem Roboter . . . 377
6.4.6 Job und HEC_config API . . . 378
6.5 OPC UA Interface . . . 383
6.6 KUKA Ethernet KRL Schnittstelle . . . 383
6.6.1 Konfiguration der Ethernet-Verbindung . . . 383
6.6.2 Allgemeine XML-Struktur . . . 384
6.6.3 Services . . . 385
6.6.4 Parameter . . . 389
6.6.5 Migration zu Firmware Version 22.01 . . . 391
6.6.6 Beispielanwendungen . . . 391
6.6.7 Fehlerbehebung . . . 391
6.7 GigE Vision 2.0/GenICam-Schnittstelle . . . 392
6.7.1 GigE Vision Ports . . . 393

6.7.2 Wichtige Parameter der GenICam-Schnittstelle . . . 393
6.7.3 Wichtige Standardparameter der GenICam-Schnittstelle . . . 393
6.7.4 Besondere Parameter der GenICam-Schnittstelle des *rc_visard* . . . 397
6.7.5 Chunk-Daten . . . 400
6.7.6 Verfügbare Bild-Streams . . . 401
6.7.7 Umwandlung von Bild-Streams . . . 401
6.8 gRPC Bilddatenschnittstelle . . . 402
6.8.1 gRPC Servicedefinition . . . 403
6.8.2 Beispielclient . . . 406
6.9 Zeitsynchronisierung . . . 406
6.9.1 NTP . . . 407
6.9.2 PTP . . . 407
6.9.3 Manuelles Setzen der Zeit . . . 407

7 Wartung **408**
7.1 Reinigung der Kameralinsen . . . 408
7.2 Kamerakalibrierung . . . 408
7.3 Backup der Einstellungen . . . 408
7.4 Aktualisierung der Softwarelizenz . . . 409
7.5 Download der Logdateien . . . 409
7.6 Aktualisierung der Firmware . . . 409
7.7 Wiederherstellung der vorherigen Firmware-Version . . . 411
7.8 Neustart des *rc_visard* . . . 411

8 Zubehör **412**
8.1 Anschlussset . . . 412
8.2 Verkabelung . . . 412
8.2.1 Ethernet-Anschluss . . . 412
8.2.2 Stromanschluss . . . 413
8.2.3 Netzteile . . . 413
8.3 Ersatzteile . . . 413

9 Fehlerbehebung **414**
9.1 LED-Farben . . . 414
9.2 Probleme mit der Hardware . . . 414
9.3 Probleme mit der Konnektivität . . . 415
9.4 Probleme mit den Kamerabildern . . . 416
9.5 Probleme mit Tiefen-/Disparitäts-, Fehler- oder Konfidenzbildern . . . 417
9.6 Probleme mit der Zustandsschätzung . . . 418
9.7 Probleme mit GigE Vision/GenICam . . . 418

10 Kontakt **419**
10.1 Support . . . 419
10.2 Downloads . . . 419
10.3 Adresse . . . 419

11 Anhang **420**
11.1 Formate für Posendaten . . . 420
11.1.1 Rotationsmatrix und Translationsvektor . . . 421
11.1.2 ABB Posenformat . . . 421
11.1.3 FANUC XYZ-WPR Format . . . 422
11.1.4 Franka Posenformat . . . 422
11.1.5 Fruitcore HORST Posenformat . . . 424
11.1.6 Kawasaki XYZ-OAT Format . . . 424
11.1.7 KUKA XYZ-ABC Format . . . 425
11.1.8 Mitsubishi XYZ-ABC Format . . . 426
11.1.9 Universal Robots Posenformat . . . 426
11.1.10 Yaskawa Posenformat . . . 427

HTTP Routing Table **429**

Stichwortverzeichnis **431**

1 Einführung

Hinweise im Handbuch

Um Schäden an der Ausrüstung zu vermeiden und die Sicherheit der Benutzer zu gewährleisten, enthält das vorliegende Handbuch Sicherheitshinweise, die mit dem Symbol *Warnung* gekennzeichnet werden. Zusätzliche Informationen sind als *Bemerkung* gekennzeichnet.

Warnung: Die mit *Warnung* gekennzeichneten Sicherheitshinweise geben Verfahren und Maßnahmen an, die befolgt bzw. ergriffen werden müssen, um Verletzungsgefahren für Bediener/Benutzer oder Schäden am Gerät zu vermeiden. Beziehen sich die angegebenen Sicherheitshinweise auf Softwaremodule, dann weisen diese auf Verfahren hin, die befolgt werden müssen, um Störungen oder ein Fehlverhalten der Software zu vermeiden.

Bemerkung: Bemerkungen werden in diesem Handbuch eingesetzt, um zusätzliche relevante Informationen zu vermitteln.

1.1 Überblick

Der 3D-Sensor *rc_visard* ist eine selbstregistrierende Stereokamera, die über eine integrierte Recheneinheit verfügt und der Schutzklasse IP 54 angehört.

Der *rc_visard* stellt Echtzeit-Kamerabilder und Tiefenbilder bereit, die zur Berechnung von 3D-Punktwolken verwendet werden können. Zudem erstellt er Konfidenz- und Fehlerbilder, mit denen sich die Qualität der Bilderfassung messen lässt. Dank der standardisierten Schnittstellen ist er mit allen großen Bildverarbeitungsbibliotheken kompatibel und bietet darüber hinaus eine intuitive, web-basierte Bedienoberfläche an.

Mit optional erhältlichen Softwaremodulen bietet der *rc_visard* Standardlösungen für Anwendungen in der Objekterkennung oder für robotische Pick-and-Place-Applikationen.

Der *rc_visard* kann sich darüber hinaus aufgrund von Bild- und Trägheitsdaten selbst lokalisieren. Mit dem integrierten SLAM-Modul lässt sich eine mobile Navigationslösung umsetzen.

Der *rc_visard* wird mit zwei verschiedenen Basisabständen angeboten: Der *rc_visard* 65 eignet sich ideal für die Montage auf Roboter-Manipulatoren, wohingegen der *rc_visard* 160 als Navigationsgerät oder extern befestigter Sensor eingesetzt werden kann.

Dank der intuitiven Kalibrierung, Konfiguration und Bedienung macht der *rc_visard* 3D-Wahrnehmung für jedermann möglich.

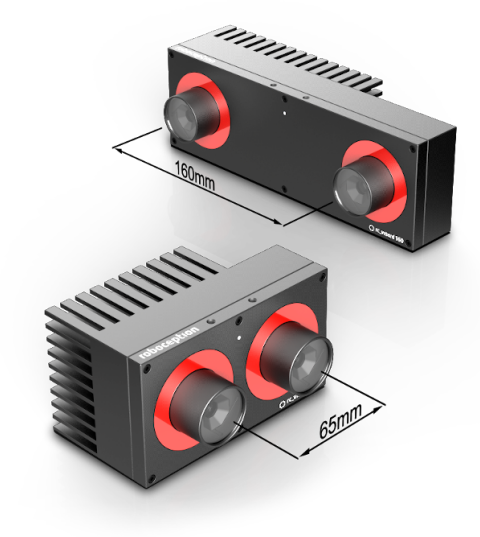


Abb. 1.1: *rc_visard* 65 und *rc_visard* 160

Werden im vorliegenden Handbuch die Begriffe „Sensor“, „*rc_visard* 65“ und „*rc_visard* 160“ verwendet, so beziehen sich diese auf die von Roboception angebotene *rc_visard*-Produktfamilie an selbstregistrierenden Kameras. Die Installation und Steuerung all dieser Sensoren sind absolut identisch. Zudem verwenden alle den gleichen Montagesockel.

Bemerkung: Sofern nicht anders angegeben, gelten die in diesem Handbuch enthaltenen Informationen für beide Versionen des Roboception-Sensors, d.h. für den *rc_visard* 65 und den *rc_visard* 160.

Bemerkung: Das vorliegende Handbuch nutzt das metrische System und verwendet vorrangig die Maßeinheiten Meter und Millimeter. Sofern nicht anders angegeben, sind Abmessungen in technischen Zeichnungen in Millimetern angegeben.

1.2 Garantie

Jede Änderung oder Modifikation der Hard- oder Software dieses Produkts, die nicht ausdrücklich von Roboception genehmigt wurde, kann zum Verlust der Gewährleistungs- und Garantierechte führen.

Warnung: Der *rc_visard* arbeitet mit komplexer Hardware- und Software-Technologie, die sich ggf. nicht immer so verhält, wie es der Benutzer beabsichtigt. Der Käufer muss seine Anwendung so gestalten, dass eine Fehlfunktion des *rc_visard* nicht zu Körperverletzungen, Sachschäden oder anderen Verlusten führt.

Warnung: Der *rc_visard* darf nicht zerlegt, geöffnet, instand gesetzt oder verändert werden, da dies eine Stromschlaggefahr oder andere Risiken nach sich ziehen kann. Kann nachgewiesen werden, dass der Benutzer versucht hat, das Gerät zu öffnen und/oder zu modifizieren, erlischt die Garantie. Dies gilt auch, wenn Typenschilder beschädigt, entfernt oder unkenntlich gemacht wurden.

Warnung: VORSICHT: Gemäß den europäischen CE-Anforderungen müssen alle Kabel, die zum Anschluss dieses Geräts verwendet werden, abgeschirmt und geerdet sein. Der Betrieb mit falschen Kabeln kann zu Interferenzen mit anderen Geräten oder zu einem unerwünschten Verhalten des Produkts führen.

Bemerkung: Dieses Produkt darf nicht über den Hausmüll entsorgt werden. Durch die korrekte Entsorgung des Produkts tragen Sie zum Umweltschutz bei. Nähere Informationen zur Wiederverwertung des Produkts erhalten Sie bei den zuständigen Behörden, bei Ihrem Entsorgungsunternehmen oder beim Händler, bei dem Sie das Produkt erworben haben.

1.3 Schnittstellen, Zulassungen und Normen

1.3.1 Schnittstellen

Der *rc_visard* unterstützt folgende Standardinterfaces:

GEN<i>CAM

Der generische Schnittstellenstandard für Kameras ist die Grundlage für die Plug-&-Play-Handhabung von Kameras und Geräten.

GiGE VISION

GigE Vision® ist ein Interfacestandard für die Übermittlung von Hochgeschwindigkeitsvideo- und zugehörigen Steuerdaten über Ethernet-Netzwerke.

1.3.2 Zulassungen

Der *rc_visard* hat folgende Zulassungen erhalten:

CE EG-Konformitätserklärung

TÜV SÜD C US NRTL-Zertifizierung durch den TÜV Süd

FC Konformitätserklärung nach FCC, Part 15 (betrifft die USA): Changes or modifications not expressly approved by the manufacturer could void the user's authority to operate the equipment. This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and
2. this device must accept any interference received, including interference that may cause undesired operation.

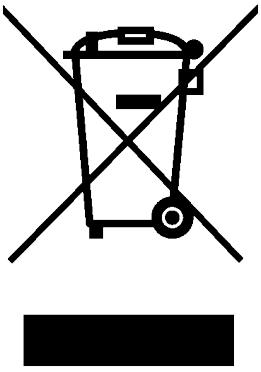
1.3.3 Normen

Der *rc_visard* wurde getestet und entspricht den Vorgaben der folgenden Normen:

- AS/NZS CISPR32 : 2015 Information technology equipment, Radio disturbance characteristics, Limits and methods of measurement
- CISPR 32 : 2015 Electromagnetic compatibility of multimedia equipment - Emission requirements
- GB 9254 : 2008 This standard is out of the accreditation scope. Information technology equipment, Radio disturbance characteristics, Limits and methods of measurement

- EN 55032 : 2015 Electromagnetic compatibility of multimedia equipment - Emission requirements
- EN 55024 : 2010 +A1:2015 Information technology equipment, Immunity characteristics, Limits and methods of measurement
- CISPR 24 : 2015 +A1:2015 International special committee on radio interference, Information technology equipment-Immunity characteristics-Limits and methods of measurement
- EN 61000-6-2 : 2005 Electromagnetic compatibility (EMC) Part 6-2:Generic standards - Immunity for industrial environments
- EN 61000-6-3 : 2007+A1:2011 Electromagnetic compatibility (EMC) - Part 6-3: Generic standards - Emission standard for residential, commercial and light-industrial environments
- Registered FCC ID: 2AVMTRCV17
- Certified for Canada according to CAN ICES-3(B)/NMB-3(B)

1.4 Informationen zur Entsorgung



1. **Entsorgung von Elektro- und Elektronikgeräten**

   Das Symbol der „durchgestrichenen Mülltonne“ bedeutet, dass Sie gesetzlich verpflichtet sind, diese Geräte einer vom unsortierten Siedlungsabfall getrennten Erfassung zuzuführen. Die Entsorgung über den Hausmüll, wie bspw. die Restmülltonne oder die Gelbe Tonne ist untersagt. Vermeiden Sie Fehlwürfe durch die korrekte Entsorgung in speziellen Sammel- und Rückgabestellen.

2. **Entnahme von Batterien**

   Enthalten die Produkte Batterien und Akkus, die aus dem Altgerät zerstörungsfrei entnommen werden können, müssen diese vor der Entsorgung entnommen werden und getrennt als Batterie entsorgt werden.

   Folgende Batterien bzw. Akkumulatoren sind im rc_visard enthalten: Keine

3. **Möglichkeiten der Rückgabe von Altgeräten**

   Besitzer von Altgeräten können diese an den Hersteller zurückgeben, damit eine ordnungsgemäße Entsorgung sichergestellt ist.“

   Bitte kontaktieren Sie den *Support* (Abschnitt 10) wegen der Rücknahme des Gerätes.

4. **Datenschutz**

Endnutzer von Elektro- und Elektronikaltgeräten werden darauf hingewiesen, dass Sie für das Löschen personenbezogener Daten auf den zu entsorgenden Altgeräten selbst verantwortlich sind.

5. **WEEE-Registrierungsnummer**

   Roboception ist unter der Registrierungsnummer DE 33323989 bei der stiftung elektro-altgeräte register, Nordostpark 72, 90411 Nürnberg, als Hersteller von Elektro- und/ oder Elektronikgeräten registriert.

6. **Sammel- und Verwertungsquoten**

   Die EU-Mitgliedsstaaten sind nach der WEEE-Richtlinie verpflichtet, Daten zu Elektro- und Elektronikaltgeräten zu erheben und diese an die Europäische Kommission zu übermitteln. Auf der Webseite des Bundesministeriums für Umwelt- und Naturschutz finden Sie weitere Informationen hierzu.

Information zur Entsorgung außerhalb der Europäischen Union

Das Symbol der durchgestrichenen Mülltonne ist nur in der Europäischen Union gültig. Für die Entsorgung in anderen Ländern außerhalb der Europäischen Union können die örtlichen Behörden oder der Hersteller Auskunft über die richtige Entsorgungsmethode geben.

1.5 Glossar

DHCP Das Dynamic Host Configuration Protocol (DHCP) wird verwendet, um einem Netzwerkgerät automatisch eine *IP*-Adresse zuzuweisen. Einige DHCP-Server akzeptieren lediglich bekannte Geräte. In diesem Fall muss der Administrator die feste *MAC-Adresse* eines Gerätes im DHCP-Server erfassen.

DNS

mDNS Das Domain Name System (DNS) verwaltet die Host-Namen und *IP*-Adressen aller Netzwerkgeräte. Es dient dazu, den Host-Namen zur Kommunikation mit einem Gerät in die IP-Adresse zu übersetzen. Das DNS kann so konfiguriert werden, dass diese Informationen entweder automatisch abgerufen werden, wenn ein Gerät in einem Netzwerk erscheint, oder manuell von einem Administrator zu erfassen sind. Im Gegensatz hierzu arbeitet *multicast DNS* (mDNS) ohne einen zentralen Server, wobei jedes Mal, wenn ein Host-Name aufgelöst werden muss, alle Geräte in einem Netzwerk abgefragt werden. mDNS ist standardmäßig für die Betriebssysteme Linux und macOS verfügbar und wird verwendet, wenn „.local‘ an einen Host-Namen angehängt wird.

DOF Als Freiheitsgrade (Degrees of Freedom, DOF) wird die Anzahl unabhängiger Translations- und Rotationsparameter bezeichnet. Im 3D-Raum genügen 6 Freiheitsgrade (drei für Translation und drei für Rotation), um eine beliebige Position und Orientierung zu definieren.

GenICam GenICam ist eine generische Standard-Schnittstelle für Kameras. Sie fungiert als einheitliche Schnittstelle für andere Standards, wie *GigE Vision*, Camera Link, USB, usw. Für nähere Informationen siehe http://genicam.org.

GigE Gigabit Ethernet (GigE) ist eine Netzwerktechnologie, die mit einer Übertragungsrate von einem Gigabit pro Sekunde arbeitet.

GigE Vision GigE Vision® ist ein Standard für die Konfiguration von Kameras und Übertragung der Bilder über eine *GigE* Netzwerkverbindung. Für nähere Informationen siehe http://gigevision.com.

IMU Eine inertiale Messeinheit (Inertial Measurement Unit, IMU) dient zur Messung der Linearbeschleunigungen und Drehraten in allen drei Dimensionen. Sie besteht aus drei Beschleunigungsaufnehmern und drei Gyroskopen.

INS Ein Trägheitsnavigationssystem oder inertiales Navigationssystem (INS) ist ein 3D-Messsystem, das Positions- und Orientierungsdaten über Inertialsensoren (Beschleunigungs- und Drehratensensoren) berechnet. In diesem Dokument wird die Kombination aus Stereobildverarbeitung und inertialer Navigation ebenfalls INS genannt.

IP

IP-Adresse Das Internet Protocol (IP) ist ein Standard für die Übertragung von Daten zwischen verschiedenen Geräten in einem Computernetzwerk. Jedes Gerät benötigt eine IP-Adresse, die innerhalb des Netzwerks nur einmal vergeben werden darf. Die IP-Adresse lässt sich über *DHCP*, über *Link-Local* oder manuell konfigurieren.

Link-Local Link-Local ist eine Technologie, mit der sich ein Netzwerkgerät selbst eine *IP-Adresse* aus dem Adressbereich 169.254.0.0/16 zuweist und überprüft, ob diese im lokalen Netzwerk eindeutig ist. Link-Local kann verwendet werden, wenn *DHCP* nicht verfügbar ist oder die manuelle IP-Konfiguration nicht vorgenommen wurde bzw. werden kann. Link-Local ist besonders nützlich, wenn ein Netzwerkgerät direkt an einen Host-Computer angeschlossen werden soll. Windows 10 greift automatisch auf Link-Local zurück, wenn DHCP nicht verfügbar ist (Fallback-Option). Unter Linux muss Link-Local manuell im Netzwerkmanager aktiviert werden.

MAC-Adresse Bei der MAC-Adresse (Media Access Control Address) handelt es sich um die eindeutige und feste Adresse eines Netzwerkgerätes. Sie wird auch als Hardware-Adresse bezeichnet. Im Gegensatz zur *IP-Adresse* wird die MAC-Adresse einem Gerät (normalerweise) fest zugewiesen; sie ändert sich nicht.

NTP Das Network Time Protocol (NTP) ist ein TCP/IP Protokoll, um Zeit über ein Netzwerk zu synchronisieren. Im Wesentlichen fordert ein Client die aktuelle Zeit von einem Server an und nutzt diese, um seine eigene Uhr zu stellen.

PTP Das Precision Time Protocol (PTP, auch als IEEE1588 bekannt) ist ein Protokoll, welches genauere und robustere Synchronisation der Uhren erlaubt als NTP.

SDK Ein Software Development Kit (SDK) ist eine Sammlung von Softwareentwicklungswerkzeugen bzw. von Softwaremodulen.

SGM SGM steht für Semi-Global Matching, einen hochmodernen Stereo-Matching-Algorithmus, der sich durch kurze Laufzeiten und eine hohe Genauigkeit – insbesondere an Objekträndern, bei feinen Strukturen und in schwach texturierten Bildbereichen – auszeichnet.

SLAM SLAM steht für Simultaneous Localization and Mapping und bezeichnet den Prozess der Kartenerstellung für eine unbekannte Umgebung und die gleichzeitige Schätzung der Sensorpose in der Karte.

TCP Der Tool Center Point (TCP) ist die Position des Werkzeugs am Endeffektor eines Roboters. Die Position und Orientierung des TCP definiert die Position und Orientierung des Werkzeugs im 3D-Raum.

UDP Das User Datagram Protocol (UDP) ist die minimale nachrichtenbasierte Transportschicht der Internetprotokollfamilie (*IP*). UDP nutzt ein einfaches Modell zur verbindungslosen Übertragung, das mit einem Minimum an Protokollmechanismen, wie Integritätschecks (über Prüfsummen), auskommt. Der *rc_visard* nutzt UDP zum Übertragen seiner *Dynamik-Zustandsschätzungen* (Abschnitt 5.3.1.2) über die *rc_dynamics-Schnittstelle* (Abschnitt 6.3). Um diese Daten zu empfangen, kann einer Anwendung ein Datagram Socket (Endpunkt der Datenübertragung) zugeordnet werden, der aus einer Kombination aus *IP-Adresse* und einer Service-Port-Nummer besteht (z.B. `192.168.0.100:49500`). Dieser Endpunkt wird im vorliegenden Handbuch in der Regel als *Ziel* eines rc_dynamics-Datenstroms bezeichnet.

URI

URL Ein Uniform Resource Identifier (URI) ist eine Zeichenfolge, mit der sich Ressourcen in der REST-API des *rc_visard* identifizieren lassen. Ein Beispiel für eine solche URI ist die Zeichenkette `/nodes/rc_camera/parameters/fps`, die auf die `fps`-Laufzeitparameter des Stereokamera-Moduls verweist.

Ein Uniform Resource Locator (URL) gibt zudem die vollständige Netzwerkadresse und das Netzwerkprotokoll an. Die oben angeführte Ressource könnte beispielsweise über `https://<ip>/api/v1/nodes/rc_camera/parameters/fps` lokalisiert werden, wobei sich `<ip>` auf die *IP-Adresse* des *rc_visard* bezieht.

XYZ+Quaternion Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *Rotationsmatrix und Translationsvektor* (Abschnitt 11.1.1).

XYZABC Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *KUKA XYZ-ABC Format* (Abschnitt 11.1.7).

2 Sicherheit

Warnung: Vor Inbetriebnahme des *rc_visard*-Produkts muss der Bediener alle Anweisungen in diesem Handbuch gelesen und verstanden haben.

Bemerkung: Der Begriff „Bediener“ bezieht sich auf jede Person, die in Verbindung mit dem *rc_visard* mit einer der folgenden Aufgaben betraut ist:

- Installation
- Wartung
- Inspektion
- Kalibrierung
- Programmierung
- Außerbetriebnahme

Das vorliegende Handbuch geht auf die verschiedenen Softwaremodule des *rc_visard* ein und erläutert allgemeine Aspekte zum Lebenszyklus des Produkts: von der Installation über die Verwendung bis hin zur Außerbetriebnahme.

Die im vorliegenden Handbuch enthaltenen Zeichnungen und Fotos sind Beispiele zur Veranschaulichung. Das ausgelieferte Produkt kann hiervon abweichen.

2.1 Allgemeine Sicherheitshinweise

Bemerkung: Wird der *rc_visard* entgegen den hierin angegebenen Sicherheitshinweisen verwendet, so kann dies zu Personen- oder Sachschäden sowie zum Verlust der Garantie führen.

Warnung:

- Der *rc_visard* muss vor der Verwendung ordnungsgemäß montiert werden.
- Alle Kabel sind am *rc_visard* und an seiner Halterung zu sichern.
- Die Länge der verwendeten Kabel darf 30 Meter nicht überschreiten.
- Die Stromversorgung für den *rc_visard* muss über eine geeignete Gleichstromquelle erfolgen.
- Jeder *rc_visard* muss an eine separate Gleichstromquelle angeschlossen werden.
- Das Gehäuse des *rc_visard* muss geerdet werden.
- Die zum *rc_visard* oder zugehöriger Ausrüstung angegebenen Sicherheitshinweise müssen stets eingehalten werden.
- Der *rc_visard* fällt nicht in den Anwendungsbereich der europäischen Maschinen-, Niederspannungs- oder Medizinproduktrichtlinie.

Risikobewertung und Endanwendung

Der *rc_visard* kann auf einem Roboter installiert werden. Der Roboter, der *rc_visard* und jede andere für die Endanwendung eingesetzte Ausrüstung müssen im Rahmen einer Risikobewertung begutachtet werden. Der Systemintegrator ist verpflichtet, die Einhaltung aller lokalen Sicherheitsmaßnahmen und Vorschriften zu gewährleisten. Je nach Anwendung kann es Risiken geben, die zusätzliche Schutz- oder Sicherheitsmaßnahmen erfordern.

2.2 Bestimmungsgemäße Verwendung

Der *rc_visard* ist ein industrieller 3D-Bildverarbeitungssensor mit Recheneinheit, der für die Integration in Automatisierungs- und Robotikanwendungen bestimmt ist. Sein bestimmungsgemäßer Zweck ist die Erfassung und Verarbeitung von Bilddaten sowie die Bereitstellung der Ergebnisse (z.B. 3D-Daten, erkannte Posen) für eine angeschlossene Robotersteuerung oder ein angeschlossenes Host-System.

Der *rc_visard* ist für den Betrieb in einer vertrauenswürdigen Umgebung mit einer direkten Punkt-zu-Punkt-Verbindung (1:1) zwischen dem Gerät und der angeschlossenen Kamera, Robotersteuerung oder dem angeschlossenen Host-System ausgelegt und bestimmt. In dieser Konfiguration ist nicht vorgesehen, die Geräte mit einem offenen oder nicht vertrauenswürdigen Netzwerk zu verbinden oder aus einem solchen Netzwerk zugänglich zu machen.

Mehrere Schnittstellen, die für die Interoperabilität mit Robotersteuerungen und industriellen Systemen erforderlich sind, basieren auf etablierten Industriestandards oder herstellerspezifischen Protokollen, die weder Verschlüsselung noch Authentifizierung unterstützen. Die Aktivierung dieser Schnittstellen außerhalb eines vertrauenswürdigen, isolierten Netzwerks kann das System Cybersicherheitsrisiken aussetzen und liegt außerhalb der bestimmungsgemäßen Verwendung des Produkts.

Der Betreiber ist dafür verantwortlich sicherzustellen, dass die Betriebsumgebung diesen Bedingungen entspricht, einschließlich der physischen Zugangskontrolle zum Gerät, der Netzwerkisolierung und gegebenenfalls der sicheren Konfiguration des Host-Systems. Ein Betrieb des Produkts außerhalb der bestimmungsgemäßen Verwendung, insbesondere wenn ungesicherte Schnittstellen einem nicht vertrauenswürdigen Netzwerk zugänglich gemacht werden, kann die Sicherheit und Zuverlässigkeit des Systems beeinträchtigen und ist nicht durch die Cybersicherheits-Risikobewertung des Produkts abgedeckt.

Warnung: Der *rc_visard* ist **NICHT** für sicherheitskritische Anwendungen bestimmt.

Der *rc_visard* darf nur im Rahmen seiner technischen Spezifikation verwendet werden. Jede andere Verwendung des Produkts gilt als nicht bestimmungsgemäße Verwendung. Roboception haftet nicht für Schäden, die aus unsachgemäßer oder nicht bestimmungsgemäßer Verwendung entstehen.

Warnung: Die lokalen und/oder nationalen Gesetze, Vorschriften und Richtlinien zu Automationssicherheit und allgemeiner Maschinensicherheit sind stets einzuhalten.

3 Hardware-Spezifikation

Bemerkung: Die folgenden Hardware-Spezifikationen sind als allgemeine Richtlinie angegeben. Das Produkt kann hiervon abweichen.

3.1 Lieferumfang

Der Lieferumfang eines *rc_visard* umfasst üblicherweise lediglich den *rc_visard*-Sensor und die Kurzanleitung. Das Handbuch liegt in digitaler Form vor, ist im Sensor hinterlegt und lässt sich zudem über die *Web GUI* (Abschnitt 6.1) oder über die Roboception-Homepage https://roboception.com/resources/knowledge-base/ aufrufen.

Bemerkung: Folgende Elemente sind, sofern nicht anders angegeben, NICHT im Lieferumfang enthalten:
- Kupplungen, Adapter, Halterungen,
- Netzteil, Kabel und Sicherungen,
- Netzwerkkabel.

In Abschnitt *Zubehör* (Abschnitt 8) ist angegeben, welche Kabelanbieter empfohlen werden.

Für den *rc_visard* ist ein Anschlussset verfügbar. Dieses Set umfasst das M12/RJ45-Netzwerkkabel, ein 24-V-Netzteil und einen DC/M12-Adapter. Für nähere Informationen siehe *Zubehör* (Abschnitt 8).

Bemerkung: Das Anschlussset ist lediglich für die Ersteinrichtung, nicht jedoch für die dauerhafte Installation im industriellen Umfeld gedacht.

Das folgende Bild zeigt die wichtigsten Bauteile des *rc_visard*, auf die in diesem Handbuch Bezug genommen wird.

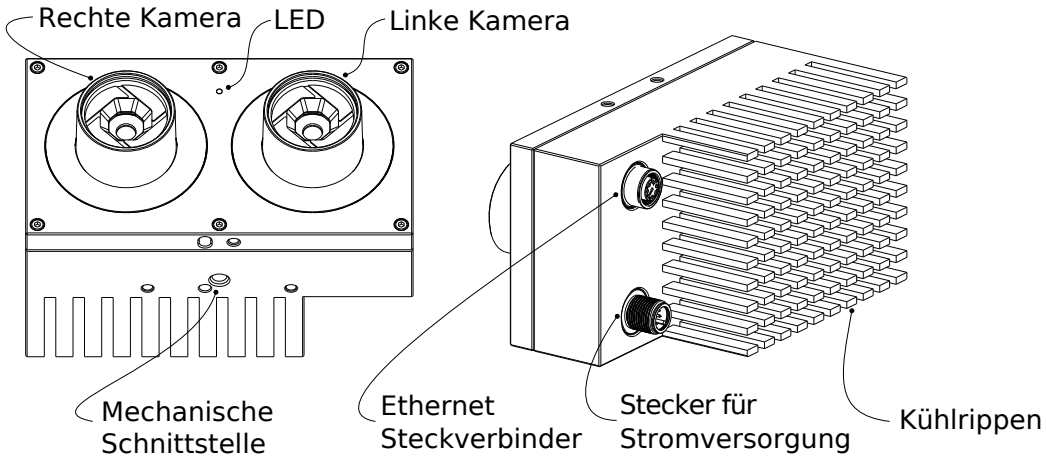


Abb. 3.1: Beschreibung der Bauteile

3.2 Technische Spezifikation

Tab. 3.1 enthält die gemeinsame technische Spezifikation für alle *rc_visard*-Varianten. Der *rc_visard* 160 ist mit zwei Arten von Objektiven erhältlich: 4 mm und 6 mm Brennweite. Der *rc_visard* 65 ist nur mit 4 mm Objektiven erhältlich.

Tab. 3.1: Gemeinsame technische Spezifikation für beide *rc_visard* Basisabstände.

	rc_visard 65 / *rc_visard* 160
Bildauflösung	1280 x 960 Pixel, farbig oder monochrom
Sichtfeld	4 mm Objektiv: Horizontal: 61°, Vertikal: 48° 6 mm Objektiv: Horizontal: 43°, Vertikal: 33°
IR Filter	650 nm
Tiefenbild	1280 x 960 pixel (Full) bei 1 Hz (mit StereoPlus-Lizenz) 640 x 480 Pixel (High) bei 3 Hz 320 x 240 Pixel (Medium) bei 15 Hz 214 x 160 Pixel (Low) bei 25 Hz
Eigenbewegung	200 Hz, geringe Latenz
GPU/CPU	Nvidia Tegra K1
Stromversorgung	18–30 V
Kühlung	Passiv

Der *rc_visard* 65 und der *rc_visard* 160 unterscheiden sich in ihren Basisabständen, was sich einerseits auf den Tiefenmessbereich und die Auflösung und andererseits auf die Größe und das Gewicht des Sensors auswirkt.

Tab. 3.2: Unterschiedliche technische Spezifikation für die *rc_visard*-Varianten

	rc_visard 65	*rc_visard* 160
Basisabstand	65 mm	160 mm
Tiefenmessbereich	0,2 m bis unendlich	0,5 m bis unendlich
Abmessungen (B x H x L)	135 mm x 75 mm x 96 mm	230 mm x 75 mm x 84 mm
Gewicht	0,68 kg	0,84 kg

Die Kombination der Basisabstände und Objektive führt zu verschiedenen Auflösungen und Genauigkeiten.

Tab. 3.3: Auflösung und Genauigkeit der *rc_visard*-Varianten in Millimetern, mit Stereo-Matching in voller Auflösung und Random-Dot-Projektion auf nicht-reflektierenden und nicht-transparenten Objekten.

	Abstand (mm)	*rc_visard* 65-4	*rc_visard* 160-4	*rc_visard* 160-6
laterale Auflösung (mm)	200 500 1000 2000 3000	0.2 0.5 0.9 1.9 2.8	- 0.5 0.9 1.9 2.8	- 0.3 0.6 1.3 1.9
Tiefenauflösung (mm)	200 500 1000 2000 3000	0.04 0.2 0.9 3.6 8.0	- 0.1 0.4 1.5 3.3	- 0.06 0.3 1.0 2.2
Mittlere Tiefengenauigkeit (mm)	200 500 1000 2000 3000	0.2 0.9 3.6 14.2 32.1	- 0.4 1.5 5.8 13.0	- 0.3 1.0 3.9 8.8

Der *rc_visard* kann für zusätzliche Funktionalitäten mit On-Board-Softwaremodulen ausgestattet werden. Diese Softwaremodule können bei Roboception bestellt werden und benötigen ein Lizenz-Update.

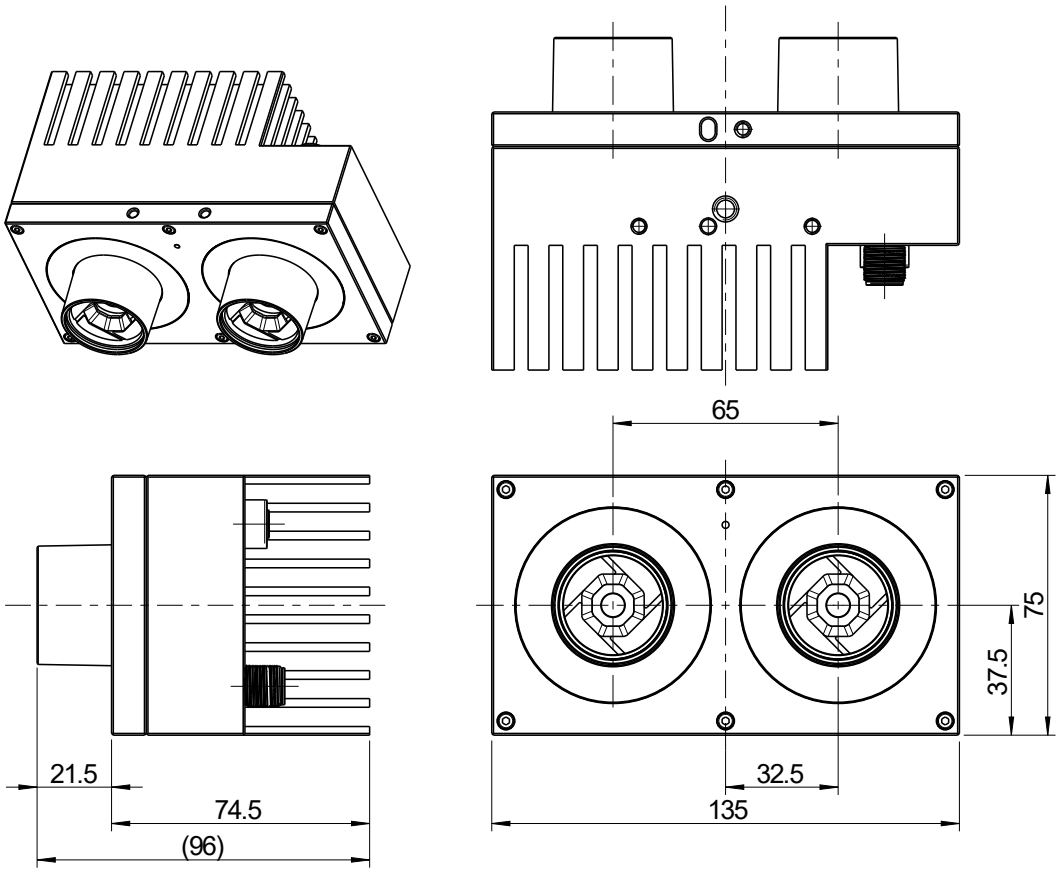


Abb. 3.2: Abmessungen des *rc_visard* 65

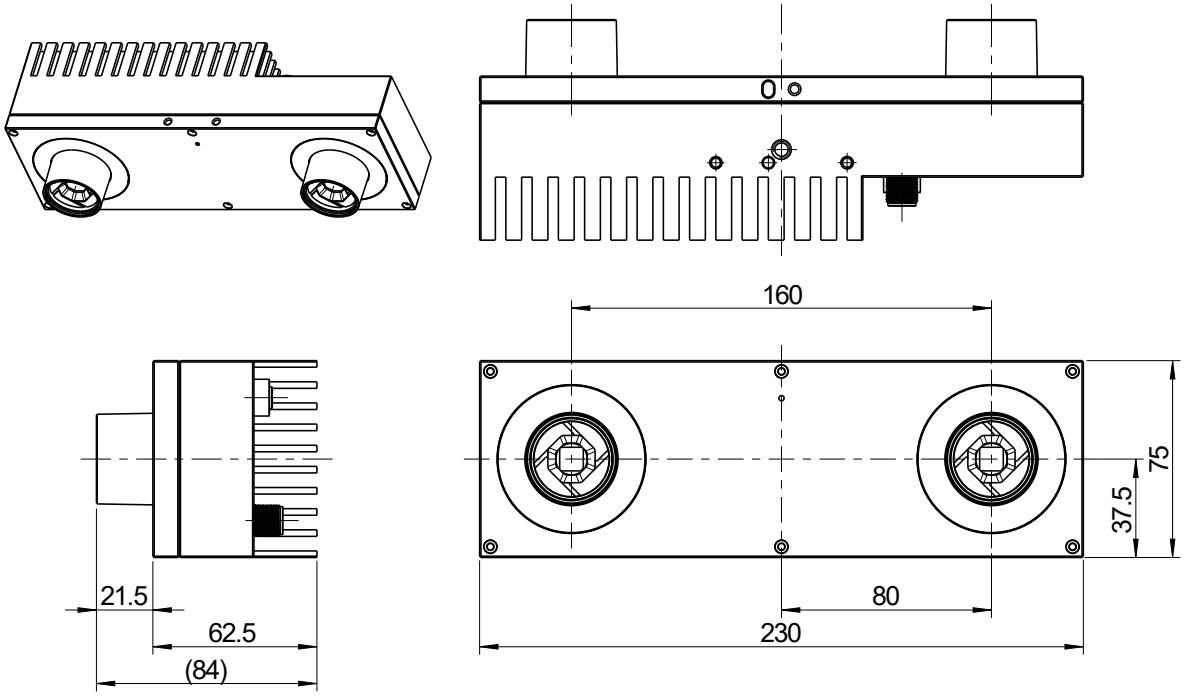


Abb. 3.3: Abmessungen des *rc_visard* 160

CAD-Modelle des *rc_visard* können von der Roboception-Homepage heruntergeladen werden: http://www.roboception.com/download. Die CAD-Modelle werden nach bestem Wissen und Gewissen, aber ohne Garantie für die Richtigkeit bereitgestellt. Wird als Materialeigenschaft Aluminium zugewiesen (Dichte: $2.76 \frac{\mathrm{g}}{\mathrm{cm}^3}$), weicht das CAD-Modell in Bezug auf Gewicht und Massenschwerpunkt nicht mehr als fünf Prozent und in Bezug auf das Trägheitsmoment nicht mehr als zehn Prozent vom Produkt ab.

3.3 Umwelt- und Betriebsbedingungen

Der *rc_visard* ist für industrielle Anwendungen konzipiert worden. Die in Tab. 3.4 angegebenen Umweltbedingungen für die Lagerung, den Transport und den Betrieb sind ausnahmslos einzuhalten.

Tab. 3.4: Umweltbedingungen

	rc_visard
Lager-/Transporttemperatur	-25–70 °C
Betriebstemperatur	0–50 °C
Relative Feuchte (nicht kondensierend)	20–80 %
Schwingungen	5 g
Erschütterungen	50 g
Schutzklasse	IP 54
Sonstiges	• Von korrosiven Flüssigkeiten oder Gasen fernhalten. • Von explosiven Flüssigkeiten oder Gasen fernhalten. • Von starken elektromagnetischen Störungen fernhalten.

Der *rc_visard* ist für den Betrieb bei einer Umgebungstemperatur zwischen 0 und 50 °C ausgelegt und arbeitet mit konvektiver (passiver) Kühlung. Während der Verwendung muss, insbesondere im Bereich der Kühlrippen, ein ungehinderter Luftstrom sichergestellt sein. Der *rc_visard* sollte nur mithilfe der vorgesehenen mechanischen Montageschnittstelle montiert werden. Kein Teil des Gehäuses darf während des Betriebs abgedeckt werden. Das Gehäuse muss in alle Richtungen mindestens zehn Zentimeter Abstand zu angrenzenden Elementen haben und es ist ein ausreichender Luftaustausch mit der Umgebung nötig, um eine angemessene Kühlung sicherzustellen. Die Kühlrippen müssen frei von Schmutz und anderen Verunreinigungen gehalten werden.

Die Gehäusetemperatur richtet sich nach der Verarbeitungslast, der Sensororientierung und der Umgebungstemperatur. Erreichen die frei liegenden Gehäuseflächen des Sensors eine Temperatur von mehr als 60 °C, wechselt die LED von Grün auf Rot.

Warnung: Für handgeführte Anwendungen sollte ein wärmeisolierter Griff am Sensor angebracht werden. So wird das bei Kontakt mit der 60 °C heißen Oberfläche bestehende Risiko für Brandverletzungen reduziert.

3.4 Spezifikationen für die Stromversorgung

Der *rc_visard* muss an eine Gleichspannungsquelle angeschlossen werden. Der Lieferumfang des *rc_visard* umfasst standardmäßig kein Netzteil. Das im Anschlussset enthaltene Netzteil kann für die Ersteinrichtung verwendet werden. Der Kunde ist dafür verantwortlich, bei einer dauerhaften Installation für eine geeignete Gleichspannungsquelle zu sorgen. Jeder *rc_visard* muss an eine eigene Stromquelle angeschlossen werden. Der Anschluss an ein Gebäudenetz darf nur über ein Netzteil erfolgen, das gemäß EN55011 Klasse B zertifiziert ist.

Tab. 3.5: Grenzwerte für die Stromversorgung

	Minimum	*Bemessungswert*	*Maximum*
Versorgungsspannung	18 V	24 V	30 V
Max. Leistungsaufnahme			25 W
Überstromschutz	Schutz der Stromversorgung mit einer 2-A-Sicherung		
Erfüllung der EMV-Anforderungen	siehe *Normen* (Abschnitt 1.3.3)		

Warnung: Die Überschreitung der maximalen Bemessungswerte kann zu Schäden am *rc_visard*, am Netzteil und an angeschlossener Ausrüstung führen.

Warnung: Jeder *rc_visard* muss von einem eigenen Netzteil versorgt werden.

Warnung: Der Anschluss an das Gebäudenetz darf nur über Netzteile erfolgen, die gemäß EN 55011 als Gerät der Klasse B zertifiziert sind.

3.5 Verkabelung

Die Kabel sind nicht im Standardlieferumfang des *rc_visard* enthalten. Es obliegt dem Kunden, geeignete Kabel zu beschaffen. In *Zubehör* (Abschnitt 8) ist eine Übersicht über die empfohlenen Komponenten enthalten.

Warnung: Die Richtlinien zum Kabelmanagement sind zwingend einzuhalten. Kabel sind immer mit einer Zugentlastung an der Halterung des *rc_visard* zu befestigen, sodass durch Kabelbewegungen keine Kräfte auf die M12-Anschlüsse des *rc_visard* wirken. Die verwendeten Kabel müssen lang genug sein, damit sich der *rc_visard* voll bewegen kann, ohne dass das Kabel zu stark belastet wird. Der minimale Biegeradius des Kabels muss beachtet werden.

Der *rc_visard* besitzt eine industrielle, achtpolige M12-Buchse (A-kodiert) für die Ethernet-Verbindung und einen achtpoligen M12-Stecker (A-kodiert) für den Stromanschluss und die GPIO-Konnektivität. Beide Anschlüsse befinden sich an der Rückwand des Geräts. Ihre Position (Abstand von Mittellinien) ist beim *rc_visard* 65 und beim *rc_visard* 160 identisch. Die Lage der beiden Anschlüsse am *rc_visard* wird in Abb. 3.4 dargestellt.

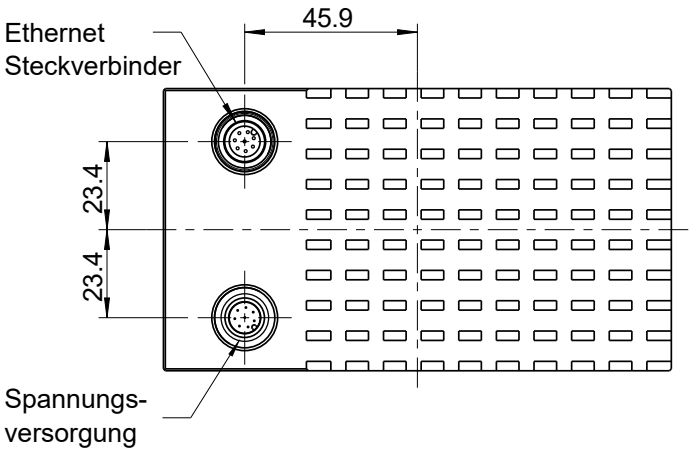


Abb. 3.4: Lage der elektrischen Anschlüsse des *rc_visard* für die Ethernetverbindung (oben) und die Stromversorgung (unten)

Die Anschlüsse sind so gedreht, dass die üblicherweise 90° abgewinkelten Stecker horizontal abgehen und von der Kamera (und den Kühlrippen) wegzeigen.

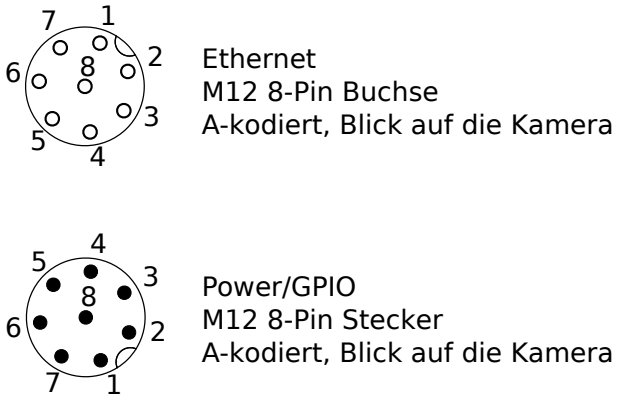


Abb. 3.5: Steckerbelegung für den Strom- und Ethernetanschluss

Die Steckerbelegung für den Ethernetanschluss ist in Abb. 3.6 angegeben.

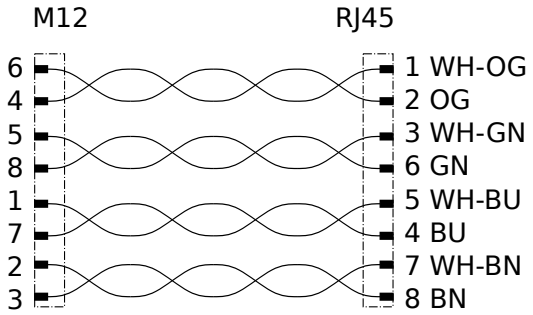


Abb. 3.6: Steckerbelegung für die M12/Ethernet-Verkabelung

Die Steckerbelegung für den Stromanschluss ist in Tab. 3.6 angegeben.

Tab. 3.6: Steckerbelegung für den Stromanschluss

Pos.	*Belegung*
1	GPIO Eingang 2
2	Stromzufuhr
3	GPIO Eingang 1
4	GPIO Masse
5	GPIO Vcc
6	GPIO Ausgang 1 (Bildbelichtung)
7	Masse
8	GPIO Ausgang 2

Die GPIO-Signale werden über Optokoppler entkoppelt. *GPIO Ausgang 1* bietet standardmäßig ein Signal zur Belichtungssynchronisierung und hat für die Dauer der Belichtung einen logischen HIGH-Pegel. Alle GPIOs können über das IOControl-Modul kontrolliert werden (*IOControl und Projektor-Kontrolle*, Abschnitt 5.5.4). Pins von unbenutzten GPIOs sollten ungeerdet bleiben.

Warnung: Es ist besonders wichtig, dass *GPIO Eingang 1* während des Boot-Vorgangs ungeerdet oder auf LOW gesetzt ist. Der *rc_visard* fährt nicht hoch, wenn der Pin während des Boot-Vorgangs auf HIGH gesetzt ist.

Das GPIO-Schaltschema und die zugehörigen Spezifikationen sind in Abb. 3.7 angegeben. Die maximale Spannung für *GPIO Eingang* und *GPIO Vcc* beträgt 30 V.

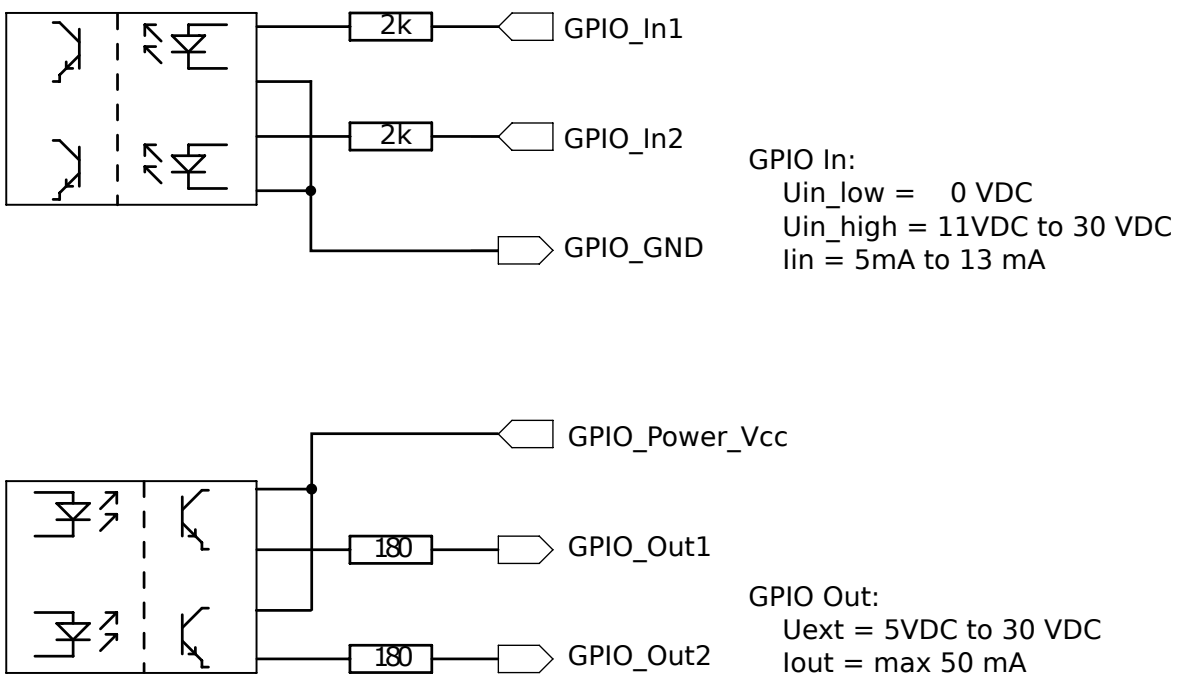


Abb. 3.7: GPIO-Schaltschema und zugehörige Spezifikationen: Keine Signale über 30 V anschließen!

Warnung: Schließen Sie keine Signale mit Spannungen über 30 V an den *rc_visard* an.

3.6 Mechanische Schnittstelle

Der *rc_visard* 65 und der *rc_visard* 160 verfügen an der Unterseite über eine identische Montageschnittstelle.

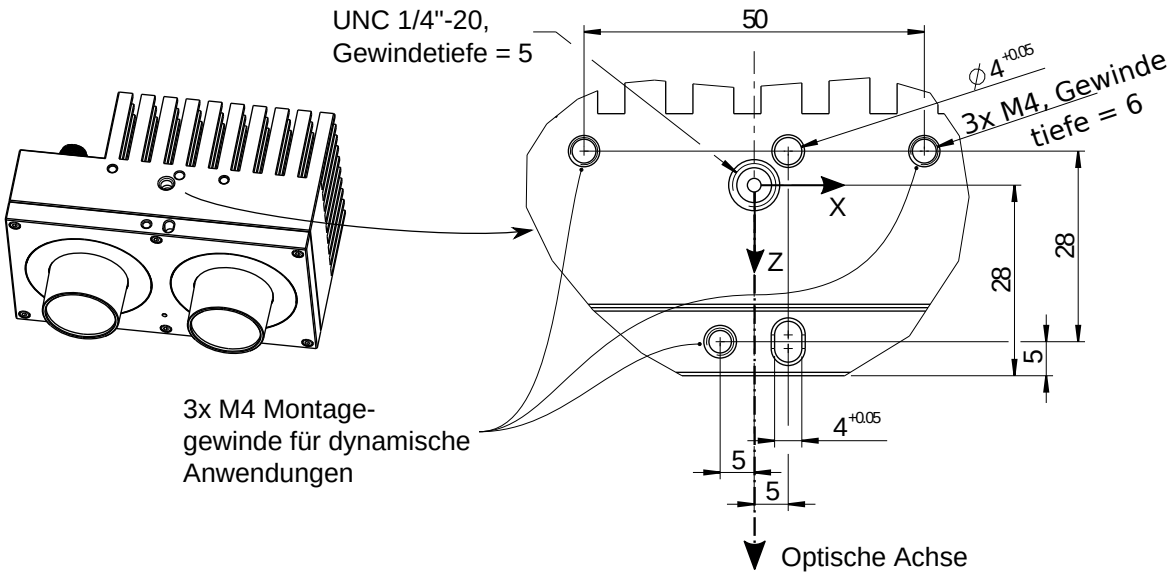


Abb. 3.8: Montagepunkt für den Anschluss des *rc_visard* an Roboter oder andere Vorrichtungen

Zur Fehlerbehebung sowie zu Konfigurationszwecken kann der Sensor über die am Koordinatenursprung angegebene, genormte Stativaufnahme (Gewinde: 1/4 Zoll x 20) montiert werden. Für dynamische Anwendungen, wie für die Montage an einem Roboterarm, muss der Sensor mit drei M4-8.8-Maschinenschrauben befestigt werden, die mit einem Drehmoment von 2,5 Nm anzuziehen und mit einer mittelfesten Gewindesicherung, wie Loctite 243, zu sichern sind. Die maximale Einschraubtiefe beträgt 6 mm. Die beiden Löcher mit einem Durchmesser von 4 mm können für Positionsstifte (ISO 2338 4 m6) verwendet werden, damit der Sensor präzise positioniert wird.

Warnung: Für dynamische Anwendungen muss der *rc_visard* mit drei M4-8.8-Maschinenschrauben befestigt werden, die mit einem Drehmoment von 2,5 Nm anzuziehen und mit einer mittelfesten Gewindesicherung zu sichern sind. Es dürfen keine hochfesten Schrauben verwendet werden. Die Einschraubtiefe muss wenigstens 5 mm betragen.

3.7 Koordinatensysteme

Der Ursprung des *rc_visard*-Koordinatensystems liegt in der Austrittspupille der linken Kameralinse. Dieses System wird auch als Sensor- oder Kamerakoordinatensystem bezeichnet. Die ungefähre Lage für den *rc_visard* wird auf dem nächsten Bild gezeigt.

Das Montagepunkt-Koordinatensystem für den *rc_visard* sitzt an der Unterseite, zentriert auf dem Gewinde, wobei die Ausrichtung der des Sensor-Koordinatensystems entspricht.

Abb. 3.9 und Abb. 3.10 zeigen den ungefähren Versatz.

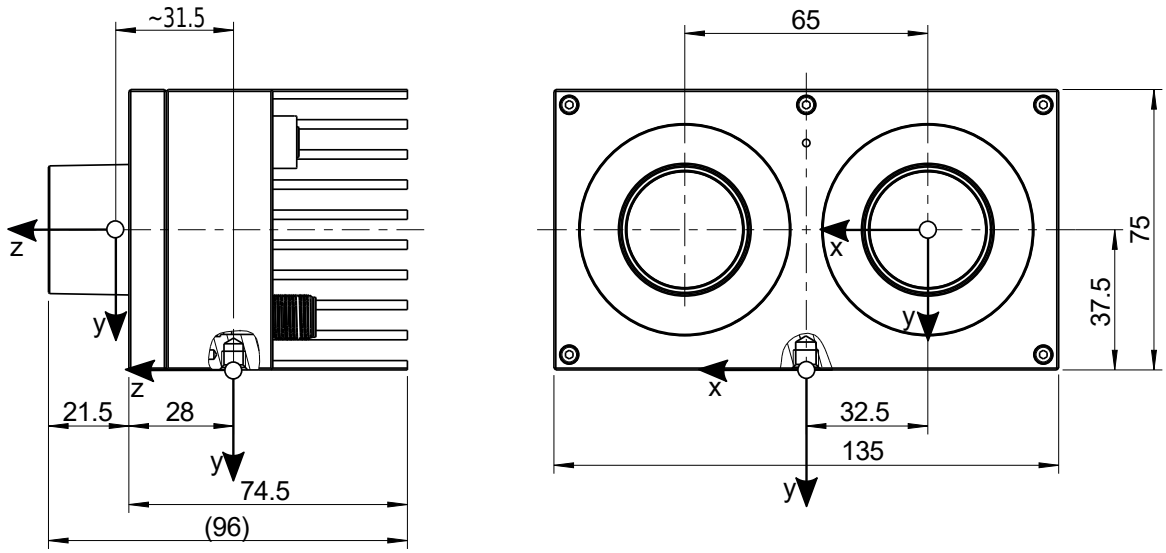


Abb. 3.9: Ungefähre Position des Sensor-/Kamerakoordinatensystems (in der linken Kameralinse) und des Montagepunkt-Koordinatensystems (am Stativgewinde) für den *rc_visard* 65

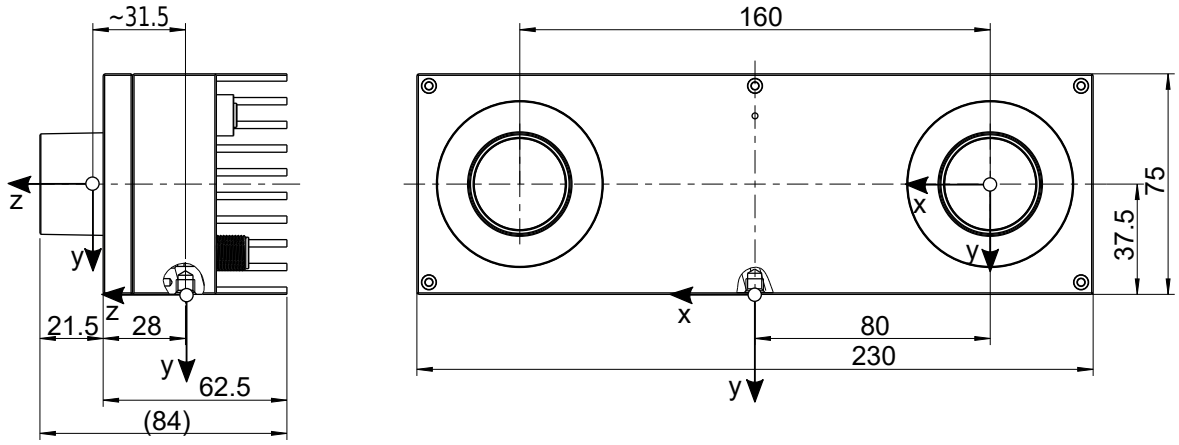


Abb. 3.10: Ungefähre Position des Sensor-/Kamerakoordinatensystems (in der linken Kameralinse) und des Montagepunkt-Koordinatensystems (am Stativgewinde) für den *rc_visard* 160

Bemerkung: Der korrekte Versatz zwischen dem Sensor-/Kamerakoordinatensystem und einem Roboterkoordinatensystem kann über die *Hand-Auge Kalibrierung* (Abschnitt 5.5.1) bestimmt werden.

4 Installation

Warnung: Vor Installation des Gerätes müssen die Hinweise zur *Sicherheit* (Abschnitt 2) des *rc_visard* gelesen und verstanden werden.

Für den Anschluss an ein Computernetzwerk verfügt der *rc_visard* über eine Gigabit-Ethernet-Schnittstelle. Die gesamte Kommunikation mit dem Gerät wird über diese Schnittstelle abgewickelt. Der *rc_visard* besitzt zudem eine eigene Prozessierungseinheit, welche nach dem Starten des Geräts eine gewisse Zeit für den Boot-Vorgang benötigt.

4.1 Softwarelizenz

Jeder *rc_visard* wird mit einer vorinstallierten Lizenzdatei zur Lizenzierung und zum Schutz der installierten Softwarepakete ausgeliefert. Die Lizenz ist an den spezifischen *rc_visard* gebunden und kann nicht auf andere Geräte übertragen oder mit diesen verwendet werden.

Die Funktionalität des *rc_visard* kann jederzeit durch ein *Upgrade der Lizenz* (Abschnitt 7.4) erweitert werden – zum Beispiel für zusätzlich erhältliche, optionale Softwaremodule.

Bemerkung: Der *rc_visard* muss neu gestartet werden, sobald die Softwarelizenz geändert wurde.

Bemerkung: Der Status der Softwarelizenz kann über die verschiedenen Schnittstellen des *rc_visard* abgefragt werden, zum Beispiel über die Seite *System* → *Firmware & Lizenz* in der *Web GUI* (Abschnitt 6.1).

4.2 Einschalten

Bemerkung: Vergewissern Sie sich, *bevor* Sie die Stromzufuhr einschalten, dass der M12-Stromanschluss am *rc_visard* sicher befestigt ist.

Sobald der *rc_visard* an den Strom angeschlossen ist, schaltet sich die LED an der Gerätefront ein. Während des Boot-Vorgangs ändert sich die Farbe der LED, bis sie schließlich grün leuchtet. Dies bedeutet, dass alle Prozesse laufen und der *rc_visard* einsatzbereit ist.

Ist kein Netzwerkkabel angeschlossen bzw. das Netzwerk nicht ordnungsgemäß konfiguriert, blinkt die LED alle fünf Sekunden rot. In diesem Fall muss die Netzwerkkonfiguration des Geräts überprüft werden. Für nähere Informationen zu den LED-Farbcodes siehe *LED-Farben* (Abschnitt 9.1).

4.3 Aufspüren von *rc_visard*-Geräten

Roboception-*rc_visard*-Geräte, die eingeschaltet und mit dem lokalen Netzwerk oder direkt mit einem Computer verbunden sind, können über den Discover-Mechanismus von GigE Vision® ausfindig gemacht werden.

Das Open-Source-Tool `rcdiscover-gui` kann für Windows und Linux kostenlos heruntergeladen werden: https://github.com/roboception/rcdiscover/releases. Dieses Tool besteht für Windows 7, 10 und 11 aus einer einzigen ausführbaren Datei, die ohne Installation direkt ausgeführt werden kann. Für Linux ist ein Installationspaket für Ubuntu erhältlich.

Nach dem Start wird jedes verfügbare GigE Vision®-Gerät, und damit auch jeder verfügbare *rc_visard*, mit seinem Namen, seiner Seriennummer, der aktuellen IP-Adresse und der eindeutigen MAC-Adresse aufgelistet. Das Discovery-Tool findet alle Geräte, die sich über globale Broadcasts erreichen lassen. Es kann vorkommen, dass falsch konfigurierte Geräte aufgeführt werden, die anderen Subnetzen als dem des Computers angehören. Ein Häkchen im Discovery-Tool gibt an, ob ein Gerät richtig konfiguriert und damit auch über einen Webbrowser erreichbar ist.

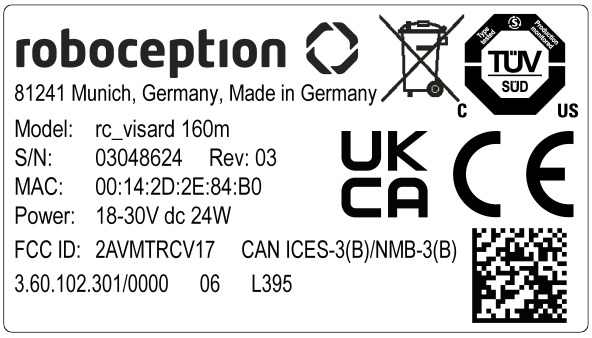


Abb. 4.1: Typenschild des *rc_visard* mit Modellart, Seriennummer und MAC-Adresse

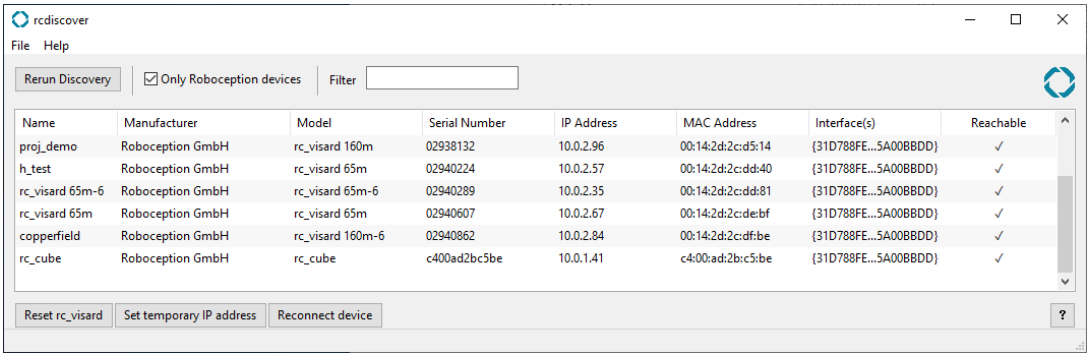


Abb. 4.2: `rcdiscover-gui`-Tool zum Aufspüren angeschlossener GigE Vision®-Geräte

Wurde das Gerät erfolgreich gefunden, öffnet sich nach einem Doppelklick auf den Geräteeintrag die *Web GUI* (Abschnitt 6.1) des Geräts im Standard-Browser des Betriebssystems. Wir empfehlen, Google Chrome oder Mozilla Firefox als Webbrowser zu verwenden.

4.3.1 Zurücksetzen der Konfiguration

Ein falsch konfiguriertes Gerät lässt sich über die Schaltfläche *Reset rc_visard* im Discovery-Tool zurücksetzen. Der Rücksetzmechanismus ist jedoch nur in den ersten beiden Minuten nach dem Gerätestart verfügbar. Daher kann es sein, dass der *rc_visard* neu gestartet werden muss, um seine Konfiguration zurückzusetzen.

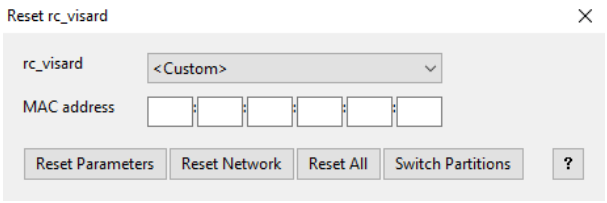


Abb. 4.3: Reset-Dialog des `rcdiscover-gui`-Tools

Wird ein *rc_visard* trotz falscher Konfiguration vom Discovery-Mechanismus erkannt, kann er aus der *rc_visard*-Dropdown-Liste gewählt werden. Anderenfalls kann die auf dem *rc_visard* aufgedruckte MAC-Adresse manuell im vorgesehenen Feld eingegeben werden.

Nach Eingabe der MAC-Adresse kann aus vier Optionen gewählt werden:

- *Reset Parameters*: Setzt alle Parameter des *rc_visard*, die über die *Web GUI* (Abschnitt 6.1) konfiguriert werden können (z.B. Bildwiederholrate), zurück.
- *Reset Network*: Setzt die Netzwerkeinstellungen und den benutzerdefinierten Namen zurück.
- *Reset All*: Setzt sowohl die *rc_visard*-Parameter als auch die Netzwerkeinstellungen und den benutzerdefinierten Namen zurück.
- *Switch Partitions*: Ermöglicht es, einen Rollback vorzunehmen, siehe *Wiederherstellung der vorherigen Firmware-Version* (Abschnitt 7.7).

Wird die LED weiß und das Gerät neu gestartet, war der Reset erfolgreich. Ist keine Reaktion erkennbar, war möglicherweise das zweiminütige Zeitfenster abgelaufen, sodass das Gerät neu gestartet werden muss.

Bemerkung: Der Rücksetzmechanismus ist nur in den ersten beiden Minuten nach dem Gerätestart verfügbar.

4.4 Netzwerkkonfiguration

Für die Kommunikation mit anderen Netzwerkgeräten muss dem *rc_visard* eine Internet-Protokoll-Adresse (*IP*) zugewiesen werden. Jede IP-Adresse darf innerhalb des lokalen Netzwerks nur einmal vergeben werden. Sie kann entweder manuell mittels einer durch den Nutzer zugewiesenen persistenten (d.h. statischen) IP-Adresse festgelegt oder automatisch per *DHCP* zugewiesen werden. Ist keine der Methoden verfügbar oder aktiviert, greift der *rc_visard* auf eine *Link-Local*-Adresse zurück.

Nach dem *GigE Vision*®-Standard wird die Konfiguration der IP-Adresse in folgender Priorität und Reihenfolge durchgeführt:

1. Persistente IP (falls aktiviert)
2. DHCP (falls aktiviert)
3. Link-Local

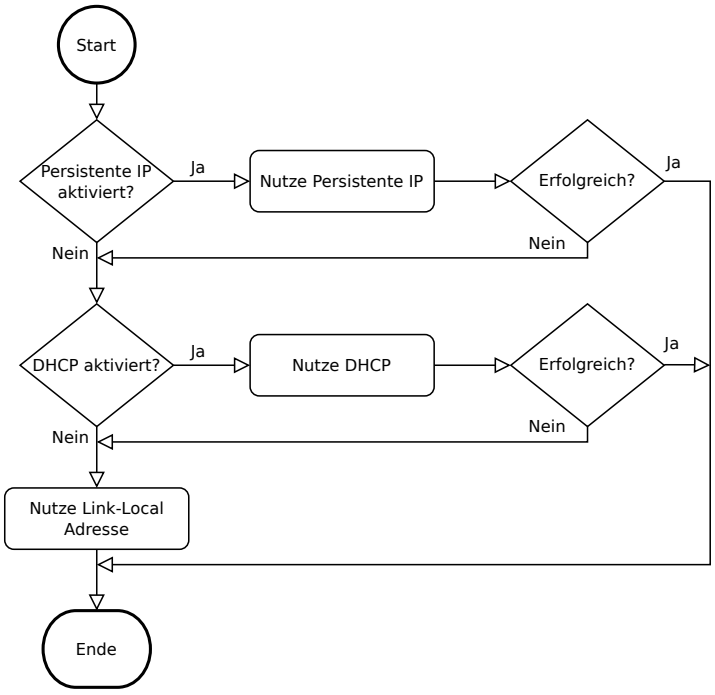


Abb. 4.4: Flussdiagramm für die Auswahl der IP-Konfigurationsmethoden des *rc_visard*

Zur Konfiguration der Netzwerkeinstellungen und IP-Adresse des *rc_visard* stehen folgende Möglichkeiten zur Verfügung:

- die Seite *System* → *Netzwerk* der *rc_visard*-Web GUI – falls diese im lokalen Netzwerk bereits erreichbar ist, siehe *Web GUI* (Abschnitt 6.1)
- alle Konfigurationsprogramme, welche mit *GigE Vision*® 2.0 kompatibel sind, oder das Roboception Kommandozeilenprogramm `gc_config`. Üblicherweise wird nach dem Start dieser Programme das Netzwerk nach allen verfügbaren GigE Vision®-Sensoren durchsucht. Alle *rc_visard*-Geräte können über ihre Seriennummer und MAC-Adresse, die beide auf dem Gerät aufgedruckt sind, eindeutig identifiziert werden.
- das Roboception `rcdiscover-gui`-Tool, welches temporäre Änderungen oder das vollständige Zurücksetzen der Netzwerkkonfiguration des *rc_visard* erlaubt, siehe *Aufspüren von rc_visard-Geräten* (Abschnitt 4.3)

Bemerkung: Das Kommandozeilenprogramm `gc_config` ist Bestandteil der Roboception Open-Source-Bibliothek `rc_genicam_api`, welche kostenlos für Windows und Linux von folgender Seite heruntergeladen werden kann: http://www.roboception.com/download.

4.4.1 Host-Name

Der Host-Name des *rc_visard* basiert auf dessen Seriennummer, welche auf dem Gerät aufgedruckt ist, und lautet `rc-visard-<serial number>`.

4.4.2 Automatische Konfiguration (werksseitige Voreinstellung)

Für die Zuweisung von IP-Adressen wird bevorzugt auf *DHCP* zugegriffen. Ist DHCP (werksseitige Voreinstellung auf dem *rc_visard*) aktiviert, versucht das Gerät, wann immer das Netzwerkkabel eingesteckt wird, einen DHCP-Server zu kontaktieren. Ist ein DHCP-Server im Netzwerk verfügbar, wird die IP-Adresse automatisch konfiguriert.

In einigen Netzwerken ist der DHCP-Server so konfiguriert, dass lediglich bekannte Geräte akzeptiert werden. In diesem Fall muss die auf dem Gerät aufgedruckte „Media Access Control“-Adresse, kurz *MAC-Adresse*, im DHCP-Server konfiguriert werden. Zudem ist der ebenfalls aufgedruckte Host-Name des *rc_visard* im Domain Name Server *DNS* einzustellen. Sowohl die MAC-Adresse als auch der Host-Name sind zu Konfigurationszwecken an den Netzwerkadministrator zu übermitteln.

Kann der *rc_visard* nicht innerhalb von 15 Sekunden nach dem Einschalten bzw. dem Einstecken des Netzwerkkabels Kontakt zu einem DHCP-Server aufbauen, versucht er, sich selbst eine eindeutige IP-Adresse zuzuweisen. Dieser Prozess heißt *Link-Local*. Diese Option ist besonders nützlich, wenn der *rc_visard* direkt an einen Computer angeschlossen werden soll. In diesem Fall muss auch der Computer mit einer Link-Local-Adresse konfiguriert sein. Bei manchen Betriebssystemen wie Windows 10 ist Link-Local bereits standardmäßig als Fallback eingestellt. Bei der Arbeit mit anderen Betriebssystemen, wie z.B. Linux, muss die Link-Local-Adresse direkt im Netzwerkmanager konfiguriert werden.

4.4.3 Manuelle Konfiguration

In einigen Fällen kann es nützlich sein, manuell eine persistente, d.h. statische IP-Adresse einzurichten. Diese wird auf dem *rc_visard* gespeichert und beim Systemstart bzw. beim Verbindungswiederaufbau zugewiesen. Bitte stellen Sie sicher, dass die Einstellungen der IP-Adresse, der Subnetz-Maske und des Default-Gateway keine Konflikte im Netzwerk verursachen.

Warnung: Die IP-Adresse muss eindeutig sein und innerhalb des Gültigkeitsbereichs des lokalen Netzwerks liegen. Zudem muss die Subnetz-Maske dem lokalen Netzwerk entsprechen, da anderenfalls möglicherweise nicht auf den *rc_visard* zugegriffen werden kann. Dieses Problem lässt sich vermeiden, indem die unter *Automatische Konfiguration (werksseitige Voreinstellung)* (Abschnitt 4.4.2) beschriebene automatische Konfiguration genutzt wird.

Kann die gewählte IP-Adresse nicht zugewiesen werden, zum Beispiel weil ein anderes Gerät im lokalen Netzwerk diese bereits verwendet, wird auf die automatische IP-Konfiguration mittels *DHCP* (falls aktiviert) oder *Link-Local* zurückgegriffen.

5 Softwaremodule

Der *rc_visard* beinhaltet eine Reihe von Softwaremodulen mit verschiedenen Funktionalitäten. Jedes Softwaremodul bietet über seine zugehörige *Node* eine Schnittstelle über *REST-API-Schnittstelle* (Abschnitt 6.2) oder das *Generic Robot Interface* (Abschnitt 6.4) an.

Die Softwaremodule des *rc_visard* können unterteilt werden in

- ***Kamera Modul*** **(Abschnitt 5.1)** erfasst Bildpaare und führt die planare Rektifizierung durch, wodurch die Kamera als Messinstrument verwendet werden kann. Bilder werden sowohl für die weitere interne Verarbeitung durch andere Module als auch als *GenICam-Bild-Streams* für die externe Verwendung bereitgestellt.
- ***Stereo-Matching Modul*** **(Abschnitt 5.2)** welche 3D Tiefeninformationen, wie Disparitäts-, Fehler- und Konfidenzbilder, bereitstellen.
- ***Navigationsmodule*** **(Abschnitt 5.3)** welche Schätzungen der momentanen Pose, Geschwindigkeit und Beschleunigung des *rc_visard* bereitstellen,
- ***Detektions- und Messmodule*** **(Abschnitt 5.4)** welche eine Vielzahl verschiedener Detektionsfunktionen, wie Greifpunktberechnungen und Objekterkennung anbieten.
- ***Konfigurationsmodule*** **(Abschnitt 5.5)** welche es dem Nutzer ermöglichen, Kalibrierungen durchzuführen und den *rc_visard* für spezielle Anwendungen zu konfigurieren.
- ***Datenbankmodule*** **(Abschnitt 5.6)** welche dem Nutzer die Konfiguration globaler Daten ermöglichen, die in allen anderen Modulen verfügbar sind, wie Load Carrier, Regions of Interest und Greifer.

5.1 Kamera Modul

Das Kameramodul ist ein Basismodul welches auf jedem *rc_visard* verfügbar ist. Es ist für die Bildakquise und die Rektifizierung der Bilder verantwortlich. Das Modul bietet diverse Parameter um z.B. die Belichtungszeit oder die Bildwiederholrate zu verändern.

5.1.1 Parameter, Statuswerte und Services

5.1.1.1 Parameter

Das Kamera-Modul wird in der REST-API als `rc_camera` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *Kamera* in der gewünschten Pipeline dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.1: Laufzeitparameter des `rc_camera`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`exp_auto`	bool	false	true	true	Umschalten zwischen automatischer und manueller Belichtung (veraltet, nutzen Sie stattdessen exp_control)
`exp_auto_average_max`	float64	0.0	1.0	0.75	Maximaler Belichtungsmittelwert im Auto Belichtungsmodus
`exp_auto_average_min`	float64	0.0	1.0	0.25	Maximaler Belichtungsmittelwert im Auto Belichtungsmodus
`exp_auto_mode`	string	-	-	Normal	Modus für automatische Belichtung: [Normal, Out1High, AdaptiveOut1]
`exp_control`	string	-	-	Auto	Art der Belichtungsregelung: [Manual, Auto, HDR]
`exp_height`	int32	0	959	0	Höhe der Region für automatische Belichtung, 0 für das ganze Bild
`exp_max`	float64	6.6e-05	0.018	0.018	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_offset_x`	int32	0	1279	0	Erste Spalte der Region für automatische Belichtung
`exp_offset_y`	int32	0	959	0	Erste Zeile der Region für automatische Belichtung
`exp_value`	float64	6.6e-05	0.018	0.005	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_width`	int32	0	1279	0	Breite der Region für automatische Belichtung, 0 für das ganze Bild
`fps`	float64	1.0	25.0	25.0	Bildwiederholrate in Hertz
`gain_value`	float64	0.0	18.0	0.0	Verstärkung in Dezibel, wenn nicht im Auto Belichtungsmodus
`gamma`	float64	0.1	10.0	1.0	Gammafaktor
`wb_auto`	bool	false	true	true	Ein- und Ausschalten des manuellen Weißabgleichs (nur für Farbkameras)
`wb_ratio_blue`	float64	0.125	8.0	2.4	Blau-zu-Grün-Verhältnis, falls wb_auto auf false gesetzt ist (nur für Farbkameras)
`wb_ratio_red`	float64	0.125	8.0	1.2	Rot-zu-Grün-Verhältnis, falls wb_auto auf false gesetzt ist (nur für Farbkameras)

Die gleichen Parameter sind – mit leicht abweichenden Namen und teilweise mit anderen Einheiten oder Datentypen – auch über die GenICam-Schnittstelle verfügbar (siehe *GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.7).

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet.

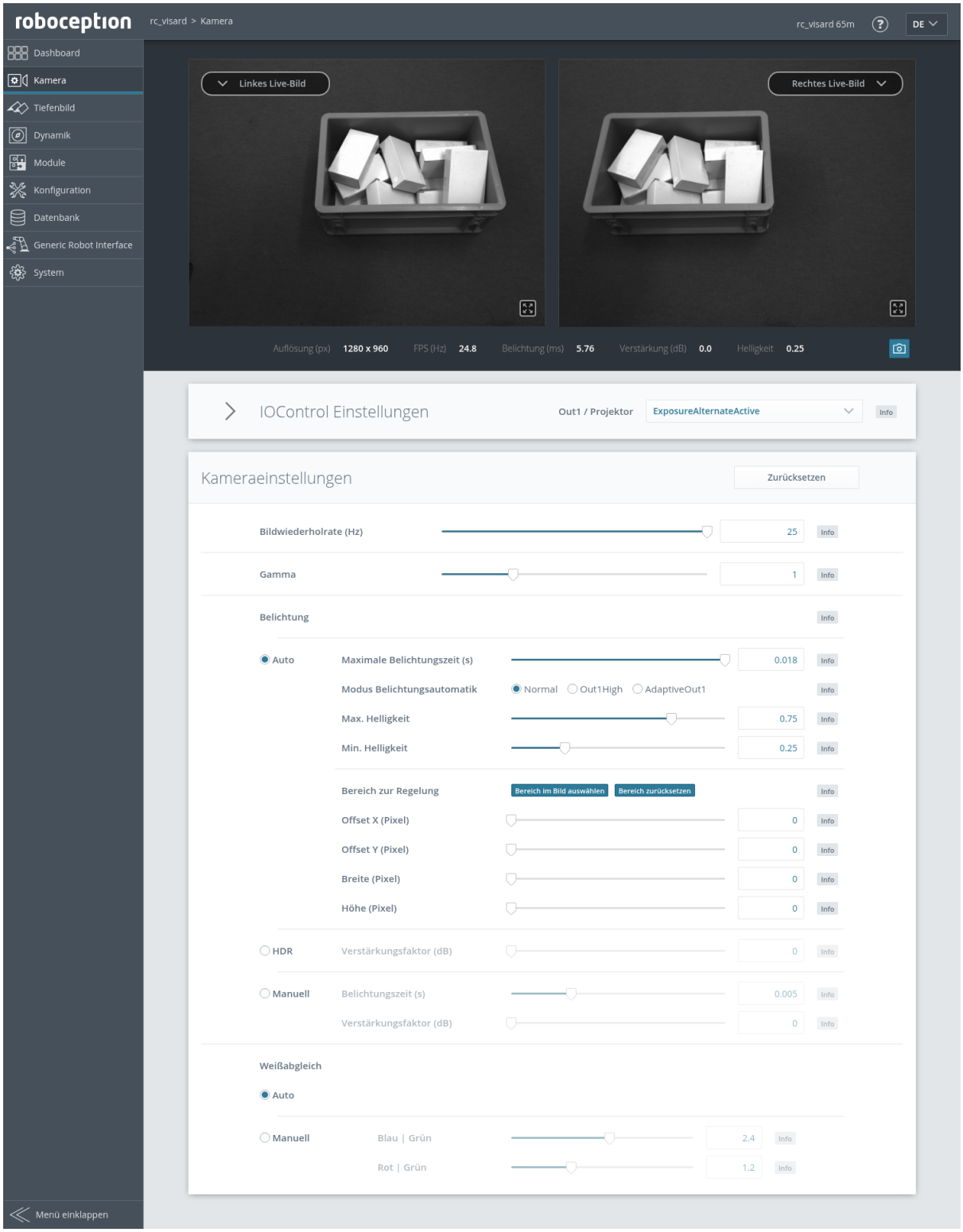


Abb. 5.1: Seite *Kamera* in der Web GUI

`fps` (*Bildwiederholrate (Hz)*)

Dieser Wert bezeichnet die Bildwiederholrate der Kamera in Bildern pro Sekunde und begrenzt zugleich die Frequenz, mit der Tiefenbilder berechnet werden können. Die Bildwiederholrate entspricht auch der Frequenz, mit welcher der *rc_visard* Bilder über GigE Vision

bereitstellt. Wird diese Frequenz verringert, reduziert sich auch die zur Übertragung der Bilder benötigte Bandbreite des Netzwerks.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?fps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?fps=<value>
```

Die Kamera läuft immer mit 25 Hz, um die Funktion von internen Modulen, die eine konstante Bildwiederholrate benötigen (wie zum Beispiel die visuelle Odometrie), sicherzustellen. Die vom Benutzer definierte Bildwiederholrate wird, wie in der folgenden Abbildung gezeigt, durch das Weglassen von Bildern erreicht, die für das Stereo-Matching und das Übertragen per GigE Vision benutzt werden. Letzteres dient der Reduktion der Bandbreite.



Abb. 5.2: Die interne Bildaufnahme geschieht immer mit 25 Hz. Der `fps` Parameter bestimmt, wie viele dieser Kamerabilder per GigE Vision versendet werden.

`gamma` (*Gamma*)

Der Gammawert bestimmt, wie das gemessene Licht auf die Helligkeit eines Pixels abgebildet wird. Ein Gammawert von 1 entspricht einem linearen Zusammenhang. Kleinere Gammawerte lassen dunkle Bildbereiche heller erscheinen. Ein Wert um 0.5 entspricht der menschlichen Wahrnehmung.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?gamma=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gamma=<value>
```

`exp_control` (Belichtung *Auto*, *HDR* oder *Manual*)

Die Belichtungsregelung kann auf *Auto*, *HDR* oder *Manual* gesetzt werden. Dies ersetzt den veralteten *exp_auto* Parameter.

Auto: Dies ist der Standard Modus der die die Belichtungszeit und Verstärkung automatisch anpasst, um Unter- und Überbelichtung zu vermeiden. Wenn die Automatik abgeschaltet wird, werden *exp_value* und *gain_value* auf die letzten von der Automatik ermittelten Werte für Belichtungszeit und Verstärkung gesetzt.

HDR: Der HDR Modus berechnet Bilder mit hohem Dynamikbereich durch Kombination von Bildern mit unterschiedlichen Belichtungszeiten um über- und unterbelichtete Bereiche zu vermeiden. Dieser Modus verringert die Bildwiederholrate und ist nur für statische Szenen geeignet.

Manual: Im manuellen Belichtungsmodus werden die Belichtungszeit und die Verstärkung konstant gehalten unabhängig von der resultierenden Bildhelligkeit.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?exp_control=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_control=<value>
```

`exp_auto_mode` (*Modus Belichtungszeitautomatik*)

Der Modus für automatische Belichtung kann auf *Normal*, *Out1High* oder *AdaptiveOut1* gesetzt werden. Diese Modi sind nur relevant, wenn der *rc_visard* mit einer externen Lichtquelle oder einem Projektor betrieben wird, der an den GPIO-Ausgang 1 der Kamera angeschlossen ist. Dieser Ausgang kann durch das IOControl-Modul (*IOControl und Projektor-Kontrolle*, Abschnitt 5.5.4) gesteuert werden.

Normal: Alle Bilder werden für die Regelung der Belichtungszeit in Betracht gezogen, außer wenn der IOControl-Modus für den GPIO-Ausgang 1 *ExposureAlternateActive* ist: Dann werden nur Bilder berücksichtigt, bei denen GPIO-Ausgang 1 HIGH ist, da diese Bilder heller sein können, falls dieser GPIO-Ausgang benutzt wird um einen externen Projektor auszulösen.

Out1High: Die Belichtungszeit wird nur anhand der Bilder mit GPIO-Ausgang 1 HIGH angepasst. Bilder bei denen GPIO-Ausgang 1 LOW ist, werden für die Belichtungszeitregelung nicht berücksichtigt. Das bedeutet, die Belichtungszeit ändert sich nicht, solange nur Bilder mit GPIO-Ausgang 1 LOW aufgenommen werden. Dieser Modus wird für die Benutzung mit dem Single+Out1 Tiefenbild Aufnahmemodus (siehe *Stereo Matching Parameters*, 5.2.1 und externem Projektor empfohlen, wenn die Helligkeit der Szene nur zu den Zeitpunkten berücksichtigt werden soll, wenn GPIO-Ausgang 1 HIGH ist. Das ist zum Beispiel der Fall, wenn kurz vor einer Objekterkennung ein heller Teil des Roboters durch das Bild fährt, der die Belichtungseinstellungen jedoch nicht beeinflussen soll.

AdaptiveOut1: Dieser Modus nutzt alle Kamerabilder und speichert die Differenz der Belichtung zwischen Bildern mit GPIO Ausgang 1 HIGH und LOW. Während der IOControl-Modus für GPIO-Ausgang 1 LOW ist, werden die Bilder um diese Differenz unterbelichtet, um eine Überbelichtung zu verhindern, sobald der externe Projektor über GPIO-Ausgang 1 ausgelöst wird. Die Differenz der Belichtung wird als *Out1 Reduktion* unter den Livebildern angezeigt. Dieser Modus wird empfohlen, wenn im Stereo-Matching-Modul der Parameter `acquisition_mode` auf `SingleFrameOut1` (*Einzelbild+Out1*) gesetzt ist (*Parameter des Stereo-Matching-Moduls*, Abschnitt 5.2.1), und ein externer Projektor an den GPIO-Ausgang 1 angeschlossen ist, und wenn die Helligkeit der Szene zu jeder Zeit zur Belichtungszeitregelung berücksichtigt werden soll.Das ist zum Beispiel in Anwendungen mit veränderlichen äußeren Lichtbedingungen der Fall.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?exp_auto_mode=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_auto_mode=<value>
```

`exp_max` (*Maximale Belichtungszeit*)

Dieser Wert gibt die maximale Belichtungszeit im automatischen Modus in Sekunden an. Die tatsächliche Belichtungszeit wird automatisch angepasst, sodass das Bild korrekt belichtet wird. Sind die Bilder trotz maximaler Belichtungszeit noch immer unterbelichtet, erhöht der *rc_visard* schrittweise die Verstärkung, um die Helligkeit der Bilder zu erhöhen. Es ist sinnvoll, die Belichtungszeit zu begrenzen, um die bei schnellen Bewegungen auftretende Bildunschärfe zu vermeiden oder zu verringern. Jedoch führt eine höhere Verstärkung auch zu mehr Bildrauschen. Welcher Kompromiss der beste ist, hängt immer auch von der Anwendung ab.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?exp_max=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_max=<value>
```

`exp_auto_average_max` (*Maximale Helligkeit*) und `exp_auto_average_min` (*Minimale Helligkeit*)

Die automatische Belichtungszeitsteuerung versucht die Belichtungszeit und den Verstärkungsfaktor so einzustellen, dass die mittlere Bildhelligkeit im Bild oder im *Bereich zur Regelung* zwischen der maximalen und minimalen Helligkeit liegt. Die maximale Helligkeit wird benutzt, wenn keine Bildteile in der Sättigung sind, d.h. keine Überbelichtung durch helle Oberflächen oder Reflexionen vorhanden sind. Falls Sättigungen auftreten, werden die Belichtungszeit und der Verstärkungsfaktor verringert, aber nur bis zur eingestellten minimalen Helligkeit.

Der Parameter für die maximale Helligkeit hat Vorrang über den Parameter der minimalen Helligkeit. Falls die minimale Helligkeit größer als die maximale ist, versucht die automatische Belichtungszeitsteuerung die mittlere Bildhelligkeit auf die maximale Helligkeit zu setzen.

Die aktuelle Helligkeit wird in der Statuszeile unter den Bildern angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?<exp_auto_
↪average_max|exp_auto_average_min>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_auto_average_max|exp_auto_
↪average_min>=<value>
```

`exp_offset_x`, `exp_offset_y`, `exp_width`, `exp_height` (*Bereich zur Regelung*)

Diese Werte definieren eine rechteckige Region im linken rektifizierten Bild, um den von der automatischen Belichtung überwachten Bereich zu limitieren. Die Belichtungszeit und der Verstärkungsfaktor werden so gewählt, dass die definierte Region optimal belichtet wird. Dies kann zu Über- oder Unterbelichtung in anderen Bildbereichen führen. Falls die Breite oder Höhe auf 0 gesetzt werden, dann wird das gesamte linke und rechte Bild von der automatischen Belichtungsfunktion berücksichtigt. Dies ist die Standardeinstellung.

Die Region wird in der Web GUI mit einem Rechteck im linken rektifizierten Bild visualisiert. Sie kann über Slider oder direkt im Bild mithilfe der Schaltfläche `Bereich im Bild auswählen` verändert werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?<exp_offset_
↪x|exp_offset_y|exp_width|exp_height>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_offset_x|exp_offset_y|exp_
↪width|exp_height>=<value>
```

`exp_value` (*Belichtungszeit*)

Dieser Wert gibt die Belichtungszeit im manuellen Modus in Sekunden an. Diese Belichtungszeit wird konstant gehalten, auch wenn die Bilder unterbelichtet sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?exp_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_value=<value>
```

`gain_value` (*Verstärkungsfaktor (dB)*)

Dieser Wert gibt den Verstärkungsfaktor im manuellen Modus in Dezibel an. Höhere Verstärkungswerte reduzieren die Belichtungszeit, führen aber zu Rauschen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?gain_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gain_value=<value>
```

`wb_auto` (*Weißabgleich Auto* oder *Manuell*)

Dieser Wert kann auf *true* gesetzt werden, um den automatischen Weißabgleich anzuschalten. Bei *false* kann das Verhältnis der Farben manuell mit `wb_ratio_red` und `wb_ratio_blue` gesetzt werden. `wb_ratio_red` und `wb_ratio_blue` werden auf die letzten von der Automatik ermittelten Werte gesetzt, wenn diese abgeschaltet wird. Der Weißabgleich ist bei monochromen Kameras ohne Funktion und wird in diesem Fall in der Web GUI nicht angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?wb_auto=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?wb_auto=<value>
```

`wb_ratio_blue` und `wb_ratio_red` (*Blau* / *Grün* and *Rot* / *Grün*)

Mit diesen Werten kann das Verhältnis von Blau zu Grün bzw. Rot zu Grün für einen manuellen Weißabgleich gesetzt werden. Der Weißabgleich ist bei monochromen Kameras ohne Funktion und wird in diesem Fall in der Web GUI nicht angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?<wb_ratio_blue|wb_ratio_
↪red>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_red>=<value>
```

5.1.1.2 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.2: Statuswerte des `rc_camera`-Moduls

Name	Beschreibung
`baseline`	Basisabstand t der Stereokamera in Metern
`brightness`	Aktuelle Helligkeit als Wert zwischen 0 und 1
`color`	0 für monochrome Kameras, 1 für Farbkameras
`device_trigger_sources`	Angabe der verfügbaren Triggerquellen
`exp`	Aktuelle Belichtungszeit in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Belichtung (ms)* angezeigt.
`focal`	Brennweitenfaktor, normalisiert auf eine Bildbreite von 1
`fps`	Aktuelle Bildwiederholrate der Kamerabilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`gain`	Aktueller Verstärkungsfaktor in Dezibel. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Verstärkung (dB)* angezeigt.
`gamma`	Aktueller Gammawert
`height`	Höhe des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
`last_timestamp_grabbed`	Zeitstempel des letzten aufgenommenen Bildes, wenn die Kamera im Triggermodus ist
`out1_reduction`	Anteil der Helligkeits-Reduktion (0.0 - 1.0) für Bilder mit GPIO-Ausgang 1=LOW, wenn exp_auto_mode=AdaptiveOut1 oder exp_auto_mode=Out1High. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Out1 Reduktion (%)* angezeigt.
`params_override_active`	1 wenn die Parameter temporär durch einen laufenden Kalibrierprozess überschrieben werden
`selfcalib_counter`	Wie oft eine Korrektur durch die Selbstkalibrierung vorgenommen wurde
`selfcalib_offset`	Aktueller Offset, der durch die Selbstkalibrierung bestimmt wurde
`temp_left`	Temperatur des linken Kamerasensors in Grad Celsius
`temp_right`	Temperatur des rechten Kamerasensors in Grad Celsius
`test`	0 for Live-Bilder und 1 für Test-Bilder
`time`	Verarbeitungszeit für die Bilderfassung in Sekunden
`width`	Breite des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

5.1.1.3 Services

Das Kamera-Modul bietet folgende Services.

`reset_defaults`

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.1.2 Rektifizierung

Eine rektifizierte Kamera kann mit der Brennweite als einzigen Modellparameter beschrieben werden. Der *rc_visard* stellt über seine verschiedenen Schnittstellen einen Brennweitenfaktor bereit. Er bezieht sich auf die Bildbreite, um verschiedene Bildauflösungen zu unterstützen. Die Brennweite f in Pixeln lässt sich leicht bestimmen, indem der Brennweitenfaktor mit der Bildbreite (in Pixeln) multipliziert wird.

5.1.3 Anzeigen und Herunterladen von Bildern

Der *rc_visard* liefert über die GenICam Schnittstelle (siehe *Verfügbare Bild-Streams*, Abschnitt 6.7.6) oder über *gRPC Bilddatenschnittstelle* (siehe Abschnitt 6.8) zeitgestempelte rektifizierte Kamerabilder.

Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 6.1) bereitgestellt.

Die Web GUI bietet weiterhin die Möglichkeit, einen Schnappschuss der aktuellen Szene als .tar.gz-Datei zu speichern, wie in *Herunterladen von Kamerabildern* (Abschnitt 6.1.4) beschrieben wird.

5.2 Stereo-Matching Modul

Das Stereo-Matching-Modul ist ein Basismodul, das auf jedem *rc_visard* verfügbar ist, und berechnet auf Grundlage des rektifizierten Stereobildpaars Disparitäts-, Fehler- und Konfidenzbilder.

Um Disparitäts-, Fehler- und Konfidenzbilder in voller Auflösung zu berechnen, wird eine gesonderte StereoPlus *Lizenz* (Abschnitt 7.4) benötigt. Diese Lizenz ist auf jedem *rc_visard* vorhanden, der nach dem 31.01.2019 gekauft wurde.

5.2.1 Parameter

Das Stereo-Matching-Modul wird in der REST-API als `rc_stereomatching` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *Tiefenbild* dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.2) ändern.

5.2.1.1 Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.3: Laufzeitparameter des `rc_stereomatching`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	Continuous	Aufnahmemodus: [Continuous, SingleFrame, SingleFrameOut1]
`double_shot`	bool	false	true	false	Kombination zweier Disparitätsbilder von zwei Stereobildpaaren
`exposure_adapt_timeout`	float64	0.0	2.0	0.0	Maximale Zeit in Sekunden, die nach Auslösen einer Aufnahme im Einzelbild-Modus gewartet wird, bis die Belichtung angepasst ist
`fill`	int32	0	4	3	Disparitätstoleranz (für das Füllen von Löchern) in Pixeln
`maxdepth`	float64	0.1	100.0	100.0	Maximaler Abstand in Metern
`maxdeptherr`	float64	0.01	100.0	100.0	Maximaler Tiefenfehler in Metern
`method`	string	-	-	SGM	Stereo-Matching-Methode, d.h. einer der Namen, die vom get_methods-Aufruf zurückgegeben werden
`minconf`	float64	0.5	1.0	0.5	Mindestkonfidenz
`mindepth`	float64	0.1	100.0	0.1	Minimaler Abstand in Metern
`quality`	string	-	-	High	Full (Voll), High (Hoch), Medium (Mittel), oder Low (Niedrig). Full benötigt eine ‚StereoPlus'-Lizenz.
`roi_height`	int32	0	960	0	Höhe der Region of Interest. 0 für das gesamte Bild.
`roi_offset_x`	int32	0	1280	0	Erste Spalte der Region of Interest
`roi_offset_y`	int32	0	960	0	Erste Zeile der Region of Interest
`roi_width`	int32	0	1280	0	Breite der Region of Interest. 0 für das gesamte Bild.
`seg`	int32	0	4000	200	Mindestgröße der gültigen Disparitätssegmente in Pixeln
`smooth`	bool	false	true	true	Glättung von Disparitätsbildern (benötigt eine ‚StereoPlus'-Lizenz)
`static_scene`	bool	false	true	false	Mitteln von Bildern in statischen Szenen, um Rauschen zu reduzieren

Diese Parameter sind – mit leicht abweichenden Namen und teilweise mit anderen Einheiten oder Datentypen – auch über die GenICam-Schnittstelle verfügbar (siehe *GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.7).

5.2.1.2 Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet:

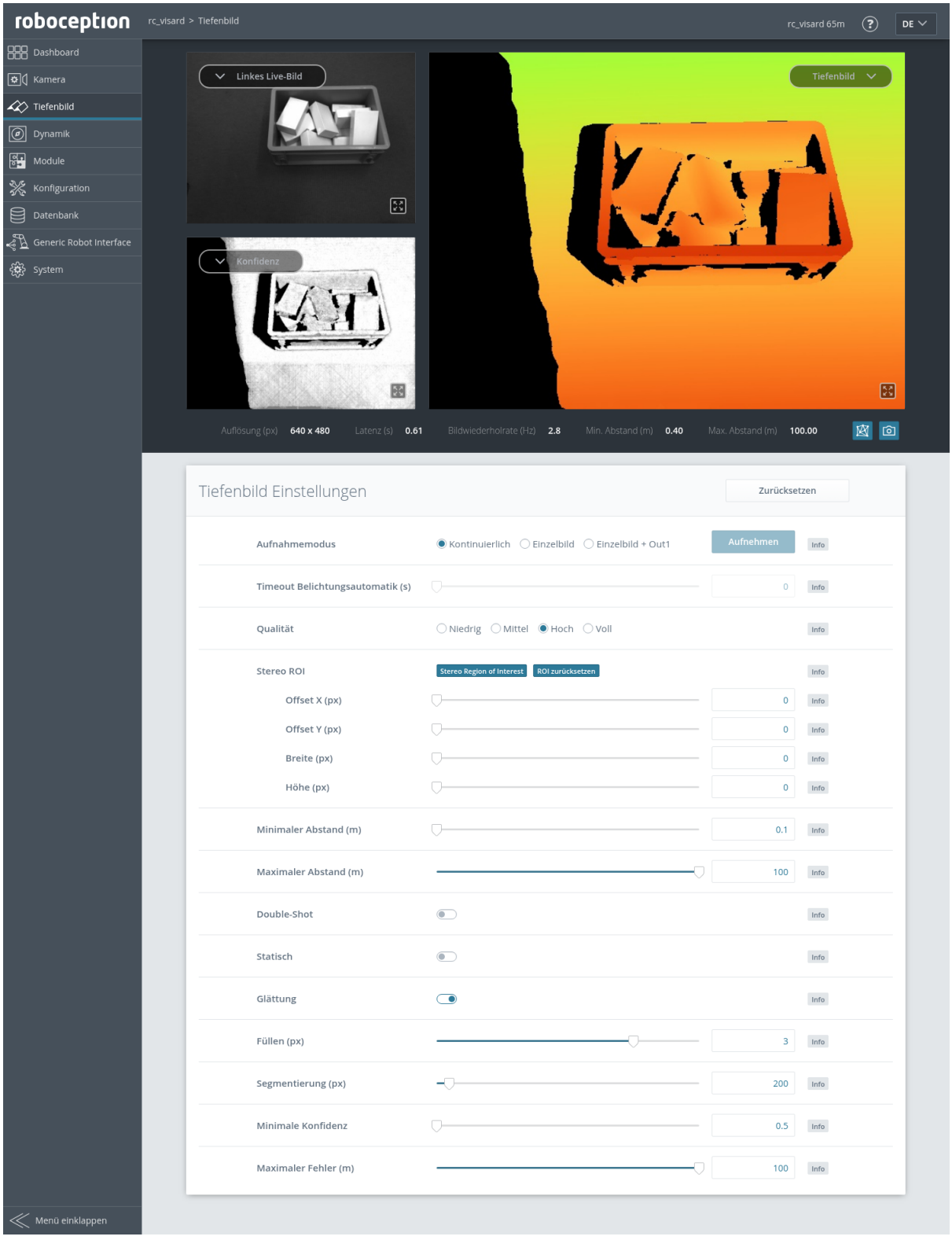


Abb. 5.3: Seite *Tiefenbild* der Web GUI

`acquisition_mode` (*Aufnahmemodus*)

Der Aufnahmemodus kann auf `Continuous` (*Kontinuierlich*), `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild + Out1*) eingestellt werden. *Kontinuier-*

lich ist die Standardeinstellung, bei der das Stereo-Matching kontinuierlich mit der vom Benutzer eingestellten Bildwiederholrate, entsprechend der verfügbaren Rechenressourcen, durchgeführt wird. Bei den beiden anderen Modi wird das Stereo-Matching bei jedem Drücken des *Aufnehmen*-Knopfes durchgeführt. Der *Einzelbild + Out1* Modus kontrolliert zusätzlich einen externen Projektor, falls dieser an GPIO-Ausgang 1 angeschlossen ist (*IOControl und Projektor-Kontrolle*, Abschnitt 5.5.4). In diesem Modus wird `out1_mode` des IOControl-Moduls automatisch bei jedem Trigger auf `ExposureAlternateActive` und nach dem Aufnehmen der Bilder für das Stereo-Matching auf `Low` gesetzt.

Bemerkung: Der *Einzelbild + Out1* Modus kann nur dann über `out1_mode` einen Projektor steuern, wenn die IOControl-Lizenz auf dem *rc_visard* verfügbar ist.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪acquisition_mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?acquisition_mode=<value>
```

`exposure_adapt_timeout` (*Timeout Belichtungsautomatik*)

Der Timeout für die Belichtungsautomatik gibt die maximale Zeitspanne in Sekunden an, die das System nach dem Auslösen einer Bildaufnahme warten wird, bis die Belichtungsautomatik die optimale Belichtungszeit gefunden hat. Dieser Timeout wird nur im Modus `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild + Out1*) bei aktiver Belichtungsautomatik verwendet. Dieser Wert sollte erhöht werden, wenn in Anwendungen mit veränderlichen Lichtbedingungen Bilder unter- oder überbelichtet werden, und das resultierende Disparitätsbild nicht dicht genug ist. In diesem Fall werden mehrere Bilder aufgenommen, bis sich die Belichtungsautomatik angepasst hat oder der Timeout erreicht ist, und erst dann wird die eigentliche Bildaufnahme ausgelöst.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪exposure_adapt_timeout=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?exposure_adapt_timeout=
↪<value>
```

`quality` (*Qualität*)

Disparitätsbilder lassen sich in verschiedenen Auflösungen berechnen: `Full` (*Voll*, volle Bildauflösung), `High` (*Hoch*, halbe Bildauflösung), `Medium` (*Mittel*, Viertel-Bildauflösung) und `Low` (*Niedrig*, Sechstel-Bildauflösung). Stereo-Matching mit voller Auflösung (`Full`) ist nur mit einer gültigen StereoPlus Lizenz möglich. Je niedriger die Auflösung, desto höher die Bildwiederholrate des Disparitätsbilds. Es ist

zu beachten, dass die Bildwiederholrate der Disparitäts-, Konfidenz- und Fehlerbilder immer höchstens der Bildwiederholrate der Kamera entspricht. Falls die Projektoreinstellung `ExposureAlternateActive` ist, kann die Wiederholrate der Bilder höchstens die halbe Bildwiederholrate der Kamera sein.

Eine Bildwiederholrate von 25 Hz lässt sich nur bei niedriger Auflösung erreichen.

Wenn volle Auflösung eingestellt ist, dann ist der mögliche Tiefenbereich intern limitiert, aufgrund von beschränktem on-board Speicherplatz. Es wird empfohlen `mindepth` and `maxdepth` auf den Tiefenbereich anzupassen der für die Applikation benötigt wird.

Tab. 5.4: Auflösung des Tiefenbilds in Abhängigkeit von der gewählten Qualität

Qualität	**Voll** (`Full`)	**Hoch** (`High`)	**Mittel** (`Medium`)	**Niedrig** (`Low`)
Auflösung (Pixel)	1280 x 960	640 x 480	320 x 240	214 x 160

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪quality=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?quality=<value>
```

`roi_offset_x`, `roi_offset_y`, `roi_width`, `roi_height` (*Stereo ROI*)

Diese Werte definieren eine rechteckige Region im linken rektifizierten Bild, die den Bereich für die Berechnung der Tiefenwerte limitiert. Dadurch verringert sich die Laufzeit für das Stereo Matching und damit auch die Latenz. Falls die Breite oder Höhe auf 0 gesetzt werden, dann wird das gesamte linke und rechte Bild für das Stereomatching verwendet. Standardmäßig sind alle Werte 0.

Die Stereo ROI wird in der Web GUI durch eine weiße Überlagerung des ungenutzten Bildbereichs im linken Kamerabild dargestellt. Sie kann über Slider oder direkt im Bild mithilfe der Schaltfläche `Stereo Region of Interest` verändert werden.

Über die REST-API können diese Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/services/parameters?<roi_
↪offset_x|roi_offset_y|roi_width|roi_height>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?<roi_offset_x|roi_offset_
↪y|roi_width|roi_height>=<value>
```

`mindepth` (*Minimaler Abstand*)

Dieser Wert bezeichnet den geringsten Abstand zur Kamera, ab dem Messungen möglich sind. Größere Werte verringern implizit den Disparitätsbereich, wodurch sich auch die Rechenzeit verkürzt. Der minimale Abstand wird in Metern angegeben.

Abhängig von den Eigenschaften des Sensors kann der tatsächliche minimale Abstand größer sein als die Benutzereinstellung. Der tatsächliche minimale Abstand wird in den Statuswerten angezeigt.

Wenn volle Auflösung (`Full`) eingestellt ist, kann der minimale Abstand auch aufgrund interner Speicherplatzlimitierungen größer sein als die Benutzereinstellung. In diesem Fall hilft es, den maximalen Abstand zu verringern, um dadurch auch den tatsächlichen minimalen Abstand zu verringern.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪mindepth=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?mindepth=<value>
```

`maxdepth` (*Maximaler Abstand*)

Dieser Wert ist der größte Abstand zur Kamera, bis zu dem Messungen möglich sind. Pixel mit größeren Distanzwerten werden auf „ungültig“ gesetzt. Wird dieser Wert auf das Maximum gesetzt, so sind Abstände bis Unendlich möglich. Der maximale Abstand wird in Metern angegeben.

Wenn volle Auflösung (`Full`) eingestellt ist, kann der minimale Abstand auch aufgrund interner Speicherplatzlimitierungen größer sein als die Benutzereinstellung. In diesem Fall hilft es, den maximalen Abstand zu verringern um dadurch auch den tatsächlichen minimalen Abstand zu verringern.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪maxdepth=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?maxdepth=<value>
```

`double_shot` (*Double-Shot*)

Das Aktivieren dieses Modus führt zu dichteren Disparitätsbildern, aber einer erhöhten Verarbeitungszeit.

Bei Szenen, die mit einem Projektor im `Single + Out1` Modus oder im kontinuierlichen Modus mit der Projektoreinstellung `ExposureAlternateActive` aufgenommen werden, werden Löcher, die durch Projektor-Reflexionen verursacht werden, gefüllt mit Tiefeninformationen aus den Bildern ohne Projektormuster. In diesem Fall darf der `double_shot` Modus nur verwendet werden, wenn sich die Szene während der Aufnahme der Bilder nicht verändert.

Bei allen anderen Szenen werden Löcher im Disparitätsbild mit Tiefeninformationen aus demselben, herunterskalierten Bild gefüllt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪double_shot=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?double_shot=<value>
```

`static_scene` (*Statisch*)

Mit dieser Option werden acht aufeinanderfolgende Kamerabilder vor dem Matching gemittelt. Dies reduziert Rauschen, was die Qualität des Stereo-Matching-Resultats verbessert. Allerdings erhöht sich auch die Latenz deutlich. Der Zeitstempel des ersten Bildes wird als Zeitstempel für das Disparitätsbild verwendet. Diese Option betrifft nur das Matching in voller und hoher Qualität. Sie darf nur verwendet werden, wenn sich die Szene während der Aufnahme der acht Bilder nicht verändert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪static_scene=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?static_scene=<value>
```

`smooth` (*Glättung*)

Diese Option aktiviert die Glättung von Disparitätswerten. Sie ist nur mit gültiger StereoPlus-Lizenz verfügbar.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪smooth=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?smooth=<value>
```

`fill` (*Füllen*)

Diese Option wird verwendet, um Löcher im Disparitätsbild durch Interpolation zu füllen. Der Füllwert gibt die maximale Disparitätsabweichung am Rand des Lochs an. Größere Füllwerte können die Anzahl an Löchern verringern, aber die interpolierten Werte können größere Fehler aufweisen. Maximal 5% der Pixel werden interpoliert. Kleine Löcher werden dabei bevorzugt interpoliert. Die Konfidenz für die interpolierten Pixel wird auf einen geringen Wert von 0,5 eingestellt. Das Auffüllen lässt sich deaktivieren, wenn der Wert auf 0 gesetzt wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪fill=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?fill=<value>
```

`seg` (*Segmentierung*)

Der Segmentierungsparameter wird verwendet, um die Mindestanzahl an Pixeln anzugeben, die eine zusammenhängende Disparitätsregion im Disparitätsbild ausfüllen muss. Isolierte Regionen, die kleiner sind, werden im Disparitätsbild auf ungültig gesetzt. Der Wert bezieht sich immer auf ein Disparitätsbild mit hoher Qualität (halbe Auflösung) und muss nicht verändert werden, wenn andere Qualitäten gewählt werden. Die Segmentierung eignet sich, um Disparitätsfehler zu entfernen. Bei größeren Werten kann es jedoch vorkommen, dass real vorhandene Objekte entfernt werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?seg=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?seg=<value>
```

`minconf` (*Minimale Konfidenz*)

Die minimale Konfidenz lässt sich einstellen, um potenziell falsche Disparitätsmessungen herauszufiltern. Dabei werden alle Pixel, deren Konfidenz unter dem gewählten Wert liegt, im Disparitätsbild auf „ungültig“ gesetzt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪minconf=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?minconf=<value>
```

`maxdeptherr` (*Maximaler Fehler*)

Der maximale Fehler wird verwendet, um Messungen, die zu ungenau sind, herauszufiltern. Alle Pixel mit einem Tiefenfehler, der den gewählten Wert überschreitet, werden im Disparitätsbild auf „ungültig“ gesetzt. Der maximale Tiefenfehler wird in Metern angegeben. Der Tiefenfehler wächst in der Regel quadratisch mit dem Abstand eines Objekts zur Kamera (siehe *Konfidenz- und Fehlerbilder*).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/parameters?
↪maxdeptherr=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?maxdeptherr=<value>
```

5.2.2 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.5: Statuswerte des `rc_stereomatching`-Moduls

Name	Beschreibung
`fps`	Tatsächliche Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Bildwiederholrate (Hz)* angezeigt.
`latency`	Zeit zwischen Bildaufnahme und Weitergabe des Disparitätsbildes in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Latenz (s)* angezeigt.
`width`	Aktuelle Breite von Disparitäts-, Fehler- und Konfidenzbild in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Wert von *Auflösung (px)* angezeigt.
`height`	Aktuelle Höhe von Disparitäts-, Fehler- und Konfidenzbild in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Wert von *Auflösung (px)* angezeigt.
`mindepth`	Tatsächlicher minimaler Arbeitsabstand in Metern. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Min. Abstand (m)* angezeigt.
`maxdepth`	Tatsächlicher maximaler Arbeitsabstand in Metern. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Max. Abstand (m)* angezeigt.
`time_matching`	Zeit in Sekunden für die Durchführung des Stereo-Matchings mittels *SGM* auf der GPU
`time_postprocessing`	Zeit in Sekunden für die Nachbearbeitung des Matching-Ergebnisses auf der CPU
`reduced_depth_range`	Gibt an, ob der Tiefenbereich aufgrund von Rechenressourcen verringert ist ist

5.2.3 Services

Das Stereo-Matching-Modul bietet folgende Services.

5.2.3.1 `acquisition_trigger`

signalisiert dem Modul, das Stereo-Matching auf den nächsten Stereobildern durchzuführen, falls `acquisition_mode` auf `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild+Out1*) eingestellt ist.

Details

Es wird ein Fehler zurückgegeben, falls `acquisition_mode` auf `Continuous` (*Kontinuierlich*) eingestellt ist.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/services/acquisition_
↪trigger
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/services/acquisition_trigger
```

Request

Dieser Service hat keine Argumente.

Response

Mögliche Rückgabewerte sind in der Tabelle unten aufgeführt.

Tab. 5.6: Mögliche Rückgabewerte des `acquisition_trigger` Serviceaufrufs.

Code	Beschreibung
0	Erfolgreich
-8	Triggern ist nur im Einzelbild-Modus möglich.
101	Triggern wird ignoriert, da bereits ein anderer Triggeraufruf stattfindet.
102	Triggern wird ignoriert, da keine Empfänger registriert sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "acquisition_trigger",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.2.3.2 `reset_defaults`

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

5.2.4 Berechnung von 3D Daten

Die folgenden Abschnitte beschreiben, wie Disparitäts-, Fehler- und Konfidenzbilder verwendet werden können, um daraus Tiefendaten und -fehler zu berechnen.

5.2.4.1 Berechnung von Tiefenbildern und Punktwolken

Die folgenden Gleichungen zeigen, wie sich die tatsächlichen 3D-Koordinaten P_x, P_y, P_z eines Objektpunkts bezogen auf das Kamera-Koordinatensystem aus den Pixelkoordinaten p_x, p_y des Disparitätsbilds und dem Disparitätswert d in Pixeln berechnen lassen:

$$\begin{aligned} P_x &= \frac{p_x \cdot t}{d} \\ P_y &= \frac{p_y \cdot t}{d} \\ P_z &= \frac{f \cdot t}{d}, \end{aligned} \tag{5.1}$$

wobei f die Brennweite nach der Rektifizierung (in Pixeln) und t der während der Kalibrierung ermittelte Stereo-Basisabstand (in Metern) ist. Diese Werte werden auch über die GenICam-Schnittstelle zur Verfügung gestellt (siehe *Besondere Parameter der GenICam-Schnittstelle des rc_visard*, Abschnitt 6.7.4).

Bemerkung: Das Kamera-Koordinatensystem des *rc_visard* ist in *Koordinatensysteme* (Abschnitt 3.7) definiert.

Bemerkung: Der *rc_visard* stellt über seine verschiedenen Schnittstellen einen Brennweitenfaktor bereit. Er bezieht sich auf die Bildbreite, um verschiedene Bildauflösungen zu unterstützen. Die Brennweite f in Pixeln lässt sich leicht bestimmen, indem der Brennweitenfaktor mit der Bildbreite (in Pixeln) multipliziert wird.

Es ist zu beachten, dass für die Gleichungen davon ausgegangen wird, dass das Bildkoordinatensystem im Bildhauptpunkt zentriert ist, der üblicherweise in der Bildmitte liegt, und dass sich p_x, p_y auf die Mitte des Pixels bezieht, durch Addieren von 0.5 auf die ganzzahligen Pixelkoordinaten. In der folgenden Abbildung ist die Definition des Bildkoordinatensystems dargestellt.

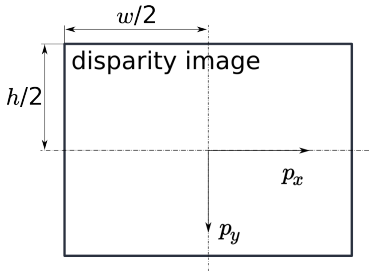


Abb. 5.4: Bildkoordinatensystem: Der Ursprung des Bildkoordinatensystems befindet sich in der Bildmitte – w ist die Bildbreite und h die Bildhöhe.

Die gleichen Formeln, aber mit den entsprechenden GenICam-Parametern, sind in *Umwandlung von Bild-Streams* (Abschnitt 6.7.7) angegeben.

Die Gesamtheit aller aus dem Disparitätsbild errechneten Objektpunkte ergibt eine Punktwolke, die für 3D-Modellierungsanwendungen verwendet werden kann. Das Disparitätsbild kann in ein Tiefenbild umgewandelt werden, indem der Disparitätswert jedes Pixels durch den Wert P_z ersetzt wird.

Bemerkung: Auf der Homepage von Roboception (http://www.roboception.com/download) stehen Software und Beispiele zur Verfügung, um Disparitätsbilder, welche über GigE Vision vom *rc_visard* empfangen werden, in Tiefenbilder und Punktwolken umzuwandeln.

5.2.4.2 Konfidenz- und Fehlerbilder

Für jedes Disparitätsbild wird zusätzlich ein Fehler- und ein Konfidenzbild zur Verfügung gestellt, um die Unsicherheit jedes einzelnen Disparitätswerts anzugeben. Fehler- und Konfidenzbilder besitzen die gleiche Auflösung und Bildwiederholrate wie das Disparitätsbild. Im Fehlerbild ist der Disparitätsfehler d_{eps} in Pixeln angegeben. Er bezieht sich auf den Disparitätswert an der gleichen Bildkoordinate im Disparitätsbild. Das Konfidenzbild gibt den entsprechenden Konfidenzwert c zwischen 0 und 1 an. Die Konfidenz gibt an, wie wahrscheinlich es ist, dass der wahre Disparitätswert innerhalb des Intervalls des dreifachen Fehlers um die gemessene Disparität d liegt, d.h. $[d - 3d_{eps}, d + 3d_{eps}]$. So lässt sich das Disparitätsbild mit Fehler- und Konfidenzwerten in Anwendungen einsetzen, für die probabilistische Folgerungen nötig sind. Die Konfidenz- und Fehlerwerte für eine ungültige Disparitätsmessung betragen 0.

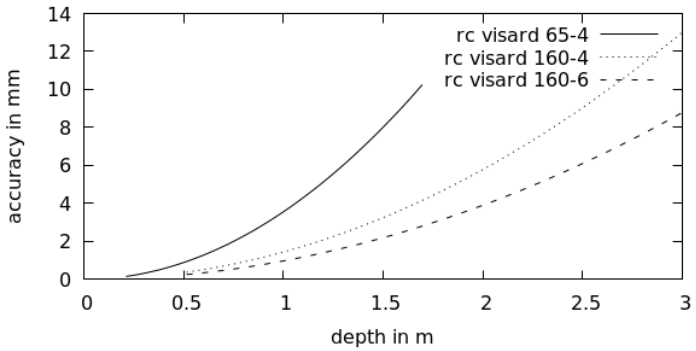
Der Disparitätsfehler d_{eps} (in Pixeln) lässt sich mithilfe der Brennweite f (in Pixeln), des Basisabstands t (in Metern) und des Disparitätswerts d (in Pixeln) desselben Pixels im Disparitätsbild in einen Tiefenfehler z_{eps} (in Metern) umrechnen:

$$z_{eps} = \frac{d_{eps} \cdot f \cdot t}{d^2}. \tag{5.2}$$

Durch Kombination der Gleichungen kann der Tiefenfehler zur Tiefe in Bezug gebracht werden:

$$z_{eps} = \frac{d_{eps} \cdot {P_z}^2}{f \cdot t}.$$

Unter Berücksichtigung der Brennweiten und Basisabstände der verschiedenen Kameramodelle sowie des typischen kombinierten Kalibrier- und Stereo-Matching-Fehlers d_{eps} von 0,25 Pixeln lässt sich die Tiefengenauigkeit wie folgt grafisch darstellen:



5.2.5 Anzeigen und Herunterladen von Tiefenbildern und Punktwolken

Der *rc_visard* stellt zeitgestempelte Disparitäts-, Fehler- und Konfidenzbilder über die GenICam-Schnittstelle (siehe *Verfügbare Bild-Streams*, Abschnitt 6.7.6) oder über *gRPC Bilddatenschnittstelle* (siehe Abschnitt 6.8) zur Verfügung.

Live-Streams in geringerer Qualität werden auf der *Tiefenbild* Seite in der *Web GUI* (Abschnitt 6.1) bereitgestellt.

Die Web GUI bietet weiterhin die Möglichkeit, einen Schnappschuss der aktuellen Szene mit den Tiefen-, Fehler und Konfidenzbildern, sowie der Punktwolke als .tar.gz-Datei zu speichern, wie in *Herunterladen von Kamerabildern* (Abschnitt 6.1.5) beschrieben wird.

5.3 Navigationsmodule

Die Navigationsmodule des *rc_visard* umfassen:

- ***Sensordynamik*** **(`rc_dynamics`, Abschnitt 5.3.1)** erstellt Schätzungen des dynamischen Zustands des *rc_visard*, d.h. seiner Pose, Geschwindigkeit und Beschleunigung. Diese Zustände werden als kontinuierliche Datenströme über die *rc_dynamics-Schnittstelle* übertragen. Zu diesem Zweck verwaltet und verknüpft das Dynamik-Modul Daten aus den folgenden Submodulen:
  - ***Visuelle Odometrie*** **(`rc_stereovisodo`, Abschnitt 5.3.2)** schätzt die Bewegung des *rc_visard* anhand der Bewegungen charakteristischer Merkmale in den Bildern der linken Kamera.
  - ***Stereo-INS*** **(`rc_stereo_ins`, Abschnitt 5.3.3)** kombiniert die per visueller Odometrie ermittelten Werte mit den Daten der integrierten inertialen Messeinheit (IMU), um auf dieser Grundlage akkurate und hochfrequente Echtzeit-Zustandsschätzungen bereitzustellen.
  - ***SLAM*** **(`rc_slam`, Abschnitt 5.3.4)** übernimmt die simultane Lokalisierung und Kartenerstellung, um akkumulierte Posendaten zu korrigieren. Die Trajektorie des *rc_visard* lässt sich über die *REST-API-Schnittstelle* (Abschnitt 6.2) abfragen.

5.3.1 Sensordynamik

Das Dynamik-Modul rc_dynamics ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist, und liefert Schätzungen des Sensorzustands, der die Pose (Position und Orientierung), Lineargeschwindigkeit, Linearbeschleunigung und Drehrate des Sensors umfasst. Mit diesem Modul lassen sich Datenströme für die verschiedenen Submodule starten, stoppen und verwalten:

- ***Visuelle Odometrie*** **(`rc_stereovisodo`)** schätzt die Kamerabewegung anhand der Bewegung von charakteristischen Bildpunkten im linken Kamerabild (Abschnitt 5.3.2).
- ***Stereo-INS*** **(`rc_stereo_ins`)** kombiniert die per visueller Odometrie ermittelten Werte mit den Daten der integrierten inertialen Messeinheit (*IMU*), um auf dieser Grundlage akkurate und hochfrequente Echtzeit-Zustandsschätzungen bereitzustellen (Abschnitt 5.3.3).
- ***SLAM*** **(`rc_slam`)** übernimmt die simultane Lokalisierung und Kartenerstellung (*SLAM*), um akkumulierte Posendaten zu korrigieren (Abschnitt 5.3.4).

Bemerkung: Die gleichzeitige Nutzung des Dynamik-Moduls und des *Stereo-Matching Modul* (Abschnitt 5.2) kann zu verringerter Genauigkeit der Bewegungsschätzung führen. Der Abschnitt *Visuelle Odometrie* (5.3.2) erklärt, wie dies vermieden werden kann.

5.3.1.1 Koordinatensysteme für die Zustandsschätzung

Das Weltkoordinatensystem für die Zustandsschätzung definiert sich wie folgt: Die Z-Achse des Koordinatensystems zeigt nach oben und ist am Gravitationsvektor ausgerichtet. Die X-Achse liegt orthogonal zur Z-Achse und zeigt in die Blickrichtung des *rc_visard* zum Zeitpunkt, zu dem die Zustandsschätzung beginnt. Der Ursprung des Weltkoordinatensystems befindet sich am Ursprung des IMU-Koordinatensystems des *rc_visard* in dem Augenblick, in dem die Zustandsschätzung aktiviert wird.

Wird die Zustandsschätzung aktiviert, wenn die Blickrichtung des *rc_visard* parallel zum Gravitationsvektor liegt (mit einem Toleranzbereich von 10 Grad), dann ist die Y-Achse des Weltkoordinatensystems

entweder an der positiven oder negativen X-Achse des IMU-Koordinatensystems ausgerichtet. In diesem Fall ist die Anfangsausrichtung des Weltkoordinatensystems nicht mehr kontinuierlich. Es ist also besondere Vorsicht geboten, wenn die Zustandsschätzung mit dieser Orientierung beginnt.

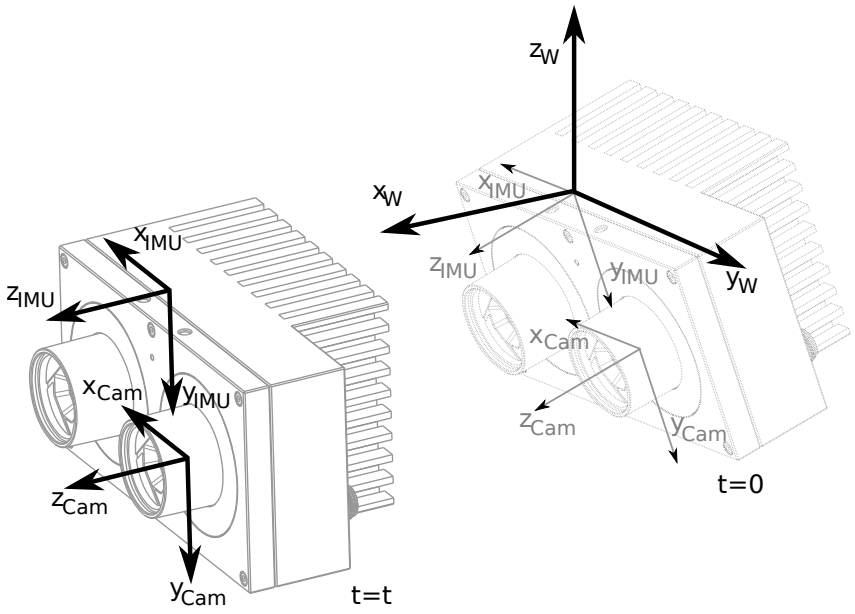


Abb. 5.5: Koordinatensysteme für die Zustandsschätzung: Das IMU-Koordinatensystem liegt im Gehäuse des *rc_visard*, das *Kamera-Koordinatensystem* (Abschnitt 3.7) im Fokuspunkt der linken Kamera.

Die Transformation zwischen dem IMU-Koordinatensystem und dem Kamera/Sensor-Koordinatensystem wird ebenfalls geschätzt und über die rc_dynamics-Schnittstelle im *Echtzeit-Dynamik-Datenstrom* bereitgestellt (siehe *Schnittstellen*, Abschnitt 6).

Warnung: Das Stereo-INS-Modul führt während der Initialisierung eine Selbstkalibrierung der IMU durch. Deswegen darf sich der *rc_visard* während des Startens des Stereo-INS-Moduls nicht bewegen und muss ausreichend viel Textur sehen können.

5.3.1.2 Verfügbare Zustandsschätzungen

Der *rc_visard* bietet über die rc_dynamics-Schnittstelle sieben verschiedene Arten an Datenströmen mit zeitgestempelten Zustandsschätzungen an (siehe *Die rc_dynamics-Schnittstelle*, Abschnitt 6.3):

Name	Frequenz	Quelle	Beschreibung
pose	25 Hz	Best Effort	genaueste Schätzung der Pose des Kamerakoordinatensystems, aber leicht zeitverzögert
pose_ins	25 Hz	*Stereo-INS*	genaueste Schätzung der Pose des Kamerakoordinatensystems, aber leicht zeitverzögert
pose_rt	200 Hz	Best Effort	Pose des Kamerakoordinatensystems
pose_rt_ins	200 Hz	*Stereo-INS*	Pose des Kamerakoordinatensystems
dynamics	200 Hz	Best Effort	Pose, Geschwindigkeit und Beschleunigung im IMU-Koordinatensystem
dynamics_ins	200 Hz	*Stereo-INS*	Pose, Geschwindigkeit und Beschleunigung im IMU-Koordinatensystem
imu	200 Hz	*Stereo-INS*	IMU-Rohdaten

Best Effort bedeutet hier für den Fall, dass das *SLAM*-Modul läuft, dass der Datenstrom per Loop-Closure korrigierte Schätzungen umfasst, bzw. dass er dem vom *Stereo-INS* bereitgestellten Datenstrom entspricht, wenn SLAM nicht läuft.

Kameraposen-Datenströme (`pose` und `pose_ins`)

Die *Kameraposen-Datenströme* heißen `pose` und `pose_ins` und sie werden mit einer Frequenz von 25 Hz mit Zeitstempeln bereitgestellt, die den Bildzeitstempeln entsprechen. `pose` bietet eine Best-Effort-Schätzung, für die *SLAM* und *Stereo-INS* kombiniert werden, wenn das SLAM-Modul läuft. Läuft das SLAM-Modul nicht, sind beide Datenströme gleichwertig. Die Posen werden in Weltkoordinaten angegeben und beziehen sich auf den Ursprung des Kamerakoordinatensystems (siehe *Koordinatensysteme für die Zustandsschätzung*, Abschnitt 5.3.1.1). Die Datenströme umfassen genaueste Schätzungen, für die alle verfügbaren Daten des *rc_visard* berücksichtigt werden. Sie können für Modellierungsanwendungen eingesetzt werden, bei denen Kamerabilder, Tiefenbilder oder Punktwolken mit höchster Genauigkeit aneinander ausgerichtet werden müssen. Um die größtmögliche Genauigkeit sicherzustellen, verzögert sich die Ausgabe dieser Datenströme, bis die zugehörigen Messwerte aus der visuellen Odometrie verfügbar sind.

Echtzeit-Datenströme der Kamerapose (`pose_rt` und `pose_rt_ins`)

Die *Echtzeit-Datenströme der Kamerapose* heißen `pose_rt` und `pose_rt_ins` und sie werden mit der IMU-Frequenz von 200 Hz bereitgestellt. `pose_rt` bietet eine Best-Effort-Schätzung, für die *SLAM* und *Stereo-INS* kombiniert werden, wenn das SLAM-Modul läuft. Läuft das SLAM-Modul nicht, sind beide Datenströme gleichwertig. Die Posen werden in Weltkoordinaten angegeben und beziehen sich auf den Ursprung des Koordinatensystems der *rc_visard*-Kamera (siehe *Koordinatensysteme für die Zustandsschätzung*, Abschnitt 5.3.1.1). Die in diesen Datenströmen enthaltenen Werte entsprechen den Werten in den *Echtzeit-Dynamik-Datenströmen*, geben aber die Pose im Sensor/Kamera-Koordinatensystem statt im IMU-Koordinatensystem an.

Echtzeit-Dynamik-Datenströme (`dynamics` und `dynamics_ins`)

Die beiden *Echtzeit-Dynamik-Datenströme* heißen `dynamics` und `dynamics_ins` und sie werden mit der IMU-Frequenz von 200 Hz bereitgestellt. `dynamics` bietet eine Best-Effort-Schätzung, für die *SLAM* and *Stereo-INS* kombiniert werden, wenn das SLAM-Modul läuft. Läuft das SLAM-Modul nicht, sind beide Datenströme gleichwertig. Die Schätzungen können für die Echtzeitregelung eines Roboters verwendet werden. Da die Werte in Echtzeit bereitgestellt werden und die Berechnung der visuellen Odometrie

eine gewisse Bearbeitungszeit erfordert, ist die letzte Odometrieschätzung möglicherweise nicht enthalten. Daher sind diese Schätzungen im Allgemeinen etwas weniger genau als die nicht in Echtzeit bereitgestellten *Kameraposen-Datenströme* (siehe oben). Dennoch sind es zu diesem Zeitpunkt die bestmöglichen Schätzungen. Die bereitgestellten Dynamik-Datenströme enthalten folgende Werte zum *rc_visard*:

- Translation $\mathbf{p} = (x, y, z)^T$ in m,
- Rotation $\mathbf{q} = (q_x, q_y, q_z, q_w)$ als Einheitsquaternion,
- Lineargeschwindigkeit $\mathbf{v} = (v_x, v_y, v_z)^T$ in $\frac{m}{s}$,
- Winkelgeschwindigkeit $\omega = (\omega_x, \omega_y, \omega_z)^T$ in $\frac{rad}{s}$,
- gravitationskompensierte Linearbeschleunigung $\mathbf{a} = (a_x, a_y, a_z)^T$ in $\frac{m}{s^2}$ und
- Transformation zwischen Kamera- und IMU-Koordinatensystem als Pose mit Frame-Namen und Parent-Frame-Namen.

Der Datenstrom enthält für jede Datenkomponente zusätzlich den Namen des Koordinatensystems, in dem die Werte angegeben sind. Translations-, Rotations- und Lineargeschwindigkeiten werden im Weltkoordinatensystem, Winkelgeschwindigkeiten und Winkelbeschleunigungen im IMU-Koordinatensystem angegeben (siehe *Koordinatensysteme für die Zustandsschätzung*, Abschnitt 5.3.1.1). Alle Werte beziehen sich auf den Ursprung des IMU-Koordinatensystems. Dies bedeutet beispielsweise, dass die Lineargeschwindigkeit der Geschwindigkeit des IMU-Koordinatenursprungs im Weltkoordinatensystem entspricht.

Schließlich enthält der Datenstrom den Statusindikator `possible_jump`, der auf *TRUE* gesetzt ist, wann immer das optional erhältliche SLAM-Modul (siehe *SLAM*, Abschnitt 5.3.4) die Zustandsschätzung nach einem Schleifenschluss (Loop Closure) korrigiert. Die Zustandsschätzung kann in diesem Fall einen Sprung machen, was beachtet werden sollte, wenn die Werte in einem Regelkreis verwendet werden. Wenn SLAM nicht läuft, bleibt der Statusindikator `possible_jump` auf *FALSE* und kann ignoriert werden.

IMU-Datenstrom (`imu`)

Der *IMU-Datenstrom* heißt `imu` und wird mit der IMU-Frequenz von 200 Hz bereitgestellt. Er umfasst die Beschleunigungen in X-, Y- und Z-Richtung sowie die Winkelgeschwindigkeiten um diese drei Achsen. Diese Werte sind kalibriert, aber nicht bias- und gravitationskompensiert, und werden im IMU-Koordinatensystem angegeben. Die Transformation zwischen dem IMU-Koordinatensystem und dem Sensorkoordinatensystem wird im *Echtzeit-Dynamik-Datenstrom* bereitgestellt.

5.3.1.3 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.7: Die Statuswerte des `rc_dynamics` Moduls

Name	Beschreibung
`state`	Der aktuelle Zustand des Sensordynamik-Moduls

5.3.1.4 Services

Das Sensordynamik-Modul bietet folgende Services zum Starten der Dynamik-/Bewegungsschätzung. Alle Services geben einen numerischen Code des eingetretenen Zustands zurück. Die Bedeutung der zurückgegebenen Zustandscodes und deren Namen werden in Tab. 5.8 angegeben.

Tab. 5.8: Mögliche Zustände des Sensordynamik-Moduls

Zustandsname	Beschreibung
IDLE	Das Dynamikmodul ist bereit aber inaktiv
WAITING_FOR_INS	Es wird auf Daten des Stereo-INS-Moduls gewartet
WAITING_FOR_INS_AND_SLAM	Es wird auf Daten des Stereo-INS- und des SLAM-Moduls gewartet
RUNNING	Das Stereo-INS-Modul läuft
WAITING_FOR_SLAM	Es wird auf den Start des SLAM-Moduls gewartet (Stereo-INS läuft)
RUNNING_WITH_SLAM	Die Module Stereo-INS und SLAM laufen
STOPPING	Übergangszustand von Kommandos, die zu (oder durch) IDLE gehen
FATAL	Ein schwerwiegender Fehler ist aufgetreten (im Stereo-INS- oder SLAM-Modul)

Das folgende Diagramm zeigt die wichtigsten Zustände und Übergänge. Zwischenzustände und der Fehlerzustand `FATAL` sind aus Gründen der Übersichtlichkeit nicht aufgeführt.

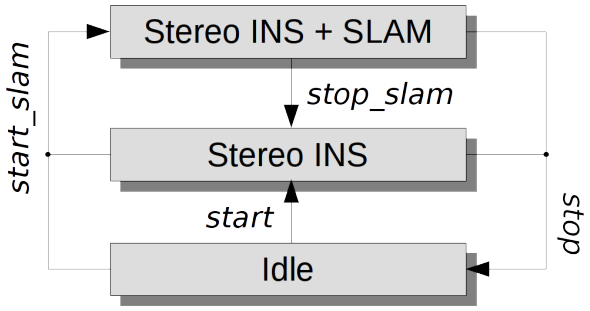


Abb. 5.6: Vereinfachtes Zustandsdiagramm

Die Services sollten schnell reagieren. Deswegen ist bei Services, die eine Zustandsänderung verursachen, der Rückgabewert `current_state` im Allgemeinen der erste neue (Zwischen-)Zustand, in den gewechselt wurde, und nicht der finale Zustand. Zum Beispiel gibt der `start` Service als `current_state` `WAITING_FOR_INS` und nicht `RUNNING` zurück. Falls der Zustandsübergang nicht innerhalb von 0.1 Sekunden vonstatten geht, wird der aktuelle Zustand zurückgegeben. In Tab. 5.8 ist die Bedeutung der zurückgegebenen Zustandscodes aufgeführt.

Bemerkung: Der Zustand `FATAL` kann nur durch den Aufruf des `stop` Services verlassen werden. Dieser Service geht in den Zustand `IDLE`. Die Services `restart` und `restart_slam` nutzen intern auch `stop` und funktionieren daher ebenso. Die Services `start` und `start_slam` funktionieren nur, wenn der aktuelle Zustand `IDLE` ist, und haben keinen Effekt, wenn der Zustand `FATAL` ist.

Bemerkung: Das Sensordynamik-Modul lässt sich auch über die Seite *Dynamik* der *Web GUI* starten und stoppen.

`start`

startet das Stereo-INS-Modul.

Details

Der Status geht von `IDLE` über `WAITING_FOR_INS` zu `RUNNING`.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

start_slam

startet SLAM und – falls es noch nicht läuft – das Stereo-INS-Modul.

Details

Ausgehend von `IDLE` geht der Zustand über `WAITING_FOR_INS_AND_SLAM` und `WAITING_FOR_SLAM` zu `RUNNING_WITH_SLAM`. Aus `RUNNING` geht er über `WAITING_FOR_SLAM` zu `RUNNING_WITH_SLAM`.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/start_slam
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/start_slam
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start_slam",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Stereo-INS-Modul und – falls es läuft – das SLAM-Modul.

Details

Die Trajektorienschätzung des SLAM-Moduls (bis zum Zeitpunkt des Stoppens) ist weiterhin verfügbar. Ausgehend von `RUNNING` oder `RUNNING_WITH_SLAM` geht der Zustand über `STOPPING` zu `IDLE`.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop_slam

stoppt das SLAM-Modul. Das Stereo-INS-Modul läuft weiter.

Details

Die Trajektorienschätzung des SLAM-Moduls (bis zum Zeitpunkt des Stoppens) ist weiterhin verfügbar. Der Zustand geht von `RUNNING_WITH_SLAM` zu `IDLE`.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/stop_slam
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/stop_slam
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop_slam",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

restart

startet die Stereo-INS neu. Äquivalent zu aufeinanderfolgendem `stop` und `start`.

Details

Aus dem Zustand `RUNNING` oder `RUNNING_WITH_SLAM`: Übergang über `STOPPING`, `IDLE` und `WAITING_FOR_INS` zu `RUNNING`.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/restart
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/restart
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "restart",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

restart_slam

startet neu in den SLAM-Modus. Äquivalent zu aufeinanderfolgendem `stop` und `start_slam`.

Details

Aus dem Zustand `RUNNING` oder `RUNNING_WITH_SLAM`: Übergang über die Zustände `STOPPING`, `IDLE`, `WAITING_FOR_INS_AND_SLAM` und `WAITING_FOR_SLAM` zu `RUNNING_WITH_SLAM`.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/restart_slam
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/restart_slam
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "restart_slam",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```


get_cam2imu_transform

gibt die Transformation zwischen Kamera- und IMU-Koordinatensystem zurück.

Details

Diese entspricht der `cam2imu_transform` der *Dynamics-Nachricht* (Abschnitt 6.3.3).

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_dynamics/services/get_cam2imu_transform
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_dynamics/services/get_cam2imu_transform
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_cam2imu_transform",
  "response": {
    "name": "string",
    "parent": "string",
    "pose": {
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      }
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

5.3.2 Visuelle Odometrie

Das Visuelle-Odometrie-Modul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

Die visuelle Odometrie ist Teil des Sensordynamik-Moduls. Sie dient dazu, die Bewegung der Kamera aus der Bewegung charakteristischer Bildpunkte (sogenannter Bildmerkmale) im linken Kamerabild zu schätzen. Bildmerkmale werden auf Basis von Eckpunkten, Bildbereichen mit hohen Intensitätsgradienten, errechnet. Mithilfe von Bildmerkmalen lassen sich Übereinstimmungen zwischen aufeinanderfolgenden Bildern finden. Deren 3D-Koordinaten werden mithilfe eines Stereo-Matching-Verfahrens (unabhängig vom Disparitätsbild) berechnet. Aus den übereinstimmenden 3D-Punkten zweier Kamerabilder wird die Bewegung der Kamera berechnet. Um die Robustheit der visuellen Odometrie zu erhöhen, werden Übereinstimmungen nicht nur zum letzten Kamerabild, sondern zu mehreren vorherigen Kamerabildern, sogenannten *Keyframes*, berechnet. Dann wird das beste Resultat ausgewählt.

Die Bildwiederholrate der visuellen Odometrie ist unabhängig von der Benutzereinstellung im Stereokamera-Modul. Sie ist intern auf 12 Hz begrenzt, kann aber je nach Anzahl der Bildmerkmale oder Keyframes auch niedriger sein. Um die Posenschätzung in einer guten Qualität berechnen zu können, sollte die Bildwiederholrate nicht signifikant unter 10 Hz fallen.

Bemerkung: Die gleichzeitige Nutzung des *Stereo-Matching Modul* mit dem Sensordynamik-Modul kann durch die hohe Prozessorlast zu verringerter Genauigkeit in der Bewegungsschätzung führen. Es wird daher empfohlen, bei gleichzeitigem Betrieb die Bildwiederholrate des *Kamera Modul* zu reduzieren, um die Rechenlast durch das Stereo-Matching zu senken.

Die Messungen aus der visuellen Odometrie lassen sich nicht direkt vom *rc_visard* aufrufen. Stattdessen werden sie intern mit den Daten der integrierten inertialen Messeinheit (IMU) kombiniert, um so die Robustheit und Frequenz der Posenschätzungen zu erhöhen und die Latenz zu verringern. Das Ergebnis der Sensordatenfusion wird in Form verschiedener Datenströme bereitgestellt (siehe *Stereo-INS*, Abschnitt 5.3.3).

5.3.2.1 Parameter

Das Odometrie-Modul heißt `rc_stereovisodo` und wird in der *Web GUI* (Abschnitt 6.1) auf der Seite *Dynamik* dargestellt. Der Benutzer kann die Parameter der visuellen Odometrie entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.9: Laufzeitparameter des `rc_stereovisodo`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`disprange`	int32	32	512	256	Disparitätsbereich in Pixeln
`ncorner`	int32	50	4000	500	Anzahl der Eckpunkte
`nfeature`	int32	50	4000	300	Anzahl der Bildmerkmale
`nkey`	int32	1	4	4	Anzahl der Keyframes

Beschreibung der Laufzeitparameter

Laufzeitparameter beeinflussen die Anzahl an Bildmerkmalen, auf deren Grundlage die Berechnungen für die visuelle Odometrie vorgenommen werden. Ein Mehr an Bildmerkmalen erhöht die Robustheit der visuellen Odometrie, geht jedoch zu Lasten einer längeren Laufzeit, was wiederum die Frequenz der visuellen Odometrie verringern kann. Doch auch wenn die resultierende Zustandsschätzung aufgrund der Kombination mit den IMU-Messdaten immer mit einer hohen Frequenz bereitgestellt wird, sind hohe Odometrie-Raten dennoch wünschenswert, da diese Messungen viel akkurater sind als IMU-Messungen allein. Daher sollte für die visuelle Odometrie eine Frequenz von 10 Hz angestrebt werden. Die Rate der visuellen Odometrie wird als Statusparameter bereitgestellt und unter der Bildvorschau in der *Web GUI* auf der Seite *Dynamik* angegeben.

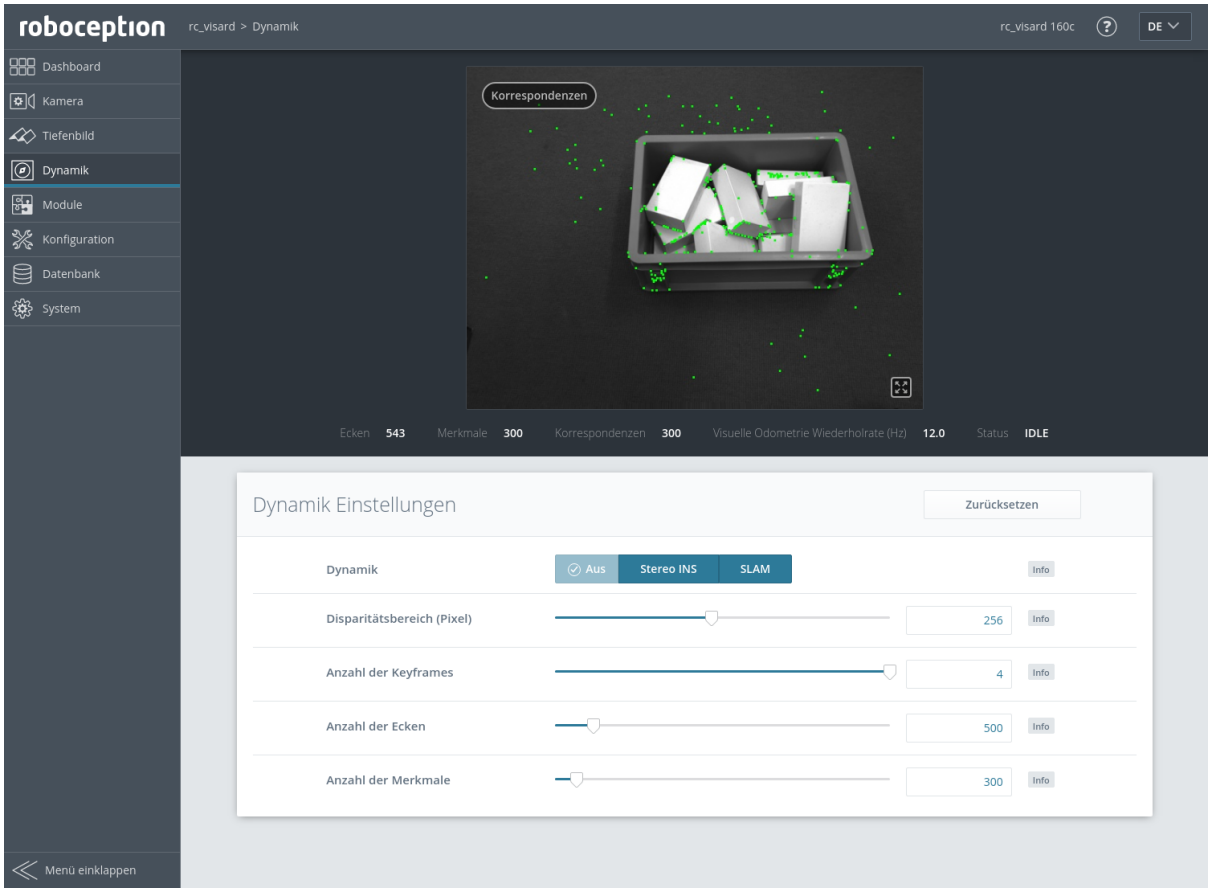


Abb. 5.7: *Dynamik*-Seite der Web GUI

Auf dem auf dieser Seite gezeigten Kamerabild werden Bildmerkmale als kleine grüne Punkte dargestellt. Die dicken grünen Punkte sind diejenigen Merkmale, für die Übereinstimmungen zu einem vorherigen Keyframe gefunden werden konnten. Grüne Linien stellen dar, wie sich diese Bildmerkmale in Bezug auf den vorherigen Keyframe bewegt haben. Diese Darstellung soll beim Finden eines guten Parametersatzes für die visuelle Odometrie helfen. Die Anzahl an übereinstimmenden Bildmerkmalen (Korrespondenzen) wird als Statusparameter bereitgestellt und unter der Bildvorschau in der *Web GUI* auf der Seite *Dynamik* angegeben. Für robuste Odometrie-Messungen sollten die Parameter derart angepasst werden, dass die resultierende Anzahl an Korrespondenzen in der Zielumgebung bei mindestens 50 liegt, wenn sich der Sensor bewegt. Die Anzahl an Korrespondenzen ist höher, wenn der *rc_visard* in Ruhe ist, und sie wird sich verändern, wenn sich der *rc_visard* durch die Umgebung bewegt. Aufgrund der Kombination mit den IMU-Messungen kann ein kurzer Ausfall der visuellen Odometrie toleriert werden. Längere Ausfälle sollten vermieden werden, da sie zu größeren Posenunsicherheiten und zu Fehlern in der Zustandsschätzung führen können.

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Dynamik* der Web GUI repräsentiert.

Der Name der Zeile ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet:

`disprange` (*Disparitätsbereich*)

Der Disparitätsbereich gibt den maximalen Disparitätswert an, den jedes Bildmerkmal im Disparitätsbild mit Qualitätsstufe *Hoch* (`High`, halbe Auflösung) annehmen kann. Der Disparitätsbereich bestimmt den Mindestabstand für die visuelle Odometrie. Ist der Disparitätsbereich klein, werden nur entferntere Merkmale für die Odometrie-Schätzungen berücksichtigt. Bei einem größeren Disparitätsbereich können auch nahe liegende Merkmale einbezogen werden. Ein größerer Disparitätsbereich erhöht die Rechenzeit, was die Frequenz der visuellen Odometrie verringern kann.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereovisodo/parameters?disprange=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereovisodo/parameters?disprange=<value>
```

`nkey` (*Anzahl der Keyframes*)

Ein Mehr an Keyframes kann die Robustheit und Genauigkeit der visuellen Odometrie erhöhen, was jedoch mit einer längeren Rechenzeit und möglicherweise mit einer geringeren Odometriefrequenz einhergehen kann.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereovisodo/parameters?nkey=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereovisodo/parameters?nkey=<value>
```

`ncorner` (*Anzahl der Ecken*)

Dieser Parameter gibt die ungefähre Anzahl an Eckpunkten an, die im linken Bild detektiert werden. Ein größerer Wert macht die visuelle Odometrie robuster und genauer, kann aber zu einer geringeren Odometriefrequenz führen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereovisodo/parameters?ncorner=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereovisodo/parameters?ncorner=<value>
```

`nfeature` (*Anzahl der Merkmale*)

Dieser Parameter beschreibt die maximale Anzahl an Bildmerkmalen, die von den Eckpunkten abgeleitet werden. Es ist hilfreich, mehr Eckpunkte zu erkennen, sodass die beste Teilmenge als Merkmale ausgewählt werden kann. Ein größerer Wert macht die visuelle Odometrie robuster und genauer, kann aber zu einer geringeren Odometrierate führen. Je nach Szene und Bewegung werden möglicherweise weniger Merkmale berechnet. Die tatsächliche Anzahl an Merkmalen wird unter der Bildvorschau in der *Web GUI* auf der Seite *Dynamik* angegeben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereovisodo/parameters?nfeature=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereovisodo/parameters?nfeature=<value>
```

Bemerkung: Die Erhöhung der Anzahl an Keyframes, Ecken oder Merkmalen erhöht zwar die Robustheit, führt aber zu mehr Rechenzeit und kann, je nachdem, welche anderen Module auf dem *rc_visard* aktiv sind, die Rate der visuellen Odometrie verringern. Die Odometriefrequenz sollte mindestens 10 Hz betragen.

5.3.2.2 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.10: Statuswerte des `rc_stereovisodo`-Moduls

Name	Beschreibung
`corner`	Anzahl der erkannten Eckpunkte. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Ecken* angezeigt.
`correspondences`	Anzahl der Übereinstimmungen. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Korrespondenzen* angezeigt.
`feature`	Anzahl der Bildmerkmale. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Merkmale* angezeigt.
`fps`	Frequenz der visuellen Odometrie in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Rate visuelle Odometrie (Hz)* angezeigt.
`time_frame`	Verarbeitungszeit in Sekunden, die zur Berechnung von Eckpunkten und Bildmerkmalen pro Frame benötigt wird
`time_vo`	Verarbeitungszeit in Sekunden, die zur Berechnung der Bewegung benötigt wird

5.3.2.3 Services

Das Modul bietet keine eigenen Funktionen zum Starten bzw. Stoppen, da es über das *Dynamik-Modul* (Abschnitt 5.3.1) gestartet bzw. gestoppt wird.

Das Visuelle-Odometrie-Modul bietet folgende Services, um Parametereinstellungen zu speichern bzw. wiederherzustellen.

`reset_defaults`

Hiermit werden die Werkseinstellungen der Parameter dieses Moduls wiederhergestellt und angewandt („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereovisodo/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereovisodo/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.3.3 Stereo-INS

Das Modul zur stereobildgestützten inertialen Navigation (*INS*) ist ein Basismodul, das auf jedem *rc_visard* verfügbar ist, und ist Teil des Dynamik-Moduls. Das System kombiniert die Messwerte der visuellen Odometrie mit den Daten aus der inertialen Messeinheit (*IMU*), um so robuste und hochfrequente Echtzeit-Zustandsschätzungen mit geringer Latenz anbieten zu können. Das IMU-Modul, das zur Messung der Linearbeschleunigungen und Drehraten in allen drei Dimensionen dient, besteht aus drei Beschleunigungsaufnehmern und drei Gyroskopen. Durch Kombination der Messdaten aus IMU und visueller Odometrie werden die Zustandsschätzungen mit der gleichen Frequenz wie die IMU-Messungen (200 Hz) vorgenommen und sind so selbst unter anspruchsvollen Lichtbedingungen und bei schnellen Bewegungen sehr robust.

Bemerkung: Um genaue Posenschätzungen zu erhalten, muss sichergestellt sein, dass ausreichend viel Textur während der Laufzeit des Stereo-INS-Moduls sichtbar ist. Falls für einen längeren Zeitraum keine Textur sichtbar ist, beendet sich das Stereo-INS-Modul, anstatt stark fehlerbehaftete Daten zu liefern.

5.3.3.1 Selbstkalibrierung

Das Stereo-INS-Modul führt während seines Startvorgangs eine Selbstkalibrierung der IMU anhand von visuellen Odometriemessungen durch. Für eine erfolgreiche Selbstkalibrierung muss während des Startvorgangs des Stereo-INS-Moduls folgendes erfüllt sein:

- Der *rc_visard* darf sich nicht bewegen und
- ausreichend viel Textur muss sichtbar sein.

Sind diese Bedingungen nicht erfüllt, kann das zu einem konstanten Drift in der Posenschätzung führen.

5.3.3.2 Parameter

Das Stereo-INS-Modul heißt `rc_stereo_ins`.

Dieses Modul besitzt keine Laufzeitparameter.

5.3.3.3 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.11: Statuswerte des `rc_stereo_ins`-Moduls

Name	Beschreibung
`freq`	Frequenz des Stereo-INS-Prozesses in Hertz.
`state`	Interner Zustand als Zeichenkette

5.3.4 SLAM

Das SLAM-Modul ist ein optionales Modul, welches intern auf dem *rc_visard* läuft, und eine separate SLAM-*Lizenz* (Abschnitt 7.4) benötigt, die erworben werden muss. Sobald eine SLAM-Lizenz auf dem *rc_visard* hinterlegt wird, erscheint das SLAM-Modul auf der Seite *System* der *Web GUI* im Bereich *Lizenz* als aktiviert.

Das SLAM-Modul ist Teil des Dynamik-Moduls und stellt genauere Posenschätzungen als das Stereo-INS-Modul bereit. Wenn sich der *rc_visard* bewegt, summieren sich mit der Zeit die bei der Posenschätzung auftretenden Fehler. Das SLAM-Modul kann diese Fehler korrigieren, indem es bereits besuchte Orte wiedererkennt.

Das Akronym SLAM steht für simultane Lokalisierung und Kartenerstellung (simultaneous localization and mapping). Das SLAM-Modul erstellt eine Karte aus den für die visuelle Odometrie genutzten Bildmerkmalen. Die Karte wird später verwendet, um kumulierte Posenfehler zu korrigieren. Am ehesten lässt sich dies in Anwendungen beobachten, bei denen der Roboter, nachdem er eine lange Strecke zurückgelegt hat, an einen besuchten Ort zurückkehrt (dies wird auch „Schleifenschluss“ oder „Loop Closure“ genannt). In diesem Fall kann der Roboter Bildmerkmale, die bereits in seiner Karte abgespeichert sind, wiedererkennen und seine Posenschätzung auf dieser Grundlage korrigieren.

Kommt es zu einem Schleifenschluss, wird nicht nur die aktuelle, sondern auch die bisherige Posenschätzung (Trajektorie des *rc_visard*) korrigiert. Die kontinuierliche Korrektur der Trajektorie sorgt dafür, dass die Karte immer mehr an Genauigkeit gewinnt. Die Genauigkeit der Trajektorie ist auch wichtig, wenn diese zum Aufbau eines integrierten Weltmodells verwendet wird, indem beispielsweise die ermittelten 3D-Punktwolken in ein gemeinsames Koordinatensystem projiziert werden (siehe *Berechnung von Tiefenbildern und Punktwolken*). Zu diesem Zweck kann die gesamte Trajektorie des *rc_visard* beim SLAM-Modul abgefragt werden.

5.3.4.1 Verwendung

Das SLAM-Modul kann jederzeit entweder über das rc_dynamics Interface (siehe Dokumentation der zugehörigen *Services*, Abschnitt 5.3.1.4) oder über die *Dynamik*-Seite der *Web GUI* eingeschaltet werden.

Die Posenschätzung des SLAM-Moduls wird durch die aktuelle Schätzung des Stereo-INS-Moduls initialisiert. Der Ursprung der Posenschätzung befindet sich also an dem Punkt, an dem das Stereo-INS-Modul gestartet wurde.

Da das SLAM-Modul auf den Bewegungsschätzungen der Stereo-INS aufbaut, wird das Stereo-INS-Modul automatisch gestartet, sobald SLAM gestartet wird, sollte es nicht bereits laufen.

Wenn das SLAM-Modul läuft, sind die korrigierten Posenschätzungen über die Datenströme *pose*, *pose_rt*, und *dynamics* des Sensordynamik-Moduls verfügbar.

Die komplette Trajektorie kann durch den `get_trajectory` Service abgefragt werden, siehe *Services* (Abschnitt 5.3.4.5) für weitere Details.

Um die Karte mit den Bildmerkmalen auf dem *rc_visard* zu speichern, bietet das SLAM-Modul den `save_map` Service an. Dieser kann nur während der Laufzeit (Zustand „RUNNING“) oder nach dem Stoppen (Zustand „HALTED“) verwendet werden.

Eine gespeicherte Karte kann vor dem Starten (Zustand „IDLE“) mit dem Service `load_map` geladen werden. Um vom Zustand „HALTED“ zurück zu Zustand „IDLE“ zu gelangen, kann der `reset` Service verwendet werden. Es ist zu beachten, dass eine falsche Lokalisierung an (visuell) ähnlichen Orten bei der initialen Lokalisierung in der bereits geladenen Karte leichter passieren kann als während des weiteren Betriebs. Das Auswählen eines Startpunkts mit eindeutiger visueller Erscheinung vermeidet dieses Problem. Das SLAM-Modul nimmt daher an, dass der *rc_visard* in der ungefähren Umgebung des Ursprungs der Karte gestartet wird, d.h. innerhalb weniger Meter. Der Ursprung der Karte befindet sich dort, wo das Stereo-INS-Modul gestartet wurde, als die Karte aufgezeichnet wurde.

5.3.4.2 Speicherbeschränkungen

Das SLAM-Modul muss, im Gegensatz zu anderen Softwaremodulen auf dem *rc_visard*, Daten über die Zeit akkumulieren, z.B. Bewegungsmessungen und Bildmerkmale. Weiterhin benötigt die Optimierung der Trajektorie beträchtliche Speichermengen, besonders wenn große Schleifen geschlossen werden müssen. Deswegen steigen die Speicheranforderungen des SLAM-Moduls mit der Zeit.

Durch die Speicherbeschränkungen der Hardware muss das SLAM-Modul seinen Speicherbedarf reduzieren, wenn es kontinuierlich läuft. Wenn der verfügbare Speicher knapp wird, fixiert das SLAM-Modul Teile seiner Trajektorie. Das heißt, diese Teile werden nicht weiter optimiert. Die letzten 10 Minuten der aktuellen Trajektorie sind von der Fixierung jedoch immer ausgenommen.

Wenn trotz der oben genannten Maßnahmen der Speicher knapp wird, stehen zwei Optionen zur Verfügung. Als erste Option kann das SLAM-Modul in den HALTED-Zustand gehen, in dem es keine weiteren Verarbeitungsschritte durchführt, aber die Trajektorie weiterhin verfügbar ist (bis zu dem Zeitpunkt, in dem das Modul in den HALTED-Zustand gewechselt ist). Dies ist das Standardverhalten.

Die zweite Option ist, dass das SLAM-Modul weiterläuft, bis der Speicher aufgebraucht ist. In diesem Fall wird das SLAM-Modul neu gestartet. Wenn der `autorecovery` Parameter auf `true` gesetzt ist, wird das SLAM-Modul seine letzte Position wieder herstellen und die Kartierung fortsetzen. Andernfalls wird das Modul in den FATAL-Zustand wechseln und muss über das rc_dynamics Interface neu gestartet werden (siehe *Services*, Abschnitt 5.3.1.4).

Die Laufzeit bis zum Erreichen des Speicherlimits hängt stark von der Trajektorie des Sensors ab.

Warnung: Aufgrund des limitierten internen Speichers des *rc_visard* ist es nicht empfohlen, *Stereo-Matching Modul* in voller Auflösung (`Full`) gleichzeitig mit dem SLAM-Modul laufen zu lassen. Durch den Speicherverbrauch des Stereo-Matchings ist die maximale Laufzeit des SLAM-Moduls stark eingeschränkt. Eventuell kann ein SLAM-Prozess, der schon länger läuft, durch Starten des Stereo-Matchings in voller Auflösung einen `reset` erfahren.

5.3.4.3 Parameter

Das SLAM-Modul wird in der REST-API als `rc_slam` bezeichnet. Der Benutzer kann die SLAM-Parameter über die *REST-API-Schnittstelle* (Abschnitt 6.2) setzen.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.12: Laufzeitparameter des `rc_slam`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`autorecovery`	bool	false	true	true	Stellt korrigierte Position im Fall von fatalen Fehlern wieder her und startet die Kartierung neu.
`halt_on_low_memory`	bool	false	true	true	Gehe in den HALTED-Zustand, wenn der Speicher knapp wird.

5.3.4.4 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.13: Statuswerte des `rc_slam`-Moduls

Name	Beschreibung
`map_frames`	Anzahl der Posen aus denen die Karte besteht
`state`	Der aktuelle Zustand des SLAM-Moduls
`trajectory_poses`	Anzahl der Posen in der Trajektorienschätzung

Der Parameter `state` kann folgende Werte annehmen:

Tab. 5.14: Mögliche Zustände des SLAM-Moduls

Zustandsname	Beschreibung
IDLE	Das Modul ist bereit, aber inaktiv. Keine Trajektorie ist verfügbar.
WAITING_FOR_DATA	Das Modul wurde gestartet, aber wartet auf Daten des Stereo-INS-Moduls oder der VO.
RUNNING	Das Modul läuft.
HALTED	Das Modul wurde gestoppt. Die Trajektorie ist noch verfügbar. Es werden keine neuen Informationen verarbeitet.
RESETTING	Das Modul wird gestoppt und die internen Daten werden gelöscht.
RESTARTING	Das Modul wird neu gestartet.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

5.3.4.5 Services

Bemerkung: Die Aktivierung und Deaktivierung des SLAM-Moduls wird über das Service-Interface von rc_dynamics gesteuert (siehe *Services*, Abschnitt 5.3.1.4).

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service (außer `reset`) einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von 0 quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind.

Das SLAM-Modul bietet folgende Services.

`get_trajectory`

gibt die Trajektorie zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_slam/services/get_trajectory
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_slam/services/get_trajectory
```

Request

Die Service-Argumente `start_time` und `end_time` ermöglichen die Auswahl eines Trajektorienabschnitts. Sie sind optional und können weggelassen oder mit Null-Werten gefüllt sein. In diesem Fall beginnt der Ausschnitt am Trajektorienanfang bzw. schließt mit dem Trajektorienende. Im anderen Fall stellen sie entweder einen absoluten Zeitstempel dar, oder sie sind über die Flags `start_time_relative` und `end_time_relative` relativ zur Trajektorie zu interpretieren. Ist eine relative Zeitangabe angegeben, entscheidet das Vorzeichen der entsprechenden Werte, auf welchen Zeitpunkt der Trajektorie sie sich bezieht: Positive Werte werden als Offset auf den Zeitpunkt des Trajektorienstarts interpretiert, negative Werte auf den Zeitpunkt des Trajektorienendes. Das folgende Diagramm zeigt drei Beispielparametrisierungen des Services mit relativen Zeitangaben.

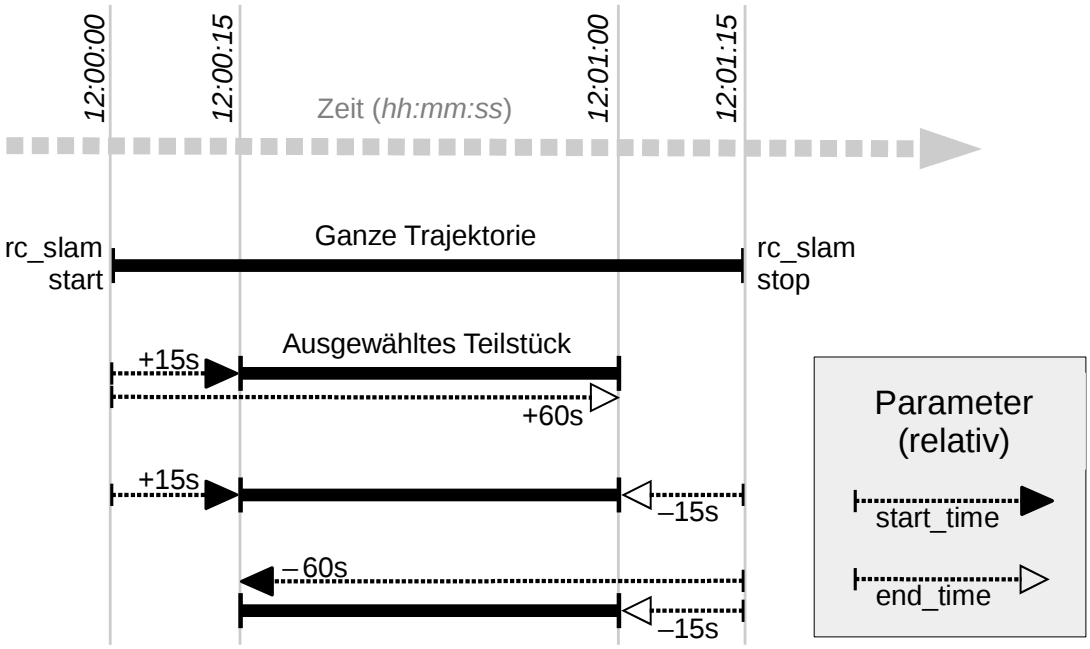


Abb. 5.8: Beispielhafte Kombinationen positiver und negativer relativer Zeitangaben. Alle gezeigten Kombinationen resultieren im gleichen Teilstück.

Bemerkung: Bei relativen Zeitangaben wird eine `start_time` von Null als der Anfang der Trajektorie interpretiert, eine `end_time` von Null wird hingegen als das Ende der Trajektorie gewertet. Absolutangaben von Null werden effektiv genauso behandelt. Sind alle Zeiten Null, wird somit immer die gesamte Trajektorie ausgegeben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "end_time": {
      "nsec": "int32",
      "sec": "int32"
    },
    "end_time_relative": "bool",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "start_time": {
      "nsec": "int32",
      "sec": "int32"
    },
    "start_time_relative": "bool"
  }
}
```

Response

Das `producer`-Feld gibt an, woher die Daten kommen und ist immer `slam`.

Das Feld `return_code` enthält mögliche Warnungen oder Fehlercodes und Nachrichten. Mögliche Werte für `return_code` sind in der Tabelle unten angegeben.

Code	Beschreibung
0	Erfolg
-1	Die Eingabeargumente sind ungültig (z.B. eine ungültige Angabe für den Zeitraum)
101	Die Trajektorie ist leer, weil es keine Daten im angegebenen Zeitraum gibt
102	Die Trajektorie ist leer, weil keine Daten verfügbar sind (z.B. wenn SLAM noch im Zustand IDLE ist)

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_trajectory",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "trajectory": {
      "name": "string",
      "parent": "string",
      "poses": [
        {
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "timestamp": {
            "nsec": "int32",
            "sec": "int32"
          }
        }
      ],
      "producer": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      }
    }
  }
}
```

save_map

speichert die aktuelle Karte im persistenten Speicher. Die Karte besteht aus mehreren unveränderlichen Kartenabschnitten. Die Trajektorie ist nicht Teil der Karte.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_slam/services/save_map
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_slam/services/save_map
```

Bemerkung: Nur abstrakte Beschreibungen von Bildmerkmalen und deren Positionen werden in der Karte gespeichert. Weder werden Kameraaufnahmen in der Karte gespeichert, noch ist es möglich, Bilder oder Bildteile aus den gespeicherten Informationen zu rekonstruieren.

Warnung: Die Karte wird nicht über Software Aktualisierungen oder Rollbacks hinweg behalten.

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "save_map",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

load_map

lädt eine vorher gespeicherte Karte.

Details

Dies ist nur möglich, wenn sich das SLAM-Modul im Zustand IDLE befindet, d.h. es ist nicht möglich eine Karte in ein laufendes System zu laden. Eine geladene Karte kann mit dem `reset` Serviceaufruf zurückgesetzt werden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_slam/services/load_map
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_slam/services/load_map
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "load_map",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

remove_map

entfernt die gespeicherte Karte aus dem persistenten Speicher.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_slam/services/remove_map
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_slam/services/remove_map
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "remove_map",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`reset`

löscht den internen Zustand des SLAM-Moduls.

Details

Dieser Service sollte genutzt werden, wenn das SLAM-Modul über das rc_dynamics Interface (siehe *Services*, Abschnitt 5.3.1.4) gestoppt wurde. Das SLAM-Modul behält die Schätzung der kompletten Trajektorie, auch wenn es gestoppt ist. Dieser Service löscht diese Trajektorie und gibt den zugehörigen Speicher frei. Der zurückgegebene Zustand ist RESETTING.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_slam/services/reset
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_slam/services/reset
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

5.4 Detektions- und Messmodule

Der *rc_visard* bietet Softwaremodule für unterschiedliche Detektions- und Messanwendungen:

- ***Measure*** **(`rc_measure`, Abschnitt 5.4.1)** ermöglicht die Messung von Tiefenwerten
- ***LoadCarrier*** **(`rc_load_carrier`, Abschnitt 5.4.2)** ermöglicht die Erkennung von Load Carriern (Behältern) und ihres Füllstands.
- ***TagDetect*** **(`rc_april_tag_detect` und `rc_qr_code_detect`, Abschnitt 5.4.3)** ermöglicht die Erkennung von AprilTags und QR-Codes sowie die Schätzung von deren Pose.
- ***ItemPick*** **(`rc_itempick`, Abschnitt 5.4.4)** bietet eine Standardlösung für robotische Pick-and-Place-Anwendungen für oder unbekannte Objekte.
- ***BoxPick*** **(`rc_boxpick`, Abschnitt 5.4.5)** bietet eine Standardlösung für robotische Pick-and-Place-Anwendungen für Boxen oder texturierte Boxen.
- ***Pack*** **(`rc_pack`, Abschnitt 5.4.6)** berechnet Platzierungsposen für Objekte innerhalb eines Packvolumens oder Load Carriers.
- ***SilhouetteMatch*** **(`rc_silhouettematch`, Abschnitt 5.4.7)** bietet eine Objekterkennungslösung für Objekte, die sich auf einer Ebene befinden, oder gestapelte planare Objekt.

Diese Module sind optional und können durch Kauf einer separaten *Lizenz* (Abschnitt 7.4) aktiviert werden.

5.4.1 Measure

5.4.1.1 Einleitung

Das Measure Modul ermöglicht die Messung von Tiefenwerten in einer Region of Interest und die Schätzung von Ebenen aus den aktuellen Sensordaten.

Das Measure Modul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

5.4.1.2 Tiefe messen

Das Measure Modul bietet Funktionen zum Messen von Tiefenwerten in der aktuellen Szene in einer 2D Region of Interest. Optional kann die Region of Interest in bis zu 100 Zellen unterteilt werden, für die separate Tiefenmessungen erfolgen, die zusätzlich zu den Tiefenmessungen für die gesamte Region of Interest zurückgegeben werden.

Die Tiefenmessung besteht aus der durchschnittlichen Tiefe `mean_z`, der minimalen Tiefe `min_z` und der maximalen Tiefe `max_z`, die jeweils 3D-Koordinaten enthalten. Die Koordinaten der Messungen `min_z` und `max_z` entsprechen dem Punkt in der Zelle oder der gesamten Region of Interest mit dem minimalen bzw. maximalen Abstand von der Kamera. Die `x`- und `y`-Koordinaten der `mean_z`-Messungen definieren einen Punkt in der Mitte der Zelle oder der gesamten Region of Interest und die `z`-Koordinate wird aus dem Durchschnitt aller Tiefenwerte (Entfernungen von der Kamera) in diesem Bereich bestimmt. Darüber hinaus wird für jede Zelle und die gesamte Region of Interest ein `coverage` Wert zurückgegeben. Dabei handelt es sich um eine Zahl zwischen 0 und 1, die den Anteil der gültigen Tiefenwerte innerhalb der jeweiligen Region widerspiegelt. Ein `coverage` Wert von 0 bedeutet, dass die Zelle ungültig ist und kein Tiefenwert berechnet werden konnte.

Wenn als Referenzkoordinatensystem (`pose_frame`) `external` für die Tiefenmessungen verwendet wird, werden alle 3D-Koordinaten wie oben beschrieben berechnet, dann aber in das externe Koordinatensystem transformiert. Das heißt, die Tiefe wird immer entlang der Sichtlinie der Kamera gemessen, unabhängig vom gewählten Referenzkoordinatensystem.

5.4.1.3 Schätzung von Ebenen

Das Measure Modul bietet die Möglichkeit, eine Ebene in der aktuellen Szene zu schätzen, z.B. die Oberfläche eines Förderbands oder eines Tischs.

Die Ebene wird in Hessescher Normalform mit einer Normalen `normal` und einem Abstand `distance` zurückgegeben (siehe *Ebene*, Abschnitt 5.6.3.2). Jede Ebenenschätzung löst eine neue Bildaufnahme aus.

Die Ebene kann mit zwei verschiedenen Methoden geschätzt werden (`plane_estimation_method`):

- `DEPTH`: Die Ebene wird aus der Punktwolke geschätzt. Werden mehrere Ebenen gefunden, legt `plane_preference` fest, ob die am weitesten entfernte (`FARTHEST`, Standardwert) oder die nächstgelegene (`CLOSEST`) Ebene zurückgegeben wird.
- `APRILTAG`: Die Ebene wird aus mindestens drei AprilTags geschätzt, die auf einer gemeinsamen Ebene liegen. Diese Methode erfordert eine gültige Lizenz für die AprilTag-Erkennung (siehe *TagDetect*, Abschnitt 5.4.3). Die AprilTags werden in Bildern ohne Projektormuster erkannt. Wird ein Projektor verwendet, muss `out1_mode` des IOControl Moduls auf `Low` oder `ExposureAlternateActive` gesetzt sein.

Optional kann die Ebenenschätzung auf eine 2D Region of Interest beschränkt werden. Mit `offset` kann eine Verschiebung in Metern angegeben werden, um die die zurückgegebene Ebene entlang der Sichtlinie der Kamera verschoben wird. Ein positiver Wert verschiebt die Ebene in Richtung der Kamera.

Die Ebenenschätzung schlägt fehl, wenn keine Ebene gefunden wird, wenn weniger als drei AprilTags sichtbar sind oder diese nicht auf einer gemeinsamen Ebene liegen, oder wenn die Ebene um mehr als 45° gegen die optische Achse der Kamera geneigt ist.

Die Ebene kann mit dem Service `set_plane` im *PrimitiveDB* (Abschnitt 5.6.3) gespeichert und dann z.B. als Referenzebene für die Dimensionierung in *BoxPick* (Abschnitt 5.4.5.2) verwendet werden.

5.4.1.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das Measure Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des Measure Moduls haben.

Tiefenbilder

Folgende Daten werden vom Measure Modul verarbeitet:

- Die Disparitätsbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 5.2).

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das Measure Modul automatisch Punkte im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.4.1.7) kann das Koordinatensystem der berechneten Punkte mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Punkte sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Punkte der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Punkte sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Measure Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

5.4.1.5 Parameter

Das Measure Modul wird in der REST-API als `rc_measure` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Module* → *Measure* dargestellt.

Übersicht über die Parameter

Dieses Modul besitzt keine Laufzeitparameter.

5.4.1.6 Statuswerte

Das Measure Modul meldet folgende Statuswerte:

Tab. 5.15: Statuswerte des `rc_measure` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Tiefenbilddaten gewartet werden musste
`last_return_code_message`	Rückgabecode-Nachricht der letzten Detektion
`last_return_code_value`	Rückgabecode-Wert der letzten Detektion
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Tiefenbilds
`processing_time`	Berechnungszeit für die letzte Messung in Sekunden

5.4.1.7 Services

Die angebotenen Services des Measure Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) auf der Seite *Measure* unter dem Menüpunkt *Module* ausprobiert und getestet werden.

Das Measure Modul stellt folgende Services zur Verfügung.

`measure_depth`

Berechnet die durchschnittliche, minimale und maximale Tiefe in einer angegebenen Region of Interest, die optional in Zellen unterteilt werden kann.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_measure/services/measure_depth
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_measure/services/measure_depth
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.1.4).

Optionale Serviceargumente:

`region_of_interest_2d_id`: Die ID der 2D Region of Interest (siehe *RoiDB*, Abschnitt 5.6.2), innerhalb welcher die Tiefenmessung durchgeführt werden soll.

`region_of_interest_2d` ist eine alternative Definition der Region of Interest für die Tiefenmessung. Diese Region of Interest wird ignoriert, falls eine `region_of_interest_2d_id` gesetzt ist. Die Region of Interest wird immer auf dem Kamerabild mit voller Auflösung definiert, wobei `offset_x` und `offset_y` die Pixelkoordinaten der oberen linken Ecke der rechteckigen Region of Interest sind, und `width` und `height` die Breite und Höhe des Rechtecks in Pixeln angeben. Der Standardwert ist eine Region of Interest, die das gesamte Bild abdeckt.

`cell_count` ist die Anzahl der Zellen in x und y Richtung, in die die Region of Interest für die Tiefenmessung unterteilt wird. Falls nicht angegeben, wird ein `cell_count` von 0, 0 angenommen und es werden nur die Gesamtwerte `overall`

berechnet. Die Gesamtanzahl der Zellen, die als Produkt aus den x und y Werten des `cell_count` berechnet werden kann, darf nicht größer sein als 100.

`data_acquisition_mode`: Falls der Aufnahmemodus auf `CAPTURE_NEW` (Standardwert) gesetzt ist, wird ein neuer Bild-Datensatz für die Messung aufgenommen. Falls der Modus auf `USE_LAST` gesetzt wird, wird der Datensatz der vorherigen Messung erneut verwendet.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.1.4).

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "cell_count": {
      "x": "uint32",
      "y": "uint32"
    },
    "data_acquisition_mode": "string",
    "pose_frame": "string",
    "region_of_interest_2d": {
      "height": "uint32",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    },
    "region_of_interest_2d_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

`cells` enthält die Tiefenmessungen aller gewünschter Zellen. Die Zellen sind immer von links nach rechts und oben nach unten in Bildkoordinaten sortiert.

`overall` enthält die Tiefenmessung der gesamten Region of Interest.

`coverage` enthält den Anteil der Pixel mit gültigen Tiefenmesswerten, wie in *Tiefe messen* (Abschnitt 5.4.1.2) beschrieben.

`mean_z`, `min_z` und `max_z` enthalten die gemessenen Koordinaten wie in *Tiefe messen* (Abschnitt 5.4.1.2).

`region_of_interest_2d` gibt die Definition der angeforderten Region of Interest für die Tiefenmessung zurück.

`pose_frame` enthält das Koordinatensystem der Tiefenmessungen.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "measure_depth",
  "response": {
    "cell_count": {
      "x": "uint32",
      "y": "uint32"
    },
    "cells": [
      {
        "coverage": "float64",
        "max_z": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "mean_z": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "min_z": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    ],
    "overall": {
      "coverage": "float64",
      "max_z": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "mean_z": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "min_z": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "pose_frame": "string",
    "region_of_interest_2d": {
      "height": "uint32",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

compute_plane

schätzt eine Ebene aus den aktuellen Sensordaten wie in *Schätzung von Ebenen* (Abschnitt 5.4.1.3) beschrieben und gibt sie zurück, ohne sie zu speichern.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_measure/services/compute_plane
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_measure/services/compute_plane
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.1.4).

`plane_estimation_method`: Methode zur Schätzung der Ebene, entweder `DEPTH` oder `APRILTAG`.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.1.4).

Optionale Serviceargumente:

`depth`: Optionen für die Methode `DEPTH`. `plane_preference` legt fest, ob die am weitesten entfernte (`FARTHEST`, Standardwert) oder die nächstgelegene (`CLOSEST`) Ebene zurückgegeben wird, falls mehrere Ebenen gefunden werden.

`region_of_interest_2d_id`: Die ID der 2D Region of Interest (siehe *RoiDB*, Abschnitt 5.6.2), innerhalb welcher die Ebenenschätzung durchgeführt werden soll. Standardmäßig wird das ganze Bild verwendet.

`offset`: Verschiebung der zurückgegebenen Ebene in Metern entlang der Sichtlinie der Kamera. Ein positiver Wert verschiebt die Ebene in Richtung der Kamera.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "depth": {
      "plane_preference": "string"
    },
    "offset": "float64",
    "plane_estimation_method": "string",
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

Response

`plane` enthält die geschätzte Ebene mit `normal`, `distance` und `pose_frame` (siehe *Ebene*, Abschnitt 5.6.3.2).

`timestamp`: Aufnahmezeitpunkt der Daten, aus denen die Ebene geschätzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "compute_plane",
  "response": {
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

5.4.1.8 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.16: Rückgabecodes der Services des Measure Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-8	Die Anfrage ist nicht anwendbar, z.B. wird die Methode `APRILTAG` in der aktuellen Pipeline nicht unterstützt.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-101	Es konnte keine Ebene geschätzt werden, z.B. weil keine Ebene gefunden wurde oder die AprilTags nicht auf einer gemeinsamen Ebene liegen.
-104	Die geschätzte Ebene ist zu stark gegen die optische Achse der Kamera geneigt.

5.4.2 LoadCarrier

5.4.2.1 Einleitung

Das LoadCarrier Modul ermöglicht die Erkennung von Load Carriern, was oftmals der erste Schritt für die Erkennung von Objekten oder Berechnung von Greifpunkten in einem Behälter ist. Die Modelle der zu erkennenden Load Carrier müssen im *LoadCarrierDB* (Abschnitt 5.6.1) Modul definiert werden.

Das LoadCarrier Modul ist ein optionales Modul, welches intern auf dem *rc_visard* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5) oder *SilhouetteMatch* (Abschnitt 5.4.7) vorhanden ist. Andernfalls benötigt es eine gesonderte LoadCarrier *Lizenz* (Abschnitt 7.4), welche erworben werden muss.

5.4.2.2 Erkennung von Load Carriern

Der Algorithmus zur Erkennung von Load Carriern detektiert Load Carrier, die einem speziellen Load Carrier Modell entsprechen, welches im *LoadCarrierDB* (Abschnitt 5.6.1) Modul definiert werden muss. Das Load Carrier Modell wird über seine ID referenziert, die bei der Load Carrier Detektion übergeben wird. Die Erkennung von Load Carriern basiert auf der Erkennung des rechteckigen Load Carrier Rands. Dazu werden detektierte Linien im linken Kamerabild und die Tiefenwerte des Load Carrier Randes genutzt. Daher sollte der Rand einen Kontrast zum Hintergrund bilden und das Disparitätsbild auf dem Rand dicht sein.

Wenn mehrere Load Carrier mit der angegeben Load Carrier ID in der Szene sichtbar sind, werden alle von ihnen detektiert und zurückgeliefert.

Standardmäßig, wenn `assume_gravity_aligned` aktiv ist und Gravitationsmessungen verfügbar sind, wird nach Load Carriern gesucht, deren Randebene senkrecht zur gemessenen Gravitationsrichtung ausgerichtet ist. Um geneigte Load Carrier zu erkennen, muss `assume_gravity_aligned` deaktiviert werden oder deren ungefähre Orientierung als Pose `pose` in einem Referenzkoordinatensystem `pose_frame` angegeben werden, und der Posentyp `pose_type` auf `ORIENTATION_PRIOR` gesetzt werden.

Load Carrier können höchstens bis zu einer Entfernung von 3 Metern von der Kamera erkannt werden.

Wenn eine 3D Region of Interest (siehe *RoiDB*, Abschnitt 5.6.2) genutzt wird, um das Volumen für die Load Carrier Erkennung einzuschränken, müssen nur die Ränder der Load Carrier vollständig in der Region of Interest enthalten sein.

Die Erkennung liefert die Posen der Load Carrier Koordinatensysteme (siehe *Load Carrier Definition*, Abschnitt 5.6.1.2) im gewünschten Referenzkoordinatensystem zurück.

Bei der Erkennung wird auch ermittelt, ob die Load Carrier überfüllt (`overfilled`) sind, was bedeutet, dass Objekte aus dem Load Carrier herausragen.

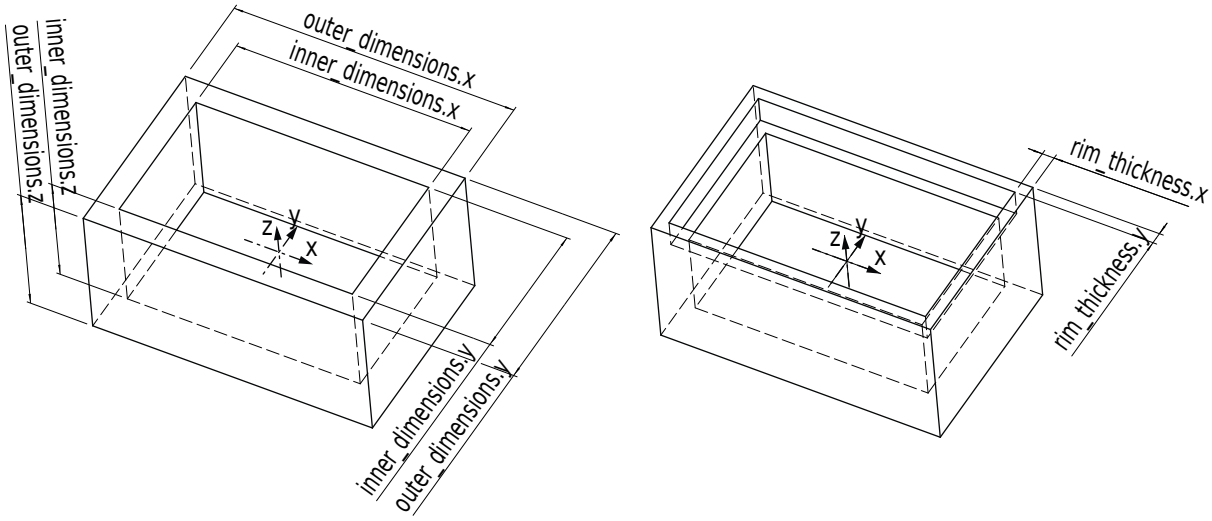


Abb. 5.9: Illustration verschiedener Load Carrier Modelle und deren Koordinatensysteme

5.4.2.3 Füllstandserkennung

Das LoadCarrier Modul bietet den Service `detect_filling_level` an, um den Füllstand aller erkannten Load Carrier zu berechnen.

Dazu werden die Load Carrier in eine konfigurierbare Anzahl von Zellen unterteilt, welche in einem 2D-Raster angeordnet sind. Die maximale Anzahl beträgt 10x10 Zellen. Für jede Zelle werden folgende Werte ermittelt:

- `level_in_percent`: Minimum, Maximum und Mittelwert des Füllstands vom Boden in Prozent. Diese Werte können größer als 100% sein, falls die Zelle überfüllt ist.
- `level_free_in_meters`: Minimum, Maximum und Mittelwert in Metern des freien Teils der Zelle vom Rand des Load Carriers gemessen. Diese Werte können negativ sein, falls die Zelle überfüllt ist.
- `cell_size`: Abmessungen der 2D-Zelle in Metern.
- `cell_position`: Position des Mittelpunkts der Zelle in Metern (entweder im Koordinatensystem `camera` oder `external`, siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.4.4.5). Die z-Koordinate liegt auf der Ebene des Load Carrier Randes.
- `coverage`: Anteil der gültigen Pixel in dieser Zelle. Dieser Wert reicht von 0 bis 1 in Schritten von 0.1. Ein niedriger Wert besagt, dass die Zelle fehlende Daten beinhaltet (d.h. nur wenige Punkte konnten in der Zelle gemessen werden).

Diese Werte werden auch für den gesamten Load Carrier berechnet. Falls keine Zellunterteilung angegeben ist, wird nur der Gesamtfüllstand (`overall_filling_level`) berechnet.

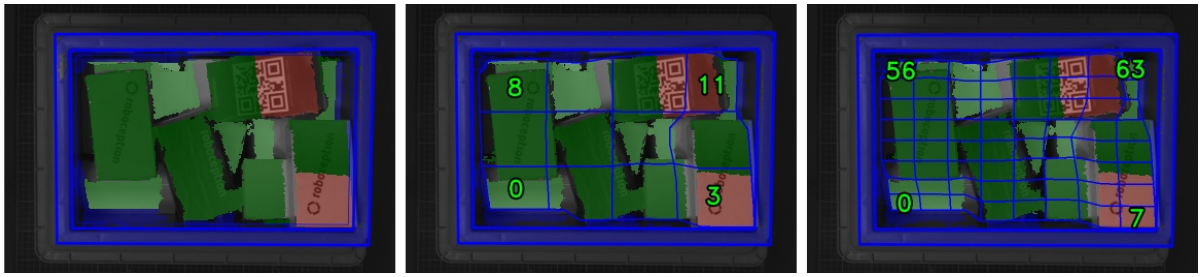


Abb. 5.10: Visualisierungen des Behälterfüllstands: Gesamtfüllstand ohne Raster (links), 4x3 Raster (Mitte), 8x8 Raster (rechts). Der Inhalt wird mit einem Farbverlauf von weiß (auf dem Boden) nach dunkelgrün dargestellt. Die überfüllten Bereiche sind rot dargestellt. Die Nummern stellen die Zell-IDs dar.

5.4.2.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das LoadCarrier Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des LoadCarrier Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom LoadCarrier Modul verarbeitet:

- Die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 5.1);
- Die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 5.2).

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_visard* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.5.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.5.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Darüber hinaus sind keine weiteren Änderungen für diesen Anwendungsfall notwendig.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die Load Carrier Komponente automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.4.2.7) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Load Carrier Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

5.4.2.5 Parameter

Das LoadCarrier Modul wird in der REST-API als `rc_load_carrier` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Module* → *LoadCarrier* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Modul bietet folgende Laufzeitparameter:

Tab. 5.17: Laufzeitparameter des `rc_load_carrier` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`assume_gravity_aligned`	bool	false	true	true	Wenn dieser Parameter aktiv ist, werden nur waagerechte Load Carrier erkannt falls eine Gravitationsmessung verfügbar ist.
`crop_distance`	float64	0.0	0.05	0.005	Sicherheitsspielraum um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren
`min_plausibility`	float64	0.0	0.99	0.8	Gibt an, wie viel von der Ebene um den Load Carrier Rand herum mindestens frei sein muss, um als gültige Erkennung zu zählen.
`model_tolerance`	float64	0.003	0.025	0.008	Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden in der Web GUI zeilenweise im Abschnitt *LoadCarrier Einstellungen* auf der Seite *LoadCarrier* unter *Module* dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet. Wenn die Parameter außerhalb des `rc_load_carrier`

Moduls über die *REST-API-Schnittstelle* (Abschnitt 6.2) eines anderen Moduls verwendet werden, sind sie durch den Präfix `load_carrier_` gekennzeichnet.

`assume_gravity_aligned` (*Gravitationsausgerichtet*)

Wenn dieser Parameter aktiv ist, werden nur waagerechte Load Carrier erkannt. Dies kann die Erkennung beschleunigen. Wenn dieser Parameter nicht aktiv ist, werden auch Load Carrier mit Verkippung detektiert.

Für Load Carrier mit einem Orientierungsprior wird dieser Parameter ignoriert.

`model_tolerance` (*Modelltoleranz*)

Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/parameters?model_tolerance=
↪<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?model_tolerance=<value>
```

`crop_distance` (*Cropping*)

setzt den Sicherheitsspielraum, um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren (siehe Abb. 5.41).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/parameters?crop_distance=
↪<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?crop_distance=<value>
```

`min_plausibility` (*Minimale Plausibilität*):

Die minimale Plausibilität gibt an, wie viel von der Ebene um den Load Carrier Rand herum mindestens frei sein muss, um als gültige Erkennung zu zählen. Erhöhen Sie die minimale Plausibilität um falsch-positive Erkennungen zu vermeiden, und verringern Sie den Wert, wenn ein deutlich sichtbarer Load Carrier nicht erkannt wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/parameters?min_plausibility=
↪<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?min_plausibility=<value>
```

5.4.2.6 Statuswerte

Das LoadCarrier Modul meldet folgende Statuswerte:

Tab. 5.18: Statuswerte des `rc_load_carrier` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_return_code_message`	Rückgabecode-Nachricht der letzten Detektion
`last_return_code_value`	Rückgabecode-Wert der letzten Detektion
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden

5.4.2.7 Services

Die angebotenen Services des LoadCarrier Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) auf der Seite *LoadCarrier* unter dem Menüpunkt *Module* ausprobiert und getestet werden.

Das LoadCarrier Modul stellt folgende Services zur Verfügung.

`detect_load_carriers`

löst die Erkennung von Load Carriern aus, wie in *Erkennung von Load Carriern* (Abschnitt 5.4.2.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/services/detect_load_
↪carriers
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/detect_load_carriers
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.2.4).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier. Aktuell kann nur eine ID angegeben werden.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.2.4).

Optionale Serviceargumente:

`region_of_interest_id`: Die ID der 3D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

`region_of_interest_2d_id`: Die ID der 2D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

Bemerkung: Es kann nur eine Art von Region of Interest kann angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ],
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_load_carriers",
  "response": {
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


detect_filling_level

löst eine Load Carrier Füllstandserkennung aus, wie in *Füllstandserkennung* (Abschnitt 5.4.2.3) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/services/detect_filling_
↪level
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/detect_filling_level
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.2.4).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier. Aktuell kann nur eine ID angegeben werden.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.2.4).

Optionale Serviceargumente:

`filling_level_cell_count`: Anzahl der Zellen im Füllstandsraster.

`region_of_interest_id`: Die ID der 3D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

`region_of_interest_2d_id`: Die ID der 2D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

Bemerkung: Es kann nur eine Art von Region of Interest kann angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "filling_level_cell_count": {
      "x": "uint32",
      "y": "uint32"
    },
    "load_carrier_ids": [
      "string"
    ],
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

`load_carriers`: Liste an erkannten Load Carriern und deren Füllstand.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_filling_level",
  "response": {
    "load_carriers": [
      {
        "cells_filling_levels": [
          {
            "cell_position": {
              "x": "float64",
              "y": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "cell_size": {
            "x": "float64",
            "y": "float64"
          },
          "coverage": "float64",
          "level_free_in_meters": {
            "max": "float64",
            "mean": "float64",
            "min": "float64"
          },
          "level_in_percent": {
            "max": "float64",
            "mean": "float64",
            "min": "float64"
          }
        }
      ],
      "filling_level_cell_count": {
        "x": "uint32",
        "y": "uint32"
      },
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overall_filling_level": {
        "cell_position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "cell_size": {
          "x": "float64",
          "y": "float64"
        },
        "coverage": "float64",
        "level_free_in_meters": {
          "max": "float64",
          "mean": "float64",
          "min": "float64"
        },
        "level_in_percent": {
          "max": "float64",
          "mean": "float64",
          "min": "float64"
        }
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


detect_filling_level_as_image

löst dieselbe Load Carrier Füllstandserkennung wie `detect_filling_level` aus, liefert die Füllstandswerte jedoch als Bilder statt als Liste von Zellen zurück. Dies führt zu kürzeren Antwortzeiten, insbesondere wenn die Anzahl der Zellen groß ist. Die Mittelwert-, Minimum- und Maximumwerte von `level_free_in_meters` sowie die `coverage` werden als base64-kodierte, glib-komprimierte Bilder zurückgegeben. Jedes Pixel entspricht einer Zelle im Füllstandsraster, geordnet nach dem Zellenindex.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/services/detect_filling_
↪level_as_image
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/detect_filling_level_as_image
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.2.4).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier. Aktuell kann nur eine ID angegeben werden.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.2.4).

Optionale Serviceargumente:

`filling_level_cell_count`: Anzahl der Zellen im Füllstandsraster.

`region_of_interest_id`: Die ID der 3D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

`region_of_interest_2d_id`: Die ID der 2D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

Bemerkung: Es kann nur eine Art von Region of Interest kann angegeben werden.

Response

`load_carriers`: Liste an erkannten Load Carriern und deren Füllstand.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_filling_level_as_image",
  "response": {
    "load_carriers": [
      {
        "filling_level_cell_count": {
          "x": "uint32",
          "y": "uint32"
        },
        "filling_level_cell_size": {
          "x": "float64",
          "y": "float64"
        },
        "filling_level_images": {
          "coverage": "string",
          "level_free_in_meters_max": "string",
          "level_free_in_meters_mean": "string",
          "level_free_in_meters_min": "string"
        },
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overall_filling_level": {
          "cell_position": {
            "x": "float64",
            "y": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "cell_size": {
            "x": "float64",
            "y": "float64"
          },
          "coverage": "float64",
          "level_free_in_meters": {
            "max": "float64",
            "mean": "float64",
            "min": "float64"
          },
          "level_in_percent": {
            "max": "float64",
            "mean": "float64",
            "min": "float64"
          }
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_load_carrier/services/reset_defaults
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`set_load_carrier` (veraltet)

speichert einen Load Carrier auf dem *rc_visard*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_load_carrier* (Abschnitt 5.6.1.5) in `rc_load_carrier_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/set_load_carrier
```

Die Definitionen von Request und Response sind dieselben wie in *set_load_carrier* (Abschnitt 5.6.1.5) in `rc_load_carrier_db` beschrieben.

`get_load_carriers` (veraltet)

gibt die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_load_carriers* (Abschnitt 5.6.1.5) in `rc_load_carrier_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/get_load_carriers
```

Die Definitionen von Request und Response sind dieselben wie in *get_load_carriers* (Abschnitt 5.6.1.5) in `rc_load_carrier_db` beschrieben.

`delete_load_carriers` (veraltet)

löscht die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_load_carriers* (Abschnitt 5.6.1.5) in `rc_load_carrier_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/delete_load_carriers
```

Die Definitionen von Request und Response sind dieselben wie in *delete_load_carriers* (Abschnitt 5.6.1.5) in `rc_load_carrier_db` beschrieben.

`set_region_of_interest` (veraltet)

speichert eine 3D Region of Interest auf dem *rc_visard*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_region_of_interest* (Abschnitt 5.6.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/set_region_of_interest
```

Die Definitionen von Request und Response sind dieselben wie in *set_region_of_interest* (Abschnitt 5.6.2.4) in `rc_roi_db` beschrieben.

`get_regions_of_interest` (veraltet)

gibt die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D Regions of Interest zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_regions_of_interest* (Abschnitt 5.6.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/get_regions_of_interest
```

Die Definitionen von Request und Response sind dieselben wie in *get_regions_of_interest* (Abschnitt 5.6.2.4) in `rc_roi_db` beschrieben.

`delete_regions_of_interest` (veraltet)

löscht die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D Regions of Interest.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_regions_of_interest* (Abschnitt 5.6.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/delete_regions_of_interest
```

Die Definitionen von Request und Response sind dieselben wie in *delete_regions_of_interest* (Abschnitt 5.6.2.4) in `rc_roi_db` beschrieben.

`set_region_of_interest_2d` (veraltet)

speichert eine 2D Region of Interest auf dem *rc_visard*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_region_of_interest_2d* (Abschnitt 5.6.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/set_region_of_interest_2d
```

Die Definitionen von Request und Response sind dieselben wie in *set_region_of_interest_2d* (Abschnitt 5.6.2.4) in `rc_roi_db` beschrieben.

`get_regions_of_interest_2d` (veraltet)

gibt die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D Regions of Interest zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_regions_of_interest_2d* (Abschnitt 5.6.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/get_region_of_interest_2d
```

Die Definitionen von Request und Response sind dieselben wie in *get_regions_of_interest_2d* (Abschnitt 5.6.2.4) in `rc_roi_db` beschrieben.

`delete_regions_of_interest_2d` (veraltet)

löscht die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D Regions of Interest.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_regions_of_interest_2d* (Abschnitt 5.6.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/delete_regions_of_interest_2d
```

Die Definitionen von Request und Response sind dieselben wie in *delete_regions_of_interest_2d* (Abschnitt 5.6.2.4) in `rc_roi_db` beschrieben.

5.4.2.8 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.19: Rückgabecodes der Services des LoadCarrier Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-302	Es wurde mehr als ein Load Carrier an den `detect_load_carriers` oder `detect_filling_level` Service übergeben, aber nur einer wird unterstützt.
3	Der Timeout während der Load Carrier Erkennung wurde erreicht. Die Modelltoleranz sollte reduziert werden.
10	Die maximal speicherbare Anzahl an Load Carriern wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
102	Der erkannte Load Carrier enthält keine 3D-Punkte
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.

5.4.3 TagDetect

5.4.3.1 Einführung

Die TagDetect-Module sind optionale Module, die intern auf dem *rc_visard* laufen, und benötigen gesonderte *Lizenzen* (Abschnitt 7.4), die erworben werden müssen. Diese Lizenzen sind auf jedem *rc_visard*, der nach dem 01.07.2020 gekauft wurde, vorhanden.

Die TagDetect-Module laufen intern auf dem *rc_visard* und ermöglichen es, 2D-Matrixcodes und Marker (Tags) zu erkennen. Derzeit gibt es TagDetect-Module für *QR-Codes* und *AprilTags*. Neben der Erkennung berechnen die Module die Position und Orientierung jedes Tags im 3D-Kamerakoordinatensystem, um diesen beispielsweise mit einem Roboter zu manipulieren oder die Pose der Kamera in Bezug auf den Tag zu berechnen.

Die Tagerkennung besteht aus drei Schritten:

1. Tagerkennung auf dem 2D-Bildpaar (siehe *Tagerkennung*, Abschnitt 5.4.3.2).
2. Schätzung der Pose jedes Tags (siehe *Posenschätzung*, Abschnitt 5.4.3.3).
3. Wiedererkennung von bisher gesehenen Tags (siehe *Tag-Wiedererkennung*, Abschnitt 5.4.3.4).

Im Folgenden werden die zwei unterstützten Tagtypen näher beschrieben, gefolgt von einem Vergleich.

QR-Code



Abb. 5.11: Beispiel eines QR-Codes

QR-Codes sind zweidimensionale Matrixcodes, welche beliebige, benutzerspezifizierte Daten enthalten können. Viele Alltagsgeräte, wie beispielsweise Smartphones, unterstützen die Erkennung von QR-Codes. Zusätzlich stehen Online- und Offlinetools zur Verfügung, um QR-Codes zu generieren.

Die „Pixel“ eines QR-Codes werden *Module* genannt. Das Aussehen und die Auflösung von QR-Codes ändert sich mit der Menge der in ihnen gespeicherten Daten. Während die speziellen Muster in den drei Ecken immer 7 Module breit sind, erhöht sich die Anzahl der Module dazwischen, je mehr Daten gespeichert sind. Der am niedrigsten aufgelöste QR-Code besitzt eine Größe von 21x21 Modulen und kann bis zu 152 Bits speichern.

Auch wenn viele QR-Code-Generatoren speziell designte QR-Codes erzeugen können (bspw. mit einem Logo, mit runden Ecken oder mit Punkten als Module), wird eine zuverlässige Erkennung solcher Tags mit dem TagDetect-Modul nicht garantiert. Gleiches gilt für QR-Codes, welche Zeichen außerhalb des ASCII-Zeichensatzes beinhalten.

AprilTag

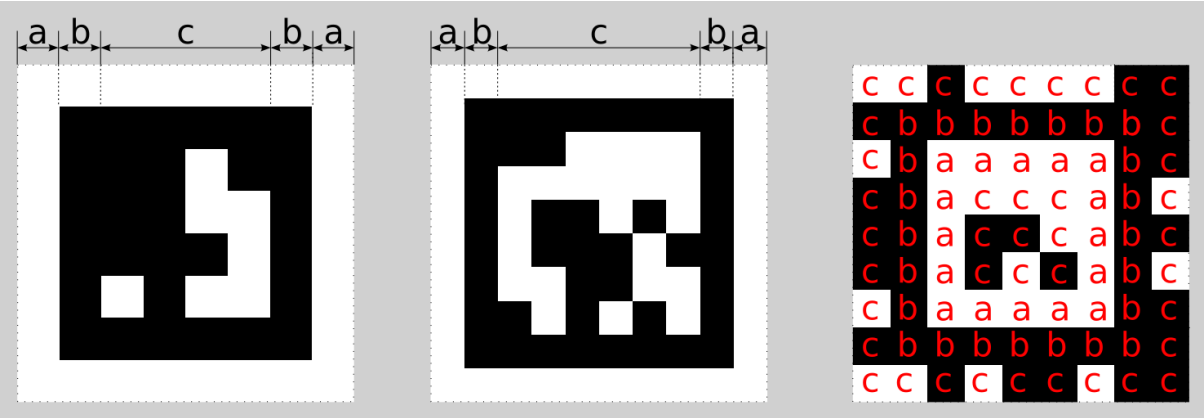


Abb. 5.12: Ein 16h5 Tag (links), ein 36h11 Tag (Mitte) und ein 41h12 Tag (rechts). AprilTags bestehen aus einem obligatorischen weißen (a) und schwarzen (b) Rahmen und einer variablen Menge an Datenmodulen (c).

AprilTags sind ähnlich zu QR-Codes. Sie wurden allerdings speziell zur robusten Identifikation auf weite Entfernungen entwickelt. Wie bei QR-Codes werden die „Pixel“ *Module* genannt. Abb. 5.12 veranschaulicht den Aufbau von AprilTags. Sie haben einen obligatorischen weißen und schwarzen Rahmen, welcher jeweils ein Modul breit ist. Tags der Familien 16h5, 25h9, 36h10 und 36h11 sind von diesem Rahmen umschlossen und enthalten innen eine variable Menge an Datenmodulen. Bei Tags der Familie 41h12 ist der Rahmen nach innen verschoben und die Datenmodule befinden sich sowohl innerhalb als auch außerhalb des Rahmens. Anders als QR-Codes speichern AprilTags keine benutzerdefinierten Informationen, sondern werden durch eine vordefinierte *Familie* und *ID* identifiziert. Die Tags in Abb. 5.12 sind zum Beispiel aus Familie 16h5, 36h11 bzw. 41h12 und besitzen ID 0, 11 bzw. 0. Alle unterstützten Familien werden in Tab. 5.20 aufgelistet.

Tab. 5.20: AprilTag-Familien

Familie	Anzahl IDs	Empfohlen
16h5	30	-
25h9	35	o
36h10	2320	o
36h11	587	+
41h12	2115	+

Die Zahl vor dem „h“ jeder Familie bezeichnet die Anzahl der Datenmodule, welche im Tag enthalten sind: Während ein 16h5 Tag 16 (4x4) Datenmodule enthält ((c) in Abb. 5.12) und ein 36h11 Tag 36 (6x6), beinhaltet ein 41h12 Tag 41 Datenmodule (3x3 innen und 4x8 außen). Die Zahl hinter dem „h“ bezeichnet den Hamming-Abstand zwischen zwei Tags der Familie. Je höher, desto höher ist die Robustheit, aber desto weniger IDs stehen bei gleicher Anzahl an Datenmodulen zur Verfügung (siehe Tab. 5.20).

Der Vorteil von Familien mit weniger Modulen (bspw. 16h5 im Vergleich zu 36h11) ist die niedrigere Auflösung der Tags. Jedes Modul ist somit größer, weshalb der Tag auf eine größere Distanz erkannt werden kann. Dies hat allerdings auch Nachteile: Zum einen stehen bei niedrigerer Zahl an Datenmodulen auch weniger IDs zur Verfügung. Wichtiger aber ist, dass die Robustheit der Tagerkennung signifikant reduziert wird, da es zu einer höheren Falsch-Positiv-Rate kommt. Dies bedeutet, dass Tags verwechselt werden oder nicht existierende Tags in zufälliger Bildtextur oder im Bildrauschen erkannt werden. Die 41h12 Familie hat ihren Rahmen nach innen verschoben, was im Vergleich zur 36h11 Familie mehr Datenmodule bei einer geringen Gesamtmodulanzahl ermöglicht.

Aus diesen Gründen empfehlen wir die Verwendung der 42h12 und 36h11-Familien und raten ausdrücklich von der Familie 16h5 ab. Letztgenannte Familie sollten nur benutzt werden, wenn eine große

Erkennungsdistanz für die Anwendung unbedingt erforderlich ist. Jedoch ist die maximale Erkennungsdistanz nur ca. 25% größer, wenn anstelle der 36h11-Familie die 16h5-Familie verwendet wird.

Vorgenerierte AprilTags können von der Webseite https://github.com/AprilRobotics/apriltag-imgs heruntergeladen werden. Jede Familie besteht aus mehreren PNGs, welche jeweils einen AprilTag enthalten. Jedes Pixel im PNG entspricht dabei einem Modul des AprilTags. Beim Drucken der Tags der Familien 36h11, 36h10, 25h9 und 16h5 sollte darauf geachtet werden, den weißen Rand um den AprilTag mit einzuschließen – dieser ist in den PNGs enthalten (siehe (a) in Abb. 5.12). Die Tags müssen außerdem ohne Interpolation auf die Druckgröße skaliert werden, sodass die scharfen Kanten erhalten bleiben.

Vergleich

Sowohl QR-Codes als auch AprilTags haben ihre Vor- und Nachteile. Während QR-Codes die Speicherung von benutzerdefinierten Daten erlauben, sind die Tags bei AprilTags vordefiniert und in ihrer Anzahl limitiert. Andererseits haben AprilTags eine niedrigere Auflösung und können daher auf eine größere Distanz erkannt werden. Zusätzlich hilft die durchgängige weiß-zu-schwarz-Kante in jedem AprilTag bei einer präziseren Posenschätzung.

Bemerkung: Falls die Speicherung von benutzerdefinierten Daten nicht benötigt wird, sollten AprilTags QR-Codes vorgezogen werden.

5.4.3.2 Tagerkennung

Der erste Schritt der Tagerkennung ist die Detektion der Tags auf dem Stereo-Bildpaar. Dieser Schritt benötigt die meiste Zeit und seine Präzision ist entscheidend für die Präzision der finalen Tagpose. Um die Dauer dieses Schritts zu kontrollieren, kann der Parameter `quality` vom Benutzer konfiguriert werden. Er hat ein Herunterskalieren des Stereo-Bildpaares vor der Tagerkennung zur Folge. *Hoch* (`High`) ergibt die höchste maximale Erkennungsdistanz und Präzision, aber auch die längste Dauer der Erkennung. *Niedrig* (`Low`) führt zur kleinsten maximalen Erkennungsdistanz und Präzision, aber benötigt auch nur weniger als die halbe Zeit. *Mittel* (`Medium`) liegt dazwischen. Es sollte beachtet werden, dass dieser `quality`-Parameter keine Verbindung zum `quality`-Parameter des *Stereo-Matching Modul* (Abschnitt 5.2) hat.

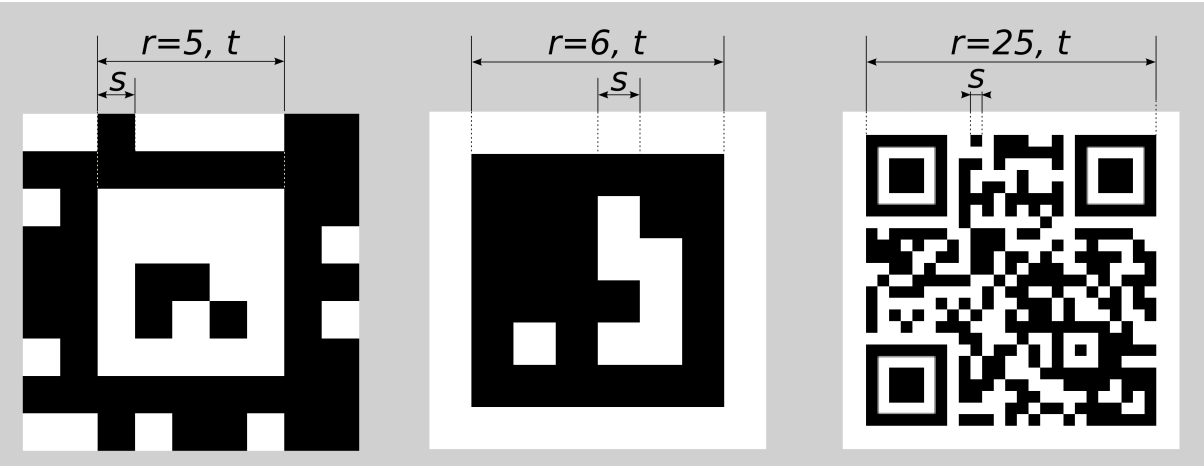


Abb. 5.13: Visualisierung der Modulgröße s, der Größe eines Tags in Modulen r und der Größe eines Tags in Metern t für AprilTags (links und Mitte) und QR-Codes (rechts)

Die maximale Erkennungsdistanz z für Qualität *Hoch* (`High`) kann mit folgenden Formeln angenähert werden:

$$z = \frac{fs}{p},$$

$$s = \frac{t}{r},$$

wobei f die *Brennweite* (Abschnitt 5.1.2) in Pixeln und s die Größe jedes Moduls in Metern bezeichnet. s kann leicht mit letztgenannter Formel berechnet werden, in welcher t der Taggröße in Metern und r der Breite des Tags in Modulen entspricht (bei AprilTags ohne den weißen Rahmen). Abb. 5.13 veranschaulicht diese Variablen. p bezeichnet die Zahl der Bildpixel pro Modul, welche für eine Erkennung erforderlich sind. Sie unterscheidet sich zwischen QR-Codes und AprilTags. Auch der Winkel des Tags zur Kamera und die Beleuchtung spielen eine Rolle. Ungefähre Werte für eine robuste Erkennung sind:

- AprilTag: $p = 5$ Pixel/Modul
- QR-Code: $p = 6$ Pixel/Modul

Die folgenden Tabellen enthalten Beispiele für die maximale Erkennungsdistanz in unterschiedlichen Situationen. Die Brennweite des *rc_visard* wird dafür mit 1075 Pixeln, die Qualität mit `High` angenommen.

Tab. 5.21: Beispiele zur maximalen Erkennungsdistanz für AprilTags mit einer Breite von $t = 4$ cm

AprilTag-Familie	Tagbreite	Maximale Distanz
36h11 (empfohlen)	8 Module	1.1 m
16h5	6 Module	1.4 m
41h12 (empfohlen)	5 Module	1.7 m

Tab. 5.22: Beispiele zur maximalen Erkennungsdistanz für QR-Codes mit einer Breite von $t = 8$ cm

Tagbreite	Maximale Distanz
29 Module	0.49 m
21 Module	0.70 m

5.4.3.3 Posenschätzung

Für jeden erkannten Tag wird dessen Pose im Kamerakoordinatensystem geschätzt.

Eine Bedingung dafür ist, dass der Tag vollständig im linken und rechten Bild zu sehen ist.

Das Koordinatensystem ist wie unten gezeigt am Tag ausgerichtet.

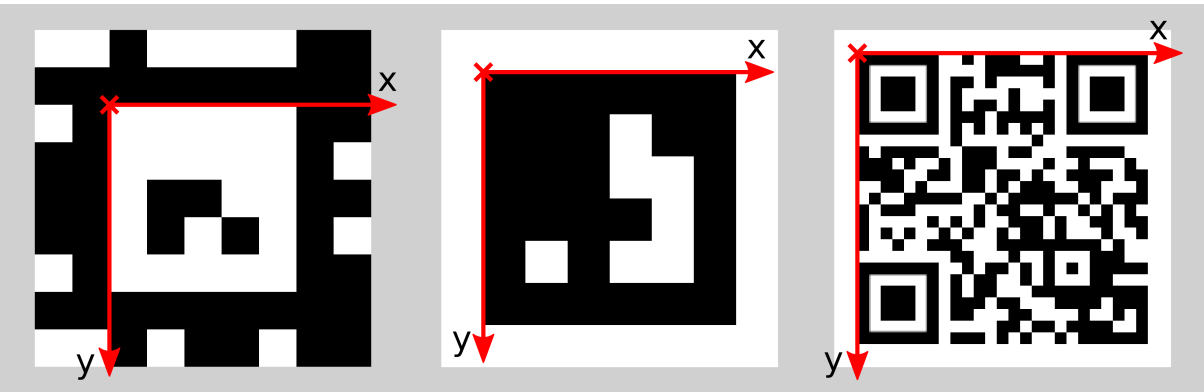


Abb. 5.14: Koordinatensysteme für AprilTags (links und Mitte) bzw. QR-Codes (rechts)

Die z-Achse zeigt „in“ den Tag. Es ist zu beachten, dass, auch wenn AprilTags den weißen Rand in ihrer Definition enthalten, der Ursprung des Koordinatensystems trotzdem am Übergang des weißen zum schwarzen Rand liegt. Da AprilTags keine offensichtliche Orientierung haben, liegt der Ursprung in der oberen linken Ecke des vorgenerierten AprilTags.

Während der Posenschätzung wird auch die Größe des Tags geschätzt unter der Annahme, dass der Tag quadratisch ist. Bei QR-Codes bezieht sich die Größe auf den gesamten Tag, bei AprilTags dagegen nur auf den Bereich innerhalb des Übergangs vom schwarzen zum weißen Rand. Das heißt, dass bei Tags der Familien 16h5, 25h9, 36h10 und 36h11 der äußere weiße Rand ignoriert wird.

Der Benutzer kann auch die ungefähre Größe ($\pm 10\%$) eines Tags angeben. Alle Tags, die dieser Einschränkung nicht entsprechen, werden automatisch herausgefiltert. Weiter hilft diese Information in bestimmten Situationen, Mehrdeutigkeiten in der Posenschätzung aufzulösen, die entstehen können, wenn mehrere Tags mit derselben ID im linken und rechten Bild sichtbar und diese Tags parallel zu den Bildzeilen ausgerichtet sind.

Bemerkung: Für beste Ergebnisse der Posenschätzung sollte der Tag sorgfältig gedruckt und auf einem steifen und möglichst ebenen Untergrund angebracht werden. Jegliche Verzerrung des Tags oder Unebenheit der Oberfläche verschlechtert die geschätzte Pose.

Bemerkung: Wir empfehlen, die ungefähre Größe der Tags anzugeben. Ansonsten, falls mehrere Tags mit derselben ID im linken oder rechten Bild sichtbar sind, kann es zu einer fehlerhaften Posenschätzung kommen, wenn die Tags gleich orientiert sind und sie ungefähr parallel zu den Bildzeilen angeordnet sind. Auch wenn die Größe nicht angegeben sein sollte, versuchen die TagDetect-Module jedoch, solche Situationen zu erkennen und verwerfen betroffene Tags.

Unten stehende Tabellen enthalten grobe Angaben zur Präzision der geschätzten Posen von AprilTags. Wir unterscheiden zwischen lateraler Präzision (also in x- und y-Richtung) und Präzision in z-Richtung. Es wird angenommen, dass `quality` auf `High` gesetzt ist, und dass die Blickrichtung der Kamera parallel zur Normalen des Tags ist. Die Größe eines Tags hat keinen signifikanten Einfluss auf die Präzision in lateraler und z-Richtung. Im Allgemeinen verbessert ein größerer Tag allerdings die Präzision. Im Bezug auf die Präzision der Rotation, im speziellen um die x- und y-Achsen, übertreffen große Tags kleinere deutlich.

Tab. 5.23: Ungefähre Präzision der Position von AprilTag Messungen mit Qualität `Hoch` in einem idealen Szenario für verschiedene Basisabstände

Distanz	*rc_visard* 65 - lateral	*rc_visard* 65 - z	*rc_visard* 160 - lateral	*rc_visard* 160 - z
0.5 m	0.05 mm	0.5 mm	0.05 mm	0.3 mm
1.0 m	0.15 mm	1.8 mm	0.15 mm	1.4 mm
2.0 m	1.5 mm	14.5 mm	0.5 mm	3.7 mm

Tab. 5.24: Ungefähre Präzision der Orientierung von AprilTag Messungen mit Qualität `Hoch` in einem idealen Szenario für verschiedene Tag-Größen

Distanz	60 x 60 mm	120 x 120 mm
0.5 m	0.2°	–
1.0 m	0.8°	0.3°
2.0 m	2.0°	0.8°
3.0 m	–	1.8°

5.4.3.4 Tag-Wiedererkennung

Jeder Tag besitzt eine ID: bei AprilTags ist dies die *Familie* zusammen mit der AprilTag-*ID*, bei QR-Codes die enthaltenen Daten. Diese IDs sind jedoch nicht einzigartig, da mehrere Tags mit derselben ID in einer Szene vorkommen können.

Zur Unterscheidung dieser Tags weisen die TagDetect-Module jedem Tag einen eindeutigen Identifikator zu. Um den Benutzer dabei zu unterstützen, denselben Tag über mehrere Tagerkennungsläufe hinweg zu identifizieren, versucht das TagDetect-Modul Tags wiederzuerkennen. Falls erfolgreich, wird einem Tag derselbe Identifikator zugewiesen.

Die Tag-Wiedererkennung vergleicht die Positionen der Ecken der Tags in einem statischen Koordinatensystem, um identische Tags wiederzufinden. Tags werden als identisch angenommen, falls sie sich nicht oder nur geringfügig in diesem statischen Koordinatensystem bewegt haben. Damit das statische Koordinatensystem verfügbar ist, muss das *Dynamik-Modul* (Abschnitt 5.3.1) angeschaltet sein. Falls dies nicht der Fall ist, wird der Sensor als statisch angenommen. Die Tag-Wiedererkennung funktioniert in diesem Fall nicht über Bewegungen des Sensors hinweg.

Über den `max_corner_distance`-Parameter kann der Benutzer festlegen, wie weit ein Tag sich zwischen zwei Erkennungsläufen bewegen darf, um als identisch zu gelten. Der Parameter definiert die maximale Distanz zwischen den Ecken zweier Tags, was in Abb. 5.15 dargestellt ist. Die euklidischen Abstände der vier zusammengehörenden Tagecken in 3D werden berechnet. Falls keiner dieser Abstände den Grenzwert überschreitet, gilt der Tag als wiedererkannt.

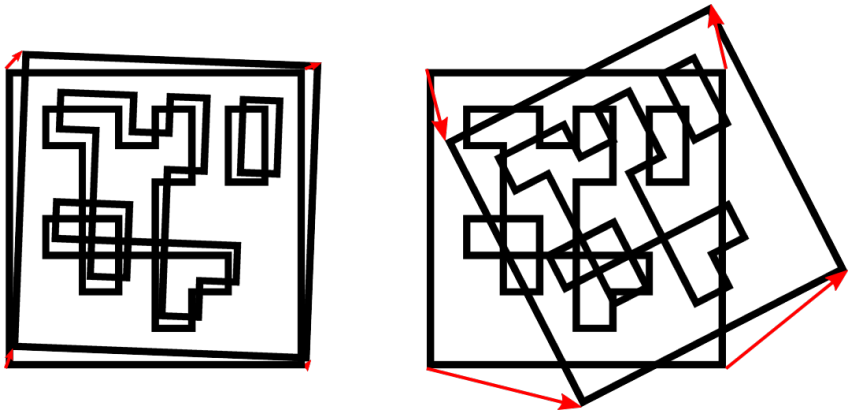


Abb. 5.15: Vereinfachte Darstellung der Tag-Wiedererkennung. Die euklidischen Abstände zwischen zusammengehörigen Tagecken in 3D werden berechnet (rote Pfeile).

Nach einer bestimmten Anzahl von Tagerkennungsläufen werden vorher gesehene Tags verworfen, falls diese in der Zwischenzeit nicht mehr erkannt wurden. Dies kann über den Parameter `forget_after_n_detections` festgelegt werden.

5.4.3.5 Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das TagDetect-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.4.3.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene `pose_frame`-Werte können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben.
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

5.4.3.6 Parameter

Es stehen zwei getrennte Module für die Tagerkennung zur Verfügung, eines für AprilTag- und eines für QR-Code-Erkennung: `rc_april_tag_detect` bzw. `rc_qr_code_detect`. Abgesehen vom Modulnamen

teilen beide die gleiche Schnittstellendefinition.

Neben der *REST-API-Schnittstelle* (Abschnitt 6.2) stellen die TagDetect-Module außerdem Seiten in der Web GUI unter *Module* → *AprilTag* und *Module* → *QR Code* bereit, über welche sie manuell ausprobiert und konfiguriert werden können.

Im Folgenden sind die Parameter am Beispiel von `rc_qr_code_detect` aufgelistet. Sie gleichen denen von `rc_april_tag_detect`.

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.25: Laufzeitparameter des `rc_qr_code_detect`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`detect_inverted_tags`	bool	false	true	false	Erkennt Tags, bei denen Schwarz und Weiß vertauscht sind
`forget_after_n_detections`	int32	1	1000	30	Anzahl an Erkennungsläufen, nach denen ein vorher gesehener Tag während der Tag-Wiedererkennung verworfen wird
`max_corner_distance`	float64	0.001	0.01	0.005	Maximale Distanz zusammengehöriger Ecken zweier Tags während der Tag-Wiedererkennung
`quality`	string	-	-	High	Qualität der Tagerkennung: [Low, Medium, High]
`use_cached_images`	bool	false	true	false	Benutze das zuletzt empfangene Stereo-Bildpaar, anstatt auf ein neues zu warten

Über die REST-API können diese Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪parameters/parameters?<parameter-name>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/parameters?
↪<parameter-name>=<value>
```

5.4.3.7 Statuswerte

Die TagDetect-Module melden folgende Statuswerte:

Tab. 5.26: Statuswerte der `rc_qr_code_detect`- und `rc_april_tag_detect`-Module

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_return_code_message`	Rückgabecode-Nachricht der letzten Detektion
`last_return_code_value`	Rückgabecode-Wert der letzten Detektion
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`processing_time`	Berechnungszeit für die letzte Erkennung in Sekunden
`state`	Der aktuelle Zustand des Moduls

Der Parameter `state` kann folgende Werte annehmen:

Tab. 5.27: Mögliche Zustände der TagDetect-Module

Zustandsname	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul läuft und ist bereit zur Tagerkennung.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

5.4.3.8 Services

Die TagDetect-Module implementieren einen Zustandsautomaten, welcher zum Starten und Stoppen genutzt werden kann. Die eigentliche Tagerkennung kann mit `detect` ausgelöst werden.

Die angebotenen Services von `rc_qr_code_detect` bzw. `rc_april_tag_detect` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

`detect`

löst eine Tagerkennung aus.

Details

Abhängig vom `use_cached_images`-Parameter arbeitet das Modul auf dem zuletzt empfangenen Bildpaar (wenn *true*) oder wartet auf ein Bildpaar, das nach dem Auslösen des Services aufgenommen wurde (wenn *false*, dies ist das Standardverhalten). Auch wenn der Parameter auf *true* steht, arbeitet die Tagerkennung niemals mehrmals auf einem Bildpaar.

Es wird empfohlen, `detect` nur im Zustand `RUNNING` aufzurufen. Es ist jedoch auch im Zustand `IDLE` möglich, was zu einem Autostart und -stop des Moduls führt. Dies hat allerdings Nachteile: Erstens dauert der Aufruf deutlich länger, zweitens funktioniert die Tag-Wiedererkennung nicht. Es wird daher ausdrücklich empfohlen, das Modul manuell zu starten, bevor `detect` aufgerufen wird.

Tags können vom `detect`-Ergebnis aus mehreren Gründen ausgeschlossen werden, z.B. falls ein Tag nur in einem der Kamerabilder sichtbar war, oder falls die Posenschätzung fehlschlug. Diese herausgefilterten Tags werden im Log aufgelistet, auf welches wie in *Download der Logdateien* (Abschnitt 7.5) beschrieben zugegriffen werden kann.

Auf den Web GUI-Seiten der TagDetect-Module wird eine Visualisierung der letzten Tagerkennung bereitgestellt. Diese Visualisierung wird allerdings erst angezeigt, sobald die Tagerkennung mindestens einmal ausgeführt wurde. In der Web GUI kann die Tagerkennung außerdem manuell ausprobiert werden, indem die *Detektieren*-Schaltfläche betätigt wird.

Aufgrund von Änderungen der Systemzeit auf dem *rc_visard* können Zeitsprünge auftreten, sowohl vorwärts als auch rückwärts. Während Vorwärtssprünge keinen Einfluss auf die TagDetect-Module haben, invalidieren Rücksprünge die bereits empfangenen Bilder. Deshalb wird, wenn ein Rücksprung erkannt wird, Fehler -102 beim nächsten `detect`-Aufruf zurückgegeben. Dies geschieht auch, um den Benutzer darauf hinzuweisen, dass die Zeitstempel in der `detect`-Antwort ebenso zurückspringen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/services/detect
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/detect
```

Request

Optionale Serviceargumente:

`tags` bezeichnet die Liste der Tag-IDs, welche erkannt werden sollen. Bei QR-Codes ist die ID gleich den enthaltenen Daten. Bei AprilTags ist es *„<Familie>_<ID>“*, also beispielsweise *„36h11_5“* für Familie 36h11 und ID 5. Natürlich kann das AprilTag-Modul nur zur Erkennung von AprilTags und das QR-Code-Modul nur zur Erkennung von QR-Codes genutzt werden.

Die `tags`-Liste kann auch leer gelassen werden. In diesem Fall werden alle erkannten Tags zurückgegeben. Dieses Feature sollte nur während der Entwicklung einer Applikation oder zur Fehlerbehebung benutzt werden. Wann immer möglich sollten die konkreten Tag-IDs aufgelistet werden, zum einen zur Vermeidung von Fehldetektionen, zum anderen auch um die Tagerkennung zu beschleunigen, da nicht benötigte Tags aussortiert werden können.

Bei AprilTags kann der Benutzer nicht nur einzelne Tags, sondern auch eine gesamte Familie spezifizieren, indem die ID auf „<family>“ gesetzt wird, bspw. „36h11“. Dadurch werden alle Tags dieser Familie erkannt. Es ist auch möglich, mehrere Familien oder eine Kombination aus Familien und einzelnen Tags anzugeben. Zum Beispiel kann `detect` mit „36h11“, „25h9_3“ und „36h10“ zur gleichen Zeit aufgerufen werden.

Zusätzlich zur ID kann auch die ungefähre Größe ($\pm 10\%$) eines Tags angegeben werden. Wie in *Posenschätzung* (Abschnitt 5.4.3.3) erklärt, verhilft dies Mehrdeutigkeiten aufzulösen, die in bestimmten Situationen auftreten können, und kann zum Herausfiltern von Tags genutzt werden, die nicht der angegebenen Größe entsprechen.

Die `tags`-Liste ist ODER-verknüpft. Es werden alle Tags zurückgegeben, die mit einem der `id-size`-Paare in der `tags`-Liste übereinstimmen.

Das Feld `pose_frame` gibt an, ob die Posen im Kamera- oder im externen Koordinatensystem zurückgegeben werden (siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.4.3.5). Der Standardwert ist `camera`.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "tags": [
      {
        "id": "string",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "size": "float64"
      }
    ]
  }
}
```

Response

`timestamp` wird auf den Zeitstempel des Bildpaares gesetzt, auf dem die Tagerkennung gearbeitet hat.

`tags` enthält alle erkannten Tags.

`id` ist die ID des Tags, vergleichbar zur `id` in der Anfrage.

`instance_id` ist der zufällige, eindeutige Identifikator eines Tags, welcher von der Tag-Wiedererkennung zugewiesen wird.

`pose` enthält `position` und `orientation`. Die Orientierung ist im Quaternionen-Format angegeben.

`pose_frame` bezeichnet das Koordinatensystem, auf welches obige Pose bezogen ist, und hat den Wert `camera` oder `external`.

`size` wird auf die gemessene Taggröße gesetzt.

`return_code` enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "tags": [
      {
        "id": "string",
        "instance_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "size": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        }
      }
    ],
    "timestamp": {
      "nsec": "int32",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "sec": "int32"
    }
  }
}
```

start

startet das Modul durch einen Übergang von `IDLE` nach `RUNNING`.

Details

Wenn das Modul läuft, empfängt es die Bilder der Kamera und ist bereit, Tags zu erkennen. Um Rechenressourcen zu sparen, sollte das Modul nur laufen, wenn dies nötig ist.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul durch einen Übergang zu `IDLE`.

Details

Dieser Übergang kann auf dem Zustand `RUNNING` und `FATAL` durchgeführt werden. Alle Tag-Wiedererkennungs-Informationen werden beim Stoppen gelöscht.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

restart

startet das Modul neu.

Details

Wenn im Zustand `RUNNING` oder `FATAL`, wird das Modul erst gestoppt und dann wieder gestartet. In `IDLE` wird das Modul nur gestartet.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/services/restart
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/restart
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "restart",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/reset_
↪defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.4.3.9 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Code	Beschreibung
0	Erfolg
-1	Ein ungültiges Argument wurde übergeben.
-4	Die maximale Wartezeit auf ein Stereo-Bildpaar wurde überschritten.
-9	Die Lizenz ist ungültig.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-101	Ein interner Fehler trat während der Tagerkennung auf.
-102	Ein Rückwärtssprung der Systemzeit trat auf
-103	Ein interner Fehler trat während der Posenschätzung auf.
-200	Ein schwerwiegender interner Fehler trat auf.
200	Mehrere Warnungen traten auf. Siehe die Auflistung in `message`.
201	Das Modul war nicht im Zustand `RUNNING`.

5.4.4 ItemPick

5.4.4.1 Einführung

Das ItemPick Modul liefert eine gebrauchsfertige Perzeptionslösung, um robotische Pick-and-Place-Anwendungen zu realisieren. ItemPick detektiert ebene Oberflächen unbekannter Objekte für die Posi-

tionierung eines Sauggreifers.

Darüber hinaus bietet das Modul:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_visard* *Web GUI* (Abschnitt 6.1)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *RoiDB*, Abschnitt 5.6.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 5.4.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Fächern, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- eine Kollisionsprüfung zwischen Greifer und Load Carrier und/oder der Punktwolke
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.
- einen Qualitätswert für jeden vorgeschlagenen Greifpunkt, der die Ebenheit der für das Greifen verfügbaren Oberfläche bewertet
- Auswahl einer Strategie zum Sortieren der zurückgelieferten Greifpunkte
- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

Bemerkung: In diesem Kapitel werden die Begriffe Cluster und Oberfläche synonym verwendet und bezeichnen eine Menge von Punkten (oder Pixeln) mit ähnlichen geometrischen Eigenschaften.

Das Modul ist ein optional erhältliches Module, welches intern auf dem *rc_visard* läuft und eine gesonderte ItemPick *Lizenz* (Abschnitt 7.4) benötigt.

5.4.4.2 Berechnung der Greifpunkte

Das ItemPick Modul bietet einen Service, um Greifpunkte für Sauggreifer zu berechnen. Der Sauggreifer ist durch die Länge und Breite der Greiffläche definiert.

Das ItemPick-Modul identifiziert ebene Flächen in der Szene und unterstützt flexible und/oder deformierbare Objekte. Der Typ (`type`) dieser Objektmodelle (`item_models`) ist als unbekannt (`UNKNOWN`) definiert, da sie keine gebräuchliche geometrische Form aufweisen müssen. Optional kann eine minimale und maximale Größe angegeben werden.

Optional können den Modulen zu einer Greifpunktberechnung weitere Informationen übergeben werden:

- die ID des Load Carriers, welcher die zu greifenden Objekte enthält
- ein Unterabteil (`load_carrier_compartment`) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 5.6.1.3).
- die ID der 3D Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die 3D Region of Interest, innerhalb der Greifpunkte berechnet werden
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 5.4.4.5) gegeben.

Ein vom BoxPick-Modul ermittelter Greifpunkt repräsentiert die empfohlene Pose des TCP (Tool Center Point) des Sauggreifers. Der Greifpunkt `type` ist immer auf `SUCTION` gesetzt.

Für ItemPick mit dem Objektmodelltyp `UNKNOWN` liegt der Ursprung der berechneten Greifposen `pose` im Mittelpunkt der größten von der jeweiligen Greiffläche umschlossenen Ellipse.

Die Orientierung des Greifpunkts ist ein rechtshändiges Koordinatensystem, sodass die z-Achse orthogonal zur Greiffläche in das zu greifende Objekt zeigt und die x-Achse entlang der längsten Ausdehnung ausgerichtet ist.Da die x-Achse zwei mögliche Richtungen haben kann, wird diejenige ausgewählt, die besser zur bevorzugten TCP-Ausrichtung passt (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.4.4). Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, wird die x-Achse nicht zwangsweise an der maximalen Dehnung der greifbaren Ellipse ausgerichtet, sondern kann eine beliebige Drehung um die z-Achse aufweisen. In diesem Fall hat der zurückgegebene Greifpunkt die Ausrichtung, die am besten zur bevorzugten TCP-Ausrichtung passt und kollisionsfrei ist, wenn die Kollisionsprüfung aktiviert ist.

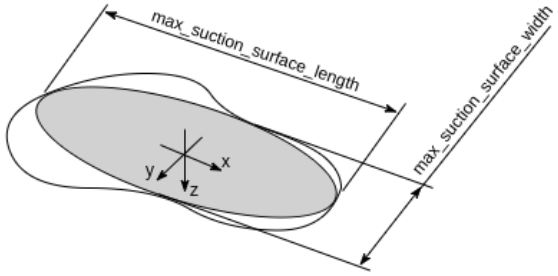


Abb. 5.16: Veranschaulichung eines berechneten Greifpunktes mit Koordinatensystem und der zugehörigen Ellipse, welche die Greiffläche bestmöglich beschreibt.

Zusätzlich enthält jeder Greifpunkt die Abmessungen der maximal verfügbaren Greiffläche, die als Ellipse mit den Achslängen `max_suction_surface_length` und `max_suction_surface_width` beschrieben wird. Der Nutzer kann Greifpunkte mit zu kleinen Greifflächen herausfiltern, indem die minimalen Abmessungen der Greiffläche, die vom Sauggreifer benötigt wird, angegeben werden. Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, dann sind die Achslängen `max_suction_surface_length` und `max_suction_surface_width` gleich und entsprechen der kürzeren Achse der größtmöglichen Greifellipse.

Jeder Greifpunkt enthält auch einen Qualitätswert (`quality`), der einen Hinweis auf die Ebenheit der Greiffläche gibt. Dieser Wert reicht von 0 bis 1, wobei höhere Werte für eine ebenere rekonstruierte Oberfläche stehen.

Jeder berechnete Greifpunkt lässt sich anhand einer `uuid` (Universally Unique Identifier) eindeutig identifizieren und enthält zusätzlich den Zeitstempel der ältesten Bildaufnahme, auf der die Greifpunktberechnung durchgeführt wurde.

Die Greifpunkte werden gemäß der benutzerdefinierten Sortierstrategie sortiert (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.4.3).

5.4.4.3 Setzen der Sortierstrategie

Die Greifpunkte, die von den Services zurückgeliefert werden, werden gemäß einer Sortierstrategie sortiert, die vom Nutzer gewählt werden kann. Folgende Sortierstrategien sind verfügbar und können in der *Web GUI* (Abschnitt 6.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `gravity`: die höchsten Greifpunkte entlang der Gravitationsrichtung werden zuerst zurückgeliefert.
- `surface_area`: Greifpunkte mit den größten Oberflächen werden zuerst zurückgeliefert.
- `direction`: Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Richtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

- `distance_to_point`: Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `gravity` und `surface_area`.

5.4.4.4 Setzen der bevorzugten TCP-Orientierung

Das ItemPick-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die ItemPick Seite in der Web GUI gesetzt werden. Die daraus resultierende Richtung der z-Achse des TCPs wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann. Außerdem wird die bevorzugte Orientierung genutzt, um aus mehreren möglichen Symmetrien den Greifpunkt auszuwählen, der für den Roboter am besten erreichbar ist.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und die Kamera am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Koordinatensysteme) als die bevorzugte TCP-Orientierung genutzt.

5.4.4.5 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das ItemPick-Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des ItemPick-Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom ItemPick-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 5.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 5.2)

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_visard* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.5.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.5.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das ItemPick-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.4.4.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das ItemPick- oder BoxPick-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame` und der Definition der Sortierrichtung, des Sortierpunktes oder der bevorzugten Orientierung nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die bevorzugte Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

LoadCarrier

Das ItemPick-Module nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 5.4.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine 3D Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das ItemPick-Modul verwendet wird.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des ItemPick-Moduls aktiviert werden, indem das `collision_detection` Argument an den `compute_grasps` oder `compute_grasps_extended` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer

muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 5.6.4.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.5.2.2) gegeben.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in den Visualisierungen auf der ItemPick Seite der Web GUI kollidierende Greifpunkte als schwarze Ellipsen dargestellt.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 5.5.2.3) beschrieben.

5.4.4.6 Parameter

Das ItemPick-Modul wird in der REST-API als `rc_itempick` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Module* → *ItemPick* und *Modules* → *ItemPickAI* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) ändern.

Die angebotenen Services von `rc_itempick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard* *Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.28: Laufzeitparameter des `rc_itempick` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`allow_any_grasp_pose`	bool	false	true	false	Bestimmt, ob die Greifpunkte beliebig dort auf dem Objekt platziert sein dürfen, wo ebene Greifflächen detektiert werden
`allow_any_grasp_z_rotation`	bool	false	true	false	Bestimmt, ob die Greifpunkte beliebige Orientierung haben dürfen, anstatt an der Hauptachse der greifbaren Ellipse ausgerichtet zu sein
`check_collisions_with_point_cloud`	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
`cluster_max_curvature`	float64	0.005	0.5	0.11	Maximal erlaubte Krümmung für Greifflächen
`cluster_max_dimension`	float64	0.05	2.0	0.3	Maximum allowed diameter for a cluster in meters. Clusters with a diameter larger than this value are not used for grasp computation.
`clustering_discontinuity_factor`	float64	0.1	5.0	1.0	Erlaubte Unebenheit von Greifflächen
`clustering_max_surface_rmse`	float64	0.0005	0.01	0.004	Maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern
`clustering_patch_size`	int32	3	10	4	Pixelgröße der Patches für die Unterteilung des Tiefenbildes im ersten Clustering-Schritt
`discard_cut_off_items`	bool	false	true	false	Gibt an, ob Objekte, die vom Bildrand, der Stereo ROI, der fehlenden Stereo-Überlappung oder der 3D Region of Interest abgeschnitten werden, verworfen werden sollen
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
`max_grasps`	int32	1	100	5	Maximale Anzahl von bereitgestellten Greifpunkten

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der ItemPick Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_grasps` (*Anzahl Greifpunkte*)

ist die maximale Anzahl von bereitgestellten Greifpunkten.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/parameters/parameters?max_
↪grasps=<value>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?max_grasps=<value>
```

`cluster_max_dimension` (*Maximale Größe*)

is the maximum allowed diameter for a cluster in meters. Clusters with a diameter larger than this value are not used for grasp computation.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/parameters/parameters?cluster_
↪max_dimension=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?cluster_max_dimension=<value>
```

`cluster_max_curvature` (*Maximale Krümmung*)

ist die maximal erlaubte Krümmung für Greifflächen. Je kleiner dieser Wert ist, desto mehr mögliche Greifflächen werden in kleinere Flächen mit weniger Krümmung aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/parameters/parameters?cluster_
↪max_curvature=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?cluster_max_curvature=<value>
```

`clustering_patch_size` (*Patchgröße*)

ist die Pixelgröße der Patches für die Unterteilung des Tiefenbildes im ersten Clustering-Schritt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/parameters/parameters?
↪clustering_patch_size=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_patch_size=<value>
```

`clustering_discontinuity_factor` (*Unstetigkeitsfaktor*)

beschreibt die erlaubte Unebenheit von Greifflächen. Je kleiner dieser Wert ist, umso mehr werden mögliche Greifflächen in kleinere Flächen mit weniger Unebenheiten aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/parameters/parameters?
↪clustering_discontinuity_factor=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_discontinuity_factor=
↪<value>
```

`clustering_max_surface_rmse` (*Maximaler RMSE*)

ist die maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/parameters/parameters?
↪clustering_max_surface_rmse=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_max_surface_rmse=
↪<value>
```

`grasp_filter_orientation_threshold` (*Grasp Orientation Threshold*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der bevorzugten TCP-Orientierung in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist. Falls der Wert auf Null gesetzt wird, sind alle Abweichungen valide.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?grasp_filter_
↪orientation_threshold=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?grasp_filter_orientation_
↪threshold=<value>
```

`allow_any_grasp_z_rotation` (*Beliebige Greifrotation um Z*)

Wenn der Wert auf True gesetzt ist, werden die x-Achsen der zurückgegebenen Greifpunkte nicht mehr notwendigerweise an der maximalen Ausdehnung der greifbaren Ellipse ausgerichtet, sondern können eine beliebige Drehung um die z-Achse haben. Die zurückgegebenen Werte von `max_suction_surface_length` und `max_suction_surface_width` sind dann

gleich und entsprechen dem kleinsten Durchmesser der größten greifbaren Ellipsenfläche. Dieser Parameter eröffnet dem Roboter mehr Optionen zum Greifen von Objekten, insbesondere in Szenen, in denen es zu Kollisionen kommen kann. Da der Greifpunkt jedoch nicht mehr mit der greifbaren Ellipse ausgerichtet ist, muss bei Objektmodellen vom Typ `UNKNOWN` die korrekte Ausrichtung zum Platzieren des Objekts auf andere Weise bestimmt werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?allow_any_
↪grasp_z_rotation=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?allow_any_grasp_z_rotation=
↪<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `compute_grasps` oder `compute_grasps_extended` Service aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?check_collisions_with_point_
↪cloud=<value>
```

5.4.4.7 Statuswerte

Das `rc_itempick` Modul meldet folgende Statuswerte:

Tab. 5.29: Statuswerte des `rc_itempick` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste.
`grasp_computation_time`	Laufzeit für die Greifpunktberechnung beim letzten Aufruf in Sekunden
`last_return_code_message`	Rückgabecode-Nachricht der letzten Detektion
`last_return_code_value`	Rückgabecode-Wert der letzten Detektion
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Laufzeit für die letzte Load Carrier Erkennung in Sekunden
`processing_time`	Laufzeit für die letzte Erkennung (einschließlich Load Carrier Detektion) in Sekunden
`state`	Aktueller Zustand des rc_itempick Moduls

Folgende `state`-Werte werden gemeldet.

Tab. 5.30: Mögliche Werte für den Zustand des ItemPick Moduls

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier zu erkennen und Greifpunkte zu berechnen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

5.4.4.8 Services

Die angebotenen Services von `rc_itempick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

Das ItemPick Modul stellt folgende Services zur Verfügung.

`compute_grasps`

löst die Erkennung von Greifpunkten für einen Sauggreifer aus, wie in *Berechnung der Greifpunkte* (Abschnitt 5.4.4.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/compute_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/compute_grasps
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.4.5).

`suction_surface_length`: Länge der Greiffläche des verwendeten Vakuum-Greifsystems.

`suction_surface_width`: Breite der Greiffläche des verwendeten Vakuum-Greifsystems.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.4.5).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu greifenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Greifpunkte berechnet werden sollen (siehe *Load Carrier Abteile*, Abschnitt 5.6.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, innerhalb der nach den Greifpunkten gesucht wird.

`item_models`: Liste von Objektmodellen, die erkannt werden sollen. Im Fall von ItemPick wird aktuell nur ein einzelnes Objektmodell vom Typ `UNKNOWN` mit minimaler und maximaler Größe unterstützt, wobei die minimale Größe kleiner als die maximale Größe sein muss.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.5.2.2)

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "dimensioning": {
      "plane_ids": [
        "string"
      ]
    },
    "item_models": [
      {
        "type": "string",
        "unknown": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        }
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```

Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten.

`items`: Liste von erkannten Objekten, die zu den zurückgelieferten Greifpunkten gehören. Im Fall von ItemPick ist diese Liste immer leer.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
      {
        "item_uuid": "string",
        "max_suction_surface_length": "float64",
        "max_suction_surface_width": "float64",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "quality": "float64",
        "tcp_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string"
      }
    ],
    "items": [
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    {
      "bounding_box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "grasp_uuids": [
        "string"
      ],
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_ledge": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


compute_grasps_extended

löst die Erkennung von Greifpunkten für einen Sauggreifer aus. Dieser Service verhält sich analog zu `compute_grasps`, gibt aber die Objektinformationen `item` für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn Objektinformationen für die Greifpunkte benötigt werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/compute_grasps_extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/compute_grasps_extended
```

Request

Siehe `compute_grasps` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "dimensioning": {
      "plane_ids": [
        "string"
      ]
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    },
    "item_models": [
      {
        "type": "string",
        "unknown": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        }
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```


Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten. Jeder Greifpunkt enthält auch die `item` Information, falls verfügbar.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "compute_grasps_extended",
  "response": {
    "grasps": [
      {
        "item": {
          "bounding_box": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "pose_frame": "string",
          "type": "string",
          "uuid": "string"
        },
        "max_suction_surface_length": "float64",
        "max_suction_surface_width": "float64",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "quality": "float64",
        "tcp_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string"
      }
    ],
    "load_carriers": [
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `compute_grasps` und `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.4.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/set_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional für die Sortierung der vom `compute_grasps` und `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.4.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/get_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_sorting_strategies

speichert dauerhaft die Sortierstrategie zur Sortierung der Greifpunkte (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.4.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/set_sorting_strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


`get_sorting_strategies`

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der Greifpunkte verwendet wird (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.4.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/get_sorting_strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_sorting_strategies",
  "response": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```


start

startet das Modul und versetzt es in den Zustand `RUNNING`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul und versetzt es in den Zustand `IDLE`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter, der Sortierstrategie und der bevorzugten Orientierung dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_itempick/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.4.4.9 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.31: Rückgabecodes der Services des ItemPick Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Boxerkennung, wenn der Bereich der angegebenen Abmessungen zu groß ist.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-8	Das Template wurde während der Detektion gelöscht.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
-301	Für die Anfrage zur Greifpunktberechnung `compute_grasps` oder `compute_grasps_extended` wurden mehrere Objektmodelle (`item_models`) übergeben.
10	Die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` oder `set_region_of_interest` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Es wurden keine gültigen Greifflächen in der Szene gefunden.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
112	Die Detektionen eines oder mehrerer Cluster wurden verworfen, da die minimale Clusterabdeckung nicht erreicht wurde.
115	Objekte ohne Greifpunkte wurden gefunden
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

5.4.5 BoxPick

5.4.5.1 Einführung

Das BoxPick Modul liefert eine gebrauchsfertige Perzeptionslösung, um robotische Pick-and-Place-Anwendungen zu realisieren. Es erkennt rechteckige Oberflächen und bestimmt ihre Position, Orientierung und Größe für das Greifen. Mit der +Match-Erweiterung kann BoxPick zur Detektion von texturierten Rechtecken mit konsistenten Orientierungen verwendet werden, z.B. für bedruckte Produktverpackungen, Etiketten, Broschüren oder Bücher.

Darüber hinaus bietet das Modul:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_visard* *Web GUI* (Abschnitt 6.1)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *RoiDB*, Abschnitt 5.6.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 5.4.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Fächern, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- eine Kollisionsprüfung zwischen Greifer und Load Carrier und/oder der Punktwolke
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.
- einen Qualitätswert für jeden vorgeschlagenen Greifpunkt, der die Ebenheit der für das Greifen verfügbaren Oberfläche bewertet
- Auswahl einer Strategie zum Sortieren der zurückgelieferten Greifpunkte
- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

Bemerkung: In diesem Kapitel werden die Begriffe Cluster und Oberfläche synonym verwendet und bezeichnen eine Menge von Punkten (oder Pixeln) mit ähnlichen geometrischen Eigenschaften.

Das ItemPick Modul ist ein optional erhältliches Module, welches intern auf dem *rc_visard* läuft und eine gesonderte BoxPick-*Lizenzen* (Abschnitt 7.4) benötigt. Die +Match-Erweiterung von BoxPick bedarf einer separaten Lizenz.

5.4.5.2 Erkennung von Rechtecken

Es gibt zwei verschiedene Typen von Objektmodellen für die Erkennung von Rechtecken im BoxPick Modul.

Standardmäßig unterstützt BoxPick nur Objektmodelle (`item_models`) des Typs (`type`) `RECTANGLE`. Mit der +Match-Erweiterung können auch Objektmodelle des Typs `TEXTURED_BOX` detektiert werden. Die Erkennung der verschiedenen Objektmodelltypen wird weiter unten beschrieben.

Optional können dem BoxPick-Modul folgende Informationen übergeben werden:

- die ID des Load Carriers, welcher die Objekte enthält
- ein Teilbereich innerhalb eines Load Carriers, in dem Objekte detektiert werden sollen
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, in der nach Objekten gesucht wird

- Die ID einer Referenzebene zur Messung der Abmessungen der erkannten Objekte (siehe *Dimensionierung*, Abschnitt 5.4.5.2).
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die gewählte Region of Interest im externen Koordinatensystem definiert ist

Die zurückgegebene Pose `pose` eines detektierten Objekts `item` ist die Pose des Mittelpunkts des erkannten Rechtecks im gewünschten Koordinatensystem `pose_frame`, wobei die z-Achse in Richtung der Kamera zeigt und die x-Achse parallel zu langen Seite des Rechtecks ausgerichtet ist. Diese Pose hat eine 180° Mehrdeutigkeit in der Rotation um die z-Achse, welche durch Nutzung der +Match-Erweiterung im BoxPick Modul aufgelöst werden kann. Jedes erkannte Rechteck beinhaltet eine `uuid` (Universally Unique Identifier) und den Zeitstempel `timestamp` des ältesten Bildes, das für die Erkennung benutzt wurde.

Die Objekte werden gemäß der gewählten Sortierstrategie sortiert (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.5.4).

Erkennung von Objekten des Typs RECTANGLE

Das BoxPick-Modul unterstützt mehrere Objektmodelle (`item_models`) vom Typ (`type`) Rechteck (`RECTANGLE`). Jedes Rechteck ist durch seine minimale und maximale Größe definiert, wobei die minimale Größe kleiner als die maximale Größe sein muss. Die Abmessungen sollten relativ genau angegeben werden, um Fehldetektionen zu verhindern, jedoch eine gewisse Toleranz beinhalten, um Messunsicherheiten und mögliche Produktionsabweichungen zu berücksichtigen.

Die Erkennung der Rechtecke läuft in mehreren Schritten ab. Zuerst wird die Punktwolke in möglichst ebene Segmente (Cluster) unterteilt. Dann werden gerade Liniensegmente in den 2D Bildern erkannt und auf die zugehörigen Clusterflächen projiziert. Die Cluster und die erkannten Linien werden in der „Zwischenergebnis“ Visualisierung auf der *BoxPick* Seite in der Web GUI angezeigt. Schließlich werden für jedes Cluster die am besten zu den erkannten Linien passenden Rechtecke extrahiert.

Erkennung von Objekten des Typs RECTANGLE (BoxPick+Match)

Mit der +Match-Erweiterung unterstützt BoxPick zusätzlich Objektmodelle (`item_models`) des Typs (`type`) `TEXTURED_BOX`. Wenn dieser Objektmodelltyp verwendet wird, kann nur ein einzelnes Objektmodell pro Anfrage angegeben werden.

Das `TEXTURED_BOX` Objektmodell sollte für die Detektion mehrerer Rechtecke mit gleicher Textur, d.h. gleichem Aussehen oder Aufdruck, verwendet werden, wie zum Beispiel bedruckte Produktverpackungen, Etiketten, Broschüren oder Bücher. Es wird vorausgesetzt, dass die Textur bei allen Objekten gleich positioniert ist in Bezug auf die Objektgeometrie. Weiterhin sollte die Textur nicht repetitiv sein.

Ein Objekt vom Typ `TEXTURED_BOX` wird definiert durch die exakten Abmessungen `dimensions` des Objekts in `x`, `y` und `z` (wobei nur `z` 0 sein darf) sowie eine Toleranz `dimensions_tolerance_m` die angibt, wie stark die Abmessungen der erkannten Rechtecke von den gegebenen Dimensionen abweichen dürfen. Als Standardwert wird eine Toleranz von 0.01 m angenommen. Des Weiteren muss eine `template_id` angegeben werden, über die die spezifizierten Abmessungen und die Texturen der erkannten Rechtecke referenziert werden. Zusätzlich können die maximal mögliche Verformung der Objekte (`max_deformation_m`) in Metern angegeben werden (Standardwert 0.004 m), um steifere oder flexiblere Objekte zu beschreiben.

Wird eine `template_id` zum ersten Mal verwendet, dann detektiert BoxPick die Rechtecke so wie für den Objektmodelltyp `RECTANGLE` beschrieben, und nutzt die angegebene Toleranz um den Abmessungsbereich für die Erkennung festzulegen. Wenn zusätzlich zu x und y auch die z Abmessungen gegeben sind, werden Rechtecke mit allen möglichen Kombinationen der drei Abmessungen erkannt. Aus den erkannten Rechtecken werden sogenannte *Views* erzeugt, die die Form und die Bildintensitätswerte der Rechtecke beinhalten, und werden in einem neu erzeugten Template mit der angegebenen `template_id` gespeichert. Die Views werden schrittweise erzeugt: Beginnend bei dem Rechteck mit dem höchsten Erkennungs-Score wird ein View erzeugt und direkt verwendet, um weitere Rechtecke mit derselben

Textur zu finden. Dann werden in allen verbleibenden Clustern weitere Rechtecke mit den angegebenen Abmessungen detektiert und es wird wiederum aus dem besten Rechteck ein View generiert, der für weitere Erkennungen genutzt wird. Jedes Template kann bis zu 10 verschiedene Views speichern, zum Beispiel um verschiedene Sorten derselben Produktverpackung abzubilden. Jeder View hat eine eindeutige ID (`view_uuid`) und alle Rechtecke mit gleicher Textur erhalten dieselbe `view_uuid`. Das bedeutet auch, dass alle Objekte (`items`) mit derselben `view_uuid` konsistente Orientierungen haben, da die Orientierung jedes Objekts an der Textur ausgerichtet ist. Die Views können angezeigt und gelöscht werden, und ihre Orientierungen können über die *Web GUI* (Abschnitt 6.1) geändert werden, indem das Template oder sein Editierbutton in der Templateübersicht angeklickt wird. Jedes erkannte Objekt hat ein Feld `view_pose_set`, welches angibt, ob die Orientierung des zum Objekt zugeordneten Views explizit gesetzt wurde, oder ob sie unbestimmt in einem zufälligen Zustand ist, welcher eine 180° Mehrdeutigkeit hat. Weiterhin kann ein benutzerdefinierter Name für jeden View gesetzt werden, der gemeinsam mit der `view_uuid` zurückgegeben wird und die Identifikation bestimmter Views vereinfacht. Der Typ `type` eines zurückgelieferten Objekts mit einer `view_uuid` lautet `TEXTURED_RECTANGLE`.

Wenn ein Template mit der angegebenen `template_id` bereits existiert, werden die vorhandenen Views verwendet, um Rechtecke anhand ihrer Textur zu erkennen. Wenn weitere Rechtecke gefunden werden, die ebenfalls passende Abmessungen haben, aber eine andere Textur, dann werden neue Views generiert und dem Template hinzugefügt. Wenn die maximale Anzahl Views erreicht ist, werden zu selten detektierte Views gelöscht, damit neu generierte Views dem Template hinzugefügt werden können, und das Template aktuell gehalten wird. Um zu verhindern, dass ein Template durch neue Views aktualisiert wird, kann das automatische Updaten der Views in der Web GUI aus- und eingeschaltet werden. Die Dimensionstoleranz `dimensions_tolerance_m` und die maximale Verformung `max_deformation_m` können dort ebenso für jedes Template geändert werden. Die maximale Verformung bestimmt die Toleranz für die Texturerkennung, die nötig ist, wenn sich durch flexible Objektoberflächen Teile der Textur relativ zueinander verschieben. Für steife Objekte sollte die maximale Verformung möglichst niedrig gesetzt werden, um eine hohe Erkennungsgenauigkeit zu erreichen.

Die Abmessungen `dimensions` des Templates können nur beim Erstellen eines neuen Templates angegeben werden. Sobald das Template erzeugt wurde, können die Abmessungen nicht mehr geändert werden und müssen beim Aufruf der Erkennung auch nicht angegeben werden. Wenn die Abmessungen dennoch beim Aufruf angegeben werden, müssen sie mit den Abmessungen im vorhandenen Template übereinstimmen. Die Toleranz `dimensions_tolerance_m` und die maximale Verformung `max_deformation_m` können jedoch für jeden Detektionsaufruf unterschiedlich angegeben werden und ihre Werte werden auch im gespeicherten Template entsprechend aktualisiert.

Dimensionierung

BoxPick erkennt nur die Oberseite eines Objekts als Rechteck. Um die vollständigen Abmessungen der erkannten Objekte zu messen, kann mit dem Argument `dimensioning` der Services `detect_items`, `compute_grasps` und `compute_grasps_extended` eine Referenzebene angegeben werden, auf der die Objekte liegen, z.B. ein Förderband oder ein Tisch. `dimensioning.plane_ids` muss genau eine ID einer Ebene enthalten, die im *PrimitiveDB* (Abschnitt 5.6.3) gespeichert ist.

Für jedes erkannte Objekt wird aus den vier Eckpunkten des erkannten Rechtecks eine Box berechnet, wobei die Eckpunkte bis zur Referenzebene herunter gemessen werden. Die Box ist die an der Referenzebene ausgerichtete Bounding Box der Eckpunkte des Rechtecks.

Das zurückgegebene Objekt hat den `type BOX` (bzw. `TEXTURED_BOX` mit der +Match-Erweiterung) und enthält seine Abmessungen im Feld `box` statt im Feld `rectangle`. Die `pose` des Objekts ist der Mittelpunkt der Box. Die zurückgegebenen Greifpunkte werden von der Dimensionierung nicht beeinflusst.

Für Objektmodelle vom Typ `TEXTURED_BOX`, bei denen alle drei Abmessungen angegeben sind, ist keine Dimensionierung möglich, da die Abmessungen dieser Objekte bereits bekannt sind.

Objekte, die hinter der Referenzebene liegen, werden verworfen. Dies wird mit dem `return_code`-Wert 115 gemeldet.

5.4.5.3 Berechnung der Greifpunkte

Das BoxPick-Modul bietet einen Service, um Greifpunkte für Sauggreifer zu berechnen. Der Sauggreifer ist durch die Länge und Breite der Greiffläche definiert.

Die Greifpunkte werden auf den erkannten Rechtecken `items` berechnet (siehe *Erkennung von Rechtecken*, Abschnitt 5.4.5.2).

Optional können dem Modul weitere Informationen zur Greifpunktberechnung übergeben werden:

- die ID des Load Carriers, welcher die zu greifenden Objekte enthält
- ein Unterabteil (`load_carrier_compartment`) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 5.6.1.3).
- die ID der 3D Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die 3D Region of Interest, innerhalb der Greifpunkte berechnet werden
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 5.4.5.6) gegeben.
- Die ID einer Referenzebene zur Messung der Abmessungen der erkannten Objekte (siehe *Dimensionierung*, Abschnitt 5.4.5.2).

Ein vom BoxPick-Modul ermittelter Greifpunkt repräsentiert die empfohlene Pose des TCP (Tool Center Point) des Sauggreifers. Der Greifpunkt `type` ist immer auf `SUCTION` gesetzt. Für jeden Greifpunkt liegt der Ursprung der Greifpose `pose` im Mittelpunkt der größten von der jeweiligen Greiffläche umschlossenen Ellipse. Die Orientierung des Greifpunkts ist ein rechtshändiges Koordinatensystem, sodass die z-Achse orthogonal zur Greiffläche in das zu greifende Objekt zeigt und die x-Achse entlang der längsten Ausdehnung ausgerichtet ist.Da die x-Achse zwei mögliche Richtungen haben kann, wird diejenige ausgewählt, die besser zur bevorzugten TCP-Ausrichtung passt (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.5.5). Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, wird die x-Achse nicht zwangsweise an der maximalen Dehnung der greifbaren Ellipse ausgerichtet, sondern kann eine beliebige Drehung um die z-Achse aufweisen. In diesem Fall hat der zurückgegebene Greifpunkt die Ausrichtung, die am besten zur bevorzugten TCP-Ausrichtung passt und kollisionsfrei ist, wenn die Kollisionsprüfung aktiviert ist.

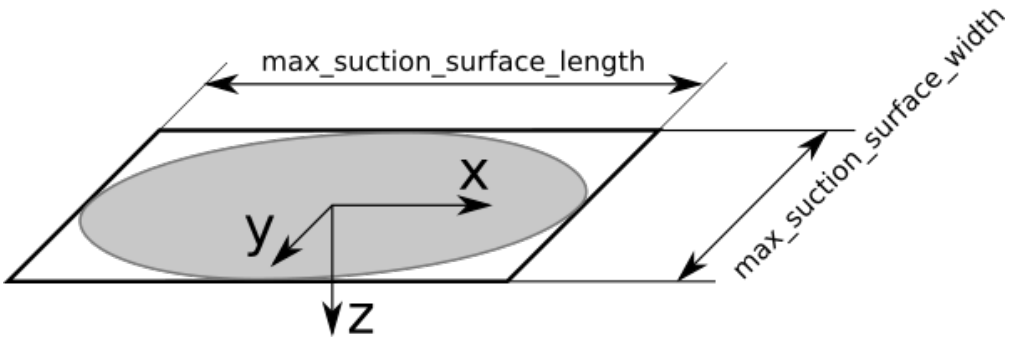


Abb. 5.17: Veranschaulichung eines berechneten Greifpunktes mit Koordinatensystem und der zugehörigen Ellipse, welche die größtmögliche Greiffläche beschreibt.

Zusätzlich enthält jeder Greifpunkt die Abmessungen der maximal verfügbaren Greiffläche, die als Ellipse mit den Achslängen `max_suction_surface_length` und `max_suction_surface_width` beschrieben wird. Der Nutzer kann Greifpunkte mit zu kleinen Greifflächen herausfiltern, indem die minimalen Abmessungen der Greiffläche, die vom Sauggreifer benötigt wird, angegeben werden. Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, dann sind die Achslängen `max_suction_surface_length` und `max_suction_surface_width` gleich und entsprechen der kürzeren Achse der größtmöglichen Greifellipse.

Im BoxPick-Modul entspricht der Greifpunkt dem Zentrum des detektierten Rechtecks. Wenn BoxPick mit einem Objektmodell vom Typ `RECTANGLE` aufgerufen wird, entsprechen die Achslängen der Greiffläche der Länge und Breite des erkannten Rechtecks. In diesem Fall erhalten Rechtecke keinen Greifpunkt, wenn mehr als 15% ihrer Fläche durch andere Objekte verdeckt ist oder ungültige Datenpunkte hat.

Wenn BoxPick mit einem Objektmodell vom Typ `TEXTURED_BOX` aufgerufen wird, können Greifpunkt auch auf teilweise verdeckten Rechtecken berechnet werden. Die zurückgegebene maximale Sauggreiffläche entspricht dann der freien Oberfläche des Rechtecks, die nicht durch andere Cluster verdeckt ist.

Jeder Greifpunkt enthält auch einen Qualitätswert (`quality`), der einen Hinweis auf die Ebenheit der Greiffläche gibt. Dieser Wert reicht von 0 bis 1, wobei höhere Werte für eine ebenere rekonstruierte Oberfläche stehen.

Jeder berechnete Greifpunkt lässt sich anhand einer `uuid` (Universally Unique Identifier) eindeutig identifizieren und enthält zusätzlich den Zeitstempel der ältesten Bildaufnahme, auf der die Greifpunktberechnung durchgeführt wurde.

Die Sortierung der Greifpunkte basiert auf der ausgewählten Sortierstrategie (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.5.4).

5.4.5.4 Setzen der Sortierstrategie

Die Objekte und Greifpunkte, die von den Services zurückgeliefert werden, werden gemäß einer Sortierstrategie sortiert, die vom Nutzer gewählt werden kann. Folgende Sortierstrategien sind verfügbar und können in der *Web GUI* (Abschnitt 6.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `gravity`: die höchsten Objekte oder Greifpunkte entlang der Gravitationsrichtung werden zuerst zurückgeliefert.
- `surface_area`: Objekte oder Greifpunkte mit den größten Oberflächen werden zuerst zurückgeliefert.
- `direction`: Objekte oder Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Richtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.
- `distance_to_point`: Objekte oder Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `gravity` und `surface_area`.

5.4.5.5 Setzen der bevorzugten TCP-Orientierung

Das BoxPick-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die BoxPick-Seite in der Web GUI gesetzt werden. Die daraus resultierende Richtung der z-Achse des TCPs wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann. Außerdem wird die bevorzugte Orientierung genutzt, um aus mehreren möglichen Symmetrien den Greifpunkt auszuwählen, der für den Roboter am besten erreichbar ist.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und die Kamera am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Koordinatensysteme) als die bevorzugte TCP-Orientierung genutzt.

5.4.5.6 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das BoxPick-Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieses Moduls kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des Boxpick-Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom BoxPick-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 5.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 5.2)

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_visard* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.5.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.5.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das BoxPick-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.4.5.9) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das ItemPick- oder BoxPick-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame` und der Definition der Sortierrichtung, des Sortierpunktes oder der bevorzugten Orientierung nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die bevorzugte Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

LoadCarrier

Das BoxPick-Modul nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 5.4.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine 3D Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das BoxPick-Modul verwendet wird.

Der Load Carrier wird verwendet um Fehldetektionen zu filtern, wenn BoxPick mit einem Objektmodell vom Typ `TEXTURED_BOX` aufgerufen wird, und alle drei Dimensionen x, y und z angegeben werden. In diesem Fall werden intern 3D Boxen generiert, indem die erkannten Rechtecken um die fehlende Dimension erweitert werden. Es werden dann nur die erkannten Rechtecke zurückgeliefert, bei denen die entsprechende 3D Box vollständig im Load Carrier enthalten ist.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des BoxPick-Moduls aktiviert werden, indem das `collision_detection` Argument an den `compute_grasps` oder `compute_grasps_extended` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 5.6.4.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.5.2.2) gegeben.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in den Visualisierungen auf der *BoxPick*-Seite der Web GUI kollidierende Greifpunkte als schwarze Ellipsen dargestellt.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 5.5.2.3) beschrieben.

5.4.5.7 Parameter

Das BoxPick-Modul wird in der REST-API als `rc_boxpick` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Module* → *BoxPick* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.32: Laufzeitparameter des `rc_boxpick` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`allow_any_grasp_z_rotation`	bool	false	true	false	Bestimmt, ob die Greifpunkte beliebige Orientierung haben dürfen, anstatt an der Hauptachse der greifbaren Ellipse ausgerichtet zu sein
`allow_untextured_detections`	bool	false	true	false	Gibt an, ob auch untexturierte Rechtecke zurückgegeben werden sollen, wenn ein Modell vom Typ TEXTURED_BOX angegeben wurde
`check_collisions_with_point_cloud`	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
`cluster_max_curvature`	float64	0.005	0.5	0.11	Maximal erlaubte Krümmung für Greifflächen
`clustering_discontinuity_factor`	float64	0.1	5.0	1.0	Erlaubte Unebenheit von Greifflächen
`clustering_max_surface_rmse`	float64	0.0005	0.01	0.004	Maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern
`discard_cut_off_items`	bool	false	true	false	Gibt an, ob Objekte, die vom Bildrand, der Stereo ROI, der fehlenden Stereo-Überlappung oder der 3D Region of Interest abgeschnitten werden, verworfen werden sollen
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
`line_sensitivity`	float64	0.1	1.0	0.1	Empfindlichkeit des Liniendetektors
`manual_line_sensitivity`	bool	false	true	false	Gibt an, ob die benutzerdefinierte Linienempfindlichkeit oder die automatische genutzt werden soll
`max_grasps`	int32	1	100	5	Maximale Anzahl von bereitgestellten Greifpunkten
`min_cluster_coverage`	float64	0.0	0.99	0.0	Bestimmt den minimalen Anteil an Punkten pro Cluster, die durch Detektionen abgedeckt sein müssen
`mode`	string	-	-	Unconstrained	Modus der Rechteckerkennung: [Unconstrained, PackedGridLayout, PackedLayers]
`prefer_splits`	bool	false	true	false	Gibt an, ob Rechtecke in kleinere Rechtecke gesplittet werden sollen, falls möglich

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der *BoxPick*-Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_grasps` (*Anzahl Greifpunkte*)

ist die maximale Anzahl von bereitgestellten Greifpunkten.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?max_grasps=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?max_grasps=<value>
```

`cluster_max_curvature` (*Maximale Krümmung*)

ist die maximal erlaubte Krümmung für Greifflächen. Je kleiner dieser Wert ist, desto mehr mögliche Greifflächen werden in kleinere Flächen mit weniger Krümmung aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?cluster_
↪max_curvature=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?cluster_max_curvature=<value>
```

`clustering_discontinuity_factor` (*Unstetigkeitsfaktor*)

beschreibt die erlaubte Unebenheit von Greifflächen. Je kleiner dieser Wert ist, umso mehr werden mögliche Greifflächen in kleinere Flächen mit weniger Unebenheiten aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?clustering_
↪discontinuity_factor=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?clustering_discontinuity_factor=
↪<value>
```

`clustering_max_surface_rmse` (*Maximaler RMSE*)

ist die maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?clustering_
↪max_surface_rmse=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?clustering_max_surface_rmse=
↪<value>
```

`mode` (*Modus*)

legt den Modus der Rechteckerkennung fest. Mögliche Werte sind `Unconstrained` (*Unbeschränkt*), `PackedGridLayout` (*Dichtes Gitterlayout*) und `PackedLayer` (*Dicht geschichtet*). Im Modus `PackedGridLayout` werden Rechtecke eines Clusters in einem dichten Gittermuster erkannt. Im Modus `PackedLayers` wird angenommen, dass die Boxen Schichten (Layer) bilden, und die Erkennung der Boxen startet an den Ecken des Clusters. Dieser Modus sollte für Depalettierszenarien genutzt werden. Im Modus `Unconstrained` (Standardwert) werden Rechtecke unabhängig von ihren relativen Positionen zueinander und ihren Positionen im Cluster erkannt. Abb. 5.18 zeigt die Modi für verschiedene Szenarien.

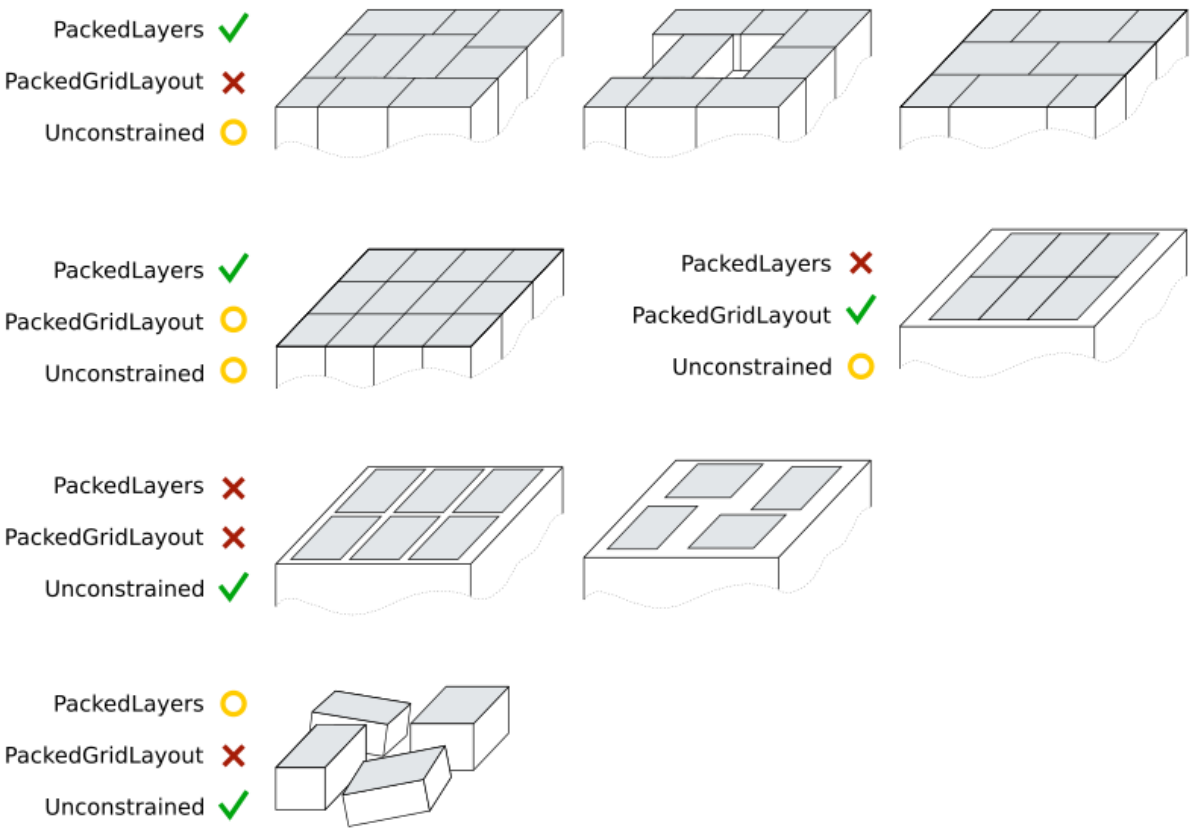


Abb. 5.18: Darstellung geeigneter BoxPick Modi für unterschiedliche Szenen. Gelb markierte Modi sind anwendbar, aber nicht empfohlen für das jeweilige Szenario. Die grauen Flächen markieren die Rechtecke, die erkannt werden sollen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?mode=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?mode=<value>
```

`manual_line_sensitivity` (*Manuelle Linienempfindlichkeit*)

legt fest, ob die benutzerdefinierte Linienempfindlichkeit für die Liniendetektion zur Rechteckerkennung verwendet werden soll. Wenn dieser Parameter auf true gesetzt ist, wird der benutzerdefinierte Wert in `line_sensitivity` (Linienempfindlichkeit) zur Detektion verwendet, andernfalls wird die Linienempfindlichkeit automatisch ermittelt. Dieser Parameter sollte auf true gesetzt werden, wenn die automatische Linienempfindlichkeit nicht genügend Linien an den Rändern der Boxen liefert, sodass Boxen nicht erkannt werden. Die detektierten Linien werden in der „Zwischenergebnis“ Visualisierung auf der *BoxPick* Seite in der Web GUI angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?manual_
↪line_sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?manual_line_sensitivity=<value>
```

`line_sensitivity` (*Linienempfindlichkeit*)

legt die Empfindlichkeit für die Detektion von Linien für die Rechteckerkennung fest, wenn der Parameter `manual_line_sensitivity` (Manuelle Linienempfindlichkeit) auf true gesetzt ist. Andernfalls hat dieser Parameter keinen Einfluss auf die Rechteckerkennung. Höhere Werte liefern mehr Liniensegmente, aber erhöhen auch die Laufzeit der Detektion. Dieser Parameter sollte erhöht werden, wenn Boxen nicht erkannt werden können, weil ihre Ränder nicht als Linien detektiert werden. Die erkannten Linien werden in der „Zwischenergebnis“ Visualisierung auf der *BoxPick* Seite in der Web GUI angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?line_
↪sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?line_sensitivity=<value>
```

`prefer_splits` (*Splitting bevorzugen*)

bestimmt, ob Rechtecke in kleinere Rechtecke aufgesplittet werden, falls die kleineren Rechtecke ebenfalls den angegebenen Objektmodellen entsprechen. Dieser Parameter sollte auf true gesetzt werden, wenn Boxen dicht beieinander liegen, und die Objektmodelle auch zu einem Rechteck der Größe von zwei angrenzenden Boxen passen. Wenn dieser Parameter auf false steht, werden in solch einem Fall die Rechtecke bevorzugt, die sich aus zwei angrenzenden Boxen ergeben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?prefer_
↪splits=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?prefer_splits=<value>
```

`min_cluster_coverage` (*Minimale Clusterabdeckung*)

bestimmt den Anteil von Punkten in jedem segmentierten Cluster, der durch Rechtecksdetektionen abgedeckt sein muss, um diese Detektionen als valide anzunehmen. Wird die minimale Clusterabdeckung unterschritten, werden für das jeweilige Cluster keine Detektionen zurückgeliefert und eine Warnung ausgegeben. Dieser Parameter sollte genutzt werden, um in einem Depalettierszenario zu verifizieren, dass alle Objekte in einem Layer detektiert wurden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?min_
↪cluster_coverage=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?min_cluster_coverage=<value>
```

`allow_untextured_detections` (Nur für BoxPick+Match, *Untexturierte Detektionen*)

ermöglicht die Rückgabe aller Rechtecke, die den angegebenen Templateabmessungen entsprechen, auch wenn sie mit keinem vorhandenen View gematcht werden können oder wenn sie nicht über genügend Textur verfügen, um daraus einen neuen View zu generieren. Das Deaktivieren dieses Parameters führt zu schnellerer Laufzeit, wenn ein Template verwendet wird, für das automatische View Updates gesperrt sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?allow_
↪untextured_detections=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?allow_untextured_detections=
↪<value>
```

`discard_cut_off_items` (*Abgeschnittene Objekte verwerfen*)

bestimmt, ob Objekte, die vom Bildrand, der Stereo Region of Interest (siehe Abschnitt *Stereo ROI Parameter*, 5.2.1), der fehlenden Stereo-Überlappung zwischen dem linken und rechten Kamerabild oder der 3D Region of Interest abgeschnitten werden, verworfen werden sollen.

Dieser Parameter hat keine Auswirkung, wenn der Objektmodell Typ `UNKNOWN` verwendet wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?discard_
↪cut_off_items=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?discard_cut_off_items=<value>
```

`grasp_filter_orientation_threshold` (*Grasp Orientation Threshold*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der bevorzugten TCP-Orientierung in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist. Falls der Wert auf Null gesetzt wird, sind alle Abweichungen valide.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?grasp_
↪filter_orientation_threshold=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?grasp_filter_orientation_
↪threshold=<value>
```

`allow_any_grasp_z_rotation` (*Allow Any Grasp Z Rotation*)

Wenn der Wert auf True gesetzt ist, werden die x-Achsen der zurückgegebenen Greifpunkte nicht mehr notwendigerweise an der maximalen Ausdehnung der greifbaren Ellipse ausgerichtet, sondern können eine beliebige Drehung um die z-Achse haben. Die zurückgegebenen Werte von `max_suction_surface_length` und `max_suction_surface_width` sind dann gleich und entsprechen dem kleinsten Durchmesser der größten greifbaren Ellipsenfläche. Dieser Parameter eröffnet dem Roboter mehr Optionen zum Greifen von Objekten, insbesondere in Szenen, in denen es zu Kollisionen kommen kann. Da der Greifpunkt jedoch nicht mehr mit der greifbaren Ellipse ausgerichtet, muss die richtige Orientierung zum Platzieren des Objekts anhand der Pose des zugehörigen `item` ermittelt werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?allow_any_
↪grasp_z_rotation=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?allow_any_grasp_z_rotation=<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `compute_grasps` oder `compute_grasps_extended` Service aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur

Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/parameters/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?check_collisions_with_point_cloud=
↪<value>
```

5.4.5.8 Statuswerte

Das `rc_boxpick` Modul meldet folgende Statuswerte:

Tab. 5.33: Statuswerte des `rc_boxpick` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste.
`grasp_computation_time`	Laufzeit für die Greifpunktberechnung beim letzten Aufruf in Sekunden
`last_return_code_message`	Rückgabecode-Nachricht der letzten Detektion
`last_return_code_value`	Rückgabecode-Wert der letzten Detektion
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Laufzeit für die letzte Load Carrier Erkennung in Sekunden
`processing_time`	Laufzeit für die letzte Erkennung (einschließlich Load Carrier Detektion) in Sekunden
`state`	Aktueller Zustand des BoxPick-Moduls

Folgende `state`-Werte werden gemeldet.

Tab. 5.34: Mögliche Werte für den Zustand des BoxPick Moduls

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier zu erkennen und Greifpunkte zu berechnen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

5.4.5.9 Services

Die angebotenen Services von `rc_boxpick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

Das BoxPick-Modul stellt folgende Services zur Verfügung.

`detect_items`

löst die Erkennung von Rechtecken aus, wie in *Erkennung von Rechtecken* (Abschnitt 5.4.5.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/detect_items
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/detect_items
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.5.6).

`item_models`: Liste der zu erkennenden Objektmodelle. Der Typ `type` der Modelle muss immer `RECTANGLE` oder `TEXTURED_BOX` sein. Für den Typ `RECTANGLE` muss das Feld `rectangle` gefüllt werden, wohingegen für `TEXTURED_BOX` das Feld `textured_box` angegeben werden muss. Siehe *Erkennung von Rechtecken* (Abschnitt 5.4.5.2) für eine ausführliche Beschreibung der Objektmodelle.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.5.6).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 5.6.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, in der nach Objekten gesucht wird.

`dimensioning`: `plane_ids` enthält die ID der Referenzebene zur Messung der Abmessungen der erkannten Objekte (siehe *Dimensionierung*, Abschnitt 5.4.5.2).

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "dimensioning": {
      "plane_ids": [
        "string"
      ]
    },
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "textured_box": {
          "dimensions": {
            "x": "float64",
            "y": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "dimensions_tolerance_m": "float64",
          "max_deformation_m": "float64",
          "template_id": "string"
        },
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`items`: Liste von erkannten Rechtecken. Falls `dimensioning` angegeben ist, Liste von erkannten Boxen.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_items",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
"response": {
  "items": [
    {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rectangle": {
        "x": "float64",
        "y": "float64"
      },
      "template_id": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string",
      "view_name": "string",
      "view_pose_set": "bool",
      "view_uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


compute_grasps

löst die Erkennung von Rechtecken und Berechnung von Greifposen für diese Rechtecke aus, wie in *Berechnung der Greifpunkte* (Abschnitt 5.4.5.3) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/compute_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/compute_grasps
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.5.6).

`item_models`: Liste der zu erkennenden Objektmodelle. Der Typ `type` der Modelle muss immer `RECTANGLE` oder `TEXTURED_BOX` sein. Für den Typ `RECTANGLE` muss das Feld `rectangle` gefüllt werden, wohingegen für `TEXTURED_BOX` das Feld `textured_box` angegeben werden muss. Siehe *Erkennung von Rechtecken* (Abschnitt 5.4.5.2) für eine ausführliche Beschreibung der Objektmodelle.

`suction_surface_length`: Länge der Greiffläche des verwendeten Vakuum-Greifsystems.

`suction_surface_width`: Breite der Greiffläche des verwendeten Vakuum-Greifsystems.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.5.6).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu greifenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 5.6.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, innerhalb der Greifpunkte berechnet werden.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.5.2.2)

`dimensioning`: `plane_ids` enthält die ID der Referenzebene zur Messung der Abmessungen der erkannten Objekte (siehe *Dimensionierung*, Abschnitt 5.4.5.2).

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "dimensioning": {
      "plane_ids": [
        "string"
      ]
    },
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "textured_box": {
          "dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "dimensions_tolerance_m": "float64",
          "max_deformation_m": "float64",
          "template_id": "string"
        },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```


Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten.

`items`: Liste von erkannten Rechtecken, die zu den zurückgelieferten Greifpunkten gehören. Falls `dimensioning` angegeben ist, Liste von erkannten Boxen.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    {
      "item_uuid": "string",
      "max_suction_surface_length": "float64",
      "max_suction_surface_width": "float64",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "quality": "float64",
      "tcp_id": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "items": [
    {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "grasp_uuids": [
        "string"
      ],
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rectangle": {
        "x": "float64",
        "y": "float64"
      },
      "template_id": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "type": "string",
        "uuid": "string",
        "view_name": "string",
        "view_pose_set": "bool",
        "view_uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

compute_grasps_extended

löst die Erkennung von Rechtecken und Berechnung von Greifposen für diese Rechtecke aus. Dieser Service verhält sich analog zu `compute_grasps`, gibt aber die Objektinformationen für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn Objektinformationen für die Greifpunkte benötigt werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/compute_grasps_extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/compute_grasps_extended
```

Request

Siehe `compute_grasps` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "dimensioning": {
      "plane_ids": [
        "string"
      ]
    },
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "textured_box": {
          "dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "dimensions_tolerance_m": "float64",
          "max_deformation_m": "float64",
          "template_id": "string"
        },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```


Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten. Jeder Greifpunkt enthält die `item` Information des zugehörigen erkannten Rechtecks.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "compute_grasps_extended",
  "response": {
    "grasps": [
      {
        "item": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rectangle": {
          "x": "float64",
          "y": "float64"
        },
        "template_id": "string",
        "type": "string",
        "uuid": "string",
        "view_name": "string",
        "view_pose_set": "bool",
        "view_uuid": "string"
      },
      "max_suction_surface_length": "float64",
      "max_suction_surface_width": "float64",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "quality": "float64",
      "tcp_id": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `compute_grasps` oder `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.5.5).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/set_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional für die Sortierung der vom `detect_object` und `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.5.5).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/get_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_sorting_strategies

speichert dauerhaft die Sortierstrategie zur Sortierung der Objekte und Greifpunkte (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.5.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/set_sorting_strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`get_sorting_strategies`

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der Objekte und Greifpunkte verwendet wird (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.5.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/get_sorting_strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_sorting_strategies",
  "response": {
    "direction": {
      "pose_frame": "string",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```


start

startet das Modul und versetzt es in den Zustand `RUNNING`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "start",
  "response": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul und versetzt es in den Zustand `IDLE`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter, der Sortierstrategie und der bevorzugten Orientierung dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_boxpick/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


5.4.5.10 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.35: Rückgabecodes der Services des BoxPick-Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Boxerkennung, wenn der Bereich der angegebenen Abmessungen zu groß ist.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-8	Das Template wurde während der Detektion gelöscht.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
-301	Für die Anfrage zur Greifpunktberechnung `compute_grasps` oder `compute_grasps_extended` wurden mehrere Objektmodelle (`item_models`) übergeben.
10	Die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` oder `set_region_of_interest` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Es wurden keine gültigen Greifflächen in der Szene gefunden.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
112	Die Detektionen eines oder mehrerer Cluster wurden verworfen, da die minimale Clusterabdeckung nicht erreicht wurde.
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

5.4.5.11 BoxPick Template API“

BoxPick Templates sind nur mit der +Match-Erweiterung von BoxPick verfügbar. Für den Upload, Download, das Auflisten und Löschen von Templates werden spezielle REST-API-Endpunkte zur Verfügung

gestellt. Templates können auch über die Web GUI hoch- und runtergeladen werden. Die Templates beinhalten die Greifpunkte und Posenvorgaben, falls Greifpunkte oder Posenvorgaben konfiguriert wurden. Bis zu 100 Templates können gleichzeitig auf dem *rc_visard* gespeichert werden.

`GET /templates/rc_boxpick`
listet alle rc_cadmatch-Templates auf.

Musteranfrage

```
GET /api/v2/templates/rc_boxpick HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array der Templates)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.2.4)

`GET /templates/rc_boxpick/{id}`
ruft ein rc_boxpick-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/templates/rc_boxpick/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Templates *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 404 Not Found – Modul oder Template wurden nicht gefunden.

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.2.4)

PUT /templates/rc_boxpick/{id}
erstellt oder aktualisiert ein rc_boxpick-Template.

Musteranfrage

```
PUT /api/v2/templates/rc_boxpick/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Formularparameter

- **file** – Template-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.2.4)

DELETE /templates/rc_boxpick/{id}
entfernt ein rc_boxpick-Template.

Musteranfrage

```
DELETE /api/v2/templates/rc_boxpick/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

5.4.6 Pack

5.4.6.1 Einführung

Das Pack-Modul liefert eine gebrauchsfertige Perzeptionslösung für robotische Pick-and-Place-Anwendungen, bei denen zuvor gegriffene Objekte in einem Packvolumen, z. B. einem Karton, einer Kiste oder auf einer Palette, platziert werden sollen. Anhand der Bounding Box des zu platzierenden Objekts und des bzw. der Greifpunkte, mit denen es aufgenommen werden kann, berechnet Pack eine Pose, an der das Objekt platziert werden kann, wobei der aktuelle Füllstand des Packvolumens berücksichtigt wird. Die Platzierungspose wird so gewählt, dass sie die niedrigste Position mit stabiler Unterstützung für das Objekt darstellt und dadurch den nach der Platzierung verbleibenden Freiraum maximiert.

Darüber hinaus bietet das Modul:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_visard* *Web GUI* (Abschnitt 6.1)
- die Definition einer Region of Interest, um das relevante Packvolumen in der Szene festzulegen oder den Suchbereich für den Load Carrier einzuschränken (siehe *RoiDB*, Abschnitt 5.6.2)
- eine Load-Carrier-Erkennungsfunktion (siehe *LoadCarrier*, Abschnitt 5.4.2), um Objekte ausschließlich innerhalb eines erkannten Load Carriers zu platzieren
- einen konfigurierbaren Startpunkt, um zu steuern, wo das Packen beginnt
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1) kombiniert werden, um Platzierungsposen in einem benutzerdefinierten externen Koordinatensystem zu liefern
- eine 3D-Visualisierung des Packvolumens und der berechneten Platzierungspose in der Web GUI

Das Modul ist ein optional erhältliches Modul, welches intern auf dem *rc_visard* läuft und eine gesonderte Pack *Lizenz* (Abschnitt 7.4) benötigt.

5.4.6.2 Berechnung von Platzierungsposen

Das Pack-Modul bietet einen Service zur Berechnung einer Platzierungspose für ein Objekt, unabhängig davon, ob dieses bereits gegriffen wurde oder erst noch gegriffen werden soll. Das zu platzierende Objekt wird durch seine `bounding_box` beschrieben. Optional können zusätzlich die aktuelle `pose` des Objekts und ein oder mehrere Greifpunkte (`grasps`), mit denen es gegriffen werden kann, angegeben werden; diese Pose und diese Greifpunkte können aus einem anderen Detektionsmodul stammen, z. B. ItemPick oder BoxPick. Alternativ können die Greifpunkte relativ zum Mittelpunkt der Bounding Box des Objekts angegeben werden.

Das Modul diskretisiert das Packvolumen in x- und y-Richtung entsprechend dem Laufzeitparameter `resolution`. Ausgehend von der Ecke des Packvolumens, die dem konfigurierten *Startpunkt* am nächsten liegt, sucht es die niedrigste Position, an der das Objekt – vergrößert um den Laufzeitparameter `object_padding` als Sicherheitsspielraum – ohne Überlappung mit bereits platzierten Objekten oder

dem Load Carrier passt. Dadurch wird der nach der Platzierung verbleibende Freiraum maximiert und es werden möglichst viele Packoptionen für das nächste zu platzierende Objekt offengehalten.

Optional können dem Modul zu einer Berechnung einer Platzierungspose weitere Informationen übergeben werden:

- die ID des Load Carriers, der das Packvolumen definiert
- die ID der 3D Region of Interest, die das Packvolumen definiert, falls kein Load Carrier angegeben ist, oder die den Suchbereich für den Load Carrier einschränkt

Die Platzierungspose enthält zusätzlich den `timestamp` der ältesten Bildaufnahme, die für ihre Berechnung verwendet wurde.

5.4.6.3 Festlegen des Startpunkts

Das Pack-Modul bestimmt anhand des *Startpunkts*, wo das Befüllen des Packvolumens beginnt. Das Modul vergleicht den konfigurierten Startpunkt mit den Ecken des Packvolumens, das durch den konfigurierten Load Carrier oder die Region of Interest definiert wird, und beginnt das Befüllen an derjenigen Ecke, die ihm am nächsten liegt.

Der Startpunkt kann über den Service `set_starting_point` oder auf der Pack-Seite der Web GUI festgelegt werden und wird dauerhaft gespeichert. Er kann im Kamera-Koordinatensystem oder, falls eine Hand-Auge-Kalibrierung vorliegt, im externen Koordinatensystem angegeben werden.

5.4.6.4 Festlegen der bevorzugten TCP-Orientierung

Beim Platzieren einer Box besteht für jede Platzierungspose eine 180°-Rotationsmehrdeutigkeit um die z-Achse. Von den angegebenen Greifpunkten (`grasps`) wählt das Pack-Modul diejenigen aus, deren Orientierungen zum Platzieren des Objekts der *bevorzugten TCP-Orientierung* am nächsten kommen, wodurch die 180°-Platzierungsmehrdeutigkeit aufgelöst wird. Die bevorzugte TCP-Orientierung kann über den Service `set_preferred_orientation` oder auf der Pack-Seite der Web GUI festgelegt werden. Greifpunkte, die um mehr als den Laufzeitparameter `grasp_filter_orientation_threshold` davon abweichen, werden verworfen.

Die bevorzugte TCP-Orientierung kann im Kamera-Koordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist und der Sensor robotergeführt ist, muss bei jedem Aufruf zur Berechnung einer Platzierungspose die aktuelle Roboterpose angegeben werden.

Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Koordinatensysteme) als bevorzugte TCP-Orientierung verwendet.

5.4.6.5 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das Pack-Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des Pack-Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom Pack-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 5.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 5.2)

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_visard* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.5.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.5.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das Pack-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.4.6.8) des Pack-Knotens kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Es kann zwischen den folgenden zwei Werten für `pose_frame` gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Load Carrier Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wurde keine Hand-Auge-Kalibrierung durchgeführt, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Für den Fall eines robotergeführten Sensors ist es abhängig vom Wert von `pose_frame` nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

LoadCarrier

Das Pack-Modul nutzt die Funktionalität zur Load-Carrier-Erkennung aus dem *LoadCarrier*-Modul (`rc_load_carrier`, Abschnitt 5.4.2) mit den für dieses Modul festgelegten Laufzeitparametern, um die Grenzen des Packvolumens zu bestimmen. Befinden sich jedoch mehrere passende Load Carrier in der Szene, wird nur einer davon verwendet. Sind mehrere Load Carrier desselben Typs sichtbar, sollte

eine 3D Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das Pack-Modul verwendet wird.

5.4.6.6 Parameter

Das Pack-Modul wird in der REST-API als `rc_pack` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Module* → *Pack* dargestellt. Der Benutzer kann die Laufzeitparameter des `rc_pack`-Moduls, z. B. für Entwicklung und Test, über die Web GUI oder die *REST-API-Schnittstelle* (Abschnitt 6.2) erkunden und konfigurieren.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.36: Laufzeitparameter des `rc_pack` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`drop_height`	float64	0.0	0.5	0.0	Höhe über der Zielplatzierungsposition (in z-Richtung des Packvolumens) zum Loslassen des Objekts
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
`object_padding`	float64	0.001	1.0	0.005	Breite in Metern zur Vergrößerung der Objektabmessungen in x- und y-Richtung als Sicherheitsspielraum
`resolution`	float64	0.001	0.1	0.005	Auflösung zur Diskretisierung des Packvolumens in x- und y-Richtung

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der Pack-Seite in der Web GUI dargestellt. Der Name des Parameters in der Web GUI wird dabei in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`resolution` (*Resolution*)

ist die Auflösung in Metern, mit der das Packvolumen in x- und y-Richtung bei der Suche nach einer freien Platzierungsposition diskretisiert wird. Je kleiner dieser Wert ist, desto feiner ist das Suchraster, allerdings auf Kosten einer längeren Berechnungszeit.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/parameters/parameters?resolution=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/parameters?resolution=<value>
```

`object_padding` (*Object Padding*)

ist die Breite in Metern, um die die `bounding_box` des Objekts in x- und y-Richtung als Sicherheitsspielraum vergrößert wird, wenn geprüft wird, ob es an einer möglichen Platzierungsposition passt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/parameters/parameters?object_
↪padding=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/parameters?object_padding=<value>
```

`drop_height` (*Drop Height*)

legt die Höhe in Metern über der Zielplatzierungsposition entlang der z-Achse des Packvolumens fest, in der der Greifer das Objekt loslässt, z. B. um es ein kurzes Stück fallen zu lassen, statt es vollständig kontrolliert bis zur Oberfläche abzusetzen. Die vom Pack-Modul bereitgestellte Platzierungspose entspricht der empfohlenen Pose des Objekts im Moment des Loslassens.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/parameters/parameters?drop_height=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/parameters?drop_height=<value>
```

`grasp_filter_orientation_threshold` (*Grasp Orientation Threshold*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der *bevorzugten TCP-Orientierung* in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist. Falls der Wert auf Null gesetzt wird, sind alle Abweichungen valide.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/parameters/parameters?grasp_filter_
↪orientation_threshold=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/parameters?grasp_filter_orientation_threshold=
↪<value>
```

5.4.6.7 Statuswerte

Das `rc_pack` Modul meldet folgende Statuswerte:

Tab. 5.37: Statuswerte des rc_pack Moduls

Name	Beschreibung
data_acquisition_time	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
last_return_code_message	Rückgabecode-Nachricht der letzten Detektion
last_return_code_value	Rückgabecode-Wert der letzten Detektion
last_timestamp_processed	Zeitstempel des letzten verarbeiteten Bilddatensatzes
load_carrier_detection_time	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden
processing_time	Berechnungszeit für die letzte Detektion (einschließlich Load Carrier Detektion) in Sekunden

5.4.6.8 Services

Die angebotenen Services von rc_pack können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

Das Pack Modul stellt folgende Services zur Verfügung.

detect_place_pose

löst die Berechnung einer Platzierungspose aus, wie in *Berechnung von Platzierungsposen* (Abschnitt 5.4.6.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/services/detect_place_pose
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/services/detect_place_pose
```

Request

Obligatorische Serviceargumente:

pose_frame: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.6.5).

objects: das zu platzierende Objekt, beschrieben durch seine bounding_box und optional die pose, an der es sich aktuell befindet. Derzeit wird nur ein einzelnes Objekt unterstützt.

grasps: ein oder mehrere Greifpunkte, mit denen das in objects aufgeführte Objekt gegriffen wurde oder gegriffen werden kann, referenziert über object_id. Wird kein pose_frame angegeben, wird die pose jedes Greifpunkts relativ zum Mittelpunkt der Bounding Box des Objekts interpretiert. Wird ein pose_frame angegeben, benötigt das Objekt zusätzlich eine pose. Dadurch können Detektionsergebnisse anderer Module, wie ItemPick oder BoxPick, für Pack wiederverwendet werden.

Potentiell obligatorische Serviceargumente:

robot_pose: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.6.5).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, der das Packvolumen definiert.

`region_of_interest_id`: ID der 3D Region of Interest, die das Packvolumen definiert, falls `load_carrier_id` nicht gesetzt ist, oder die den Bereich zur Erkennung des Load Carriers einschränkt.

Mindestens eines von `load_carrier_id` oder `region_of_interest_id` muss angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "object_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "tcp_id": "string"
      }
    ],
    "load_carrier_id": "string",
    "objects": [
      {
        "bounding_box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string"
      }
    ],
    "pose_frame": "string",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "region_of_interest_id": "string"
  }
}
```

Response

`pose`: die berechnete Platzierungspose.

`grasps`: der/die in der Anfrage übergebene(n) Greifpunkt(e), mit dem/denen die endgültige Platzierung erreicht werden kann.

`load_carriers`: die Liste, die den erkannten Load Carrier enthält.

`timestamp`: Zeitstempel des Bildes, das für die Erkennung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_place_pose",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
        "id": "string",
        "object_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        }
      }
    ],
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


get_starting_point

gibt den Startpunkt zurück, der zur Auswahl der Ecke des Packvolumens verwendet wird, ab der das Befüllen beginnt (siehe *Festlegen des Startpunkts*, Abschnitt 5.4.6.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/services/get_starting_point
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/services/get_starting_point
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_starting_point",
  "response": {
    "point": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_starting_point

speichert den Startpunkt, der zur Auswahl der Ecke des Packvolumens verwendet wird, ab der das Befüllen beginnt (siehe *Festlegen des Startpunkts*, Abschnitt 5.4.6.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/services/set_starting_point
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/services/set_starting_point
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "point": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_starting_point",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die bei der Auswahl des Greifpunkts zum Platzieren des Objekts verwendet wird (siehe *Festlegen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.6.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/services/get_preferred_orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung, die bei der Auswahl des Greifpunkts zum Platzieren des Objekts verwendet wird (siehe *Festlegen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.6.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/services/set_preferred_orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an. Setzt außerdem den Startpunkt und die bevorzugte TCP-Orientierung zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_pack/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_pack/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


5.4.6.9 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.38: Rückgabecodes der Services des Pack Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
100	Der angefragte Load Carrier wurde in der Szene nicht gefunden
115	Es wurde keine gültige Platzierungspose innerhalb des Packvolumens gefunden
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

5.4.7 SilhouetteMatch

5.4.7.1 Einführung

Das SilhouetteMatch Modul ist ein optionales Modul, welches intern auf dem *rc_visard* läuft, und benötigt eine eigene *Lizenz* (Abschnitt 7.4), welche erworben werden muss.

Das Modul erkennt Objekte, indem eine vordefinierte Silhouette („Template“) mit Kanten im Bild verglichen wird.

Das SilhouetteMatch Modul kann Objekte in zwei verschiedenen Szenarien erkennen:

Mit kalibrierter Basisebene: Die Objekte befinden sich auf einer gemeinsamen Basisebene, die vor der Objekterkennung kalibriert werden muss, und die Objekte haben prägnante Kanten auf einer gemeinsamen Ebene, welche parallel zu der Basisebene ist.

Mit Objektebenenerkennung: Die Objekte können sich auf verschiedenen, vorab unbekannten Ebenen befinden, falls die Objekte eine planare Oberfläche haben und ihre Konturen gut in den Kamerabildern sichtbar sind (z.B. gestapelte flache Objekte).

Templates für die Objekterkennung können erstellt werden, indem eine DXF Datei hochgeladen und die Objekthöhe angegeben wird. Die korrekte Skalierung und Einheit der Konturen wird aus der DXF

Datei extrahiert. Falls die DXF Datei keine Einheit enthält, muss der Nutzer die korrekte Einheit angeben. Wenn die Außenkontur des Objekts in der DXF Datei geschlossen ist, wird automatisch ein 3D Kollisionsmodell erstellt, indem die Kontur auf die Objekthöhe extrudiert wird. Dieses Modell wird dann zur Kollisionsprüfung und 3D-Visualisierung verwendet. Das Hochladen der DXF Datei kann in der Web GUI über guilabel:*+ Neues Template erstellen* im Abschnitt *SilhouetteMatch Templates und Greifpunkte* auf der *Module* → *SilhouetteMatch* oder *Datenbank* → *Templates* Seite erfolgen.

Roboception bietet hierfür auch einen Template-Generierungsservice auf ihrer Website (https://roboception.com/de/template-request-de/) an, auf der der Benutzer CAD-Daten oder mit dem System aufgenommene Daten hochladen kann, um Templates generieren zu lassen.

Templates bestehen aus den prägnanten Kanten eines Objekts. Die Kanten des Templates werden mit den erkannten Kanten im linken und rechten Kamerabild abgeglichen, wobei die Größe der Objekte und deren Abstand zur Kamera mit einbezogen wird. Die Posen der erkannten Objekte werden zurückgegeben und können beispielsweise benutzt werden, um die Objekte zu greifen.

Das SilhouetteMatch Modul bietet:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_visard Web GUI* (Abschnitt 6.1)
- eine *REST-API-Schnittstelle* (Abschnitt 6.2) und eine *KUKA Ethernet KRL Schnittstelle* (Abschnitt 6.6)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche des Kamerabilds auszuwählen (siehe *Setzen einer Region of Interest*, Abschnitt 5.4.7.3)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 5.4.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Speicherung von bis zu 50 Templates
- die Definition von bis zu 50 Greifpunkten für jedes Template über eine interaktive Visualisierung in der Web GUI
- eine Kollisionsprüfung zwischen Greifer und Load Carrier, der kalibrierten Basisebene, anderen erkannten Objekten, und/oder der Punktwolke
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern
- Auswahl einer Strategie zum Sortieren der erkannten Objekte und zurückgelieferten Greifpunkte
- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

Taugliche Objekte

Das SilhouetteMatch Modul ist für Objekte ausgelegt, die prägnante Kanten auf einer Ebene besitzen, welche parallel zu der Ebene ist, auf der die Objekte liegen. Das trifft beispielsweise auf flache, nichttransparente Objekte zu, wie gefräste, lasergeschnittene oder wasserstrahlgeschnittene Teile. Komplexere Objekte können auch erkannt werden, solange sie prägnante Kanten auf einer Ebene besitzen, z.B. ein gedrucktes Muster auf einer ebenen Fläche.

Falls die Objekte nicht auf einer gemeinsamen Ebene liegen oder die Basisebene nicht vorab kalibriert werden kann, brauchen die Objekte eine planare Oberfläche und ihre Konturen müssen gut im linken und rechten Kamerabild sichtbar sein. Weiterhin müssen die Templates für diese Objekte eine geschlossene Außenkontur haben.

Taugliche Szene

Eine für das SilhouetteMatch Modul taugliche Szene muss folgende Bedingungen erfüllen:

- Die zu erkennenden Objekte müssen, wie oben beschrieben, tauglich für das SilhouetteMatch Modul sein.
- Nur Objekte, die zum selben Template gehören, dürfen gleichzeitig sichtbar sein (sortenrein). Falls auch andere Objekte sichtbar sind, muss eine passende Region of Interest (ROI) festgelegt werden.
- Im Falle einer kalibrierten Basisebene: Die Verkippung der Basisebene zur Blickrichtung der Kamera darf 10 Grad nicht übersteigen.
- Im Falle von verschiedenen oder unbekannten Basisebenen: Die Verkippung der planaren Oberfläche der Objekte zur Blickrichtung der Kamera darf 25 Grad nicht übersteigen.
- Die Objekte sind weder teilweise noch komplett verdeckt. .
- Alle sichtbaren Objekte liegen richtig herum.
- Die Objektkanten, welche abgeglichen werden sollen, sind sowohl im linken als auch im rechten Kamerabild zu sehen.

5.4.7.2 Kalibrierung der Basisebene

Falls alle Objekte auf einer gemeinsamen Ebene liegen, die vorab bekannt ist, sollte diese Ebene kalibriert werden, bevor die Objekterkennung gestartet wird. Hierbei wird die Distanz und der Winkel der Ebene, auf welcher die Objekte liegen, gemessen und persistent auf dem *rc_visard* gespeichert.

Durch die Trennung der Kalibrierung der Basisebene von der eigentlichen Objekterkennung werden beispielsweise Szenarien ermöglicht, in denen die Basisebene zeitweise verdeckt ist. Darüber hinaus wird die Berechnungszeit der Objekterkennung für Szenarien verringert, in denen die Basisebene für eine gewisse Zeit fixiert ist – die Basisebene muss in diesem Fall nicht fortlaufend neu detektiert werden.

Die Kalibrierung der Basisebene kann mit drei unterschiedlichen Verfahren durchgeführt werden, auf die im Folgenden näher eingegangen wird:

- AprilTag-basiert
- Stereo-basiert
- Manuell

Die Kalibrierung ist erfolgreich, solange der Normalenvektor der Basisebene höchstens 10 Grad gegen die Blickrichtung der Kamera verkippt ist. Eine erfolgreiche Kalibrierung wird persistent auf dem *rc_visard* gespeichert, bis sie entweder gelöscht wird oder eine neue Kalibrierung durchgeführt wird.

Bemerkung: Um Datenschutzproblemen entgegenzuwirken, wird die Visualisierung der Kalibrierung der Basisebene nach einem Neustart des *rc_visard* verschwommen dargestellt.

In Szenarien, in denen die Basisebene nicht direkt kalibriert werden kann, ist es auch möglich, zu einer zur Basisebene parallel liegenden Ebene zu kalibrieren. In diesem Fall kann der Parameter `offset` benutzt werden, um die geschätzte Ebene auf die eigentliche Basisebene zu verschieben. Der Parameter `offset` gibt die Distanz in Metern an, um welche die geschätzte Ebene in Richtung der Kamera verschoben wird.

In der REST-API ist eine Ebene durch eine Normale (`normal`) und einen Abstand (`distance`) definiert. `normal` ist ein normalisierter 3-Vektor, welcher die Normale der Ebene spezifiziert. Die Normale zeigt immer von der Kamera weg. `distance` repräsentiert den Abstand der Ebene von der Kamera in Richtung der Normale. `normal` und `distance` können auch als a, b, c, bzw. d der Ebenengleichung interpretiert werden:

$$ax + by + cz + d = 0$$

AprilTag-basierte Kalibrierung der Basisebene

Die AprilTag-Erkennung (siehe *TagDetect*, Abschnitt 5.4.3) wird benutzt, um AprilTags in der Szene zu finden und eine Ebene durch diese zu legen. Mindestens drei AprilTags müssen so auf der Basisebene platziert werden, dass sie im linken und rechten Kamerabild zu sehen sind. Die AprilTags sollten ein möglichst großes Dreieck aufspannen. Je größer das Dreieck ist, desto höher wird die Genauigkeit der Schätzung der Basisebene. Diese Methode sollte benutzt werden, wenn die Basisebene untexturiert und kein externer Projektor mit Zufallsmuster angeschlossen ist. Diese Kalibriermethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 6.2) als auch über die *rc_visard* Web GUI verfügbar.

Stereo-basierte Kalibrierung der Basisebene

Die 3D-Punktwolke, welche vom Stereo-Matching-Modul berechnet wird, wird benutzt um eine Ebene in den 3D-Punkten zu finden. Die Region of Interest (ROI) sollte für diese Methode deshalb so gewählt werden, dass nur die relevante Basisebene eingeschlossen wird. Der Parameter `plane_preference` erlaubt es auszuwählen, ob die zur Kamera am nächsten gelegene oder die von der Kamera am weitesten entfernte Ebene als Basisebene benutzt wird. Die am nächsten gelegene Ebene kann in Szenarien ausgewählt werden, in denen die Basisebene vollständig von Objekten verdeckt wird oder für die Kalibrierung nicht erreichbar ist. Diese Methode sollte benutzt werden, wenn die Basisebene texturiert ist oder ein Projektor mit Zufallsmuster angeschlossen ist. Diese Kalibriermethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 6.2) als auch über die *rc_visard* Web GUI verfügbar.

Manuelle Kalibrierung der Basisebene

Die Basisebene kann manuell gesetzt werden, falls die Parameter bekannt sind – beispielsweise von einer vorangegangenen Kalibrierung. Diese Kalibriermethode ist nur über die *REST-API-Schnittstelle* (Abschnitt 6.2) und nicht über die *rc_visard* Web GUI verfügbar.

5.4.7.3 Setzen einer Region of Interest

Falls Objekte nur in einem Teil des Sichtfelds der Kamera erkannt werden sollen, kann eine 2D Region of Interest (ROI) gesetzt werden, wie in *Region of Interest* (Abschnitt 5.6.2.2) beschrieben wird.

5.4.7.4 Setzen von Greifpunkten

Um das SilhouetteMatch Modul direkt in einer Roboteranwendung zu nutzen, können für jedes Template bis zu 50 Greifpunkte definiert werden. Ein Greifpunkt repräsentiert die gewünschte Position und Orientierung des Roboter-TCPs (Tool Center Point), mit der das Objekt gegriffen werden kann (siehe Abb. 5.19).

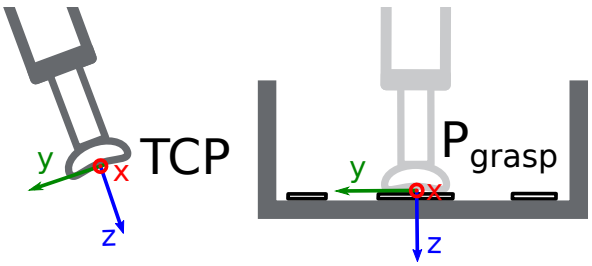


Abb. 5.19: Definition von Greifpunkten bezogen auf den Roboter-TCP

Jeder Greifpunkt enthält eine `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, die ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und die Greifpose (`pose`) im Koordinatensystem des Templates. Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt

6.2), oder über die interaktive Visualisierung in der Web GUI definiert werden. Zudem kann einem Greifpunkt eine Priorität (von -2 für sehr niedrig bis 2 für sehr hoch) zugewiesen werden. Prioritäten können Roboteranwendungen vereinfachen, oder die Rechenzeit der Kollisionsprüfung verkürzen, wenn der Parameter `only_highest_priority_grasp` aktiviert ist. In diesem Fall endet die Kollisionsprüfung, wenn Greifpunkte mit der höchsten Priorität gefunden sind. Weiterhin können Greifpunkte unterschiedlichen Greifern zugewiesen werden, indem die ID des Greifers (`gripper_id`) spezifiziert wird. Dieser Greifer wird dann anstelle des Greifers, welcher im `detect_object` oder `detect_object_extended` Service definiert ist, für die Kollisionsprüfung des zugehörigen Greifpunkts verwendet.

Wenn für einen Greifpunkt eine `gripper_id` angegeben wird, und der zugehörige Greifer Elemente vom Typ (`function_type`) `FINGER` besitzt, kann jeder Greifpunkt auch Werte für `stroke_per_finger_approach_mm` und `stroke_per_finger_grasp_mm` festlegen. Diese Werte geben die Verschiebung eines Fingers in Millimetern an, um die das Fingerelement und alle seine Kind-Elemente von der `zero_pose` zur `pose` des Fingerelements bewegt werden. Der Wert `stroke_per_finger_approach_mm` gibt die Greiferöffnung während der Annäherung an und wird zur Kollisionsprüfung verwendet. Der Wert `stroke_per_finger_grasp_mm` wird nicht für die Kollisionsprüfung verwendet, sondern enthält Informationen über die Greiferöffnung während des Greifens. Dieses Feld definiert somit implizit die Bewegungsrichtung des Fingers beim Greifen. Wenn weder `stroke_per_finger_approach_mm` noch `stroke_per_finger_grasp_mm` angegeben werden, dann wird der Greifer mit den Fingern in der Standardpose zur Kollisionsprüfung verwendet.

Wird ein Greifpunkt auf einem symmetrischen Objekt definiert, werden alle Greifpunkte, die zu diesem symmetrisch sind, automatisch im `detect_object` und `detect_object_extended` Service des SilhouetteMatch Moduls mit berücksichtigt. Symmetrische Greifpunkte zu einem gegebenen Greifpunkt können mittels des `get_symmetric_grasps` Services abgefragt werden und in der Web GUI visualisiert werden.

Benutzer können ebenfalls Replikationen eines Greifpunktes um eine selbst-definierte Achse definieren. Eine Replikation generiert mehrere Greifpunkte und sorgt dafür, dass Benutzer nicht zu viele Greifpunkte manuell setzen müssen. Der Ursprung der Replikation ist als Koordinatensystem im Objektkoordinatensystem definiert und die x-Achse dieses Koordinatensystems entspricht der Replikationsachse. Der Greifpunkt wird repliziert, indem er ausgehend von seiner ursprünglichen Pose um diese x-Achse gedreht wird. Die Replikation erfolgt in `step_x_deg`-Grad Schritten. Der Bereich wird durch die minimalen und maximalen Endpunkte `min_x_deg` und `max_x_deg` bestimmt. Der minimale (maximale) Endpunkt muss nicht-positiv (nicht-negativ) sein.

Setzen von Greifpunkten in der Web GUI

Die *rc_visard* Web GUI bietet eine interaktive und intuitive Möglichkeit, Greifpunkte für Objekt-Templates zu setzen. Im ersten Schritt muss das Objekt-Template auf den *rc_visard* hochgeladen werden. Das kann über die Web GUI in einer beliebigen Kamerapipeline unter *Module* → *SilhouetteMatch* erfolgen, indem im Abschnitt *Templates und Greifpunkte* auf *+ Neues Template hinzufügen* geklickt wird, oder unter *Datenbank* → *Templates* im Abschnitt *SilhouetteMatch Templates und Greifpunkte*. Wenn der Upload abgeschlossen ist, erscheint ein Fenster mit einer 3D-Visualisierung des Templates, in dem Greifpunkte hinzugefügt oder existierende Greifpunkte bearbeitet werden können. Dasselbe Fenster erscheint, wenn ein vorhandenes Template bearbeitet wird. Wenn das Template ein Kollisionsmodell oder ein Visualisierungsmodell enthält, wird dieses Modell ebenfalls angezeigt.

Dieses Fenster bietet zwei Möglichkeiten, um Greifpunkte zu setzen:

1. **Greifpunkte manuell hinzufügen**: Durch Klicken auf das *+* Symbol wird ein neuer Greifpunkt im Ursprung des Templates angelegt. Diesem Greifpunkt kann ein eindeutiger Name gegeben werden, der seiner ID entspricht. Die gewünschte Pose des Greifpunkts im Koordinatensystem des Templates kann in den Feldern für *Position* und *Roll/Pitch/Yaw* eingegeben werden. Die Greifpunkte können frei platziert werden, auch außerhalb oder innerhalb des Templates, und werden mit ihrer Orientierung zur Überprüfung in der Visualisierung veranschaulicht.
2. **Greifpunkte interaktiv hinzufügen**: Greifpunkte können interaktiv zu einem Template hinzugefügt werden, indem zuerst auf den Button *Greifpunkt hinzufügen* oben rechts in der Visualisierung und anschließend auf den gewünschten Punkt auf dem Template geklickt wird. Wenn ein 3D-Modell angezeigt wird, wird der Greifpunkt an die Oberfläche des Modells angeheftet, andernfalls

an die Template-Oberfläche. Die Orientierung des Greifpunkts entspricht einem rechtshändigen Koordinatensystem, sodass die z-Achse senkrecht auf der Template-Oberfläche steht und in das Template hinein gerichtet ist. Die Position und Orientierung des Greifpunkts im Koordinatensystem des Templates ist auf der rechten Seite angezeigt. Die Position und Orientierung des Greifpunkts kann auch interaktiv verändert werden. Für den Fall, dass *An Oberfläche anheften* in der Visualisierung deaktiviert ist (das ist der Standardwert), kann der Greifpunkt in allen drei Dimensionen frei verschoben und gedreht werden, indem in der Visualisierung auf *Greifpunkt bewegen* geklickt wird und der Greifpunkt dann entlang der Achse zur gewünschten Position verschoben wird. Die Orientierung des Greifpunkts kann ebenfalls interaktiv verändert werden, indem die Achse mit der Maus rotiert wird. Wenn *An Oberfläche anheften* nicht aktiv ist, kann der Greifpunkt nur auf der Objektoberfläche verschoben und rotiert werden.

Benutzer können auch eine Greifpunktpriorität festlegen, indem sie den Schieberegler *Priorität* ändern. Ein dedizierter Greifer kann im Dropdown-Feld *Greifer* ausgewählt werden.

Durch Aktivieren des Kontrollkästchens *Replizieren* können Benutzer den Greifpunkt um eine benutzerdefinierte Achse replizieren. Die Replikationsachse und die generierten Greifpunkte werden visualisiert. Die Lage und Ausrichtung der Replikationsachse relativ zum Objektkoordinatensystem kann interaktiv angepasst werden, indem im Visualisierungsmenü auf *Replikationsachse bewegen* geklickt und die Achse an die gewünschte Position und Ausrichtung gezogen wird. Die Greifpunkte werden innerhalb des angegebenen Drehbereichs mit der ausgewählten Schrittgröße repliziert. Benutzer können eine Visualisierung die replizierten Greifpunkte durchlaufen, indem sie die Leiste unter *Durchlaufen n repl. Greifpunkte* im Abschnitt *Ansichtsoptionen* des Visualisierungsmenüs ziehen. Wenn für den Greifpunkt ein Greifer ausgewählt ist oder im Visualisierungsmenü ein Greifer ausgewählt wurde, wird der Greifer auch am aktuell ausgewählten Greifpunkt angezeigt.

Wenn das Template Symmetrien hat, können die Greifpunkte, die symmetrisch zum definierten Greifpunkt sind, zusammen mit ihren Replikationen (sofern definiert) durch Aktivieren von *. . . Symmetrien* im Abschnitt *Ansichtsoptionen* des Visualisierungsmenüs angezeigt werden. Visualisierungen der symmetrischen Greifpunkte können ebenfalls durchlaufen werden, indem die Leiste unter *Durchlaufe n symm. Greifpunkte* bewegt wird.

Setzen von Greifpunkten über die REST-API

Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 6.2) mithilfe des `set_grasp` oder `set_all_grasps` Services gesetzt werden (siehe *Interne Services*, Abschnitt 5.4.7.12). Ein Greifpunkt besteht aus der `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, der ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und der Greifpose (`pose`). Die Pose ist im Koordinatensystem des Templates angegeben und besteht aus einer Position (`position`) in Metern und einer Orientierung (`orientation`) als Quaternion. Ein dedizierter Greifer kann durch Setzen des Feldes `gripper_id` angegeben werden. Die `priority` wird durch einen ganzzahligen Wert angegeben, der von -2 für sehr niedrig bis 2 für sehr hoch reicht. Der Replikationsursprung ist als eine Transformation im Koordinatensystem des Objekts definiert und die x-Achse der Transformation entspricht der Replikationsachse. Der Replikationsbereich wird durch die Felder `min_x_deg` und `max_x_deg` und die Schrittweite `step_x_deg` gesteuert.

5.4.7.5 Setzen der bevorzugten TCP-Orientierung

Das SilhouetteMatch Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die *SilhouetteMatch*-Seite in der Web GUI gesetzt werden. Die resultierende Richtung der z-Achse des TCP wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann. Weiterhin kann die bevorzugte Orientierung genutzt werden, um die erreichbaren Greifpunkte zu sortieren, indem die entsprechende Sortierstrategie ausgewählt wird.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und die Kamera am Roboter montiert ist,

muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Koordinatensysteme) als die bevorzugte TCP-Orientierung genutzt.

5.4.7.6 Setzen der Sortierstrategie

Die vom `detect_object` und `detect_object_extended` Service zurückgelieferten Objekte und Greifpunkte werden gemäß einer Sortierstrategie sortiert, die vom Nutzer gewählt werden kann. Folgende Sortierstrategien sind verfügbar und können über die *Web GUI* (Abschnitt 6.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `preferred_orientation`: Matches und Greifpunkte mit der geringsten Rotationsänderung einer gewählten Achse (`axis`), oder aller Achsen, wenn `axis` leer ist, bezogen auf die bevorzugte TCP-Orientierung werden zuerst zurückgeliefert.
- `direction`: Objekte und Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Richtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.
- `distance_to_point`: Objekte und Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `preferred_orientation` und dem kleinsten Abstand entlang der z-Achse der bevorzugten TCP-Orientierung von der Kamera.

5.4.7.7 Objekterkennung

Um eine Objekterkennung durchzuführen, müssen im Allgemeinen die folgenden Serviceargumente an das SilhouetteMatch Modul übergeben werden:

- das Template des Objekts, welches in der Szene erkannt werden soll
- das Koordinatensystem, in dem die Posen der detektierten Objekte zurückgegeben werden sollen (siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.4.7.8)

Optional können auch folgende Serviceargumente an das SilhouetteMatch Modul übergeben werden:

- Ein Flag `object_plane_detection`, welches bestimmt, ob die Oberflächenebene der Objekte für die Erkennung verwendet werden soll anstelle einer kalibrierten Basisebene.
- ein Versatz `offset`, falls Objekte nicht direkt auf der Basisebene liegen, sondern auf einer zu dieser parallelen Ebene. Der Versatz bezeichnet die Distanz beider Ebenen in Richtung der Kamera. Wenn dieser Wert nicht gesetzt wird, wird ein Versatz von 0 angenommen. Der Versatz darf nicht gesetzt werden, wenn `object_plane_detection` true ist.
- die ID des Load Carriers, der die zu detektierenden Objekte enthält
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, innerhalb der Objekte erkannt werden sollen. Wenn keine ROI gesetzt wird, werden Objekte im gesamten Kamerabild gesucht.
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die bevorzugte TCP-Orientierung im externen Koordinatensystem angegeben ist
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 5.4.7.8) gegeben.

Wenn `object_plane_detection` nicht true ist, können Objekte erst nach einer erfolgreichen Kalibrierung der Basisebene erkannt werden. Es muss sichergestellt werden, dass sich Position und Orientierung

der Basisebene zwischen Kalibrierung und Objekterkennung nicht ändern. Anderenfalls muss die Kalibrierung erneuert werden.

Wenn `object_plane_detection` auf true gesetzt ist, ist eine Kalibrierung der Basisebene nicht nötig und eine ggf. existierende Kalibrierung wird ignoriert. Während der Erkennung wird die Szene in planare Flächen unterteilt und das Matching der Templatekanten wird für jede dieser Ebenen durchgeführt, solange sie nicht mehr als 25° in Bezug auf die Sichtachse der Kamera verkippt ist, und solange ihre Größe ausreichend ist für das gewählte Template. Wenn ein Match gefunden wird, wird dessen Position und Orientierung durch Kanten im Kamerabild und durch die Punktwolke innerhalb der Außenkontur des Templates verfeinert. Aus diesem Grund muss die Außenkontur des Templates geschlossen und die Oberfläche des Objekts planar sein.

Im *Ausprobieren*-Abschnitt der Seite *SilhouetteMatch* der Web GUI kann die Objektdetektion ausprobiert werden. Verschiedene Bild-Streams können ausgewählt werden, um Zwischenergebnisse und die finalen Matches anzuzeigen.

> Das **„Template“** Bild zeigt das zu erkennende Template in Grün mit den Greifpunkten (siehe *Setzen von Greifpunkten*, Abschnitt 5.4.7.4) in Grün. Das Template wird verformt dargestellt, passend zu Abstand und Verkippung der kalibrierten Basisebene, oder - falls `object_plane_detection` auf true gesetzt war, der höchsten erkannten Ebene. Die entsprechende Ebene ist in Dunkelblau dargestellt.
>
> Das **„Zwischenergebnis“** Bild zeigt die Kanten im linken Bild, die für die Suche nach Matches verwendet wurden, in Hellblau. Die gewählte Region of Interest wird als petrolfarbenes Rechteck dargestellt. Eine blau schattierte Fläche auf der linken Seite markiert den Teil des linken Kamerabilds, welcher nicht mit dem rechten Kamerabild überlappt. In diesem Bereich können keine Objekte erkannt werden. Wenn die Objektebenenerkennung verwendet wurde (`object_plane_detection` ist true), zeigt dieses Bild auch die erkannten planaren Cluster in der Szene. Cluster, die nicht für das Matching verwendet wurden, weil sie zu klein oder zu stark geneigt sind, werden mit einem Streifenmuster dargestellt.
>
> Das **„Zwischenergebnis rechts“** Bild zeigt die Kanten im rechten Bild, die für die Suche nach Matches verwendet wurden, in Hellblau. Die gewählte Region of Interest wird als petrolfarbenes Rechteck dargestellt. Eine blau schattierte Fläche auf der rechten Seite markiert den Teil des rechten Kamerabilds, welcher nicht mit dem linken Kamerabild überlappt. In diesem Bereich können keine Objekte erkannt werden.

Die Posen der Objektursprünge werden im gewählten Koordinatensystem als Liste (`instances`) zurückgegeben. Falls die kalibrierte Basisebene für die Erkennung genutzt wurde (`object_plane_detection` nicht oder false gesetzt), wird die Orientierung der erkannten Objekte mit mit der Normalen der Basisebene ausgerichtet. Andernfalls ist die Orientierung der Objekte an der Normalen der Ebene ausgerichtet, die in die zugehörigen Objektpunkte in der 3D Punktwolke eingepasst wurde.

Wenn das ausgewählte Template auch Greifpunkte hat, dann wird zusätzlich zu den erkannten Objekten auch eine Liste von Greifpunkten (`grasps`) für alle erkannten Objekte zurückgegeben. Die Posen der Greifpunkte sind im gewünschten Koordinatensystem angegeben und die Liste ist gemäß der gewählten Sortierstrategie sortiert (siehe *Setzen der Sortierstrategie*, Abschnitt 5.4.7.6). Die erkannten Objekte und die Greifpunkte können über ihre UUIDs einander zugeordnet werden.

Falls das Template eine kontinuierliche Rotationssymmetrie aufweist (z.B. zylindrische Objekte), besitzen alle Ergebnisposen die gleiche Orientierung. Weiterhin werden alle Symmetrien eines Greifpunkts auf Erreichbarkeit und Kollisionsfreiheit geprüft, und anschließend nur der jeweilige beste gemäß der gewählten Sortierstrategie zurückgeliefert.

Für Objekte mit einer diskreten Symmetrie (z.B. prismatische Objekte), werden alle kollisionsfreien Symmetrien jedes Greifpunkts, die entsprechend der gesetzten bevorzugten TCP-Orientierung erreichbar sind, zurückgeliefert, und gemäß der gewählten Sortierstrategie sortiert.

Die Detektionsergebnisse und Berechnungszeiten werden durch Laufzeitparameter beeinflusst, welche weiter unten aufgezählt und beschrieben werden. Unsachgemäße Parameterwerte können zu Zeitüberschreitungen im Detektionsprozess des SilhouetteMatch Moduls führen.

5.4.7.8 Wechselwirkung mit anderen Modulen

Die folgenden auf dem *rc_visard* laufenden Module liefern Daten für das SilhouetteMatch Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des SilhouetteMatch Moduls haben.

Kamera- und Tiefendaten

Das SilhouetteMatch Modul verarbeitet intern die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 5.1). Es sollte deshalb auf eine passende Belichtungszeit geachtet werden, um optimale Ergebnisse zu erhalten.

Für die Kalibrierung der Basisebene mit der Stereo-Methode, für die Load Carrier Erkennung, für die automatische Objektebenenerkennung und für die Kollisionsprüfung mit der Punktwolke wird das Disparitätsbild des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 5.2) verarbeitet.

Für das Erkennen von Objekten mit einer kalibrierten Basisebene, ohne Load Carrier und ohne Kollisionsprüfung mit der Punktwolke sollte das Stereo-Matching-Modul nicht parallel zum SilhouetteMatch Modul ausgeführt werden, da die Laufzeit der Objekterkennung sonst negativ beeinflusst wird.

Für beste Ergebnisse wird empfohlen, *Glättung* (Abschnitt 5.2.1.2) für das *Stereo-Matching Modul* zu aktivieren.

IOControl und Projektor-Kontrolle

Wenn der *rc_visard* in Verbindung mit einem externen Musterprojektor und dem Modul *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.5.4) betrieben wird, sollte der Projektor für die stereobasierte Kalibrierung der Basisebene, für die automatische Objektebenenerkennung und für die Kollisionsprüfung mit der Punktwolke benutzt werden.

Das projizierte Muster darf während der Objektdetektion nicht im linken oder rechten Kamerabild sichtbar sein, da es den Detektionsvorgang behindert. Daher wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.5.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Wenn die Kamera zu einem Roboter kalibriert ist, kann das SilhouetteMatch Modul die Ergebnisposen automatisch im Roboterkoordinatensystem liefern. Für die *Services* (Abschnitt 5.4.7.11) des SilhouetteMatch Moduls kann das Referenzkoordinatensystem aller Posen über das Argument `pose_frame` angegeben werden.

Es kann zwischen den folgenden zwei Werten für `pose_frame` gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen und Ebenenparameter werden im Kamera-Koordinatensystem angegeben.
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen und Ebenenparameter sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das SilhouetteMatch

Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom internen Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Bemerkung: Wurde keine Hand-Auge-Kalibrierung durchgeführt, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Bemerkung: Wird die Hand-Auge-Kalibrierung nach einer Kalibrierung der Basisebene verändert, wird die Kalibrierung der Basisebene als ungültig markiert und muss erneuert werden.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame`, der bevorzugten TCP-Orientierung und der Sortierrichtung bzw. des Sortierpunktes nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die bevorzugte TCP-Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

Wenn die aktuelle Roboterpose während der Kalibrierung der Basisebene angegeben wird, wird sie persistent auf dem *rc_visard* gespeichert. Falls für die Services `get_base_plane_calibration`, `detect_objects` oder `detect_object_extended` die dann aktuelle Roboterpose ebenfalls angegeben wird, wird die Basisebene automatisch zu der neuen Roboterpose transformiert. Das erlaubt dem Benutzer, die Roboterpose (und damit die Pose der Kamera) zwischen Kalibrierung der Basisebene und Objekterkennung zu verändern.

Bemerkung: Eine Objekterkennung kann nur durchgeführt werden, wenn die Verkippung der Basisebene zur Sichtachse der Kamera ein 10-Grad-Limit nicht übersteigt.

LoadCarrier

Das SilhouetteMatch Modul nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 5.4.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das SilhouetteMatch Modul verwendet wird.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des SilhouetteMatch Moduls aktiviert werden, indem das `collision_detection` Argument an den `detect_object` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 5.6.4.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.5.2.2) gegeben.

Alternativ können Greifpunkten individuell Greifer IDs zugewiesen werden, und die Kollisionsprüfung kann für alle Greifpunkte mit einer Greifer ID über den Laufzeitparameter `check_collisions` eingeschaltet werden.

Zusätzlich wird auf Kollisionen zwischen dem Greifer und der kalibrierten Basisebene geprüft, wenn der Laufzeitparameter `check_collisions_with_base_plane` auf true gesetzt ist. Wenn das ausgewählte Template ein Kollisionsmodell enthält und der Laufzeitparameter `check_collisions_with_matches` true ist, wird außerdem auch auf Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht begrenzt auf die Anzahl `max_number_of_detected_objects`) geprüft, wobei das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen ist.

Wenn der Laufzeitparameter `check_collisions_with_point_cloud` auf true gesetzt ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Wenn diese Funktionalität in Kombination mit Sauggreifern genutzt wird, muss sichergestellt werden, dass sich der TCP außerhalb der Greifergeometrie befindet, oder dass die Greifpunkte oberhalb der Objektoberfläche definiert sind. Andernfalls wird für jeden Greifpunkt eine Kollision zwischen Greifer und Punktwolke erkannt.

Wenn der Laufzeitparameter `check_collisions_during_retraction` auf true gesetzt ist, und ein Load Carrier sowie ein Greif-Offset angegeben wurden, wird jeder Greifpunkt auf Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme geprüft. Die Prüfung findet auf der gesamten linearen Trajektorie von der Greifposition bis zurück zur Vorgreifposition statt.

Wenn die Kollisionsprüfung aktiviert ist, werden nur Greifpunkte zurückgeliefert, die kollisionsfrei sind, oder die nicht auf Kollisionen geprüft werden konnten (z.B. weil kein Greifer angegeben wurde). In der Ergebnis-Visualisierung oben auf der *SilhouetteMatch*-Seite der Web GUI werden kollisionsfreie Greifpunkte grün dargestellt, ungeprüfte Greifpunkte gelb und kollidierende Greifpunkte rot. Die erkannten Objekte, die bei der Kollisionsprüfung betrachtet werden, werden mit grünen Kanten visualisiert.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 5.5.2.3) beschrieben.

5.4.7.9 Parameter

Das SilhouetteMatch Modul wird in der REST-API als `rc_silhouettematch` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Module* → *SilhouetteMatch* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.39: Laufzeitparameter des `rc_silhouettematch`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`check_collisions`	bool	false	true	false	Gibt an, ob Kollisionen geprüft werden sollen, wenn ein Greifer für einen Greifpunkt definiert wurde
`check_collisions_during_retraction`	bool	false	true	false	Gibt an, ob Kollisionen zwischen dem Objekt im Greifer und dem Load Carrier während der Entnahme geprüft werden
`check_collisions_with_base_plane`	bool	false	true	true	Gibt an, ob Kollisionen zwischen Greifer und der Basisebene geprüft werden
`check_collisions_with_matches`	bool	false	true	true	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
`check_collisions_with_point_cloud`	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und der Punktwolke geprüft werden
`edge_sensitivity`	float64	0.1	1.0	0.7	Empfindlichkeit der Kantenerkennung
`match_max_distance`	float64	0.1	10.0	3.0	Der maximale tolerierte Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln
`match_percentile`	float64	0.7	1.0	0.8	Der Anteil der Template-Pixel, die innerhalb der maximalen Matchingdistanz liegen müssen, um ein Objekt erfolgreich zu detektieren
`max_number_of_detected_objects`	int32	1	20	10	Maximale Anzahl der zu detektierenden Objekte
`max_object_overlap`	float64	0.0	1.0	0.05	Maximaler Anteil der Objektoberfläche, der von anderen segmentierten Objekten überlappt werden darf
`only_highest_priority_grasps`	bool	false	true	false	Falls aktiviert werden nur Greifpunkte der höchsten Priorität zurückgegeben.
`point_cloud_enhancement`	string	-	-	Off	Art der Verbesserung der Punktwolke mit der Basisebene: [Off, ReplaceBright]
`quality`	string	-	-	High	Quality: [Low, Medium, High]

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der SilhouetteMatch Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_number_of_detected_objects` (*Maximale Objektanzahl*)

Dieser Parameter gibt an, wie viele Objekte maximal in der Szene erkannt werden sollen. Falls mehr als die angegebene Zahl an Objekten gefunden wurden, werden nur die am besten zur gewählten Sortierstrategie passenden Ergebnisse zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪max_number_of_detected_objects=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?max_number_of_detected_
↪objects=<value>
```

`quality` (*Qualität*)

Die Objekterkennung kann auf Bildern mit unterschiedlicher Auflösung durchgeführt werden: `High` (*Hoch*, volle Auflösung), `Medium` (*Mittel*, halbe Auflösung) oder `Low` (*Niedrig*, Viertel-Auflösung). Je niedriger die Auflösung ist, desto niedriger ist die Berechnungszeit der Objekterkennung, aber desto weniger Objektdetails sind erkennbar.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪quality=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?quality=<value>
```

`match_max_distance` (*Maximale Matchingdistanz*)

Dieser Parameter gibt den maximal tolerierten Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln an. Falls das Objekt durch das Template nicht exakt genug beschrieben wird, wird es möglicherweise nicht erkannt, wenn dieser Wert zu klein ist. Höhere Werte können jedoch im Fall von komplexen Szenen und bei ähnlichen Objekten zu Fehldetektionen führen, und auch die Berechnungszeit erhöhen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪match_max_distance=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_max_distance=<value>
```

`match_percentile` (*Matching Perzentil*)

Dieser Parameter kontrolliert, wie strikt der Detektionsprozess sein soll. Das Matching Perzentil gibt den Anteil der Template-Pixel an, die innerhalb der maximalen Matchingdistanz liegen müssen, um ein Objekt erfolgreich zu detektieren. Je höher der Wert, desto exakter muss ein Match sein, um als gültig angesehen zu werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪match_percentile=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_percentile=<value>
```

`edge_sensitivity` (*Kantenempfindlichkeit*)

Der Parameter beeinflusst, wie viele Kanten im linken und rechten Kamerabild gefunden werden. Umso größer dieser Parameter gewählt wird, umso mehr Kanten werden für die Erkennung benutzt. Eine große Anzahl von Kanten im Bild kann die Erkennung verlangsamen. Es muss sichergestellt werden, dass die Kanten der zu erkennenden Objekte sowohl im linken als auch im rechten Kamerabild detektiert werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪edge_sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?edge_sensitivity=<value>
```

`only_highest_priority_grasps` (*Nur Greifpunkte höchster Priorität*)

Wenn dieser Parameter auf *true* gesetzt ist, werden ausschließlich Greifpunkte der höchsten Priorität zurückgegeben. Sofern die Kollisionsprüfung aktiviert ist, werden ausschließlich kollisionsfreie Greifpunkt der höchstmöglichen Priorität zurückgegeben. Dadurch kann Rechenzeit gespart und die Anzahl der applikationsseitig zu verarbeitenden Greifpunkte reduziert werden.

Ohne Kollisionsprüfung werden nur Greifpunkt der höchsten Priorität zurückgegeben.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?only_
↪highest_priority_grasps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?only_highest_priority_
↪grasps=<value>
```

`check_collisions` (*Kollisionsprüfung*)

Wenn diese Option aktiv ist, wird die Kollisionsprüfung für alle Greifpunkte durchgeführt, denen eine Greifer ID zugewiesen wurde, auch wenn kein Standardgreifer im `detect_object` Service gesetzt wurde. Wenn ein Load Carrier verwendet wird, wird die Kollisionsprüfung immer zwischen dem Greifer und dem Load Carrier durchgeführt. Kollisionen mit der Punktwolke oder anderen Matches werden nur geprüft, wenn die zugehörigen Laufzeitparameter aktiv sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions=<value>
```

`check_collisions_with_base_plane` (*Kollisionsprüfung mit Basisebene*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_base_plane` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und der kalibrierten Basisebene geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit der Basisebene wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪check_collisions_with_base_plane=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪base_plane=<value>
```

`check_collisions_with_matches` (*Kollisionsprüfung mit Matches*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_matches` auf true gesetzt ist und die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service aktiviert ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht begrenzt auf die Anzahl `max_number_of_detected_objects`) geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit anderen detektierten Objekten wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/parameters/parameters?
↪check_collisions_with_matches=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪matches=<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪point_cloud=<value>
```

`point_cloud_enhancement` (*Verbesserung mit Basisebene*)

Dieser Parameter wird nur beachtet, wenn `check_collisions_with_point_cloud` auf true gesetzt ist und die Detektion ohne Objektebenenerkennung (`object_plane_detection` ist false) ausgelöst wurde. Standardmäßig ist `point_cloud_enhancement` auf `Off` (*Aus*) gesetzt. Wenn `point_cloud_enhancement` auf `ReplaceBright` (*Helle Bildpunkte ersetzen*) gesetzt wird, wird die kalibrierte Basisebene verwendet, um die Punktwolke für die Kollisionsprüfung zu verbessern. Dazu werden Punkte, die zu hellen Pixeln im Bild oder in der gewählten 2D Region of Interest gehören, auf den Tiefenwert der kalibrierten Basisebene gesetzt. Dieser Parameter sollte genutzt werden, wenn dunkle Objekten auf texturlosem, hellem Untergrund liegen, z.B. auf einem Lichttisch.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?point_
↪cloud_enhancement=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?point_cloud_enhancement=
↪<value>
```

`check_collisions_during_retraction` (*Kollisionsprüfung während Entnahme*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_during_retraction` auf true gesetzt ist und ein Load Carrier sowie ein Greif-Offset angegeben wurden, wird jeder Greifpunkt auf Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme geprüft. Die Prüfung findet auf der gesamten linearen Trajektorie von der Greifposition bis zurück zur Vorgreifposition statt. Es werden nur kollisionsfreie Greifpunkte zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions_during_retraction=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_during_
↪retraction=<value>
```

5.4.7.10 Statuswerte

Dieses Modul meldet folgende Statuswerte.

Tab. 5.40: Statuswerte des `rc_silhouettematch`-Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_return_code_message`	Rückgabecode-Nachricht der letzten Detektion
`last_return_code_value`	Rückgabecode-Wert der letzten Detektion
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden
`processing_time`	Berechnungszeit für die letzte Detektion (einschließlich Load Carrier Detektion) in Sekunden

5.4.7.11 Services

Die angebotenen Services des `rc_silhouettematch`-Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

Das SilhouetteMatch Modul bietet folgende Services.

`detect_object`

führt eine Objekterkennung durch, wie in *Objekterkennung* (Abschnitt 5.4.7.7) beschrieben. Der Service gibt die Posen aller gefundenen Objektinstanzen zurück.

Details

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/detect_object
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/detect_object
```

Request

Obligatorische Serviceargumente:

`object_id` in `object_to_detect`: ID des Templates, welches erkannt werden soll.

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.7.8).

Potentiell obligatorische Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.7.8).

Optionale Serviceargumente:

`object_plane_detection`: false wenn Objekte auf einer kalibrierten Basisebene liegen, true wenn die Objekte planare Oberflächen haben und die Basisebene unbekannt ist oder die Objekte auf mehreren verschiedenen Ebenen liegen, z.B. auf Stapeln.

`offset`: Versatz in Metern, um welche die Basisebene in Richtung der Kamera verschoben werden soll.

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.5.2.2)

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "load_carrier_id": "string",
    "object_plane_detection": "bool",
    "object_segmentation_model": "string",
    "object_to_detect": {
      "object_id": "string",
      "region_of_interest_2d_id": "string"
    },
    "offset": "float64",
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

Die maximale Anzahl der zurückgegebenen Instanzen kann über den `max_number_of_detected_objects`-Parameter kontrolliert werden.

`object_id`: ID des erkannten Templates.

`instances`: Liste der erkannten Objektinstanzen, sortiert gemäß der gewählten Sortierstrategie.

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind gemäß der gewählten Sortierstrategie sortiert. Die `instance_uuid` gibt eine Referenz auf das detektierte Objekt in `instances` an, zu dem dieser Greifpunkt gehört. Die Liste der Greifpunkte wird

auf die 100 besten Greifpunkte gekürzt, falls mehr erreichbare Greifpunkte gefunden werden. Jeder Greifpunkt enthält ein Flag `collision_checked` und das Feld `gripper_id` (siehe *Integrierte Kollisionsprüfung in anderen Modulen* Abschnitt 5.5.2.2).

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, das für die Erkennung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_object",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
        "id": "string",
        "instance_uuid": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "priority": "int8",
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "tcp_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "instances": [
      {
        "grasp_uuids": [
          "string"
        ],
        "id": "string",
        "object_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        }
      },
      "pose_frame": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  ],
  "object_id": "string",
  "return_code": {
    "message": "string",
    "value": "int16"
  },
  "timestamp": {
    "nsec": "int32",
    "sec": "int32"
  }
}
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}
```

detect_object_extended

führt eine Objekterkennung durch. Dieser Service verhält sich analog zu `detect_object`, gibt aber die Instanzinformationen für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn z.B. die Objektposen für jeden Greifpunkt benötigt werden, um das Objekt platziert abzulegen.

Details

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/detect_object_
↪extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/detect_object_extended
```

Request

Siehe `detect_object` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "load_carrier_id": "string",
    "object_plane_detection": "bool",
    "object_segmentation_model": "string",
    "object_to_detect": {
      "object_id": "string",
      "region_of_interest_2d_id": "string"
    },
    "offset": "float64",
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      }
    }
  }
}
```

Response

Die maximale Anzahl der zurückgegebenen Instanzen kann über den `max_number_of_detected_objects`-Parameter kontrolliert werden.

`object_id`: ID des erkannten Templates.

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind gemäß der gewählten Sortierstrategie sortiert. Jeder Greifpunkt enthält ein Feld `instance` mit Informationen zum detektierten Objekt, z.B. seiner Pose. Die Liste der Greifpunkte wird auf die 100 besten Greifpunkte gekürzt, falls mehr erreichbare Greifpunkte gefunden werden. Jeder Greifpunkt enthält ein Flag `collision_checked` und das Feld `gripper_id` (siehe *Integrierte Kollisionsprüfung in anderen Modulen* Abschnitt 5.5.2.2).

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, das für die Erkennung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_object_extended",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
        "id": "string",
        "instance": {
          "object_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "pose_frame": "string",
          "uuid": "string"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "priority": "int8",
      "stroke_per_finger_approach_mm": "float64",
      "stroke_per_finger_grasp_mm": "float64",
      "tcp_id": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  ],
  "object_id": "string",
  "return_code": {
    "message": "string",
    "value": "int16"
  },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

calibrate_base_plane

führt die Kalibrierung der Basisebene durch, wie in *Kalibrierung der Basisebene* (Abschnitt 5.4.7.2) beschrieben.

Details

Eine erfolgreiche Kalibrierung der Basisebene wird persistent auf dem *rc_visard* gespeichert und vom Service zurückgegeben. Die Kalibrierung ist dauerhaft – auch über Firmware-Updates und -Wiederherstellungen hinweg – gespeichert.

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/calibrate_base_
↪plane
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/calibrate_base_plane
```

Request

Obligatorische Serviceargumente:

`plane_estimation_method`: Methode der Kalibrierung der Basisebene. Gültige Werte sind `STEREO`, `APRILTAG`, `MANUAL`.

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.7.8).

Potentiell obligatorische Serviceargumente:

`plane` wenn für `plane_estimation_method` `MANUAL` gewählt ist: Die Ebene, welche als Basisebene gesetzt wird.

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.7.8).

`region_of_interest_2d_id`: ID der Region of Interest für die Kalibrierung der Basisebene.

Optionale Serviceargumente:

`offset`: Versatz in Metern, um welchen die geschätzte Ebene in Richtung der Kamera verschoben wird.

`plane_preference` in `stereo`: Ob die der Kamera am nächsten (`CLOSEST`) gelegene oder die am weitesten entfernte (`FARTHEST`) Ebene als Basisebene benutzt wird. Diese Option kann nur gesetzt werden, falls `plane_estimation_method` auf `STEREO` gesetzt ist. Valide Werte sind `CLOSEST` und `FARTHEST`. Falls der Wert nicht gesetzt ist, wird `FARTHEST` verwendet.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "offset": "float64",
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "plane_estimation_method": "string",
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "stereo": {
      "plane_preference": "string"
    }
  }
}
```


Response

`plane`: kalibrierte Basisebene.

`timestamp`: Zeitstempel des Bildes, das für die Kalibrierung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "calibrate_base_plane",
  "response": {
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

get_base_plane_calibration

gibt die derzeitige Kalibrierung der Basisebene zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/get_base_plane_
↪calibration
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_base_plane_calibration
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.7.8).

Potentiell obligatorische Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.4.7.8).

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_base_plane_calibration",
  "response": {
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_base_plane_calibration

löscht die derzeitige Kalibrierung der Basisebene.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/delete_base_
↪plane_calibration
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_base_plane_
↪calibration
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "delete_base_plane_calibration",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.7.5).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/set_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`get_preferred_orientation`

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.4.7.5).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/get_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_sorting_strategies

speichert die gewählte Strategie zur Sortierung der erkannten Objekte und Greifpunkte, die vom `detect_object` und `detect_object_extended` Service zurückgeliefert werden (siehe *Objekterkennung*, Abschnitt 5.4.7.7).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/set_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `preferred_orientation` gesetzt ist, kann `axis` auf `x`, `y` oder `z` gesetzt werden, um nur Rotationsunterschiede zwischen diesen Achsen zu berücksichtigen. Wenn `axis` nicht gesetzt wird, wird die volle Rotationsdifferenz zur Sortierung verwendet.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "preferred_orientation": {
      "axis": "string",
      "weight": "float64"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_sorting_strategies

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Objekte und Greifpunkte verwendet wird (siehe *Objekterkennung*, Abschnitt 5.4.7.7).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/get_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_sorting_strategies",
  "response": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "preferred_orientation": {
      "axis": "string",
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


reset_defaults

stellt die Werkseinstellungen der Parameter und die bevorzugte TCP-Orientierung sowie die Sortierstrategie dieses Moduls wieder her und wendet sie an („factory reset“). Dies betrifft nicht die konfigurierten Templates und die Kalibrierung der Basisebene.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "reset_defaults",
  "response": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


5.4.7.12 Interne Services

Die folgenden Services für die Konfiguration von Greifpunkten können sich in Zukunft ohne weitere Ankündigung ändern. Es wird empfohlen, das Setzen, Abrufen und Löschen von Greifpunkten über die Web GUI vorzunehmen.

`set_grasp`

speichert einen Greifpunkt für das angegebene Template auf dem *rc_visard*. Alle Greifpunkte sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/set_grasp
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_grasp
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.4.7.4) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasp": {
      "gripper_id": "string",
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "priority": "int8",
      "replication": {
        "max_x_deg": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "min_x_deg": "float64",
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "stroke_per_finger_approach_mm": "float64",
      "stroke_per_finger_grasp_mm": "float64",
      "tcp_id": "string",
      "template_id": "string"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_grasp",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_all_grasps

Ersetzt die gesamte Liste von Greifpunkten auf dem *rc_visard* für das angegebene Template.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/set_all_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_all_grasps
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.4.7.4) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "priority": "int8",
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "tcp_id": "string",
        "template_id": "string"
      }
    ],
    "template_id": "string"
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_all_grasps",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_grasps

gibt alle definierten Greifpunkte mit den angegebenen IDs (`grasp_ids`) zurück, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/get_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_grasps
```

Request

Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte zu den angegebenen `template_ids` zurückgeliefert. Wenn keine `template_ids` angegeben werden, werden alle Greifpunkte mit den geforderten `grasp_ids` zurückgeliefert. Wenn gar keine IDs angegeben werden, werden alle gespeicherten Greifpunkte zurückgeliefert.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_grasps",
  "response": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "priority": "int8",
        "replication": {
          "max_x_deg": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "tcp_id": "string",
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_grasps

löscht alle Greifpunkte mit den angegebenen `grasp_ids`, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/delete_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_grasps
```

Request

Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte gelöscht, die zu den Templates mit den angegebenen `template_ids` gehören. Die Liste `template_ids` darf nicht leer sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_grasps",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_symmetric_grasps

gibt alle Greifpunkte zurück, die symmetrisch zum angegebenen Greifpunkt sind.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_silhouettematch/services/get_symmetric_
↪grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_symmetric_grasps
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.4.7.4) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp": {
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "replication": {
        "max_x_deg": "float64",
        "min_x_deg": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "template_id": "string"
    }
  }
}
```


Response

Der erste Greifpunkt in der Rückgabeliste ist derselbe, der dem Service übergeben wurde. Wenn das Template keine exakte Symmetrie hat, wird nur der übergebene Greifpunkt zurückgeliefert. Wenn das Template eine kontinuierliche Symmetrie hat (z.B. ein zylindrisches Objekt), werden nur 12 gleichverteilte Greifpunkte zurückgeliefert.

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.4.7.4) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_symmetric_grasps",
  "response": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "template_id": "string"
    }
  ],
  "return_code": {
    "message": "string",
    "value": "int16"
  }
 }
}
```


5.4.7.13 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind.

Tab. 5.41: Rückgabecodes und Warnungen der Services des SilhouetteMatch Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Objekterkennung.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-8	Das Modul befindet sich in einem Zustand, in welchem dieser Service nicht aufgerufen werden kann. Beispielsweise kann `detect_object` nicht aufgerufen werden, solange keine Kalibrierung der Basisebene durchgeführt wurde.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an ROIs oder Templates überschritten wurde.
-100	Ein interner Fehler ist aufgetreten.
-101	Die Erkennung der Basisebene schlug fehl.
-102	Die Hand-Auge-Kalibrierung hat sich seit der letzten Kalibrierung der Basisebene verändert.
-104	Die Verkippung zwischen der Basisebene und der Sichtachse der Kamera überschreitet das 10-Grad-Limit.
10	Die maximale Anzahl an ROIs oder Templates wurde erreicht.
11	Ein bestehendes Element wurde überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Keiner der Greifpunkte ist erreichbar.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
107	Die Basisebene wurde nicht zur aktuellen Kamerapose transformiert, z.B. weil keine Roboterpose während der Kalibrierung der Basisebene angegeben wurde.
108	Das Template ist überholt.
109	Die Ebene für die Objekterkennung passt nicht zum Load Carrier, z.B. liegen die Objekte unterhalb des Load Carrier Bodens.
111	Die Pose des Detektionsergebnisses konnte nicht mit der Punktwolke verfeinert werden, da die Außenkontur des Templates nicht geschlossen ist.
113	Kein Greifer für die Kollisionsprüfung gefunden.
114	Kollisionsprüfung während Entnahme wurde nicht durchgeführt, z.B. weil kein Load Carrier oder kein Greif-Offset angegeben wurden.
151	Das Objekt-Template hat eine kontinuierliche Symmetrie.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

5.4.7.14 Template API

Für den Upload, Download, das Auflisten und Löschen von Templates werden spezielle REST-API-Endpunkte zur Verfügung gestellt. Templates können auch über die Web GUI hoch- und runtergeladen werden. Die Templates beinhalten die Greifpunkte, falls Greifpunkte konfiguriert wurden. Bis zu 50 Templates können gleichzeitig auf dem *rc_visard* gespeichert werden.

`GET /templates/rc_silhouettematch`
listet alle rc_silhouettematch-Templates auf.

Musteranfrage

```
GET /api/v2/templates/rc_silhouettematch HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array der Templates)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.2.4)

`GET /templates/rc_silhouettematch/{id}`
ruft ein rc_silhouettematch-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Templates *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 404 Not Found – Modul oder Template wurden nicht gefunden.

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.2.4)

`PUT /templates/rc_silhouettematch/{id}`
erstellt oder aktualisiert ein rc_silhouettematch-Template.

Musteranfrage

```
PUT /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Formularparameter

- **file** – Template-Datei oder DXF-Datei *(obligatorisch)*
- **object_height** – Objekthöhe in Metern, benötigt bei DXF-Upload *(optional)*
- **units** – Einheit für DXF Datei falls nicht in Datei enthalten (mögliche Werte: `mm`, `cm`, `m`, `in`, `ft`) *(optional)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das SilhouetteMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.2.4)

DELETE /templates/rc_silhouettematch/{id}

entfernt ein rc_silhouettematch-Template.

Musteranfrage

```
DELETE /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das SilhouetteMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

5.5 Konfigurationsmodule

Der *rc_visard* bietet mehrere verschiedene Konfigurationsmodule, welche es dem Nutzer ermöglichen, den *rc_visard* für spezielle Anwendungen zu konfigurieren.

Die Konfigurationsmodule sind:

- ***Hand-Auge-Kalibrierung*** **(`rc_hand_eye_calibration`, Abschnitt 5.5.1)** ermöglicht dem Benutzer, die Kamera entweder über die Web GUI oder die REST-API zu einem Roboter zu kalibrieren.
- ***CollisionCheck*** **(`rc_collision_check`, Abschnitt 5.5.2)** bietet eine einfache Möglichkeit zu prüfen, ob ein Greifer in Kollision ist.
- ***Kamerakalibrierung*** **(`rc_stereocalib`, Abschnitt 5.5.3)** ermöglicht die Überprüfung und Durchführung der Kamerakalibrierung über die *Web GUI* (Abschnitt 6.1).
- ***IOControl und Projektor-Kontrolle*** **(`rc_iocontrol`, Abschnitt 5.5.4)** bietet die Kontrolle über die Ein- und Ausgänge der Kamera mit speziellen Betriebsarten zur Kontrolle eines externen Musterprojektors.

5.5.1 Hand-Auge-Kalibrierung

Für Anwendungen, bei denen die Kamera in eines oder mehrere Robotersysteme integriert wird, muss sie zum jeweiligen Roboter-Koordinatensystem kalibriert werden. Zu diesem Zweck wird der *rc_visard* mit einer internen Kalibrierroutine, dem Modul zur *Hand-Auge-Kalibrierung*, ausgeliefert. Dieses Modul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

Bemerkung: Für die Hand-Auge-Kalibrierung ist es völlig unerheblich, in Bezug auf welches benutzerdefinierte Roboter-Koordinatensystem die Kamera kalibriert wird. Hierbei kann es sich um einen Endeffektor des Roboters (z.B. Flansch oder Tool Center Point (Werkzeugmittelpunkt)) oder um einen beliebigen anderen Punkt in der Roboterstruktur handeln. Einzige Voraussetzung für die Hand-Auge-Kalibrierung ist, dass die Pose (d.h. Positions- und Rotationswerte) dieses Roboter-Koordinatensystems in Bezug auf ein benutzerdefiniertes externes Koordinatensystem (z.B. Welt oder Roboter-Montagepunkt) direkt von der Robotersteuerung erfasst und an das Kalibriermodul übertragen werden kann.

Die *Kalibrierroutine* (Abschnitt 5.5.1.3) ist ein benutzerfreundliches mehrstufiges Verfahren, für das mit einem Kalibriermuster gearbeitet wird. Entsprechende Kalibriermuster können von Roboception bezogen werden.

5.5.1.1 Kalibrierschnittstellen

Für die Durchführung der Hand-Auge-Kalibrierung stehen die folgenden beiden Schnittstellen zur Verfügung:

1. Alle Services und Parameter dieses Moduls, die für eine **programmgesteuerte** Durchführung der Hand-Auge-Kalibrierung benötigt werden, sind in der *REST-API-Schnittstelle* (Abschnitt 6.2) des *rc_visard* enthalten. Der REST-API-Name dieses Moduls lautet `rc_hand_eye_calibration` und seine Services werden in *Services* (Abschnitt 5.5.1.5) erläutert.

Bemerkung: Für den beschriebenen Ansatz wird eine Netzwerkverbindung zwischen dem *rc_visard* und der Robotersteuerung benötigt, damit die Steuerung die Roboterposen an das Kalibriermodul des *rc_visard* übertragen kann.

2. Für Anwendungsfälle, bei denen sich die Roboterposen nicht programmgesteuert an das Modul zur Hand-Auge-Kalibrierung des *rc_visard* übertragen lassen, sieht die Seite *Hand-Auge-Kalibrierung* unter dem Menüpunkt *Konfiguration* der *Web GUI* (Abschnitt 6.1) einen geführten Prozess vor, mit dem sich die Kalibrierroutine **manuell** durchführen lässt.

   Bemerkung: Während der Kalibrierung muss der Benutzer die Roboterposen, auf die über das jeweilige Teach-in- oder Handheld-Gerät zugegriffen werden muss, manuell in die Web GUI eingeben.

5.5.1.2 Kameramontage

Wie in Abb. 5.20 und Abb. 5.22 dargestellt, ist für die Montage der Kamera zwischen zwei unterschiedlichen Anwendungsfällen zu unterscheiden:

a. Die Kamera wird **am Roboter montiert**, d.h. sie ist mechanisch mit einem Roboterpunkt (d.h. Flansch oder flanschmontiertes Werkzeug) verbunden und bewegt sich demnach mit dem Roboter.

b. Die Kamera ist nicht am Roboter montiert, sondern an einem Tisch oder anderen Ort in der Nähe des Roboters befestigt und verbleibt daher verglichen mit dem Roboter in einer **statischen** Position.

Die allgemeine *Kalibrierroutine* (Abschnitt 5.5.1.3) ist in beiden Anwendungsfällen sehr ähnlich. Sie unterscheidet sich jedoch hinsichtlich der semantischen Auslegung der Ausgabedaten, d.h. der erhaltenen Kalibriertransformation, und hinsichtlich der Befestigung des Kalibriermusters.

Kalibrierung einer robotergeführten Kamera Soll eine robotergeführte Kamera zum Roboter kalibriert werden, so muss das Kalibriermuster in einer statischen Position zum Roboter, z.B. auf einem Tisch oder festen Sockel, befestigt werden (siehe Abb. 5.20).

Warnung: Es ist äußerst wichtig, dass sich das Kalibriermuster in Schritt 2 der *Kalibrierroutine* (Abschnitt 5.5.1.3) nicht bewegt. Daher wird dringend empfohlen, das Muster in seiner Position sicher zu fixieren, um unbeabsichtigte Bewegungen, wie sie durch Vibrationen, Kabelbewegungen oder Ähnliches ausgelöst werden, zu verhindern.

Das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 5.5.1.3) ist eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{robot}} = \mathbf{R}_{\text{camera}}^{\text{robot}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}_{\text{camera}}^{\text{robot}} \,, \tag{5.3}$$

wobei $\mathbf{p}_{\text{robot}} = (x, y, z)^T$ ein 3D-Punkt ist, dessen Koordinaten im *Roboter*-Koordinatensystem angegeben werden, $\mathbf{p}_{\text{camera}}$ denselben Punkt im *Kamera*-Koordinatensystem darstellt, und $\mathbf{R}_{\text{camera}}^{\text{robot}}$ sowie $\mathbf{t}_{\text{camera}}^{\text{robot}}$ die 3×3 Drehmatrix und den 3×1 Translationsvektor für eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$ angeben. In der Praxis wird die Rotation für das Kalibrierergebnis und die Roboterposen als Eulerwinkel oder Quaternion anstatt einer Rotationsmatrix definiert (siehe *Formate für Posendaten*, Abschnitt 11.1).

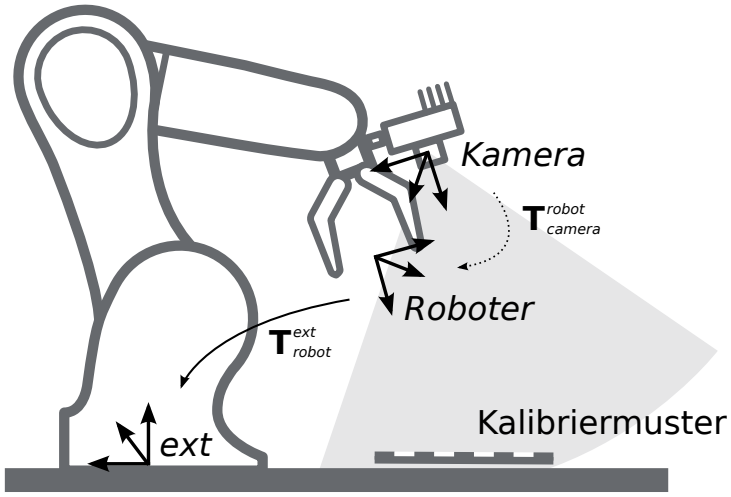


Abb. 5.20: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer robotergeführten Kamera: Sie wird mit einer festen relativen Position zu einem benutzerdefinierten *Roboter*-Koordinatensystem (z.B. Flansch oder Werkzeugmittelpunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des *Roboter*-Koordinatensystems in Bezug auf ein benutzerdefiniertes externes Referenzkoordinatensystem (*ext*) während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibriervorgangs ist die gewünschte Kalibriertransformation $\mathbf{T}^{\text{robot}}_{\text{camera}}$, d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten *Roboter*-Koordinatensystem.

Zusätzliche Benutzereingaben werden benötigt, falls die Bewegung des Roboters so beschränkt ist, dass der Tool Center Point (TCP) nur um eine Achse rotieren kann. Das ist üblicherweise für Roboter mit vier Freiheitsgraden (4DOF) der Fall, welche häufig zur Palettierung eingesetzt werden. In diesem Fall muss der Benutzer angeben, welche Achse des Roboterkoordinatensystems der Rotationsachse des TCP entspricht. Außerdem muss der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Abb. 5.21 zeigt die Situation.

Für den *rc_visard* befindet sich der Ursprung des Kamerakoordinatensystems im optischen Zentrum der linken Kamera. Die ungefähre Position wird im Abschnitt *Koordinatensysteme* (Abschnitt 3.7) angegeben.

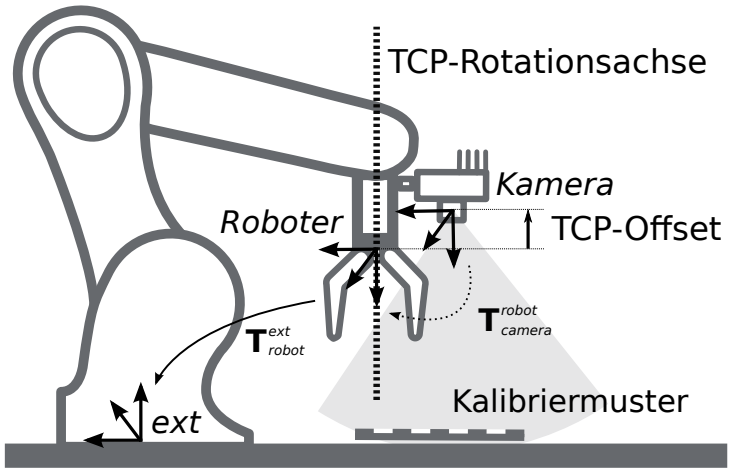


Abb. 5.21: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

Kalibrierung einer statisch montierten Kamera In Anwendungsfällen, bei denen die Kamera statisch verglichen zum Roboter montiert wird, muss das Kalibriermuster, wie im Beispiel in Abb.

5.22 und Abb. 5.23 angegeben, angebracht werden.

Bemerkung: Für das Modul zur Hand-Auge-Kalibrierung spielt es keine Rolle, wie das Kalibriermuster in Bezug auf das benutzerdefinierte *Roboter*-Koordinatensystem genau angebracht und positioniert wird. Das bedeutet, dass die relative Positionierung des Kalibriermusters zu diesem Koordinatensystem weder bekannt sein muss, noch für die Kalibrierroutine relevant ist (siehe in Abb. 5.23).

Warnung: Es ist äußerst wichtig, das Kalibriermuster sicher am Roboter anzubringen, damit sich seine relative Position in Bezug auf das in Schritt 2 der *Kalibrierroutine* (Abschnitt 5.5.1.3) vom Benutzer definierte *Roboter*-Koordinatensystem nicht verändert.

In diesem Anwendungsfall ist das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 5.5.1.3) die Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{ext}} = \mathbf{R}^{\text{ext}}_{\text{camera}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}^{\text{ext}}_{\text{camera}} \,, \tag{5.4}$$

wobei $\mathbf{p}_{\text{ext}} = (x, y, z)^T$ ein 3D-Punkt im externen Referenzkoordinatensystem *ext*, $\mathbf{p}_{\text{camera}}$ derselbe Punkt im Kamerakoordinatensystem *camera* und $\mathbf{R}^{\text{ext}}_{\text{camera}}$ sowie $\mathbf{t}^{\text{ext}}_{\text{camera}}$ die 3×3 Rotationsmatrix und 3×1 Translationsvektor der Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$ sind. In der Praxis wird die Rotation für das Kalibrierergebnis und die Roboterposen als Eulerwinkel oder Quaternion anstatt einer Rotationsmatrix definiert (siehe *Formate für Posendaten*, Abschnitt 11.1).

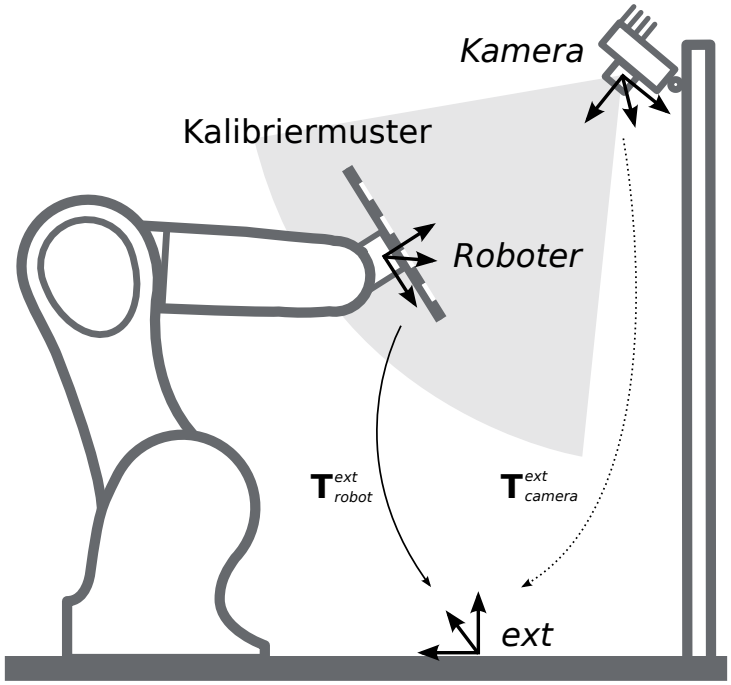


Abb. 5.22: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer statisch montierten Kamera: Sie wird mit einer festen Position relativ zu einem benutzerdefinierten externen Referenzkoordinatensystem *ext* (z.B. Weltkoordinatensystem oder Roboter-Montagepunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems in Bezug auf dieses Koordinatensystem während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibrierprozesses ist die gewünschte Kalibriertransformation $\mathbf{T}^{\text{ext}}_{\text{camera}}$, d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten externen Koordinatensystem *ext*.

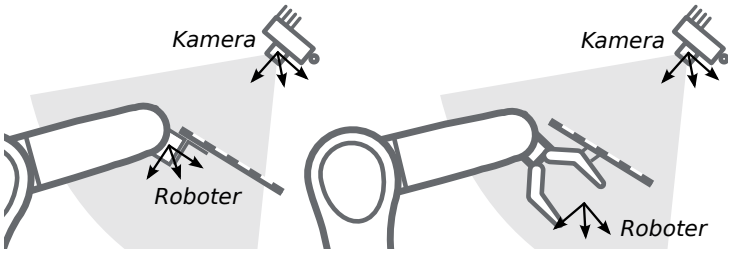


Abb. 5.23: Alternative Montageoptionen für die Befestigung des Kalibriermusters am Roboter

Zusätzliche Benutzereingaben werden benötigt, falls die Bewegung des Roboters so beschränkt ist, dass der Tool Center Point (TCP) nur um eine Achse rotieren kann. Das ist üblicherweise für Roboter mit vier Freiheitsgraden (4DOF) der Fall, welche häufig zur Palettierung eingesetzt werden. In diesem Fall muss der Benutzer angeben, welche Achse des Roboterkoordinatensystems der Rotationsachse des TCP entspricht. Außerdem muss der vorzeichenbehaftete Offset vom TCP zur sichtbaren Oberfläche des Kalibriermusters entlang der TCP-Rotationsachse angegeben werden. Das Kalibriermuster muss so angebracht werden, dass die TCP-Rotationsachse orthogonal zum Kalibriermuster verläuft. Abb. 5.24 zeigt die Situation.

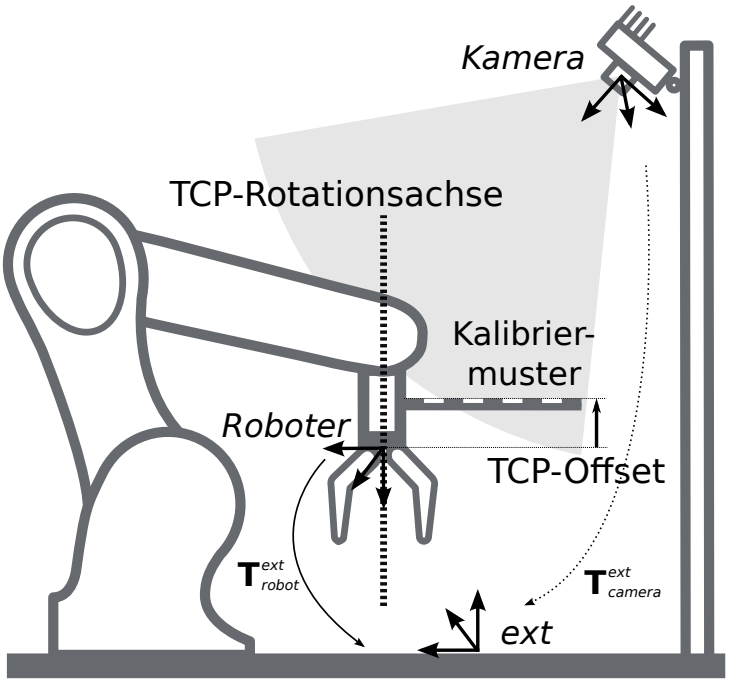


Abb. 5.24: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zur sichtbaren Oberfläche des Kalibriermusters entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

5.5.1.3 Kalibrierroutine

Die Hand-Auge-Kalibrierung kann manuell über die *Web GUI* (Abschnitt 6.1) oder programmgesteuert über die *REST-API-Schnittstelle* (Abschnitt 6.2) durchgeführt werden. Die allgemeine Vorgehensweise wird beschrieben anhand der Schritte in der Web GUI unter *Konfiguration* → *Hand-Auge-Kalibrierung*. Verweise auf die zugehörigen REST-API Aufrufe werden an den entsprechenden Stellen bereitgestellt.

Schritt 1: Hand-Auge-Kalibrierstatus

Die Startseite des Assistenten für die Hand-Auge-Kalibrierung zeigt den aktuellen Status der Hand-Auge-Kalibrierung. Wenn eine Hand-Auge-Kalibrierung auf dem *rc_visard* gespeichert ist, wird die Kalibriertransformation hier angezeigt (siehe Abb. 5.25).



Abb. 5.25: Aktueller Status der Hand-Auge-Kalibrierung falls eine Hand-Auge-Kalibrierung gespeichert ist

Um den Status der Hand-Auge-Kalibrierung programmgesteuert abzufragen bietet die REST-API den Service `get_calibration` (siehe *Services*, Abschnitt 5.5.1.5). Eine vorhandene Hand-Auge-Kalibrierung kann über *Kalibrierung entfernen* oder den REST-API Service `remove_calibration` (siehe *Services*, Abschnitt 5.5.1.5) gelöscht werden.

Durch Klick auf *Kalibrierung durchführen* wird eine neue Hand-Auge-Kalibrierung gestartet.

Schritt 2: Testen der Mustererkennung

Um gute Kalibrierergebnisse zu erzielen müssen die Bilder gut belichtet sein, damit das Kalibriermuster genau und verlässlich erkannt werden kann. In diesem Schritt kann die Erkennung des Kalibriermusters getestet werden und die Kameraeinstellungen können angepasst werden, falls nötig. Falls Teile des Kalibriermusters überbelichtet sind, werden die zugehörigen Quadrate rot hervorgehoben. Die erfolgreiche Erkennung des Kalibriermusters wird durch grüne Häkchen auf jedem Quadrat und einen dicken grünen Rahmen um das Kalibriermuster visualisiert, wie in Abb. 5.26 dargestellt ist.

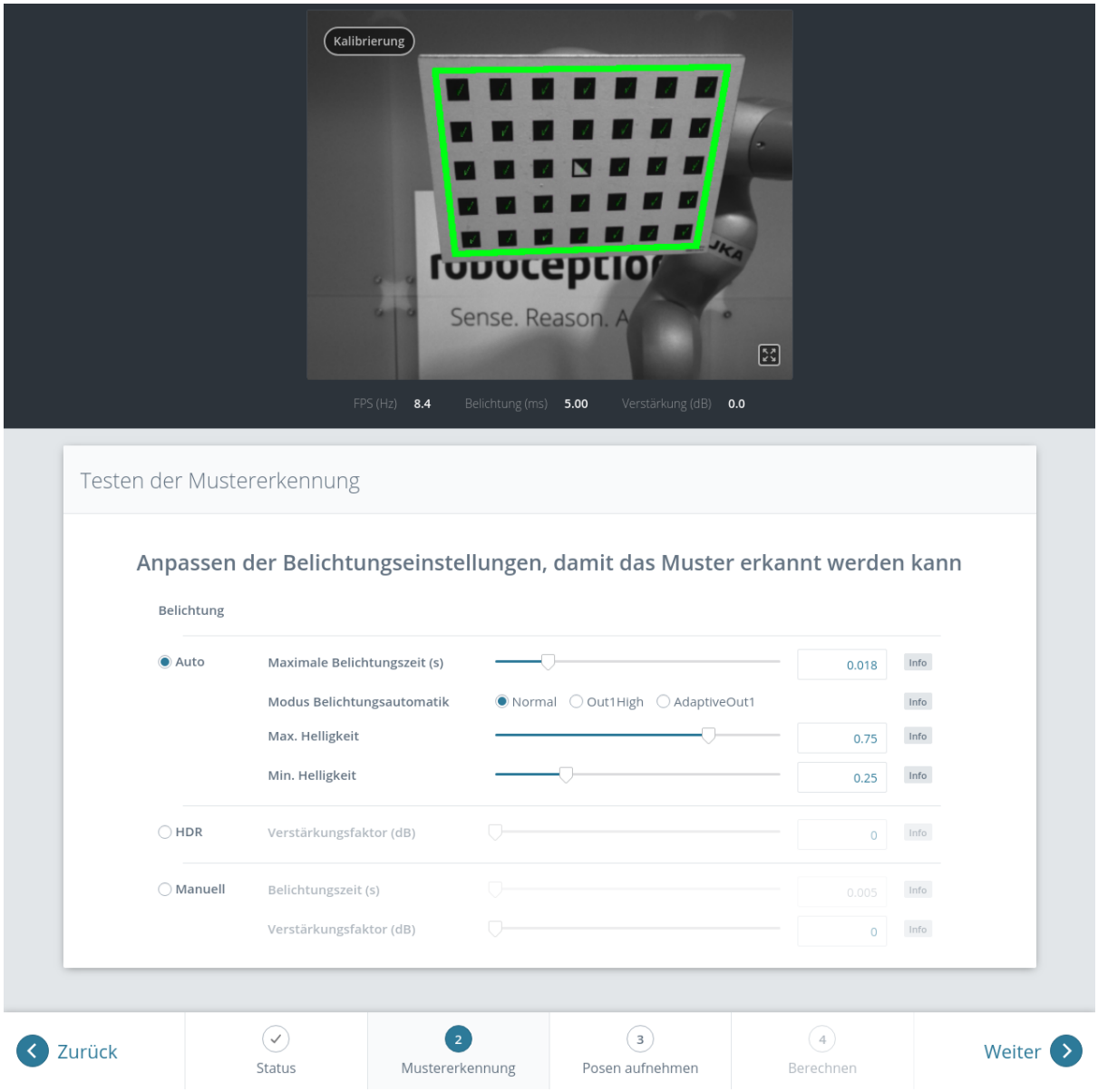


Abb. 5.26: Testen der Mustererkennung

Schritt 3: Posen aufnehmen

In diesem Schritt werden Bilder des Kalibriermusters an verschiedenen Roboterposen aufgenommen. Dabei ist sicherzustellen, dass das Kalibriermuster bei allen Posen im linken Kamerabild vollständig sichtbar ist. Zudem müssen die Roboterpositionen sorgsam ausgewählt werden, damit das Kalibriermuster aus unterschiedlichen Perspektiven aufgenommen wird. Abb. 5.27 zeigt eine schematische Darstellung der empfohlenen acht Ansichten.

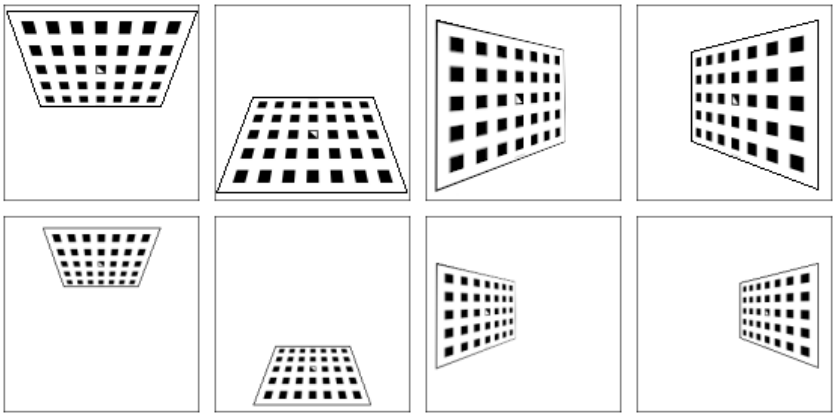


Abb. 5.27: Empfohlene Ansichten des Kalibriermusters während des Kalibriervorgangs. Im Fall von 4DOF-Robotern müssen andere Ansichten gewählt werden, welche so unterschiedlich wie möglich sein sollten.

Warnung: Die Kalibrierqualität, d.h. die Genauigkeit des berechneten Kalibrierergebnisses, hängt von den Ansichten des Kalibriermusters ab. Je vielfältiger die Perspektiven sind, desto besser gelingt die Kalibrierung. Werden sehr ähnliche Ansichten ausgewählt, d.h. wird die Pose des Roboters vor der Aufnahme einer neuen Kalibrierpose nur leicht variiert, kann dies zu einer ungenauen Schätzung der gewünschten Kalibriertransformation führen.

Nachdem der Roboter die jeweilige Kalibrierposition erreicht hat, muss die entsprechende Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems im benutzerdefinierten externen Referenzkoordinatensystem *ext* an das Modul zur Hand-Auge-Kalibrierung übertragen werden. Hierfür bietet das Softwaremodul verschiedene *Slots*, in denen die gemeldeten Posen mit den zugehörigen Bildern der linken Kamera hinterlegt werden können. Alle gefüllten Slots werden dann verwendet, um die gewünschte Kalibriertransformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Kameras) bzw. dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Kameras) zu berechnen.

In der Web GUI kann der Nutzer zwischen vielen verschiedenen Formaten für die Kalibrierposen wählen (siehe *Formate für Posendaten*, Abschnitt 11.1). Wird die Kalibrierung über die REST-API vorgenommen, dann werden die Kalibrierdaten immer im Format *XYZ+Quaternion* angegeben. Die Web GUI bietet acht Slots (*Nahaufnahme 1*, *Nahaufnahme 2*, usw.), in die der Benutzer die Posen manuell eintragen kann. Neben jedem Slot wird eine Empfehlung für die Ansicht des Kalibriermusters angezeigt. Der Roboter sollte für jeden Slot so bewegt werden, dass die empfohlene Ansicht erreicht wird.

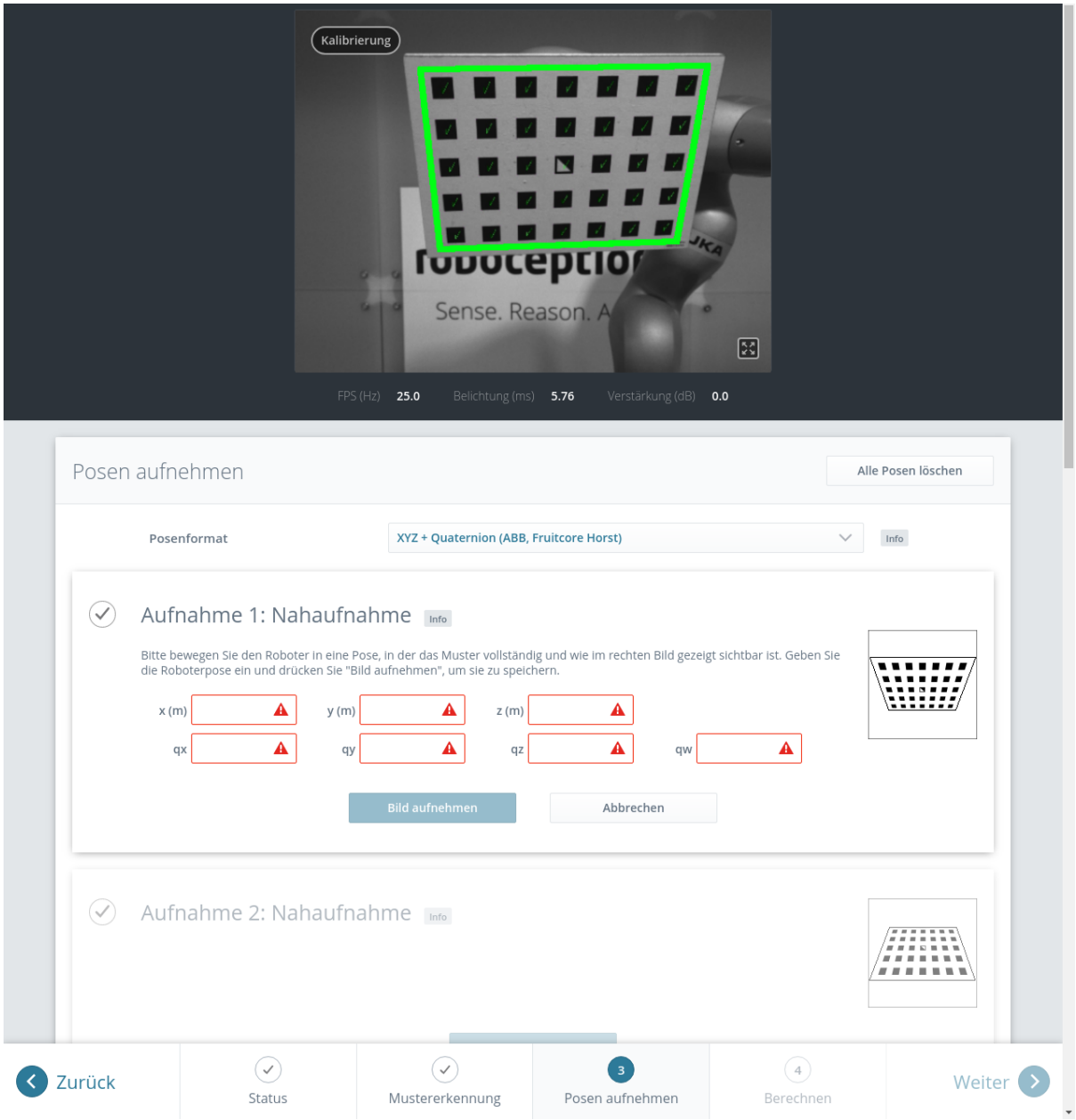


Abb. 5.28: Setzen der ersten Kalibrierpose für die Hand-Auge-Kalibrierung bei einer statisch montierten Kamera

Nach Klick auf *Pose setzen* kann die Pose des benutzerdefinierten *Roboter*-Koordinatensystems manuell in die entsprechenden Textfelder eingegeben werden. Durch *Bild aufnehmen* werden die Pose und das aktuelle Kamerabild im jeweiligen Slot gespeichert.

Um diese Posen programmgesteuert zu übertragen, bietet die REST-API den Service `set_pose` (siehe *Services*, Abschnitt 5.5.1.5).

Bemerkung: Der Zugriff auf die Posendaten des Roboters hängt vom Modell des Roboters und seinem Hersteller ab. Möglicherweise lassen sie sich über ein im Lieferumfang des Roboters enthaltenes Teach-in- oder Handheld-Gerät ablesen.

Warnung: Es ist wichtig darauf zu achten, dass genaue und korrekte Werte eingegeben werden. Selbst kleinste Ungenauigkeiten oder Tippfehler können dazu führen, dass die Kalibrierung fehl-

schlägt.

Die Web GUI zeigt die aktuell gespeicherten Kalibrierposen (nur mit den Slot-Nummern 0-7) und die zugehörigen Kamerabilder an und ermöglicht auch das Löschen von einzelnen Posen über *Pose löschen*, oder das Löschen aller gesetzten Posen über *Alle Posen löschen*. In der REST-API können die aktuell gespeicherten Kalibrierposen über `get_poses` abgefragt und über `delete_poses` oder `reset_calibration` einzeln bzw. komplett gelöscht werden (siehe *Services*, Abschnitt 5.5.1.5).

Wenn mindestens vier Posen gesetzt wurden, gelangt man über die Schaltfläche *Weiter* zur Berechnung des Kalibrierergebnisses.

Bemerkung: Um die Transformation für die Hand-Auge-Kalibrierung erfolgreich zu berechnen, müssen mindestens vier verschiedene Roboter-Kalibrierposen übertragen und in Slots hinterlegt werden. Um Kalibrierfehler zu verhindern, die durch ungenaue Messungen entstehen können, sind mindestens **acht Kalibrierposen empfohlen**.

Schritt 4: Kalibrierung berechnen

Bevor das Kalibrierergebnis berechnet werden kann, muss der Nutzer die korrekten Kalibrierparameter angeben. Diese beinhalten die exakten Abmessungen des Kalibriermusters und die Art der Sensormontage. Weiterhin kann die Kalibrierung von 4DOF-Robotern eingestellt werden. In diesem Fall müssen die Rotationsachse, sowie der Offset vom TCP zum Kamerakoordinatensystem (für Kameras am Roboter) oder zur Oberfläche des Kalibriermusters (für statische Kameras) angegeben werden. Für die REST-API sind die entsprechenden *Parameter* (Abschnitt 5.5.1.4) aufgelistet.

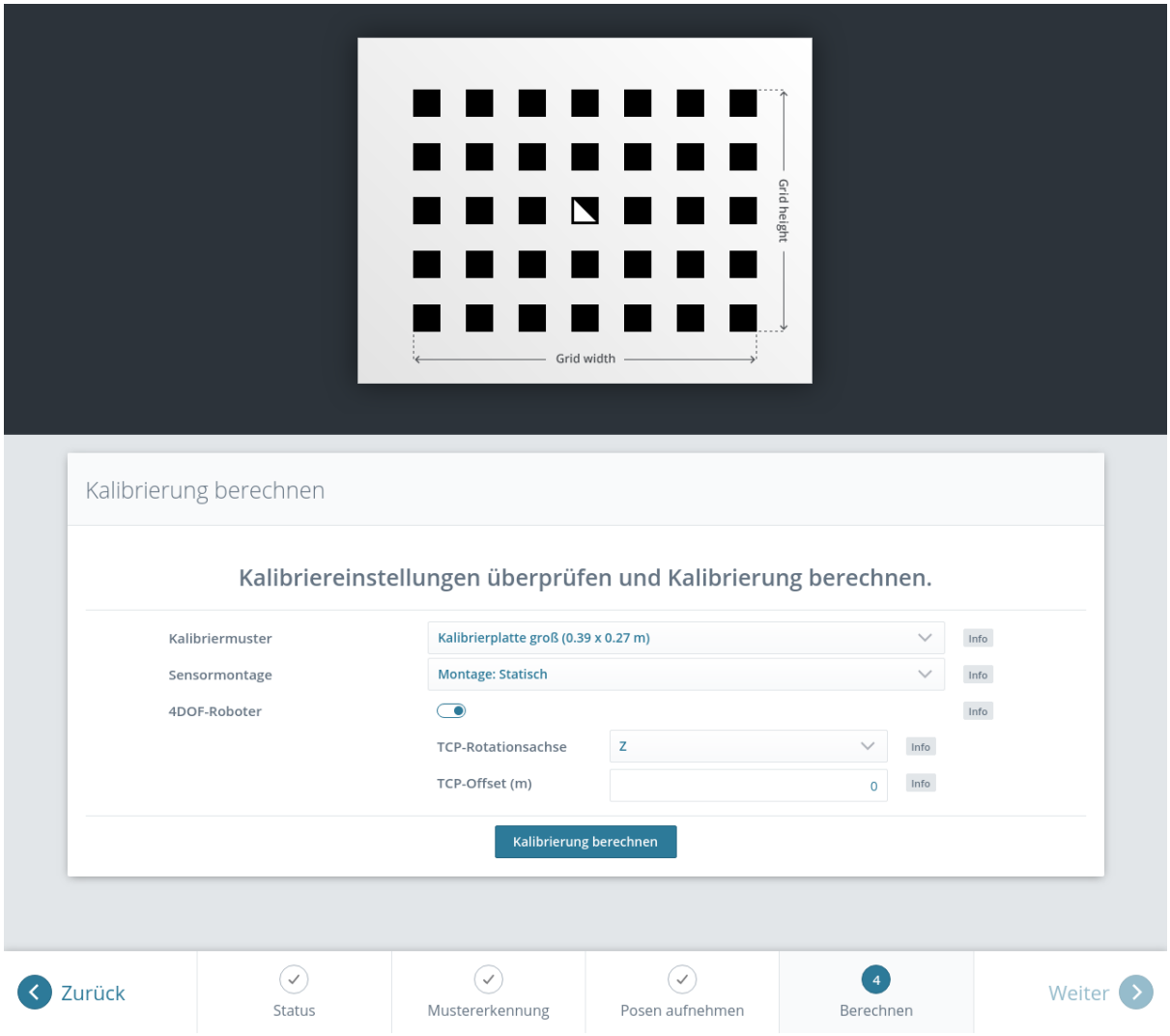


Abb. 5.29: Setzen der Parameter und Berechnen der Hand-Auge-Kalibrierung in der Web GUI des *rc_visard*

Wenn die Parameter korrekt sind, kann durch *Kalibrierung berechnen* die gewünschte Kalibriertransformation aus den aufgenommenen Kalibrierposen und den zugehörigen Kamerabildern berechnet werden. Die REST-API bietet diese Funktion über den Service `calibrate` (siehe *Services*, Abschnitt 5.5.1.5).

Je nachdem, wie die Kamera montiert ist, wird dabei die Transformation (d.h. die Pose) zwischen dem *Kamera*-Koordinatensystem und entweder dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Kameras) oder dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Kameras) berechnet und ausgegeben (siehe *Kameramontage*, Abschnitt 5.5.1.2).

Damit der Benutzer die Qualität der resultierenden Kalibriertransformation beurteilen kann, werden die translatorischen und rotatorischen Kalibrierfehler ausgegeben. Diese Werte werden aus der Varianz des Kalibrierergebnisses berechnet.

Wenn der Kalibrierfehler nicht akzeptabel ist, können die Kalibrierparameter geändert und das Ergebnis neu berechnet werden. Außerdem ist es möglich, zu Schritt 3 zurückzukehren, um mehr Posen aufzunehmen oder die vorhandenen Posen zu aktualisieren.

Durch Klicken auf *Kalibrierung speichern* oder über den REST-API Service `save_calibration` (siehe *Services*, Abschnitt 5.5.1.5) wird das Kalibrierergebnis gespeichert.

5.5.1.4 Parameter

Das Modul zur Hand-Auge-Kalibrierung wird in der REST-API als `rc_hand_eye_calibration` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Konfiguration* → *Hand-Auge Kalibrierung* dargestellt. Der Benutzer kann die Kalibrierparameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.42: Laufzeitparameter des `rc_hand_eye_calibration`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`grid_height`	float64	0.0	10.0	0.0	Höhe des Kalibriermusters in Metern
`grid_width`	float64	0.0	10.0	0.0	Breite des Kalibriermusters in Metern
`robot_mounted`	bool	false	true	true	Angabe, ob der *rc_visard* auf einem Roboter montiert ist
`tag_ids`	string	-	-	-	Optional, kommaseparierte Liste der AprilTag IDs, die mit kalibriert werden
`tcp_offset`	float64	-10.0	10.0	0.0	Offset vom TCP entlang tcp_rotation_axis
`tcp_rotation_axis`	int32	-1	2	-1	-1 für aus, 0 für x, 1 für y, 2 für z

Beschreibung der Laufzeitparameter

Für die Beschreibungen der Parameter sind die in der Web GUI gewählten Namen der Parameter in Klammern angegeben.

`grid_width` (*Breite*)

Breite des Kalibriermusters in Metern. Die Breite sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, angegeben werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/parameters?
↪grid_width=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?grid_width=<value>
```

`grid_height` (*Höhe*)

Höhe des Kalibriermusters in Metern. Die Höhe sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, angegeben werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/parameters?
↪grid_height=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?grid_height=<value>
```

`robot_mounted` (*Sensormontage*)

Ist dieser Parameter auf *true* gesetzt, dann ist die Kamera an einem Roboter montiert. Ist er auf *false* gesetzt, ist sie statisch montiert und das Kalibriermuster ist am Roboter angebracht.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/parameters?
↪robot_mounted=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?robot_mounted=<value>
```

`tcp_offset` (*TCP-Offset*)

Der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem (für Kameras auf dem Roboter) oder der sichtbaren Oberfläche des Kalibriermusters (für statische Kameras) entlang der TCP-Rotationsachse in Metern. Dies wird benötigt, falls die Roboterbewegung eingeschränkt ist, sodass der TCP nur um eine Achse gedreht werden kann (z.B. bei 4DOF-Robotern).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/parameters?
↪tcp_offset=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?tcp_offset=<value>
```

`tcp_rotation_axis` (*TCP-Rotationsachse*)

Die Achse des Roboterkoordinatensystems, um die der Roboter seinen TCP drehen kann. 0 für X-, 1 für Y- und 2 für Z-Achse. Dies wird benötigt falls, die Roboterbewegung eingeschränkt ist, sodass der TCP nur um eine Achse gedreht werden kann (z.B. bei 4DOF-Robotern). -1 bedeutet, dass der Roboter seinen TCP um zwei unabhängige Achsen drehen kann. `tcp_offset` wird in diesem Fall ignoriert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/parameters?
↪tcp_rotation_axis=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?tcp_rotation_axis=
↪<value>
```

5.5.1.5 Services

Auf die Services, die die REST-API für die programmgesteuerte Durchführung der Hand-Auge-Kalibrierung und für die Wiederherstellung der Modulparameter bietet, wird im Folgenden näher eingegangen.

get_calibration

Hiermit wird die derzeit auf dem *rc_visard* gespeicherte Hand-Auge-Kalibrierung abgerufen.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/get_
↪calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/get_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Das Feld `error` gibt den Kalibrierfehler in Pixeln an, der aus dem translatorischen Fehler `translation_error_meter` und dem rotatorischen Fehler `rotation_error_degree` berechnet wird. Dieser Wert wird nur aus Kompatibilitätsgründen mit älteren Versionen angegeben. Die translatorischen und rotatorischen Fehler sollten bevorzugt werden.

Tab. 5.43: Rückgabewerte des `get_calibration`-Services

status	success	Beschreibung
0	true	eine gültige Kalibrierung wurde zurückgegeben
2	false	die Kalibrierung ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_calibration",
  "response": {
    "error": "float64",
    "message": "string",
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
    "status": "int32",
    "success": "bool",
    "tags": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "size": "float64"
      }
    ],
    "translation_error_meter": "float64"
  }
}
```


remove_calibration

Dieser Service löscht die persistente Hand-Auge-Kalibrierung auf dem *rc_visard*. Nach diesem Aufruf gibt der `get_calibration` Service zurück, dass keine Hand-Auge-Kalibrierung vorliegt. Dieser Service löscht ebenfalls alle gespeicherten Kalibrierposen und die zugehörigen Kamerabilder.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/remove_
↪calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/remove_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Tab. 5.44: Rückgabewerte des `get_calibration`-Services

status	success	Beschreibung
0	true	persistente Kalibrierung gelöscht, Gerät nicht mehr kalibriert
1	true	keine persistente Kalibrierung gefunden, Gerät nicht kalibriert
2	false	die Kalibrierung konnte nicht gelöscht werden

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "remove_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

set_pose

Dieser Service setzt die Roboterpose als Kalibrierpose für die Hand-Auge-Kalibrierroutine und nimmt das aktuelle Bild des Kalibriermusters auf.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/set_pose
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/set_pose
```

Request

Das `slot`-Argument wird verwendet, um den verschiedenen Kalibrierpositionen eindeutige Ziffern im Wertebereich von 0-15 zuzuordnen. Wann immer der Service `set_pose` aufgerufen wird, wird ein Kamerabild aufgezeichnet. Dieser Service schlägt fehl, wenn das Kalibriermuster im aktuellen Bild nicht erkannt werden kann.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "slot": "uint32"
  }
}
```

Response

Tab. 5.45: Rückgabewerte des `set_pose`-Services

status	success	Beschreibung
1	true	Pose erfolgreich gespeichert
3	true	Pose erfolgreich gespeichert. Es wurden genügend Posen für die Kalibrierung gespeichert, d.h. die Kalibrierung kann durchgeführt werden
4	false	das Kalibriermuster wurde nicht erkannt, z.B. weil es im Kamerabild nicht vollständig sichtbar ist
8	false	keine Bilddaten verfügbar
12	false	die angegebenen Orientierungswerte sind ungültig
13	false	ungültige Slot-Nummer

Das Feld `overexposed` gibt an, ob Teile des Kalibriermusters bei diesem Bild überbelichtet sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_pose",
  "response": {
    "message": "string",
    "overexposed": "bool",
    "status": "int32",
    "success": "bool"
  }
}
```

get_poses

Dieser Service gibt die aktuell gespeicherten Kalibrierposen für die Hand-Auge-Kalibrierung zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/get_poses
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/get_poses
```

Request

Dieser Service hat keine Argumente.

Response

Tab. 5.46: Rückgabewerte des `get_poses`-Services

status	success	Beschreibung
0	true	gespeicherte Posen werden zurückgeliefert
1	true	keine Kalibrierposen verfügbar

Das Feld `overexposed` gibt an, ob Teile des Kalibriermusters bei diesem Bild überbelichtet sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_poses",
  "response": {
    "message": "string",
    "poses": [
      {
        "overexposed": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "slot": "uint32",
        "tag_ids": [
          "string"
        ]
      }
    ],
    "status": "int32",
    "success": "bool"
  }
}
```


delete_poses

Dieser Service löscht die Kalibrierposen und die zugehörigen Bilder mit den angegebenen Nummern in `slots`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/delete_
↪poses
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/delete_poses
```

Request

Das Argument `slots` gibt die Ziffern der Kalibrierposen an, die gelöscht werden sollen. Wenn `slots` leer ist, werden keine Kalibrierposen gelöscht.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "slots": [
      "uint32"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    ]
  }
}
```

Response

Tab. 5.47: Rückgabewerte des `delete_poses`-Services

status	success	Beschreibung
0	true	Posen erfolgreich gelöscht
1	true	Keine Slots angegeben

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_poses",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

reset_calibration

Hiermit werden alle zuvor aufgenommenen Posen mitsamt der zugehörigen Bilder gelöscht. Das letzte hinterlegte Kalibrierergebnis wird neu geladen. Dieser Service kann verwendet werden, um die Hand-Auge-Kalibrierung (neu) zu starten.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/reset_
↪calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/reset_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

calibrate

Dieser Service dient dazu, das Ergebnis der Hand-Auge-Kalibrierung auf Grundlage der über den Service `set_pose` konfigurierten Roboterposen zu berechnen und auszugeben.

Details

Damit die Kalibrierung für andere Module mit `get_calibration` verfügbar ist und persistent gespeichert wird, muss `save_calibration` aufgerufen werden.

Bemerkung: Zur Berechnung der Transformation der Hand-Auge-Kalibrierung werden mindestens vier Roboterposen benötigt (siehe `set_pose`). Empfohlen wird jedoch die Verwendung von acht Kalibrierposen.

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/calibrate
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/calibrate
```

Request

Dieser Service hat keine Argumente.

Response

Das Feld `error` gibt den Kalibrierfehler in Pixeln an, der aus dem translatorischen Fehler `translation_error_meter` und dem rotatorischen Fehler `rotation_error_degree` berechnet wird. Dieser Wert wird nur aus Kompatibilitätsgründen mit älteren Versionen angegeben. Die translatorischen und rotatorischen Fehler sollten bevorzugt werden.

Tab. 5.48: Rückgabewerte des `calibrate`-Services

status	success	Beschreibung
0	true	Kalibrierung erfolgreich, das Kalibrierergebnis wurde zurückgegeben.
1	false	Nicht genügend Posen gespeichert, um die Kalibrierung durchzuführen
2	false	Das berechnete Ergebnis ist ungültig, bitte prüfen Sie die Eingabewerte.
3	false	Die angegebenen Abmessungen des Kalibriermusters sind ungültig.
4	false	Ungenügende Rotation, `tcp_offset` and `tcp_rotation_axis` müssen angegeben werden
5	false	Genügend Rotation verfügbar, `tcp_rotation_axis` muss auf -1 gesetzt werden
6	false	Die Posen sind nicht unterschiedlich genug.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "calibrate",
  "response": {
    "error": "float64",
    "message": "string",
    "pose": {
      "orientation": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
    "status": "int32",
    "success": "bool",
    "tags": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "size": "float64"
      }
    ],
    "translation_error_meter": "float64"
  }
}
```


save_calibration

Hiermit wird das Ergebnis der Hand-Auge-Kalibrierung persistent auf dem *rc_visard* gespeichert und das vorherige Ergebnis überschrieben. Das gespeicherte Ergebnis lässt sich jederzeit über den Service `get_calibration` abrufen. Dieser Service löscht ebenfalls alle gespeicherten Kalibrierposen und die zugehörigen Kamerabilder.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/save_
↪calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/save_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Tab. 5.49: Rückgabewerte des `save_calibration`-Services

status	success	Beschreibung
0	true	die Kalibrierung wurde erfolgreich gespeichert
1	false	die Kalibrierung konnte nicht im Dateisystem gespeichert werden
2	false	die Kalibrierung ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "save_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

set_calibration

Hiermit wird die übergebene Transformation als Hand-Auge-Kalibrierung gesetzt.

Details

Die Kalibrierung wird im gleichen Format erwartet, in dem sie beim `calibrate` und `get_calibration` Aufruf zurückgegeben wird. Die gegebene Kalibrierung wird auch persistent gespeichert, indem intern `save_calibration` aufgerufen wird.

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/set_
↪calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/set_calibration
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "tags": [
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "size": "float64"
      }
    ]
  }
}
```


Response

Tab. 5.50: Rückgabewerte des `set_calibration`-Services

status	success	Beschreibung
0	true	Setzen der Kalibrierung war erfolgreich
12	false	die angegebenen Orientierungswerte sind ungültig

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```


reset_defaults

Hiermit werden die Werkseinstellungen der Parameter dieses Moduls wieder hergestellt und angewandt („factory reset“). Dies hat keine Auswirkungen auf das Kalibrierergebnis oder auf die während der Kalibrierung gefüllten `Slots`. Es werden lediglich Parameter, wie die Maße des Kalibriermusters oder die Montageart des Sensors, zurückgesetzt.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services/reset_
↪defaults
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.5.2 CollisionCheck

5.5.2.1 Einleitung

Das CollisionCheck Modul ist ein optionales Modul, welches intern auf dem *rc_visard* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5) oder *SilhouetteMatch* (Abschnitt 5.4.7) vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 7.4).

Das Modul ermöglicht die Kollisionsprüfung zwischen dem Greifer und dem Load Carrier, der Punktwolke (nur in Kombination mit *SilhouetteMatch* (Abschnitt 5.4.7)), oder anderen detektierten Objekten (nur in Kombination mit *SilhouetteMatch* (Abschnitt 5.4.7)). Es ist in die Module *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5) und *SilhouetteMatch* (Abschnitt 5.4.7) integriert, kann aber auch als eigenständiges Modul genutzt werden. Die Greifermodelle für die Kollisionsprüfung müssen über das *GripperDB* (Abschnitt 5.6.4) Modul definiert werden.

Warnung: Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch oder anderen Objekten. Nur wenn `check_collisions_with_point_cloud` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur in Kombination mit *SilhouetteMatch* (Abschnitt 5.4.7), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und den anderen *detektierten* Objekten geprüft. Kollisionen mit Objekten, die nicht detektiert werden können, werden nicht geprüft.

Tab. 5.51: Spezifikationen des CollisionCheck-Moduls

Kollisionsprüfung mit	detektierter Load Carrier, detektierte Objekte (nur *SilhouetteMatch* (Abschnitt 5.4.7)), Basisebene (nur *SilhouetteMatch*, Abschnitt 5.4.7), Punktwolke (nur *SilhouetteMatch* (Abschnitt 5.4.7))
Kollisionsprüfung verfügbar in	*ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5), *SilhouetteMatch* (Abschnitt 5.4.7)

5.5.2.2 Kollisionsprüfung

Stand-Alone Kollisionsprüfung

Der Service `check_collisions` triggert die Kollisionsprüfung zwischen dem angegebenen Greifer und dem angegebenen Load Carrier für jeden der übergebenen Greifpunkte. Eine Kollisionsprüfung mit anderen Objekten oder der Punktwolke ist nicht möglich. Das CollisionCheck-Modul überprüft, ob sich der Greifer in Kollision mit mindestens einem Load Carrier befindet, wenn sich der TCP an der Greifposition befindet. Es können mehrere Load Carrier gleichzeitig getestet werden. Der Griff wird als Kollision markiert, wenn es mit mindestens einem der definierten Load Carriern zu einer Kollision kommen würde.

Das Argument `pre_grasp_offset` (Greif-Offset) kann für eine erweiterte Kollisionsprüfung genutzt werden. Der Greif-Offset P_{off} ist der Offset vom Greifpunkt P_{grasp} zur Vorgreifposition P_{pre} im Koordinatensystem des Greifpunkts (siehe Abb. 5.30). Wenn der Greif-Offset angegeben wird, werden Greifpunkte auch dann als Kollisionen erkannt, wenn der Greifer an einem beliebigen Punkt während der linearen Bewegung zwischen Vorgreifposition und Greifposition in Kollision geraten würde.

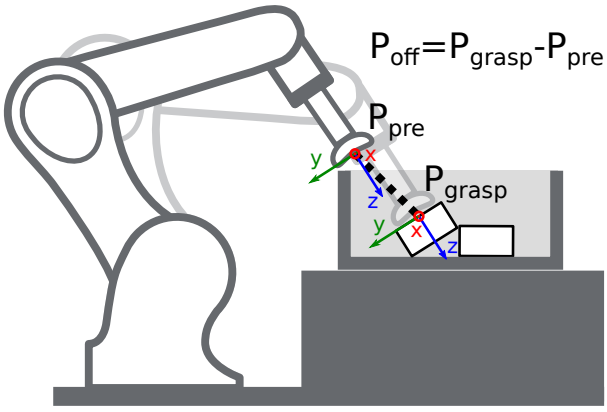


Abb. 5.30: Darstellung des Greif-Offsets für die Kollisionsprüfung. Im dargestellten Fall sind sowohl die Vorgreifposition als auch die Greifposition kollisionsfrei, aber die Trajektorie zwischen diesen Punkten hätte eine Kollision mit dem Load Carrier. Deswegen wird dieser Greifpunkt als Kollision erkannt.

Integrierte Kollisionsprüfung in anderen Modulen

Die Kollisionsprüfung ist in die Services der folgenden Softwaremodule integriert:

- *ItemPick* (Abschnitt 5.4.4): `compute_grasps` (siehe *compute_grasps*, Abschnitt 5.4.4.8
- *BoxPick* (Abschnitt 5.4.5): `compute_grasps` (siehe *compute_grasps*, Abschnitt 5.4.5.9
- *SilhouetteMatch* (Abschnitt 5.4.7): `detect_object` (siehe *detect_object*, Abschnitt 5.4.7.11)

Jedem dieser Services kann ein `collision_detection`-Argument übergeben werden, das aus der ID des Standardgreifers (`gripper_id`) und aus dem Greif-Offset (`pre_grasp_offset`, siehe *Stand-Alone Kollisionsprüfung*, Abschnitt 5.5.2.2) besteht. Der Standardgreifer, der durch das `gripper_id` Argument übergeben wird, wird nur für Greifpunkte verwendet, denen keine individuelle Greifer-ID zugewiesen wurde. Wenn das `collision_detection` Argument übergeben wird, liefern diese Services nur die Greifpunkte zurück, an denen der Greifer nicht in Kollision mit dem erkannten Load Carrier ist, oder für die keine Kollisionsprüfung durchgeführt werden konnte. Wenn eine Load Carrier ID angegeben wurde, wird die Kollisionsprüfung immer mit dem Load Carrier durchgeführt. Zusätzliche Funktionen für die Kollisionsprüfung können abhängig vom Modul aktiviert werden.

Nur in *SilhouetteMatch* (Abschnitt 5.4.7), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im entsprechenden Modul aktiviert ist, werden auch Greifpunkte, bei denen der Greifer in Kollision mit anderen *detektierten* Objekten wäre, herausgefiltert. Dabei ist das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen.

Wenn ein Greifer für einen Greifpunkt in einem Template von *SilhouetteMatch* (Abschnitt 5.4.7) definiert ist, wird dieser Greifer und nicht der im `collision_detection` Argument des `detect_object` Services

angegebene Greifer zur Kollisionsprüfung dieses Greifpunkts verwendet (siehe *Setzen von Greifpunkten*, Abschnitt 5.4.7.4). Die vom `detect_object` Service zurückgelieferten Greifpunkte enthalten ein Flag `collision_checked`, das angibt ob der jeweilige Greifpunkt auf Kollisionen geprüft wurde, und das Feld `gripper_id`. Wenn `collision_checked` true ist, enthält das Feld `gripper_id` die ID des Greifers, der für die Kollisionsprüfung dieses Greifpunkts verwendet wurde. Dies ist die ID des Greifers, der für den jeweiligen Greifpunkt definiert wurde, oder, falls kein Greifer definiert wurde, die ID des Greifers die im `collision_detection` Argument des Serviceaufrufs angegeben wurde. Wenn `collision_checked` false ist, enthält `gripper_id` die ID des Greifers, die für den Greifpunkt definiert wurde.

In *SilhouetteMatch*, Abschnitt 5.4.7 werden Kollisionen zwischen dem Greifer und der Basisebene geprüft, wenn der Parameter `check_collisions_with_base_plane` in SilhouetteMatch aktiviert ist.

Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke können geprüft werden, wenn der Parameter `check_collisions_with_point_cloud` im jeweiligen Modul aktiviert ist.

Warnung: Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch oder anderen Objekten. Nur wenn `check_collisions_with_point_cloud` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur in Kombination mit *SilhouetteMatch* (Abschnitt 5.4.7), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und den anderen *detektierten* Objekten geprüft. Kollisionen mit Objekten, die nicht detektiert werden können, werden nicht geprüft.

Die Kollisionsprüfung wird von Laufzeitparametern beeinflusst, die weiter unten aufgeführt und beschrieben werden.

5.5.2.3 Parameter

Das CollisionCheck-Modul wird in der REST-API als `rc_collision_check` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Konfiguration* → *CollisionCheck* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) ändern.

Übersicht der Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.52: Applikationsspezifische Laufzeitparameter des `rc_collision_check` Moduls

Name	Typ	Min	Max	Default	Beschreibung
`check_bottom`	bool	false	true	true	Aktiviert die Kollisionsprüfung mit dem Boden des Load Carriers.
`check_flange`	bool	false	false	true	Bestimmt, ob ein Greifpunkt als Kollision erkannt wird, sobald der Flansch innerhalb des Load Carriers ist.
`collision_dist`	float64	0.0	0.1	0.01	Minimaler Abstand in Metern zwischen einem Greiferelement und dem Load Carrier und/oder der Basisebene (nur SilhouetteMatch) für einen kollisionsfreien Griff.
`pointcloud_watertight`	bool	false	true	true	Ob ein wasserdichtes Disparitätsbild für die Kollisionsprüfung mit der Punktwolke verwendet werden soll

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile in der Web GUI im Abschnitt *Einstellungen* unter *Konfiguration* → *CollisionCheck* repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben:

`collision_dist` (*Sicherheitsabstand*)

Minimaler Abstand in Metern zwischen einem Greiferelement und dem Load Carrier und/oder der Basisebene (nur SilhouetteMatch) für einen kollisionsfreien Griff.

Bemerkung: Der Sicherheitsabstand wird nicht für die Kollisionsprüfung zwischen dem Greifer und der Punktwolke, oder dem Greifer und anderen detektierten Objekten angewendet. Er wird auch nicht verwendet um zu prüfen, ob sich der Flansch innerhalb des Load Carriers befindet (`check_flange`).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_collision_check/parameters?collision_
↪dist=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?collision_dist=<value>
```

`check_flange` (*Flansch-Check*)

Ermöglicht einen Sicherheitscheck mit dem Flansch, wie in *Flanschradius* (Abschnitt 5.6.4.2) beschrieben. Wenn dieser Parameter gesetzt ist, gelten alle Griffe, bei denen der Flansch innerhalb des Load Carriers wäre, als Kollisionen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_collision_check/parameters?check_flange=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_flange=<value>
```

`check_bottom` (*Boden-Check*)

Wenn dieser Check aktiviert ist, werden Kollisionen nicht nur mit den Load Carrier Wänden, sondern auch mit dem Boden geprüft. Falls der TCP innerhalb der Kollisionsgeometrie (z.B. innerhalb des Sauggreifers) liegt, ist es möglicherweise nötig, diesen Check zu deaktivieren.

Der Load Carrier Boden wird immer ausgenommen von der Kollisionsprüfung zwischen dem Objekt im Greifer und dem Load Carrier während der Entnahme in Kombination mit *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5), wenn `check_collisions_during_retraction` aktiv ist.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_collision_check/parameters?check_bottom=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_bottom=<value>
```

`pointcloud_watertight` (*Wasserdichte Punktwolke*)

Wenn diese Option aktiv ist, wird die Punktwolke für die Kollisionsprüfung wasserdicht gemacht. In einer wasserdichten Punktwolke werden Lücken im Disparitätsbild durch gültige Messungen benachbarter Pixel interpoliert, sodass die resultierende Punktwolke keine Löcher mehr aufweist. Dies führt zu konservativen Ergebnissen bei der Kollisionsprüfung.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_collision_check/parameters?pointcloud_
↪watertight=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?pointcloud_watertight=
↪<value>
```

5.5.2.4 Statuswerte

Statuswerte des `rc_collision_check`-Moduls:

Tab. 5.53: Statuswerte des `rc_collision_check`-Moduls

Name	Beschreibung
`last_evaluated_grasps`	Anzahl der ausgewerteten Griffe
`last_collision_free_grasps`	Anzahl der kollisionsfreien Griffe
`collision_check_time`	Laufzeit der Kollisionsprüfung

5.5.2.5 Services

Die angebotenen Services von `rc_collision_check` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der *rc_visard Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

Das CollisionCheck-Modul stellt folgende Services zur Verfügung.

`reset_defaults`

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_collision_check/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`check_collisions` (veraltet)

löst eine Kollisionsprüfung zwischen dem Greifer und einem Load Carrier aus.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_collision_check/services/check_collisions
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/check_collisions
```

Request

Obligatorische Serviceargumente:

`grasps`: Liste von Griffen, die überprüft werden sollen.

`load_carriers`: Liste von Load Carriern, die auf Kollisionen überprüft werden sollen. Die Felder der Load Carrier Definition sind in *Erkennung von Load Carriern* (Abschnitt 5.4.2.2) beschrieben. Die Griffe und die Load Carrier Positionen müssen im selben Koordinatensystem angegeben werden.

`gripper_id`: Die ID des Greifers, der in der Kollisionsprüfung verwendet werden soll. Der Greifer muss zuvor konfiguriert worden sein.

Optionale Serviceargumente:

`pre_grasp_offset`: Der Greif-Offset in Metern vom Greifpunkt zur Vorgreifposition. Wird ein Greif-Offset angegeben, dann wird die Kollisionsprüfung auf der gesamten linearen Trajektorie von der Vorgreifposition bis zur Greifposition durchgeführt.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasps": [
      {
        "pose": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "gripper_id": "string",
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        }
      }
    ],
    "pre_grasp_offset": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  }
}
```


Response

`colliding_grasps`: Liste von Griffen, die in Kollision mit einem oder mehreren Load Carriern sind.

`collision_free_grasps`: Liste von kollisionsfreien Griffen.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "check_collisions",
  "response": {
    "colliding_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "collision_free_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


`set_gripper` (veraltet)

konfiguriert und speichert einen Greifer auf dem *rc_visard*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_gripper* (Abschnitt 5.6.4.3) in `rc_gripper_db`.

API Version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/set_gripper
```

Die Definitionen von Request und Response sind dieselben wie in *set_gripper* (Abschnitt 5.6.4.3) in `rc_gripper_db` beschrieben.

`get_grippers` (veraltet)

gibt die mit `gripper_ids` spezifizierten und gespeicherten Greifer zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_grippers* (Abschnitt 5.6.4.3) in `rc_gripper_db`.

API Version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/get_grippers
```

Die Definitionen von Request und Response sind dieselben wie in *get_grippers* (Abschnitt 5.6.4.3) in `rc_gripper_db` beschrieben.

`delete_grippers` (veraltet)

löscht die mit `gripper_ids` spezifizierten, gespeicherten Greifer.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_grippers* (Abschnitt 5.6.4.3) in `rc_gripper_db`.

API Version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/delete_grippers
```

Die Definitionen von Request und Response sind dieselben wie in *delete_grippers* (Abschnitt 5.6.4.3) in `rc_gripper_db` beschrieben.

5.5.2.6 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 5.54: Fehlercodes des CollisionCheck-Services

Code	Beschreibung
0	Erfolgreich
-1	Ein ungültiges Argument wurde übergeben.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-9	Lizenz für CollisionCheck ist nicht verfügbar.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Greifern überschritten wurde.
10	Die maximal speicherbare Anzahl an Greifern wurde erreicht.
11	Bestehender Greifer wurde überschrieben.

5.5.3 Kamerakalibrierung

Das Kamerakalibrierungsmodul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

Um die Kamera als Messinstrument zu verwenden, müssen die Kameraparameter, wie die Brennweite, die Objektivverzeichnung und die Lage der Kameras zueinander, genau bekannt sein. Diese Parameter werden durch Kalibrierung bestimmt und für die Rektifizierung der Bilder, die Grundlage für alle anderen Bildverarbeitungsmodule ist, verwendet (siehe *Rektifizierung*, Abschnitt 5.1.2).

Der *rc_visard* ist bereits ab Werk kalibriert. Nichtsdestotrotz kann es vorkommen, dass die Kalibrierung überprüft und neu durchgeführt werden muss, wenn der *rc_visard* einer starken mechanischen Beanspruchung ausgesetzt war.

Mit dem Kamerakalibrierungsmodul lassen sich die Kalibrierungsüberprüfung und Kalibrierung vornehmen.

5.5.3.1 Selbstkalibrierung

Im Kamerakalibrierungsmodul läuft im Hintergrund automatisch der Selbstkalibriermodus mit niedriger Frequenz. In diesem Modus überwacht der *rc_visard* die Ausrichtung der Bildzeilen beider rektifizierten Bilder. Wirken mechanische Kräfte auf den *rc_visard* ein, wird er beispielsweise fallen gelassen, kann dies zu einer Fehlausrichtung führen. Kommt es zu einem erheblichen Ausrichtungsfehler, wird dieser automatisch korrigiert. Nach einem Neustart und einer Korrektur wird der aktuelle Kalibrierungsversatz in der Logdatei des Kameramoduls erfasst (siehe *Download der Logdateien*, Abschnitt 7.5):

> *„rc_stereocalib: Current self-calibration offset is 0.00, update counter is 0“*

Der Aktualisierungszähler (update counter) wird nach jeder automatischen Korrektur um eins erhöht. Nach einer manuellen Neukalibrierung des *rc_visard* wird der Zähler auf 0 zurückgesetzt.

Unter normalen Umständen, wenn der *rc_visard* keiner mechanischen Belastung ausgesetzt ist, dürfte die Selbstkalibrierung des *rc_visard* nicht auftreten. Die Selbstkalibrierung erlaubt dem *rc_visard*, auch nach Erkennung einer falschen Ausrichtung normal zu arbeiten, da diese automatisch korrigiert wird. Dessen ungeachtet wird empfohlen, die Kamera manuell neu zu kalibrieren, wenn der Aktualisierungszähler nicht auf 0 steht. Die Web GUI zeigt eine Warnung an, wenn Selbstkalibrierung stattgefunden hat.

5.5.3.2 Kalibriervorgang

Die Kamerakalibrierung kann über die *Web GUI* (Abschnitt 6.1) unter *Konfiguration* → *Kamera Kalibrierung* vorgenommen werden. Diese Seite bietet einen Assistenten, der den Benutzer durch den Kalibriervorgang führt.

Bemerkung: Die Kamerakalibrierung ist für den *rc_visard* in aller Regel nicht nötig, da er bereits ab Werk kalibriert ist. Eine Neukalibrierung ist nur erforderlich, wenn das Gerät einer starken mechanischen Belastung ausgesetzt war, weil es beispielsweise fallen gelassen wurde.

Während der Kalibrierung muss das Kalibriermuster in verschiedenen Posen erkannt werden. Dabei müssen alle schwarzen Quadrate des Musters müssen komplett in beiden Kameras sichtbar sein und dürfen nicht verdeckt werden. Jedes korrekt erkannte Quadrat wird mit einem grünen Haken belegt. Das Muster kann nur dann korrekt erkannt werden, wenn alle schwarzen Quadrate erkannt werden. Werden einige der Quadrate nicht oder nur für kurze Zeit erkannt, so kann dies an schlechten Lichtverhältnissen oder einem beschädigten Kalibriermuster liegen. Quadrate, die in überbelichteten Bereichen des Kalibriermusters liegen, werden rot hervorgehoben. In diesem Fall müssen die Beleuchtung oder die Belichtungseinstellungen angepasst werden. Ein dicker grüner Rahmen um das Kalibriermuster zeigt an, dass das Muster korrekt in beiden Kamerabildern erkannt wurde.

Kalibriereinstellungen

Die Qualität der Kamerakalibrierung hängt stark von der Qualität des Kalibriermusters ab. Kalibriermuster können von Roboception bezogen werden.

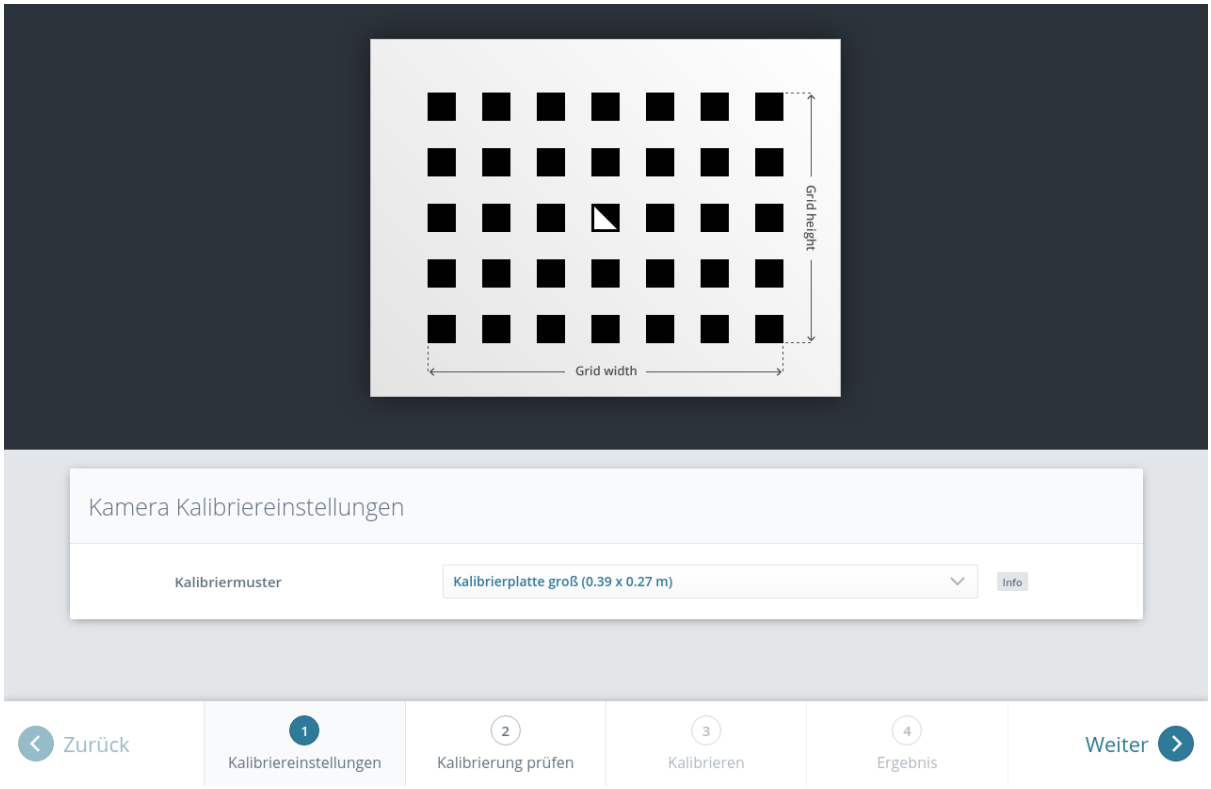


Abb. 5.31: Kalibriereinstellungen

Im ersten Schritt muss das verwendete Kalibriermuster angegeben werden. Mit Klick auf *Weiter* gelangt der Benutzer zum nächsten Schritt.

Kalibrierung prüfen

In diesem Schritt kann die aktuelle Kalibrierung überprüft werden. Um diese Prüfung vorzunehmen, muss das Muster so gehalten werden, dass es sich gleichzeitig im Sichtfeld beider Kameras befindet. Nachdem das Muster vollständig erkannt wurde, wird der Kalibrierfehler automatisch berechnet und das Ergebnis auf dem Bildschirm angegeben.

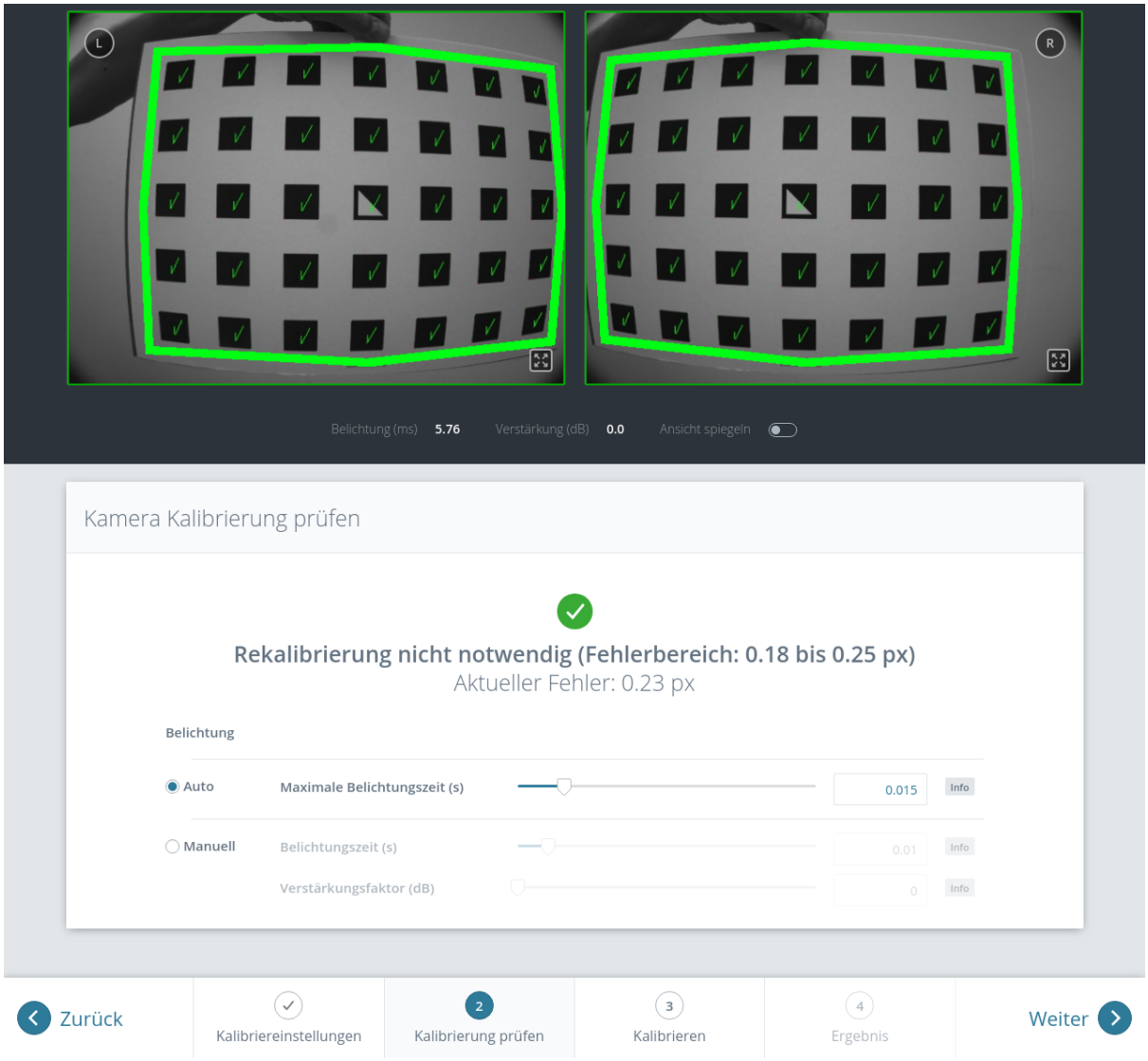


Abb. 5.32: Überprüfung der Kalibrierung

Bemerkung: Um einen aussagekräftigen Kalibrierfehler berechnen zu können, muss das Muster so nah wie möglich an die Kameras gehalten werden. Bedeckt das Muster lediglich einen kleinen Bereich der Kamerabilder, ist der Kalibrierfehler grundsätzlich geringer als wenn das Muster das gesamte Bild ausfüllt. Aus diesem Grund werden zusätzlich zum Kalibrierfehler an der aktuellen Position des Kalibriermusters auch der minimale und maximale Fehler während der Überprüfung der Kalibrierung angezeigt.

Der typische Kalibrierfehler beläuft sich auf unter 0,2 Pixel. Liegt der Fehler in diesem Bereich, kann der Kalibriervorgang übersprungen werden. Ist der errechnete Kalibrierfehler jedoch größer, sollte eine Neukalibrierung vorgenommen werden, um sicherzustellen, dass der Sensor volle Leistung erbringt. Mit Klick auf *Weiter* gelangt der Benutzer zum nächsten Schritt.

Warnung: Große Kalibrierfehler können durch falsch kalibrierte Kameras, ein unpräzises Kalibriermuster oder eine falsch eingetragene Musterbreite oder Musterhöhe verursacht werden. Bei der Verwendung eines benutzerdefinierten Kalibriermusters muss sichergestellt werden, dass das Muster präzise und die angegebenen Breiten- und Höhendaten korrekt sind. Anderenfalls kann die manuelle Kalibrierung sogar dazu führen, dass die Kameras dekalibriert werden!

Kalibrieren

Bevor die Kalibrierung vorgenommen wird, sollte die Belichtungszeit der Kamera richtig eingestellt werden. Um ein gutes Kalibrierergebnis zu erzielen, sollten die Bilder gut belichtet und Bewegungsunschärfe vermieden werden. Die maximale Belichtungszeit im automatischen Modus sollte so klein wie möglich sein, aber dennoch eine gute Belichtung ermöglichen. Die aktuelle Belichtungszeit wird, wie in Abb. 5.34 gezeigt, unter den Kamerabildern angegeben.

Für eine vollständige Kalibrierung müssen zunächst beide Kameras einzeln intrinsisch kalibriert werden (Monokalibrierung). Anschließend wird durch die Stereokalibrierung die Ausrichtung der beiden Kameras zueinander bestimmt. In den meisten Fällen wird die intrinsische Kalibrierung der beiden Kameras nicht beeinträchtigt. Daher wird die Monokalibrierung standardmäßig bei einer Neukalibrierung übersprungen, kann aber durch Klick auf *Monokalibrierung durchführen* durchgeführt werden. Dies sollte nur geschehen, wenn das Ergebnis der Stereokalibrierung nicht zufriedenstellend ist.

Stereokalibrierung

Bei der Stereokalibrierung wird die relative Rotation und Translation der Kameras zueinander ermittelt.

Die Kamerabilder können auch gespiegelt angezeigt werden, um die korrekte Ausrichtung des Kalibriermusters zu vereinfachen.

Als erstes muss das Kalibriermuster möglichst ruhig und so nah wie möglich an die Kamera gehalten werden. Es muss vollständig in beiden Bildern sichtbar sein und die Kameras sollten senkrecht auf das Kalibriermuster gerichtet sein. Wenn das Kalibriermuster nicht senkrecht zur Sichtachse der Kameras ausgerichtet ist, erscheinen kleine grüne Pfeile auf dem Kamerabild, die auf die erwarteten Positionen der Ecken des Kalibriermusters zeigen (siehe Abb. 5.33).

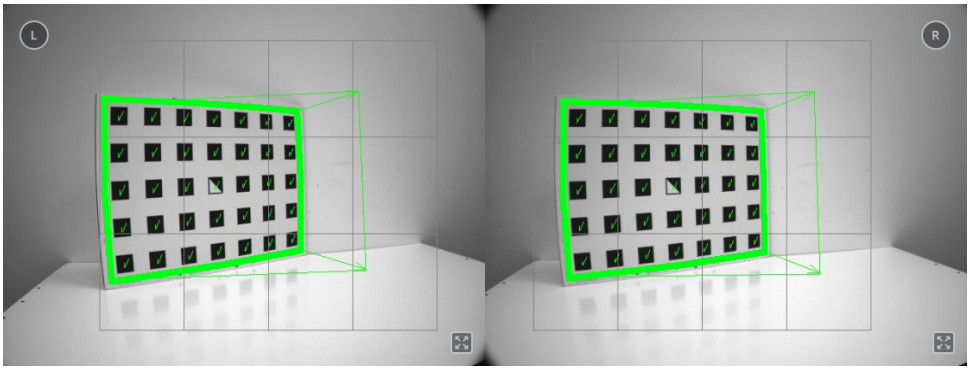


Abb. 5.33: Pfeile weisen darauf hin, wenn das Muster während der Stereokalibrierung nicht senkrecht zur Blickrichtung der Kamera gehalten wird.

Das Muster muss für die Erkennung sehr ruhig gehalten werden. Wenn Bewegungsunschärfe auftritt, wird das Muster nicht erkannt. Alle Zellen, die im Kamerabild dargestellt sind, müssen vom Kalibriermuster abgedeckt werden. Dies wird durch eine grüne Füllung der erfassten Zellen dargestellt (siehe Abb. 5.34).

Beim *rc_visard* können alle Zellen mit einer einzigen Aufnahme erfasst werden, wenn das Kalibriermuster nah genug gehalten wird.



Abb. 5.34: Stereokalibrierung: Das Muster sollte so nah wie möglich gehalten werden, um alle dargestellten Zellen zu füllen.

Bemerkung: Falls alle Häkchen auf dem Kalibriermuster verschwinden, liegt dies daran, dass die Kamerablickrichtung nicht senkrecht zum Muster steht, oder das Muster zu weit von der Kamera entfernt ist.

Sobald alle Zellen erfasst und gefüllt sind, verschwinden sie und eine einzelne entfernte Zelle wird angezeigt. Nun muss das Kalibriermuster so weit entfernt wie möglich gehalten werden, damit die kleine Zelle erfasst wird. Pfeile zeigen an, falls das Muster noch zu nah an der Kamera ist. Wenn das Kalibriermuster erfolgreich detektiert wurde, wird die Zelle grün und das Kalibrierergebnis kann berechnet werden (siehe Abb. 5.35).

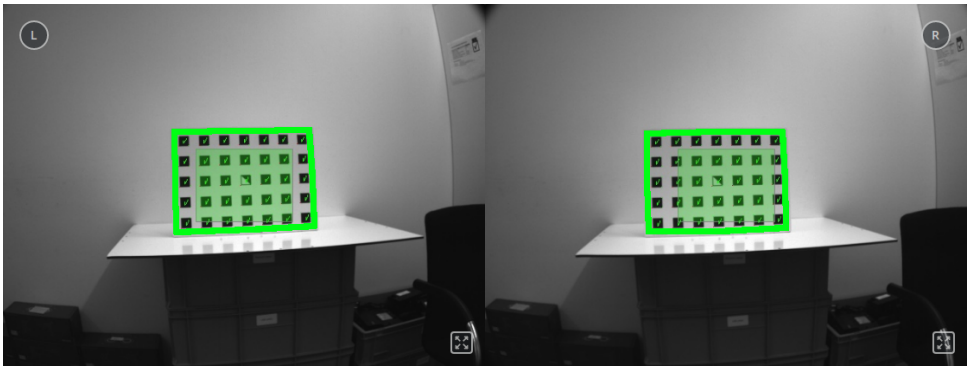


Abb. 5.35: Positionierung des Musters so entfernt wie möglich während der Stereokalibrierung

Führt die Stereokalibrierung nicht zu einem akzeptablen Kalibrierfehler, sollte die Kalibrierung erneut vorgenommen werden, jedoch mit Monokalibrierung (siehe nächster Abschnitt *Monokalibrierung*).

Monokalibrierung

Monokalibrierung ist die intrinsische Kalibrierung jeder einzelnen Kamera. Da die intrinsische Kalibrierung in der Regel nicht beeinträchtigt wird, sollte die Monokalibrierung nur durchgeführt werden, wenn das Ergebnis der Stereokalibrierung nicht zufriedenstellend ist.

Durch Klicken auf *Monokalibrierung durchführen* im Reiter *Kalibrieren* kann die Monokalibrierung gestartet werden.

Zur Kalibrierung muss das Kalibriermuster in verschiedenen Ausrichtungen vor die Kamera gehalten werden. Die Pfeile, die von den Ecken des Musters bis zu den grünen Bildschirmbereichen führen, geben an, dass alle Musterecken innerhalb der grünen Rechtecke platziert werden müssen. Diese grünen Rechtecke sind sensible Bereiche. Mit dem Schieberegler *Größe der sensiblen Bereiche* lässt sich die Größe der Rechtecke einstellen, um die Kalibrierung zu vereinfachen. Es ist jedoch zu bedenken, dass die Größe nicht zu stark erhöht werden darf, da dies auf Kosten der Kalibriergenauigkeit gehen kann.

Häufig wird der Fehler begangen, das Muster bei der Kalibrierung falsch herum zu halten. Dieser Fehler lässt sich leicht erkennen, da sich die von den Musterecken zu den grünen Rechtecken verlaufenden Linien in diesem Fall kreuzen (siehe Abb. 5.36).

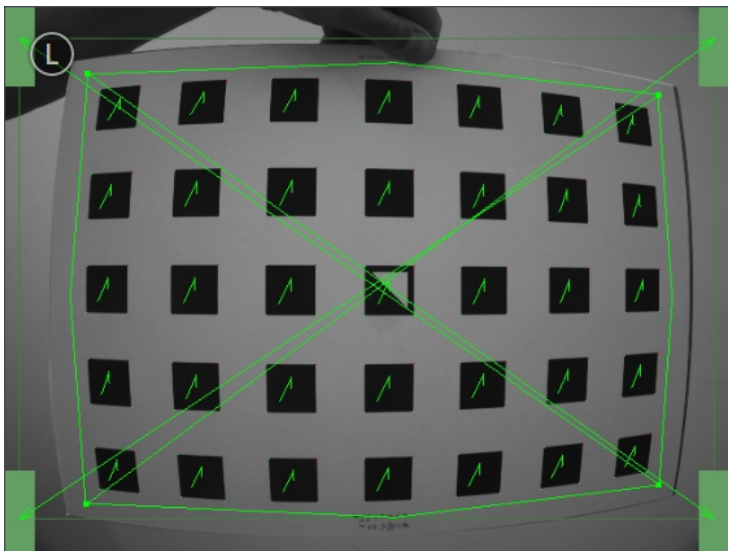


Abb. 5.36: Wird das Kalibriermuster falsch herum gehalten, kreuzen sich die grünen Linien.

Bemerkung: Die Kalibrierung mag umständlich erscheinen, da das Muster hierfür in bestimmten vordefinierten Stellungen gehalten werden muss. Dieses Vorgehen ist jedoch notwendig um ein qualitativ hochwertiges Kalibrierergebnis zu erreichen.

Für den Prozess der Monokalibrierung ist das Kalibriermuster für beide Kameras in den in Abb. 5.37 angegebenen Stellungen zu halten.

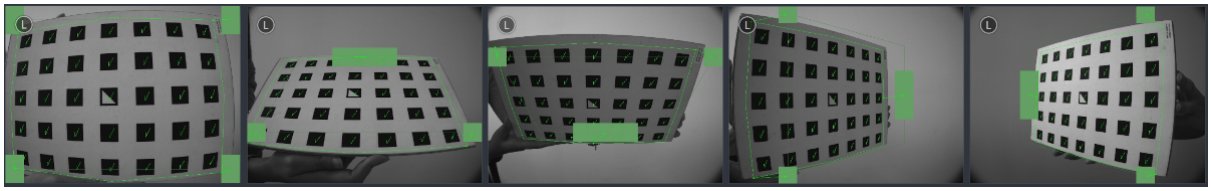


Abb. 5.37: Musterposen für die Monokalibrierung

Nachdem die Ecken oder Seiten des Kalibriermusters auf die sensiblen Bereiche ausgerichtet wurden, zeigt der Kalibriervorgang automatisch die nächste Stellung an. Sobald der Prozess für die linke Kamera abgeschlossen ist, ist er ebenso für die rechte Kamera zu wiederholen.

Anschließend folgen sind die Schritte im vorherigen Abschnitt *Stereokalibrierung* zu befolgen.

Kalibrierergebnis speichern

Mit Klick auf die Schaltfläche *Kalibrierung berechnen* wird der Kalibriervorgang beendet und das Endergebnis angezeigt. Der eingeblendete Wert ist der mittlere Reprojektionsfehler aller Kalibrierpunkte. Er ist in Pixeln angegeben und beläuft sich typischerweise auf einen Wert von unter 0,2.

Mit Klick auf *Kalibrierung speichern* wird das Kalibrierergebnis übernommen und auf dem Gerät gespeichert.

Bemerkung: Das eingeblendete Ergebnis ist der nach der Kalibrierung bestehende Mindestfehler. Der reale Fehler liegt auf keinen Fall darunter, könnte theoretisch jedoch höher sein. Dies gilt für jeden Algorithmus zur Kamerakalibrierung und ist der Grund dafür, warum das Kalibriermuster in verschiedenen Positionen vor den Sensor zu halten ist. So ist sichergestellt, dass der reale Kalibrierfehler den errechneten Fehler nicht signifikant überschreitet.

Warnung: War vor der Durchführung der Kamerakalibrierung eine Hand-Auge-Kalibrierung auf dem *rc_visard* gespeichert, so sind die Werte der Hand-Auge-Kalibrierung möglicherweise ungültig geworden. Daher ist das Hand-Auge-Kalibrierverfahren zu wiederholen.

5.5.3.3 Parameter

Dieses Modul wird in der REST-API als `rc_stereocalib` bezeichnet.

Bemerkung: Die verfügbaren Parameter und die Statuswerte des Moduls zur Kamerakalibrierung sind nur für den internen Gebrauch bestimmt und können ohne vorherige Ankündigung Änderungen unterzogen werden. Die Kalibrierung sollte gemäß den vorstehenden Anweisungen und ausschließlich in der Web GUI vorgenommen werden.

5.5.3.4 Services

Bemerkung: Die verfügbaren Services des Moduls zur Kamerakalibrierung sind lediglich für den internen Gebrauch bestimmt und können ohne vorherige Ankündigung Änderungen unterzogen werden. Die Kalibrierung sollte gemäß den vorstehenden Anweisungen und ausschließlich in der Web GUI vorgenommen werden.

5.5.4 IOControl und Projektor-Kontrolle

Das IOControl Modul ist ein Basismodul, welches auf jedem *rc_visard* läuft.

Das IOControl-Modul ermöglicht das Lesen der digitalen Eingänge und die Kontrolle der digitalen Ausgänge (GPIOs) der Kamera. Die Ausgänge können auf *aus* (LOW) oder *an* (HIGH) gesetzt werden. Sie können auch so konfiguriert werden, dass sie genau für die Belichtungszeit jedes Bildes, oder auch nur jedes zweiten Bildes, *an* sind.

Das IOControl-Modul dient der Ansteuerung einer externen Lichtquelle oder eines Projektors, der an einen der GPIO-Ausgänge der Kamera angeschlossen wird, und der mit der Bildaufnahme synchronisiert ist. Für den Fall, dass ein Musterprojektor für die Verbesserung des Stereo-Matchings verwendet wird, ist das projizierte Muster auch in den Intensitätsbildern sichtbar. Das kann für Bildverarbeitungs-Anwendungen, die auf dem Intensitätsbild basieren (z.B. Kantendetektion), von Nachteil sein. Aus diesem Grund erlaubt das IOControl-Modul auch das Setzen der Ausgänge für nur jedes zweite Kamerabild. Somit sind auch Intensitätsbilder ohne projiziertes Muster verfügbar.

Bemerkung: Details über die GPIOs des *rc_visard* werden in *Verkabelung*, Abschnitt 3.5 gegeben.

5.5.4.1 Parameter

Das IOControl-Modul wird in der REST-API als `rc_iocontrol` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Konfiguration* → *IOControl* dargestellt.

Der Benutzer kann die Parameter über die Web GUI, die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.2), oder die GenICam-Schnittstelle mit den DigitalIOControl-Parametern `LineSelector` und `LineSource` ändern (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.7).

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.55: Laufzeitparameter des `rc_iocontrol`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`out1_mode`	string	-	-	Low	Out1 mode: [Low, High, ExposureActive, ExposureAlternateActive]
`out2_mode`	string	-	-	Low	Out2 mode: [Low, High, ExposureActive, ExposureAlternateActive]

Beschreibung der Laufzeitparameter

`out1_mode` und `out2_mode` (*Ausgang 1 (Out1)* / *Projektor* und *Ausgang 2 (Out2)*)

Die Betriebsarten für GPIO-Ausgang 1 und GPIO-Ausgang 2 können individuell gesetzt werden:

`Low` schaltet den GPIO-Ausgang permanent *aus* (LOW). Das ist die Standardeinstellung.

`High` schaltet den GPIO-Ausgang permanent *an* (HIGH).

`ExposureActive` schaltet den GPIO-Ausgang für die Belichtungszeit jedes Bildes *an* (HIGH).

`ExposureAlternateActive` schaltet den GPIO-Ausgang für die Belichtungszeit jedes zweiten Bildes *an* (HIGH).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_iocontrol/parameters/parameters?<out1_
↪mode|out2_mode>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_mode|out2_mode>=<value>
```

Abb. 5.38 zeigt, welche Bilder für das Stereo-Matching und die GigE Vision-Übertragung in der Betriebsart `ExposureActive` mit einer benutzerdefinierten Bildwiederholrate von 8 Hz benutzt werden.

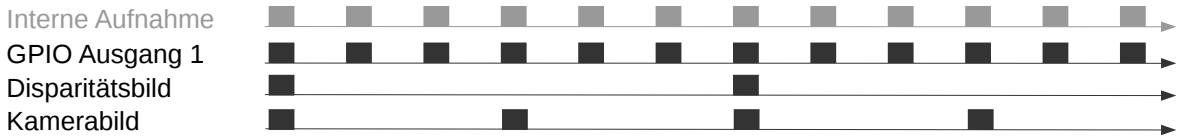


Abb. 5.38: Beispiel für die Nutzung der Betriebsart `ExposureActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes Bildes *an* (HIGH). Disparitätsbilder werden für Kamerabilder berechnet, die auch per GigE Vision in der benutzerdefinierten Bildwiederholrate versendet werden.

Die Betriebsart `ExposureAlternateActive` ist gedacht, um einen externen Musterprojektor anzusteuern, der am GPIO-Ausgang 1 der Kamera angeschlossen ist. In diesem Fall nutzt das *Stereo-Matching-Modul* (Abschnitt 5.2) nur Bilder, bei denen GPIO-Ausgang 1 *an* (HIGH) ist, d.h. der Projektor ist an. Die maximale Bildwiederholrate, welche für das Stereo-Matching genutzt wird, ist hierbei die halbe vom Benutzer konfigurierte Bildwiederholrate. Alle Module, die Intensitätsbilder benutzen, wie z.B. *TagDetect* (Abschnitt 5.4.3) und *ItemPick* (Abschnitt 5.4.4), benutzen die Intensitätsbilder, bei denen der GPIO-Ausgang 1 *aus* (LOW) ist, d.h. der Projektor ist aus. Abb. 5.39 zeigt ein Beispiel.

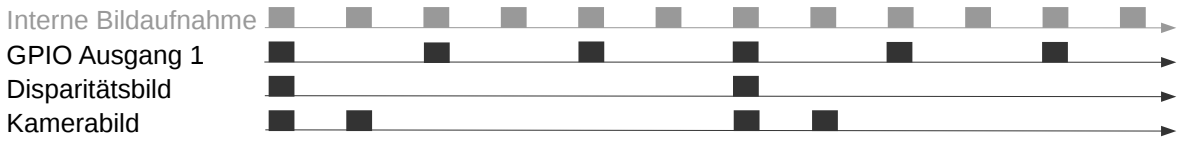


Abb. 5.39: Beispiel für die Nutzung der Betriebsart `ExposureAlternateActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes zweiten Bildes *an* (HIGH). Disparitätsbilder werden für Kamerabilder berechnet, bei denen GPIO-Ausgang 1 *an* (HIGH) ist, und die auch per GigE Vision in der benutzerdefinierten Bildwiederholrate versendet werden. In der Betriebsart `ExposureAlternateActive` werden Intensitätsbilder immer paarweise versendet: ein Bild mit GPIO-Ausgang 1 *an* (HIGH), für das ein Disparitätsbild verfügbar sein kann, und ein Bild mit GPIO-Ausgang 1 *aus* (LOW).

Bemerkung: In der Betriebsart `ExposureAlternateActive` gibt es zu einem Intensitätsbild mit angeschaltetem GPIO-Ausgang 1 (HIGH), d.h. mit projiziertem Muster, immer in 40 ms Abstand ein Intensitätsbild mit ausgeschaltetem GPIO-Ausgang 1 (LOW), d.h. ohne projiziertes Muster. Dies ist unabhängig von der benutzerdefinierten Bildwiederholrate und sollte in dieser speziellen Betriebsart für die Synchronisierung von Disparitäts- und projektionsfreien Kamerabildern berücksichtigt werden.

Die Funktionalität kann auch über die DigitalIOControl-Parameter der GenICam Schnittstelle kontrolliert werden (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.7).

5.5.4.2 Services

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von 0 quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind.

Dieses Softwaremodul bietet folgende Services.

`get_io_values`

Mit diesem Aufruf kann der aktuelle Zustand der Ein- und Ausgänge (GPIOs) der Kamera abgefragt werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_iocontrol/services/get_io_values
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/get_io_values
```

Request

Dieser Service hat keine Argumente.

Response

Das Feld `timestamp` ist der Zeitpunkt der Messung.

`input_mask` und `output_mask` sind Bitmasken, die definieren, welche Bits für die Werte der Eingänge bzw. Ausgänge verwendet werden.

`values` beinhaltet die Werte der Bits, die zu den in den Bitmasken `input_mask` und `output_mask` definierten Eingängen und Ausgängen gehören.

Das Feld `return_code` enthält mögliche Warnungen oder Fehlercodes und Nachrichten. Mögliche Werte für `return_code` sind in der Tabelle unten angegeben.

Code	Beschreibung
0	Erfolgreich
-2	Interner Fehler
-9	Lizenz für *IOControl* ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_io_values",
  "response": {
    "input_mask": "uint32",
    "inverter_mask": "uint32",
    "output_mask": "uint32",
    "ratio_mask": "uint32",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    },
    "values": "uint32"
  }
}
```


reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_iocontrol/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


5.6 Datenbankmodule

Der *rc_visard* stellt mehrere Datenbankmodule zur Verfügung, die das Konfigurieren von globalen Daten ermöglichen, die in vielen Detektionsmodulen benötigt werden, zum Beispiel Load Carrier und Regions of Interest. Über die *REST-API-Schnittstelle* (Abschnitt 6.2) sind die Datenbankmodule nur in API Version 2 verfügbar.

Die Datenbankmodule sind:

- ***LoadCarrierDB*** **(`rc_load_carrier_db`, Abschnitt 5.6.1)** ermöglicht das Erstellen, Abfragen und Löschen von Load Carriern.
- ***RoiDB*** **(`rc_roi_db` , Abschnitt 5.6.2)** ermöglicht das Erstellen, Abfragen und Löschen von 2D und 3d Regions of Interest.
- ***PrimitiveDB*** **(`rc_primitive_db`, Abschnitt 5.6.3)** ermöglicht das Erstellen, Abfragen und Löschen von geometrischen Primitiven wie Ebenen.
- ***GripperDB*** **(`rc_gripper_db`, Abschnitt 5.6.4)** ermöglicht das Erstellen, Abfragen und Löschen von Greifern für die Kollisionsprüfung.

5.6.1 LoadCarrierDB

5.6.1.1 Einleitung

Das LoadCarrierDB Modul (Load Carrier Datenbank Modul) ermöglicht die globale Definition von Load Carriern (Behältern), die dann in vielen Detektionsmodulen genutzt werden können. Die definierten Load Carrier Modelle sind in allen Modulen auf dem *rc_visard* verfügbar, die Load Carrier unterstützen.

Das LoadCarrierDB Modul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

Tab. 5.56: Spezifikationen des LoadCarrierDB Moduls

Unterstützte Load Carrier Typen	4-seitig oder 3-seitig
Mögliche Rand-Arten	durchgängig, abgestuft oder vorspringend
Min. Load Carrier Abmessungen	0.1 m x 0.1 m x 0.05 m
Max. Load Carrier Abmessungen	5 m x 5 m x 5 m
Max. Anzahl von Load Carriern	50
Load Carrier verfügbar in	*ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5) und *SilhouetteMatch* (Abschnitt 5.4.7)
Mögliche Posen-Arten	keine Pose, Orientierungsprior, exakte Pose
Mögliche Referenzkoordinatensysteme	camera, external

5.6.1.2 Load Carrier Definition

Ein sogenannter Load Carrier ist ein Behälter mit vier Wänden, einem Boden und einem rechteckigen Rand, der Objekte enthalten kann. Er kann genutzt werden, um das Volumen, in dem nach Objekten oder Greifpunkten gesucht wird, zu begrenzen.

Seine Geometrie ist durch die inneren und äußeren Abmessungen (`inner_dimensions` und `outer_dimensions`) definiert. Die maximalen `outer_dimensions` betragen 5.0 m in allen Dimensionen.

Der Ursprung des Load Carrier Koordinatensystems liegt im Zentrum des durch die *Außenmaße* definierten Quaders. Dabei ist die z-Achse senkrecht zum Boden des Load Carriers und zeigt aus dem Load Carrier heraus (siehe Abb. 5.40).

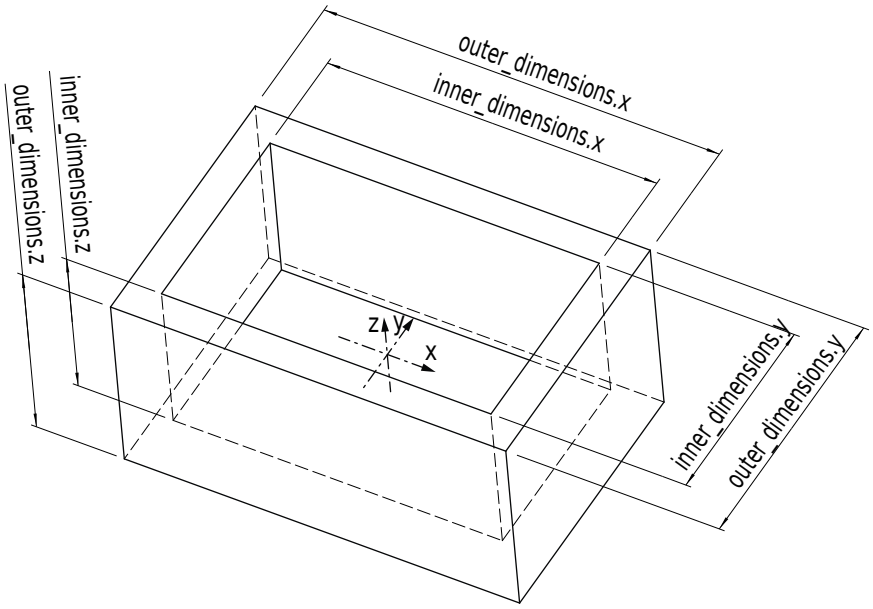


Abb. 5.40: Load Carrier mit Koordinatensystem und inneren und äußeren Abmessungen

Bemerkung: Die Innen- und Außenmaße eines Load Carriers sind typischerweise in den Angaben des jeweiligen Herstellers spezifiziert, und können im Produktblatt oder auf der Produktseite nachgeschlagen werden.

Das Innenvolumen eines Load Carriers ist durch seine Innenmaße definiert, aber enthält zusätzlich einen Bereich von 10 cm oberhalb des Load Carriers, damit Objekte, die aus dem Load Carrier herausragen, auch für die Detektion oder Greifpunktberechnung berücksichtigt werden. Weiterhin wird vom Innenvolumen in jeder Richtung ein zusätzlicher Sicherheitsabstand `crop_distance` abgezogen, welcher als Laufzeitparameter im LoadCarrier Modul konfiguriert werden kann (siehe *Parameter*, Abschnitt 5.4.2.5). Abb. 5.41 zeigt das Innenvolumen eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

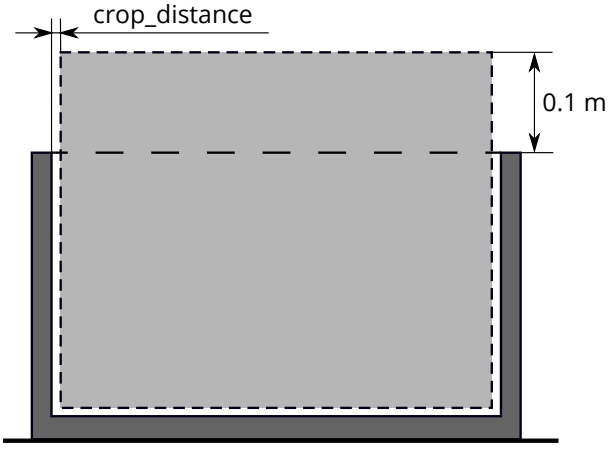


Abb. 5.41: Darstellung des Innenvolumens eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

Da die Erkennung von Load Carriern auf der Erkennung des Load Carrier Rands basiert, muss die Geometrie des Randes angegeben werden, wenn sie nicht aus der Differenz zwischen Außen- und Innenmaßen bestimmt werden kann. Dazu kann die Randstärke `rim_thickness` explizit gesetzt werden. Die Randstärke gibt die Breite des äußeren Rands in x- und y-Richtung an. Wenn eine Randstärke

gesetzt ist, kann optional auch die Randstufenhöhe `rim_step_height` angegeben werden. Die Randstufenhöhe gibt die Höhe der Stufe zwischen dem äußeren und dem inneren Teil des Load Carrier Rands an. Wenn die Stufenhöhe angegeben wird, wird sie auch bei der Kollisionsprüfung berücksichtigt (siehe *CollisionCheck*, Abschnitt 5.5.2). Beispiele abgestufter Load Carrier sind in Abb. 5.42 A, B gezeigt. Zusätzlich zur Randstärke und Randstufenhöhe kann der Randvorsprung `rim_ledge` angegeben werden, um Load Carrier zu definieren, deren innerer Rand in den Load Carrier Innenraum hineinragt, wie zum Beispiel bei Gitterboxen. Der Randvorsprung `rim_ledge` gibt die Randstärke des inneren Teils des Randes in die x- und y-Richtung an. Ein Beispiel eines Load Carriers mit vorspringendem Rand ist in Abb. 5.42 C gezeigt.

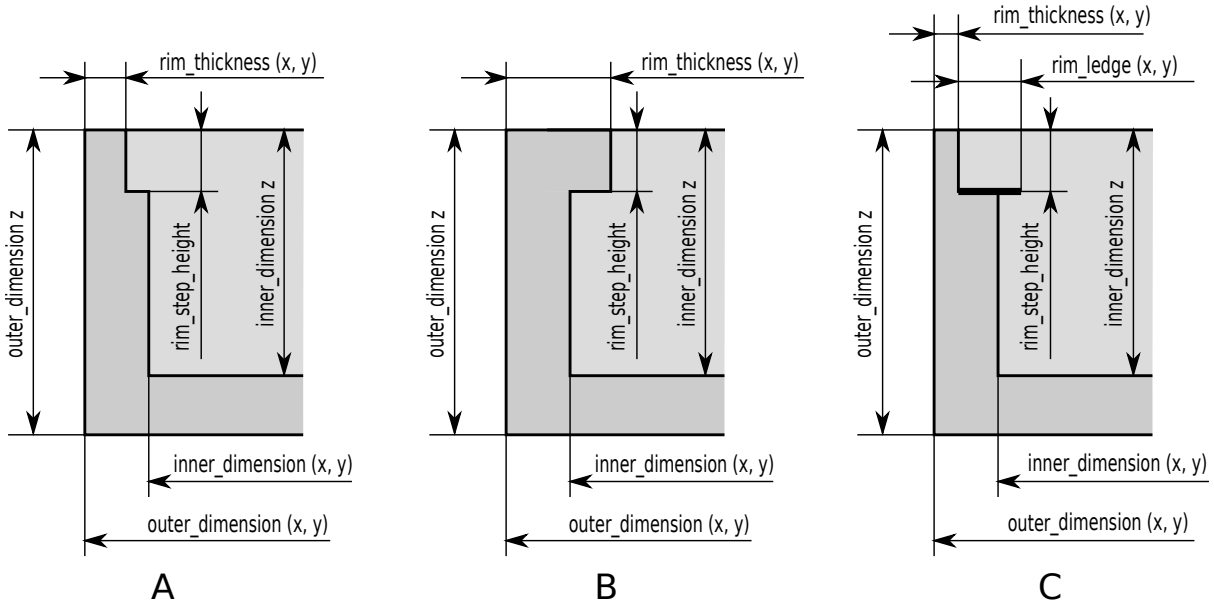


Abb. 5.42: Beispiele von Load Carriern mit abgestuftem (A, B) und vorspringendem Rand (C)

Die unterschiedlichen Randtypen können für gewöhnliche 4-seitige und 3-seitige Load Carrier angewendet werden. Für einen 3-seitigen Load Carrier muss das Feld `type` auf `THREE_SIDED` gesetzt werden. Wenn der Typ `STANDARD` oder leer ist, wird ein 4-seitiger Load Carrier angenommen. Ein 3-seitiger Load Carrier hat eine Seite, die niedriger ist als die anderen drei Seiten. Die Höhe dieser niedrigeren offenen Seite `height_open_side` wird vom äußeren Boden des Load Carriers gemessen. Die offene Seite liegt an der negativen y-Achse des Load Carrier Koordinatensystems. Beispiele der zwei unterschiedlichen Load Carrier Typen sind in Abb. 5.43 zu sehen. Die Höhe der offenen Seite wird nur während der Kollisionsprüfung berücksichtigt und ist nicht notwendig für die Erkennung des Load Carriers.

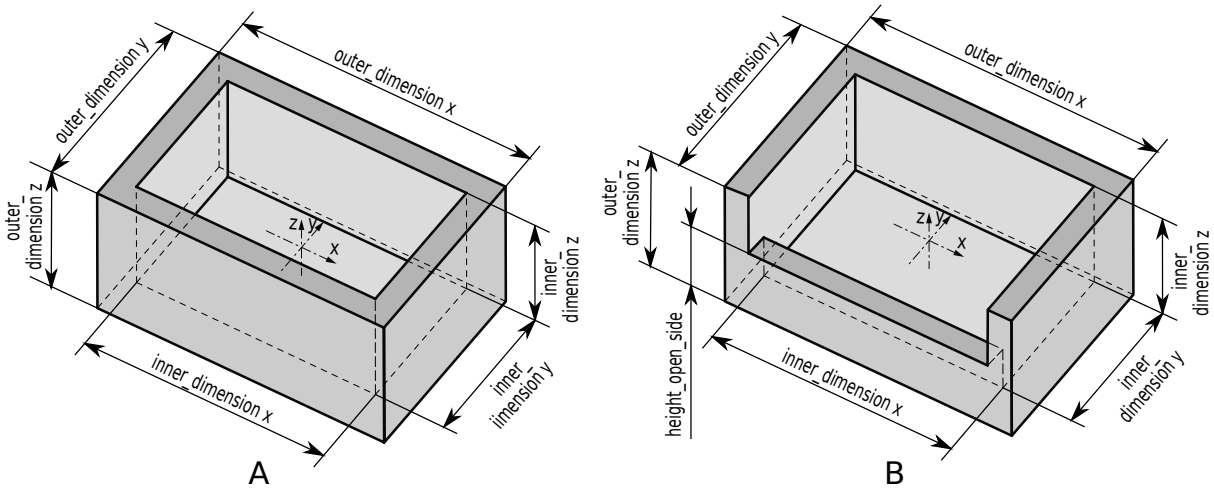


Abb. 5.43: Beispiele eines 4-seitigen (A) und 3-seitigen Load Carriers (B)

Für einen Load Carrier kann eine Pose bestehend aus `position` und `orientation` als Quaternion in einem Referenzkoordinatensystem angegeben werden. Basierend auf dem Posentyp `pose_type` wird diese Pose entweder als Vorgabe für die Load Carrier Orientierung (`pose_type` ist `ORIENTATION_PRIOR` oder leer) oder als exakte Pose (`pose_type` ist `EXACT_POSE`) verwendet.

Falls die angegebene Pose als Vorgabe (Prior) für die Orientierung dient, wird garantiert, dass die zurückgelieferte Pose des erkannten Load Carriers die minimale Rotation bezogen auf den gesetzten Prior hat. Dieser Posentyp ist nützlich für die Erkennung von geneigten Load Carriern, oder um Mehrdeutigkeiten in der x- und y-Richtung aufzulösen, die durch die Symmetrie des Load Carriers verursacht werden.

Falls der Posentyp auf `EXACT_POSE` gesetzt ist, wird keine Load Carrier Erkennung auf den Szenendaten durchgeführt, sondern die angegebene Pose wird so verwendet, als wäre der Load Carrier in dieser Pose in der Szene erkannt worden. Dieser Posentyp ist nützlich, wenn Load Carrier ihre Position nicht verändern und/oder schwer zu erkennen sind (z.B. weil ihr Rand zu schmal ist oder das Material zu stark reflektiert).

Der *rc_visard* erlaubt das Speichern von bis zu 50 verschiedenen Load Carriern, von denen jeder mit einer `id` versehen ist. Die für eine spezifische Anwendung relevanten Load Carrier können mithilfe der *rc_visard* Web GUI oder der *REST-API-Schnittstelle* (Abschnitt 6.2) konfiguriert werden.

Bemerkung: Die konfigurierten Load Carrier sind persistent auf dem *rc_visard* gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

5.6.1.3 Load Carrier Abteile

Bei einigen Detektionsmodulen kann ein Load Carrier Abteil (`load_carrier_compartment`) angegeben werden, um das Volumen für die Erkennung zu begrenzen, zum Beispiel in *ItemPick's compute_grasps Service* (siehe 5.4.4.8). Ein Load Carrier Abteil ist eine Box, deren Pose `pose` als Transformation vom Load Carrier Koordinatensystem in das Abteilkoordinatensystem, welches im Zentrum der durch das Abteil definierten Box liegt, angegeben wird (siehe Abb. 5.44). Das Load Carrier Abteil ist nicht Teil der Load Carrier Definition im LoadCarrierDB Modul, sondern muss für jeden Detektionsaufruf separat definiert werden.

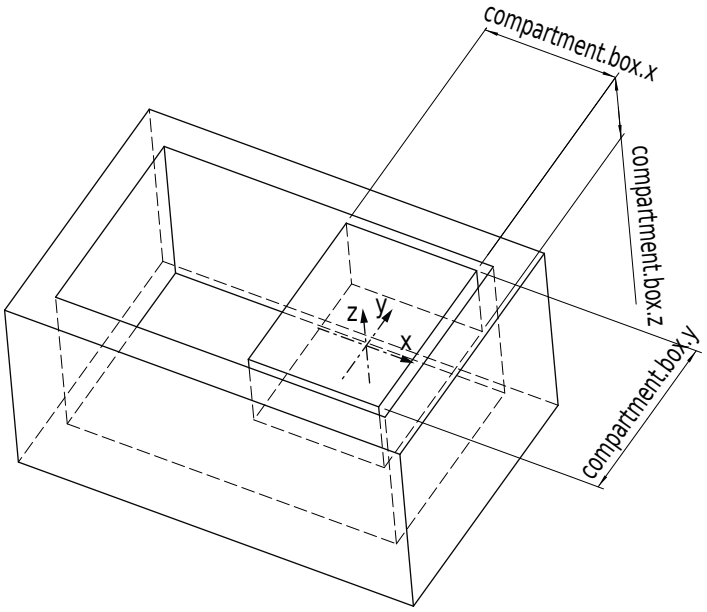


Abb. 5.44: Beispiel für ein Abteil innerhalb eines Load Carriers. Das gezeigte Koordinatensystem das Koordinatensystem des Load Carrier Abteils.

Als Volumen für die Detektion wird der Durchschnitt des Abteil-Volumens und des Load Carrier Innenraums verwendet. Wenn dieser Durchschnitt ebenfalls den Bereich von 10 cm oberhalb des Load Carriers enthalten soll, muss die Höhe der Box, die das Abteil definiert, entsprechend vergrößert werden.

5.6.1.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das LoadCarrierDB Modul oder haben Einfluss auf die Datenverarbeitung.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die exakte Pose oder der Orientierungsprior im Roboterkoordinatensystem angegeben werden, indem das Argument `pose_frame` auf `external` gesetzt wird.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Die Load Carrier Pose oder der Orientierungsprior sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Die Load Carrier Pose oder der Orientierungsprior sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1).

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

5.6.1.5 Services

Das LoadCarrierDB Modul wird in der REST-API als `rc_load_carrier_db` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Datenbank* → *Load Carrier* dargestellt. Die angebotenen Services des LoadCarrierDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der Web GUI ausprobiert und getestet werden.

Das LoadCarrierDB Modul stellt folgende Services zur Verfügung.

`set_load_carrier`

speichert einen Load Carrier auf dem *rc_visard*. Alle Load Carrier sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/set_load_carrier
```

Request

Die Definition des Typs `load_carrier` wird in *Load Carrier Definition* (Abschnitt 5.6.1.2) beschrieben.

Das Feld `type` ist optional und akzeptiert `STANDARD` und `THREE_SIDED`.

Das Feld `pose_type` ist optional und akzeptiert `NO_POSE`, `EXACT_POSE` und `ORIENTATION_PRIOR`.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "load_carrier": {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "pose_type": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_load_carrier",
  "response": {
    "return_code": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_load_carriers

gibt die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier zurück. Wenn keine `load_carrier_ids` angegeben werden, werden alle gespeicherten Load Carrier zurückgeliefert.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/get_load_carriers
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_load_carriers",
  "response": {
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "pose_type": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_load_carriers

löscht die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier. Alle zu löschenden Load Carrier müssen explizit in `load_carrier_ids` angegeben werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/delete_load_carriers
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "delete_load_carriers",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.6.1.6 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 5.57: Rückgabecodes der Services des LoadCarrierDB Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern überschritten wurde.
10	Die maximal speicherbare Anzahl an Load Carriern wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.

5.6.2 RoiDB

5.6.2.1 Einleitung

Das RoiDB Modul (Region of Interest Datenbankmodul) ermöglicht die globale Definition von 2D und 3D Regions of Interest (ROIs), die dann in vielen Detektionsmodulen verwendet werden können. Die definierten ROIs sind in allen Modulen auf dem *rc_visard* verfügbar, die 2D oder 3D ROIs unterstützen.

Das RoiDB Modul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

3D ROIs können in den *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5) Modulen verwendet werden. 2D ROIs werden vom *SilhouetteMatch* (Abschnitt 5.4.7) Modul und dem *LoadCarrier* (Abschnitt 5.4.2) Modul unterstützt.

Tab. 5.58: Spezifikationen des ROIDB Moduls

Unterstützte ROI Typen	2D, 3D
Unterstützte ROI Geometrien	2D ROI: Rechteck, 3D ROI: Box, Kugel
Max. Anzahl von ROIs	2D: 100, 3D: 100
ROIs verfügbar in	2D: *SilhouetteMatch* (Abschnitt 5.4.7), *LoadCarrier* (Abschnitt 5.4.2), 3D: *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5)
Unterstützte Referenzkoordinatensysteme	camera, external

5.6.2.2 Region of Interest

Eine sogenannte Region of Interest (ROI) definiert ein abgegrenztes Raumvolumen (3D ROI, `region_of_interest`) oder eine rechteckige Region im linken Kamerabild (2D ROI, `region_of_interest_2d`), welche für eine spezifische Anwendung relevant sind.

Eine ROI kann das Volumen, in dem ein Load Carrier gesucht wird, einschränken, oder einen Bereich definieren, der nur die zu erkennenden oder zu greifenden Objekte enthält. Die Verarbeitungszeit kann sich deutlich verringern, wenn eine ROI genutzt wird.

Folgende Arten von 3D ROIs (`type`) werden unterstützt:

- `BOX`, für quaderförmige ROIs mit den Abmessungen `box.x`, `box.y`, `box.z`.
- `SPHERE`, für kugelförmige ROIs mit dem Radius `sphere.radius`.

Die Pose `pose` einer 3D ROI kann entweder relativ zum *Kamera*-Koordinatensystem `camera` oder mithilfe der Hand-Auge-Kalibrierung im *externen* Koordinatensystem `external` angegeben werden. Das externe Koordinatensystem steht nur zur Verfügung, wenn eine *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1) durchgeführt wurde. Wenn der Sensor am Roboter montiert ist, und die ROI im externen Koordinatensystem definiert ist, dann muss jedem Detektions-Service, der diese ROI benutzt, die aktuelle Roboterpose übergeben werden.

Eine 2D ROI ist als rechteckiger Teil des linken Kamerabilds definiert und kann sowohl über die *REST-API-Schnittstelle* (Abschnitt 6.2) als auch über die *rc_visard Web GUI* (Abschnitt 6.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Datenbank* gesetzt werden. Die Web GUI bietet hierfür ein einfach zu benutzendes Werkzeug an. Jeder ROI muss ein eindeutiger Name zugewiesen werden, um diese später adressieren zu können.

In der REST-API ist eine 2D-ROI über folgende Werte spezifiziert:

- `id`: Eindeutiger Name der ROI
- `offset_x`, `offset_y`: Abstand in Pixeln von der oberen rechten Bildecke entlang der x- bzw. y-Achse
- `width`, `height`: Breite und Höhe in Pixeln

Der *rc_visard* erlaubt das Speichern von bis zu 100 verschiedenen 3D ROIs und der gleichen Anzahl von 2D ROIs. Die Konfiguration von ROIs erfolgt in der Regel offline während der Einrichtung der gewünschten Anwendung. Die Konfiguration kann mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) des RoiDB Moduls vorgenommen werden, oder über die *rc_visard Web GUI* (Abschnitt 6.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Datenbank*.

Bemerkung: Die erstellten ROIs sind persistent auf dem *rc_visard* gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

5.6.2.3 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das RoiDB Modul oder haben Einfluss auf die Datenverarbeitung.

Hand-Auge Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die Pose einer 3D ROI im Roboterkoordinatensystem angegeben werden, indem das Argument `pose_frame` auf `external` gesetzt wird.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Die Pose der 3D Region of Interest ist Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Die Pose der 3D Region of Interest ist im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1).

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

5.6.2.4 Services

Das RoiDB Modul wird in der REST-API als `rc_roi_db` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Datenbank* → *Regions of Interest* dargestellt. Die angebotenen Services des RoiDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der Web GUI ausprobiert und getestet werden.

Das RoiDB Modul stellt folgende Services zur Verfügung.

set_region_of_interest

speichert eine 3D ROI auf dem *rc_visard*. Alle ROIs sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/set_region_of_interest
```

Request

Die Definition des Typs `region_of_interest` wird in *Region of Interest* (Abschnitt 5.6.2.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "sphere": {
        "radius": "float64"
      },
      "type": "string"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_region_of_interest",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_region_of_interest_2d

speichert eine 2D ROI auf dem *rc_visard*. Alle ROIs sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/set_region_of_interest_2d
```

Request

Die Definition des Typs `region_of_interest_2d` wird in *Region of Interest* (Abschnitt 5.6.2.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_2d": {
      "height": "uint32",
      "id": "string",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_region_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_regions_of_interest

gibt die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D ROIs zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/get_regions_of_interest
```

Request

Werden keine `region_of_interest_ids` angegeben, enthält die Serviceantwort alle gespeicherten ROIs.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_regions_of_interest",
  "response": {
    "regions_of_interest": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "sphere": {
          "radius": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

get_regions_of_interest_2d

gibt die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D ROIs zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/get_regions_of_interest_2d
```

Request

Werden keine `region_of_interest_2d_ids` angegeben, enthält die Serviceantwort alle gespeicherten ROIs.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_regions_of_interest_2d",
  "response": {
    "regions_of_interest": [
      {
        "height": "uint32",
        "id": "string",
        "offset_x": "uint32",
        "offset_y": "uint32",
        "width": "uint32"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_regions_of_interest

löscht die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D ROIs.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/delete_regions_of_interest
```

Request

Alle zu löschenden ROIs müssen explizit angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_regions_of_interest",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`delete_regions_of_interest_2d`

löscht die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D ROIs.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/delete_regions_of_interest_2d
```

Request

Alle zu löschenden ROIs müssen explizit angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_regions_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "value": "int16"
    }
  }
}
```

5.6.2.5 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.59: Rückgabe-Codes der Services des RoiDB Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an ROIs überschritten wurde.
10	Die maximal speicherbare Anzahl an ROIs wurde erreicht.
11	Mit dem Aufruf von `set_region_of_interest` oder `set_region_of_interest_2d` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.

5.6.3 PrimitiveDB

5.6.3.1 Einführung

Das PrimitiveDB Modul (Datenbankmodul für geometrische Primitive) ermöglicht die globale Definition von geometrischen Primitiven, die dann in Detektionsmodulen verwendet werden können. Derzeit werden nur Ebenen unterstützt.

Eine Ebene kann z.B. als Referenzebene zur Messung der Abmessungen erkannter Objekte in *BoxPick* (Abschnitt 5.4.5.2) dienen.

Das PrimitiveDB Modul ist ein Basismodul, welches auf jedem *rc_visard* verfügbar ist.

Tab. 5.60: Spezifikationen des PrimitiveDB Moduls

Unterstützte Primitive	Ebene
Max. Anzahl von Ebenen	100
Verwendung der Ebenen	Dimensionierung erkannter Objekte
Mögliche Referenzkoordinatensysteme	camera, external

5.6.3.2 Ebene

Eine Ebene ist in Hessescher Normalform definiert

$$\mathbf{n} \cdot \mathbf{x} + d = 0,$$

wobei $\mathbf{n}$ die Normale der Ebene (`normal`) und d der vorzeichenbehaftete Abstand der Ebene vom Ursprung des Referenzkoordinatensystems (`distance`) in Metern ist. Alle Punkte $\mathbf{x}$, die diese Gleichung

erfüllen, liegen auf der Ebene. Die Normale wird als Einheitsvektor gespeichert. Wird eine Normale mit einer anderen Länge angegeben, werden die Normale und der Abstand entsprechend skaliert, was dieselbe Ebene beschreibt.

In der REST-API ist eine Ebene über folgende Werte spezifiziert:

- `id`: Eindeutiger Name der Ebene
- `normal`: Normale der Ebene mit den Komponenten `x`, `y` und `z`
- `distance`: vorzeichenbehafteter Abstand der Ebene vom Ursprung des Referenzkoordinatensystems in Metern
- `pose_frame`: Referenzkoordinatensystem der Ebene, entweder `camera` oder `external`

Das PrimitiveDB Modul speichert Ebenen nur und schätzt sie nicht. Eine Ebene kann mit dem Service `compute_plane` des Measure Moduls (siehe *Schätzung von Ebenen*, Abschnitt 5.4.1.3) aus den aktuellen Sensordaten geschätzt und dann mit `set_plane` gespeichert werden.

Der *rc_visard* erlaubt das Speichern von bis zu 100 verschiedenen Ebenen. Die Konfiguration von Ebenen erfolgt in der Regel offline während der Einrichtung der gewünschten Anwendung. Die Konfiguration kann mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) des PrimitiveDB Moduls vorgenommen werden, oder über die *rc_visard Web GUI* (Abschnitt 6.1) auf der Seite *Ebenen* unter dem Menüpunkt *Datenbank*.

Bemerkung: Die erstellten Ebenen sind persistent auf dem *rc_visard* gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

5.6.3.3 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_visard* laufenden Module liefern Daten für das PrimitiveDB Modul oder haben Einfluss auf die Datenverarbeitung.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann eine Ebene im Roboterkoordinatensystem angegeben werden, indem das Argument `pose_frame` entsprechend gesetzt wird.

Es kann zwischen den folgenden zwei Werten für `pose_frame` gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Die Ebene ist im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in ihrer Umgebung notwendig. Das bedeutet insbesondere, dass sich die konfigurierten Ebenen mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, die konfigurierten Ebenen zu aktualisieren, wenn sich das Kamera-Koordinatensystem bewegt (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Die Ebene ist im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage (statisch oder robotergeführt) und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1).

Die Ebene wird beim Speichern nicht transformiert, d.h. sie muss bereits im gewünschten `pose_frame` angegeben sein.

Bemerkung: Wurde keine Hand-Auge-Kalibrierung durchgeführt, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

5.6.3.4 Services

Das PrimitiveDB Modul wird in der REST-API als `rc_primitive_db` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Datenbank* → *Ebenen* dargestellt. Die angebotenen Services des PrimitiveDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der Web GUI ausprobiert und getestet werden.

Das PrimitiveDB Modul stellt folgende Services zur Verfügung.

`set_plane`

speichert eine Ebene auf dem *rc_visard*. Eine bestehende Ebene mit derselben `id` wird überschrieben. Alle Ebenen sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_primitive_db/services/set_plane
```

Request

Die Definition einer Ebene wird in *Ebene* (Abschnitt 5.6.3.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "distance": "float64",
    "id": "string",
    "normal": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_plane",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`get_planes`

gibt die mit `plane_ids` spezifizierten, gespeicherten Ebenen zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_primitive_db/services/get_planes
```

Request

Werden keine `plane_ids` angegeben, enthält die Serviceantwort alle gespeicherten Ebenen.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "plane_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_planes",
  "response": {
    "planes": [
      {
        "distance": "float64",
        "id": "string",
        "normal": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose_frame": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`delete_planes`

löscht die mit `plane_ids` spezifizierten, gespeicherten Ebenen.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_primitive_db/services/delete_planes
```

Request

Alle zu löschenden Ebenen müssen explizit in `plane_ids` angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "plane_ids": [
      "string"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_planes",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.6.3.5 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 5.61: Rückgabe-Codes der Services des PrimitiveDB Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Ebenen überschritten wurde.
10	Die maximal speicherbare Anzahl an Ebenen wurde erreicht.
11	Mit dem Aufruf von `set_plane` wurde eine bereits existierende Ebene mit derselben `id` überschrieben.

5.6.4 GripperDB

5.6.4.1 Einleitung

Das GripperDB Modul ist ein optionales Modul, welches intern auf dem *rc_visard* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module *ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5) oder *SilhouetteMatch* (Abschnitt 5.4.7) vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 7.4).

Das Modul bietet Services zum Anlegen, Abfragen und Löschen von Greifern, die dann für die Kollisionsprüfung mit einem Load Carrier oder anderen erkannten Objekten (nur in Kombination mit *SilhouetteMatch* (Abschnitt 5.4.7)) genutzt werden können. Die angelegten Greifer sind in allen Modulen auf dem *rc_visard* verfügbar, die eine Kollisionsprüfung anbieten.

Tab. 5.62: Spezifikationen des GripperDB Moduls

Max. Anzahl Greifer	50
Mögliche Greiferelement-Geometrien	Box, Zylinder, CAD-Element
Max. Anzahl Elemente pro Greifer	15
Kollisionsprüfung verfügbar in	*ItemPick* (Abschnitt 5.4.4) und *BoxPick* (Abschnitt 5.4.5), *SilhouetteMatch* (Abschnitt 5.4.7)

5.6.4.2 Erstellen eines Greifers

Der Greifer ist eine Kollisionsgeometrie, die für jeden greifpunkt zur Prüfung auf Kollisionen zwischen dem Greifer und dem Load Carrier, der Punktwolke oder anderen Objekten in der Szene verwendet wird. Der Greifer kann aus bis zu 15 miteinander verbundenen Elementen bestehen.

Es sind folgende Arten von Elementen möglich:

- Quader (`BOX`), mit den Abmessungen `box.x`, `box.y`, `box.z`.
- Zylinder (`CYLINDER`), mit dem Radius `cylinder.radius` und der Höhe `cylinder.height`.
- CAD-Element (`CAD`), mit der ID `cad.id` des gewählten CAD-Elements.

Jedem Greiferelement kann einer der folgenden Werte für `function_type` zugewiesen werden:

- `NONE`: Standardwert, entspricht einem leeren Eintrag. Dieses Element hat keine spezielle Funktion und wird bei der Kollisionsprüfung wie modelliert berücksichtigt.
- `FINGER`: Dieses Element ist ein beweglicher Finger oder eine Greiferbacke und besitzt neben seiner Standardpose `pose` eine Nullposition (`zero_pose`). Es kann sich linear von der Nullposition so weit in Richtung der Standardposition bewegen, wie der für jeden Griff definierte Hub (stroke) vorgibt.
- `SUCTION_CUP`: Dieses Element ist ein verformbarer Saugnapf und wird daher bei der Kollisionsprüfung ignoriert. Es dient lediglich der Visualisierung.

Weiterhin müssen für jeden Greifer der Flanschradius und die Tool Center Points (TCPs) definiert werden. Ein Greifer muss mindestens einen TCP besitzen, kann aber auch mehrere haben. Die TCPs müssen in der Reihenfolge ihrer Priorität für das Greifen angegeben werden. Jeder TCP benötigt eine ID `id` und die Pose `pose_parent` im Koordinatensystem des Elements, das mit `parent_id` definiert ist. Zusätzlich kann jedes Element die die minimal erforderlichen Abmessungen der Saugfläche in den Achsen des TCPs in `min_suction_dimensions` definieren. Bei der Kollisionsprüfung werden alle TCPs in der angegebenen Reihenfolge geprüft. Die ID des ersten kollisionsfreien TCP, dessen `min_suction_dimensions`-Anforderungen von der verfügbaren Saugfläche des Greifpunkts erfüllt werden, wird im Feld `tcp_id` des Greifpunkts zurückgegeben.

Die Konfiguration des Greifers wird in der Regel während des Setups der Zielanwendung durchgeführt. Das kann über die *REST-API-Schnittstelle* (Abschnitt 6.2) oder die *rc_visard Web GUI* (Abschnitt 6.1) geschehen.

Flanschradius

Es werden standardmäßig nur Kollisionen mit dem Greifer, nicht aber mit der Robotergeometrie geprüft. Um Kollisionen zwischen dem Load Carrier und dem Roboter zu vermeiden, kann über den Laufzeitparameter `check_flange` im CollisionCheck Modul (siehe *Übersicht der Parameter*, Abschnitt 5.5.2.3) ein zusätzlicher optionaler Test aktiviert werden. Dieser Test erkennt alle Griffe als Kollisionen, bei denen sich ein Teil des Roboterflanschs innerhalb des Load Carriers befinden würde (siehe Abb. 5.45). Der Test basiert auf der Greifergeometrie und dem Flanschradius.

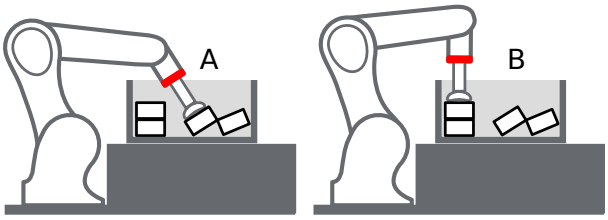


Abb. 5.45: Fall A: Der Griff wird nur als Kollision erkannt, wenn `check_flange` auf *true* gesetzt ist, denn der Flansch (rot) befindet sich im Load Carrier. Fall B: Der Griff ist in jedem Fall kollisionsfrei.

Hochladen von CAD-Greiferelementen

Ein Greifer kann aus Boxen, Zylindern und CAD-Elementen bestehen. Während Boxen und Zylinder während der Erstellung eines Greifers parametrisiert werden können, müssen CAD-Elemente im Vorfeld hochgeladen werden, um für die Greifererstellung verfügbar zu sein. Ein CAD-Element kann über die *REST-API-Schnittstelle* (Abschnitt 6.2) wie in Abschnitt *CAD-Greiferelement API* (Abschnitt 5.6.4.5) beschrieben, oder über die *rc_visard Web GUI* (Abschnitt 6.1) hochgeladen werden. Unterstützte Dateiformate sind STEP (*.stp, *.step), STL (*.stl), OBJ (*.obj) und PLY (*.ply). Die maximal hochzuladende Dateigröße ist auf 10 MB begrenzt. Die Dateien werden intern in PLY konvertiert und, falls nötig, vereinfacht. Die CAD-Elemente können dann während der Greifererstellung über ihre ID referenziert werden.

Erstellen eines Greifers über die REST-API oder die Web GUI

Bei der Greifererstellung über die *REST-API-Schnittstelle* (Abschnitt 6.2) oder die *Web GUI* (Abschnitt 6.1) hat jedes Greifer-Element ein *Parent*-Element, das die Verbindung zwischen den Elementen definiert. Der Greifer wird immer vom Roboterflansch ausgehend in Richtung TCP aufgebaut, und mindestens ein Element muss den Parent ‚flange' (Flansch) haben. Die IDs der Elemente müssen eindeutig sein und dürfen nicht ‚tcp' oder ‚flange' sein. Die Pose des Elements muss im Koordinatensystem des *Parent*-Elements angegeben werden. Das Koordinatensystem eines Elements vom Typ `CYLINDER` oder `BOX` befindet sich genau in seinem geometrischen Mittelpunkt. Damit ein Element also genau unterhalb seines *Parent*-Elements platziert wird, muss seine Position aus der Höhe des *Parent*-Elements und seiner eigenen Höhe berechnet werden (siehe Abb. 5.46).

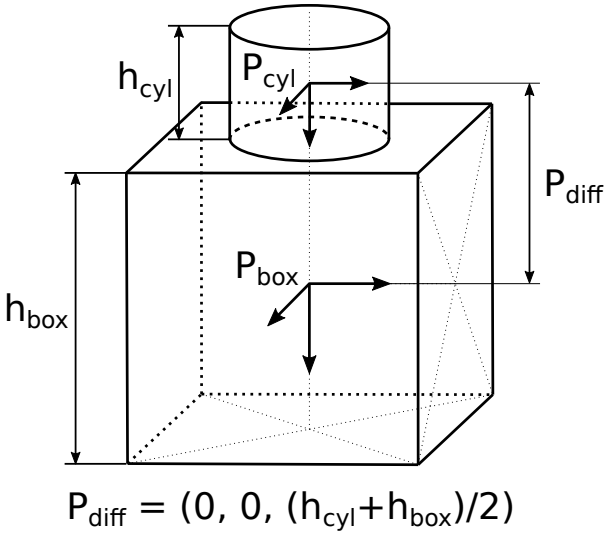


Abb. 5.46: Bezugskoordinatensysteme für das Erstellen von Greifern über die REST-API und die Web GUI

Im Falle eines CAD-Greiferelements wird der Ursprung durch die CAD-Daten bestimmt und befindet

sich nicht notwendigerweise im Mittelpunkt der Bounding Box des Elements.

Es wird empfohlen Greifer über die Web GUI zu erstellen, da diese eine 3D Visualisierung der Greifergeometrie bietet und das automatische Anheften von Kind-Element an ihre Parent-Elemente ermöglicht, indem die entsprechende Option für dieses Element aktiviert wird. In diesem Fall bleiben Elemente an ihren Parent angeheftet, auch wenn sich ihre Größen ändern. Bei CAD-Greiferelementen wird die Bounding Box des Elements als Referenz verwendet. Das automatische Anheften ist nur möglich, wenn das Kind-Element in Bezug auf seinen Parent nicht um die x- oder y-Achse rotiert ist.

Das Bezugskoordinatensystem für das erste Element liegt immer im Mittelpunkt des Roboterflanschs, wobei die z-Achse nach unten gerichtet ist. Es können Greifer mit einer Baumstruktur erstellt werden, bei denen mehrere Elemente dasselbe *Parent*-Element haben, solange alle Elemente miteinander verbunden sind.

Berechnete TCP-Position

Nach dem Erstellen des Greifers mit dem Service `set_gripper` werden die TCP-Positionen im Flanschkoordinatensystem berechnet und als `pose_flange` im jeweiligen `tcps` Element zurückgegeben. Dieser Wert muss mit den tatsächlichen TCP-Koordinaten des Roboters übereinstimmen. Wenn ein Greifer über die Web GUI erstellt wird, wird die aktuelle TCP-Position zu jeder Zeit in der 3D-Visualisierung angezeigt.

Nicht-rotationssymmetrische Greifer erstellen

Bei Greifern, die nicht rotationssymmetrisch um die z-Achse sind, muss sichergestellt werden, dass der Greifer so montiert wird, dass seine Ausrichtung mit der im GripperDB-Modul gespeicherten Darstellung übereinstimmt.

5.6.4.3 Services

Das GripperDB Modul wird in der REST-API als `rc_gripper_db` bezeichnet und in der *Web GUI* (Abschnitt 6.1) unter *Datenbank* → *Greifer* dargestellt. Die angebotenen Services des GripperDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.2) oder der Web GUI ausprobiert und getestet werden.

Das GripperDB Modul stellt folgende Services zur Verfügung.

`set_gripper`

konfiguriert und speichert einen Greifer auf dem *rc_visard*. Alle Greifer sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/set_gripper
```

Request

Obligatorische Serviceargumente:

`elements`: Liste von geometrischen Elementen, aus denen der Greifer besteht. Jedes Element muss den `type` ‚CYLINDER' oder ‚BOX' mit den zugehörigen Dimensionen im Feld `cylinder` bzw. `box`, oder den Typ ‚CAD' haben, wobei die entsprechende ID unter `id` im Feld `cad` angegeben werden muss. Die Pose jedes Elements muss im *Parent*-Koordinatensystem angegeben werden (siehe *Erstellen eines Greifers*, Abschnitt 5.6.4.2). Die `id` des Elements muss eindeutig sein und darf

nicht ‚tcp' oder ‚flange' sein. Die `parent_id` ist die ID des *Parent*-Elements, welche entweder ‚flange' ist oder der ID eines anderen Elements entsprechen muss. Jedes Element kann einen `function_type` haben, der entweder `NONE`, `FINGER` oder `SUCTION_CUP` ist. Elemente vom Typ `FINGER` benötigen zusätzlich eine `zero_pose`, deren Orientierung mit der in der `pose` des Elements übereinstimmen muss. Elemente vom Typ `SUCTION_CUP` können keine Kind-Elemente haben.

`flange_radius`: Flanschradius der benutzt wird, falls der Parameter `check_flange` aktiviert ist.

`id`: Eindeutiger Name des Greifers.

`tcps`: Liste der TCPs für den Greifer. Jedes Element muss eine `id` und die Pose `pose_parent` des TCP bezogen auf das Koordinatensystem des Elements mit der ID `parent_id` haben. Optional können die minimal nötigen Dimensionen der Sauggreiffläche in `min_suction_dimensions` für die Achsen `x`, `y` and `z` des TCP angegeben werden. Die Reihenfolge der Elemente in der Liste bestimmt die Priorität des jeweiligen TCP.

`tcp_parent_id`: veraltet, nutzen Sie `parent_id` in `tcps` stattdessen.ID des Elements, auf dem der TCP definiert ist.

`tcp_pose_parent`: veraltet, nutzen Sie `pose_parent` in `tcps` stattdessen. Die Pose des TCP im Koordinatensystem des Elements, das in `tcp_parent_id` angegeben ist.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "elements": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "cad": {
          "id": "string"
        },
        "cylinder": {
          "height": "float64",
          "radius": "float64"
        },
        "finger_zero_pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "function_type": "string",
        "id": "string",
        "parent_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "type": "string"
      }
    ],
    "flange_radius": "float64",
    "id": "string",
    "tcp_parent_id": "string",
    "tcp_pose_parent": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "tcps": [
      {
        "id": "string",
        "min_suction_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "parent_id": "string",
        "pose_parent": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        }
      }
    ]
  }
}
```


Response

`gripper`: Gibt den Greifer mit dem zusätzlichen Feld `pose_flange` in `tcps` zurück. Dieses Feld gibt die TCP-Koordinaten im Flanschkoordinatensystem an, um diese mit den Roboter-TCP-Koordinaten vergleichen zu können.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_gripper",
  "response": {
    "gripper": {
      "elements": [
        {
          "box": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "cad": {
            "id": "string"
          },
          "cylinder": {
            "height": "float64",
            "radius": "float64"
          },
          "finger_zero_pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "function_type": "string",
          "id": "string",
          "parent_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "type": "string"
        }
      ],
      "flange_radius": "float64",
      "id": "string",
      "tcp_parent_id": "string",
      "tcp_pose_flange": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "tcp_pose_parent": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "tcps": [
        {
          "id": "string",
          "min_suction_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "parent_id": "string",
          "pose_flange": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "pose_parent": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          }
        }
      ],
      "type": "string"
    },
    "return_code": {
      "message": "string",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "value": "int16"
    }
  }
}
```

get_grippers

gibt die mit `gripper_ids` spezifizierten und gespeicherten Greifer zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/get_grippers
```

Request

Wenn keine `gripper_ids` angegeben werden, enthält die Serviceantwort alle gespeicherten Greifer.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```

Response

`elements`: Liste von geometrischen Elementen, aus denen der Greifer besteht.

`flange_radius`: Flanschradius der benutzt wird, falls der Parameter `check_flange` aktiviert ist.

`id`: Eindeutiger Name des Greifers.

`tcps`: Liste der TCPs für den Greifer. `pose_flange` enthält die Pose des TCP im Roboterflansch.

`tcp_parent_id`: ID des Elements, auf dem der TCP definiert ist.

`tcp_pose_parent`: Die Pose des ersten TCP im Koordinatensystem des Elements, das in `tcp_parent_id` angegeben ist (nur für Rückwärtskompatibilität, nutzen Sie `tcps` stattdessen).

`tcp_pose_flange`: Die Pose des ersten TCP im Flansch (nur für Rückwärtskompatibilität, nutzen Sie `tcps` stattdessen).

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_grippers",
  "response": {
    "grippers": [
      {
        "elements": [
          {
            "box": {
              "x": "float64",
              "y": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "cad": {
            "id": "string"
          },
          "cylinder": {
            "height": "float64",
            "radius": "float64"
          },
          "finger_zero_pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "function_type": "string",
          "id": "string",
          "parent_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "type": "string"
        }
      ],
      "flange_radius": "float64",
      "id": "string",
      "tcp_parent_id": "string",
      "tcp_pose_flange": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "tcp_pose_parent": {
        "orientation": {
          "w": "float64",
          "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "tcps": [
          {
            "id": "string",
            "min_suction_dimensions": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "parent_id": "string",
            "pose_flange": {
              "orientation": {
                "w": "float64",
                "x": "float64",
                "y": "float64",
                "z": "float64"
              },
              "position": {
                "x": "float64",
                "y": "float64",
                "z": "float64"
              }
            },
            "pose_parent": {
              "orientation": {
                "w": "float64",
                "x": "float64",
                "y": "float64",
                "z": "float64"
              },
              "position": {
                "x": "float64",
                "y": "float64",
                "z": "float64"
              }
            }
          }
        ],
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_grippers

löscht die mit `gripper_ids` spezifizierten, gespeicherten Greifer.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/delete_grippers
```

Request

Alle zu löschenden Greifer müssen explizit angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_grippers",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.6.4.4 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 5.63: Rückgabecodes der GripperDB Services

Code	Beschreibung
0	Erfolgreich
-1	Ein ungültiges Argument wurde übergeben.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-9	Lizenz für CollisionCheck ist nicht verfügbar.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Greifern überschritten wurde.
10	Die maximal speicherbare Anzahl an Greifern wurde erreicht.
11	Bestehender Greifer wurde überschrieben.

5.6.4.5 CAD-Greiferelement API

Für den Upload, Download, das Auflisten und Löschen von CAD-Greiferelementen werden spezielle REST-API-Endpunkte zur Verfügung gestellt. CAD-Greiferelemente können auch über die Web GUI hoch- und runtergeladen werden. Bis zu 50 CAD-Greiferelemente können gleichzeitig auf dem *rc_visard* gespeichert werden.

Die maximal hochzuladende Dateigröße ist auf 10 MB begrenzt.

`GET /cad/gripper_elements`
listet alle CAD-Greiferelemente auf.

Musteranfrage

```
GET /api/v2/cad/gripper_elements HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array von GripperElement)*
- 404 Not Found – Element nicht gefunden

Referenzierte Datenmodelle

- *GripperElement* (Abschnitt 6.2.4)

`GET /cad/gripper_elements/{id}`
ruft ein CAD-Greiferelement ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Element als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/cad/gripper_elements/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Elements *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: GripperElement)*
- 404 Not Found – Element nicht gefunden

Referenzierte Datenmodelle

- *GripperElement* (Abschnitt 6.2.4)

`PUT /cad/gripper_elements/{id}`
erstellt oder aktualisiert ein CAD-Greiferelement.

Musteranfrage

```
PUT /api/v2/cad/gripper_elements/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Elements *(obligatorisch)*

Formularparameter

- **`file`** – CAD-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: GripperElement)*
- 400 Bad Request – CAD ist ungültig oder die maximale Zahl an Elementen wurde erreicht.
- 404 Not Found – Element nicht gefunden
- 413 Request Entity Too Large – Datei zu groß

Referenzierte Datenmodelle

- *GripperElement* (Abschnitt 6.2.4)

`DELETE /cad/gripper_elements/{id}`
entfernt ein CAD-Greiferelement.

Musteranfrage

```
DELETE /api/v2/cad/gripper_elements/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **`id`** (*`string`*) – ID des Elements *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung
- 404 Not Found – Element nicht gefunden

6 Schnittstellen

Es stehen die folgenden Schnittstellen zur Konfiguration und Datenübertragung des *rc_visard* zur Verfügung:

- *Web GUI* (Abschnitt 6.1)

  Leicht zu bedienendes grafisches Interface zum Konfigurieren und Kalibrieren des *rc_visard*, zum Anzeigen von Livebildern, Aufrufen von Services, Visualisieren von Ergebnissen, usw.

- *REST-API-Schnittstelle* (Abschnitt 6.2)

  Programmierschnittstelle zur Konfiguration des *rc_visard*, zur Abfrage von Statusinformationen, zum Anfordern von Datenströmen, usw.

- *rc_dynamics-Schnittstelle* (Abschnitt 6.3)

  Echtzeit-Datenströme, die Zustandsschätzungen samt Posen-, Geschwindigkeits- und anderen Daten enthalten, werden über die *rc_dynamics*-Schnittstelle bereitgestellt. Sie sendet *ProtoBuf*-kodierte Nachrichten über UDP.

- *Generic Robot Interface* (Abschnitt 6.4)

  TCP-Socketkommunikationsschnittstelle zur Konfiguration des *rc_visard* und für Serviceaufrufe.

- *OPC UA Interface* (Abschnitt 6.5)

  OPC UA Schnittstelle zur Konfiguration des *rc_visard* und Ausführen von Service-Anfragen.

- *KUKA Ethernet KRL Schnittstelle* (Abschnitt 6.6)

  API zum Konfigurieren des *rc_visard* und Ausführen von Service-Anfrage von KUKA KSS Robotern aus.

- *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 6.7)

  Konfiguration bild- und kamerabezogener Einstellungen.

- *gRPC Bilddatenschnittstelle* (Abschnitt 6.8)

  Synchronisierte Bilddaten per gRPC.

- *Zeitsynchronisierung* (Abschnitt 6.9)

  Zeitsynchronisation zwischen dem *rc_visard* und dem Anwendungs-PC.

6.1 Web GUI

Die Web GUI des *rc_visard* dient dazu, das Gerät zu testen, zu kalibrieren und zu konfigurieren.

6.1.1 Zugriff auf die Web GUI

Auf die Web GUI kann über die IP-Adresse des *rc_visard* von jedem Webbrowser aus zugegriffen werden, z.B. Firefox, Google Chrome oder Microsoft Edge. Am einfachsten lässt sich die Web GUI über die `rcdiscover-gui` aufrufen, wenn, wie in *Aufspüren von rc_visard-Geräten* (Abschnitt 4.3) beschrieben, ein Doppelklick auf das gewünschte Gerät vorgenommen wird.

Alternativ konfigurieren einige Netzwerkumgebungen den eindeutigen Host-Namen des *rc_visard* automatisch in ihrem Domain Name Server (*DNS*). In diesem Fall kann die Web GUI auch direkt über folgende *URL* aufgerufen werden: `http://<host-name>`, wobei der Platzhalter `<host-name>` gegen den Host-Namen des Geräts auszutauschen ist.

Für Linux und macOS funktioniert das ohne DNS über das Multicast-DNS-Protokoll (*mDNS*), das automatisch aktiviert wird, wenn `.local` zum Host-Namen hinzugefügt wird. So wird die URL einfach zu: `http://<host-name>.local`.

6.1.2 Kennenlernen der Web GUI

Die Dashboard-Seite der Web GUI enthält die wichtigsten Informationen über das Gerät und die Softwaremodule.

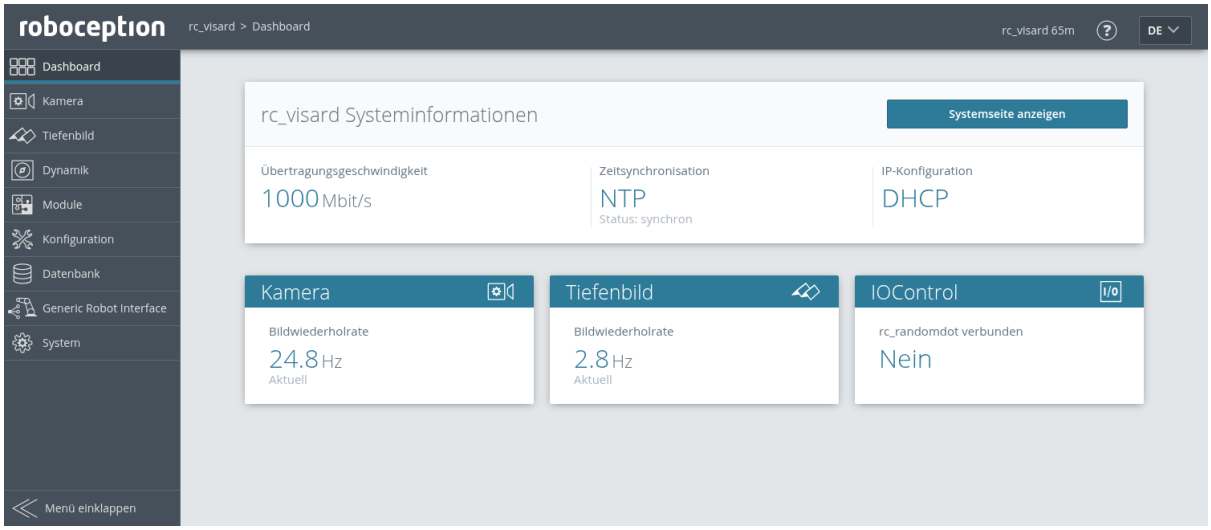


Abb. 6.1: Dashboard-Seite der Web GUI des *rc_visard*

Über das Menü kann auf die einzelnen Seiten der Web GUI des *rc_visard* zugegriffen werden:

Kamera bietet einen Live-Stream der rektifizierten Bilder und ermöglicht das Einstellen der Kameraparameter. Für nähere Informationen siehe *Kamera Modul* (Abschnitt 5.1).

Tiefenbild bietet einen Live-Stream der rektifizierten Bilder der linken Kamera sowie Disparitäts- und Konfidenzbilder. Auf der Seite lassen sich verschiedene Einstellungen zur Berechnung und Filterung von Tiefenbildern vornehmen. Für nähere Informationen siehe *Stereo-Matching Modul* (Abschnitt 5.2).

Dynamik zeigt die für die Schätzung der Eigenbewegung des *rc_visard* relevante Position und Bewegung visueller Merkmale. Über entsprechende Einstellungen lässt sich u. a. die Anzahl der zu dieser Schätzung herangezogenen Merkmale anpassen. Für nähere Informationen siehe *Parameter* (Abschnitt 5.3.2.1).

Module ermöglicht den Zugriff auf die Detektionsmodule des *rc_visard* (siehe *Detektions- und Messmodule*, Abschnitt 5.4).

Konfiguration ermöglicht den Zugriff auf die Konfigurationsmodule des *rc_visard* (siehe *Konfigurationsmodule*, Abschnitt 5.5).

Datenbank ermöglicht den Zugriff auf die Datenbankmodule des *rc_visard* (siehe *Datenbankmodule*, Abschnitt 5.6).

Generic Robot Interface zeigt die Jobs und Hand-Auge-Kalibrierkonfigurationen, die für das Generic Robot Interface definiert wurden.

System ermöglicht dem Nutzer den Zugriff auf allgemeine Systemeinstellungen, Informationen zum Gerät und den Log-Dateien, sowie die Möglichkeit, die Firmware oder Lizenzdatei zu aktualisieren.

Bemerkung: Weitere Informationen zu den einzelnen Parametern der Web GUI lassen sich über die jeweils daneben angezeigte Schaltfläche *Info* aufrufen.

6.1.3 Web GUI Zugriffskontrolle

Die Web GUI bietet einen simplen Mechanismus das User Interface zu sperren um beiläufige und unbeabsichtigte Änderungen zu vermeiden.

Beim aktivieren der Web GUI Zugriffskontrolle über die *System* Seite muss ein Passwort gesetzt werden. Jetzt ist die Web GUI in einem gesperrten Zustand wie das Schloss Symbol in der Kopfleiste anzeigt. Alle Seiten, Kamerabilder, Parameter und Detektionen können wie gewohnt eingesehen werden, Änderungen sind aber nicht möglich.

Um die Web GUI temporär zu entsperren und Änderungen vorzunehmen, klicken Sie das Schloss Symbol und geben Sie das Passwort ein. Während das aktivieren und deaktivieren der Web GUI Zugriffskontrolle jeden betrifft der diesen *rc_visard* nutzt, ist das Entsperren nur pro Browser gültig und wird durch das Symbol mit dem offenen Schloss angezeigt. Nach 10 Minutuen Inaktivität wird es automatisch wieder gesperrt.

Die Web GUI Zugriffskontrolle kann auf der *System* Seite wieder deaktiviert werden nachdem das aktuelle Passwort angegeben wurde.

Warnung: Dies ist keine Sicherheitsfunktion! Es sperrt nur die Web GUI und nicht die REST-API. Es ist dazu gedacht um unbeabsichtigte und beiläufige Änderungen, z.B. über einen angeschlossenen Bildschirm, zu verhindern.

Bemerkung: Im Fall eines vergessenen Passworts kann die Zugriffskontrolle über die REST-API mit *delete ui_lock* (Abschnitt 6.2.3.3) zurückgesetzt und deaktiviert werden.

6.1.4 Herunterladen von Kamerabildern

Die Web GUI bietet eine einfache Möglichkeit, einen Schnappschuss der aktuellen Szene als .tar.gz-Datei zu speichern. Dazu dient das Kamerasymbol unterhalb der Live-Streams auf der Seite *Kamera*. Dieser Schnappschuss beinhaltet:

- die rektifizierten Kamerabilder in voller Auflösung als .png-Dateien,
- eine Kameraparameter-Datei mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras.
- die aktuellen IMU-Messungen als imu.csv-Datei,
- eine pipeline_status.json-Datei mit Informationen aller 3D-Kamera-, Detektions- und Konfigurationsmodule, die auf dem *rc_visard* laufen,
- eine backup.json-Datei mit den Einstellungen des *rc_visard* einschließlich konfigurierter Greifer, Load Carrier und Regions of Interest,
- eine system_info.json-Datei mit Systeminformationen des *rc_visard*.

Die Dateinamen enthalten die Zeitstempel.

6.1.5 Herunterladen von Tiefenbildern und Punktwolken

Die Web GUI bietet eine einfache Möglichkeit, die Tiefendaten der aktuellen Szene als .tar.gz-Datei zu speichern. Dazu dient das Kamerasymbol unterhalb der Live-Streams auf der Seite *Tiefenbild*. Dieser Schnappschuss beinhaltet:

- die rektifizierten linken und rechten Kamerabilder in voller Auflösung als .png-Dateien,
- eine Parameterdatei für das linke Kamerabild mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras,
- die Disparitäts-, Fehler- und Konfidenzbilder in der Auflösung, die der aktuell eingestellten Qualität entspricht, als .png-Dateien,
- eine Parameterdatei zum Disparitätsbild mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras, sowie Informationen über die Disparitätswerte (ungültige Werte, Skalierung, Offset),
- die aktuellen IMU-Messungen als imu.csv-Datei,
- eine pipeline_status.json-Datei mit Informationen aller 3D-Kamera-, Detektions- und Konfigurationsmodule, die auf dem *rc_visard* laufen,
- eine backup.json-Datei mit den Einstellungen des *rc_visard* einschließlich konfigurierter Greifer, Load Carrier und Regions of Interest,
- eine system_info.json-Datei mit Systeminformationen des *rc_visard*.

Die Dateinamen enthalten die Zeitstempel.

Durch Klick auf das Mesh-Symbol unterhalb der Live-Streams auf der Seite *Tiefenbild* kann man einen Schnappschuss herunterladen, der zusätzlich ein Mesh der Punktwolke in der aktuell eingestellten Auflösung (Qualität) als *.ply Datei enthält.

Bemerkung: Das Herunterladen der Tiefenbilder löst eine Bildaufnahme aus, in der gleichen Weise wie ein Klick auf den „Aufnehmen“-Button auf der *Tiefenbild*-Seite der Web GUI. Dies kann einen Einfluss auf laufende Anwendungen haben.

6.2 REST-API-Schnittstelle

Der *rc_visard* bietet eine umfassende RESTful-Web-Schnittstelle (REST-API), auf die jeder HTTP-Client und jede HTTP-Bibliothek zugreifen kann. Während die meisten Parameter, Services und Funktionen auch über die benutzerfreundliche *Web GUI* (Abschnitt 6.1) zugänglich sind, dient die REST-API eher als Maschine-Maschine-Schnittstelle für folgende programmgesteuerte Aufgaben:

- Setzen und Abrufen der Laufzeitparameter der Softwaremodule, z.B. der Stereokamera oder von Bildverarbeitungsmodulen,
- Aufrufen von Services, z.B. zum Starten und Stoppen einzelner Softwaremodule, oder zum Nutzen spezieller Funktionen, wie der Hand-Auge-Kalibrierung,
- Abruf des aktuellen Systemstatus und des Status einzelner Softwaremodule, sowie
- Aktualisierung der Firmware des *rc_visard* oder seiner Lizenz.

Bemerkung: In der REST-API des *rc_visard* bezeichnet der Begriff *Node* ein Softwaremodul, das gewisse algorithmische Funktionen bündelt und eine ganzheitliche Benutzeroberfläche (Parameter, Services, aktueller Status) besitzt. Beispiele für solche Module sind das Stereo-Matching-Modul oder das Modul zur Hand-Auge-Kalibrierung.

6.2.1 Allgemeine Struktur der Programmierschnittstelle (API)

Der allgemeine **Einstiegspunkt** zur Programmierschnittstelle (API) des *rc_visard* ist `http://<host>/api/` wobei `<host>` entweder die IP-Adresse des Geräts ist oder sein dem jeweiligen DHCP-Server bekannter *Host-Name* (siehe *Netzwerkkonfiguration*, Abschnitt 4.4). Greift der Benutzer über einen Webbrowser auf diese Adresse zu, kann er die Programmierschnittstelle während der Laufzeit mithilfe der *Swagger UI* (Abschnitt 6.2.5) erkunden und testen.

Für die eigentlichen HTTP-Anfragen wird dem Einstiegspunkt der Programmierschnittstelle die **aktuelle Version der Schnittstelle als Postfix angehangen**, d.h. `http://<host>/api/v2`.

Alle Daten, die an die REST-API gesandt und von ihr empfangen werden, entsprechen dem JSON-Datenformat (JavaScript Object Notation). Die Programmierschnittstelle ist so gestaltet, dass der Benutzer die in *Verfügbare Ressourcen und Anfragen* (Abschnitt 6.2.3) aufgelisteten sogenannten **Ressourcen** über die folgenden HTTP-Anforderungen **anlegen, abrufen, ändern und löschen** kann.

Anfragetyp	Beschreibung
GET	Zugriff auf eine oder mehrere Ressourcen und Rückgabe des Ergebnisses im JSON-Format
PUT	Änderung einer Ressource und Rückgabe der modifizierten Ressource im JSON-Format
DELETE	Löschen einer Ressource
POST	Upload einer Datei (z.B. einer Lizenz oder eines Firmware-Images)

Je nach der Art der Anfrage und Datentyp können die **Argumente** für HTTP-Anfragen als Teil des **Pfads** (*URI*) zur Ressource, als **Abfrage**-Zeichenfolge, als **Formulardaten** oder im **Body** der Anfrage übertragen werden. Die folgenden Beispiele nutzen das Kommandozeilenprogramm *curl*, das für verschiedene Betriebssysteme verfügbar ist (siehe https://curl.haxx).se.

- Abruf des aktuellen Status eines Moduls, wobei sein Name im Pfad (URI) verschlüsselt ist

```
curl -X GET 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching'
```

- Abruf einiger Parameterwerte eines Moduls über eine Abfragezeichenfolge

```
curl -X GET 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters?
↪name=minconf&name=maxdepth'
```

- Konfiguration eines neuen Datenstroms, wobei die Zielparameter als Formulardaten übertragen werden

```
curl -X PUT --header 'Content-Type: application/x-www-form-urlencoded' -d 'destination=10.0.
↪1.14%3A30000' 'http://<host>/api/v2/datastreams/pose'
```

- Setzen eines Modulparameters als JSON-formatierter Text im Body der Anfrage

```
curl -X PUT --header 'Content-Type: application/json' -d '[{"name": "mindepth", "value": 0.
↪1}]' 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters'
```

Zur Beantwortung solcher Anfragen greift die Programmierschnittstelle des *rc_visard* auf übliche Rückgabecodes zurück:

Statuscode	Beschreibung
`200 OK`	Die Anfrage war erfolgreich. Die Ressource wird im JSON-Format zurückgegeben.
`400 Bad Request`	Ein für die API-Anfrage benötigtes Attribut oder Argument fehlt oder ist ungültig.
`404 Not Found`	Auf eine Ressource konnte nicht zugegriffen werden. Möglicherweise kann die ID einer Ressource nicht gefunden werden.
`403 Forbidden`	Der Zugriff ist (vorübergehend) verboten. Möglicherweise sind einige Parameter gesperrt, während eine GigE Vision-Anwendung verbunden ist.
`429 Too many requests`	Die Übertragungsrate ist aufgrund einer zu hohen Anfragefrequenz begrenzt.

Der folgende Eintrag zeigt eine Musterantwort auf eine erfolgreiche Anfrage, mit der Informationen zum `minconf`-Parameter des `rc_stereomatching`-Moduls angefordert werden:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 157

{
    "name": "minconf",
    "min": 0,
    "default": 0,
    "max": 1,
    "value": 0,
    "type": "float64",
    "description": "Minimum confidence"
}
```

Bemerkung: Das tatsächliche Verhalten, die zulässigen Anfragen und die speziellen Rückgabecodes hängen in hohem Maße von der gewählten Ressource, vom Kontext und von der Aktion ab. Siehe die *verfügbaren Ressourcen* (Abschnitt 6.2.3) des *rc_visard* und einzelnen Parameter und Services jedes *Softwaremoduls* (Abschnitt 5).

6.2.2 Migration von API Version 1

API Version 1 ist seit dem 22.01 Firmware-Release des *rc_visard* veraltet. Die folgenden Änderungen wurden in API Version 2 vorgenommen.

- Alle 3D-Kamera-, Navigations-, Detektions- und Konfigurationsmodule, die in API Version 1 unter `/nodes` zu finden waren, befinden sich jetzt unter `/pipelines/0/nodes/`, z.B. `/pipelines/0/nodes/rc_camera`.
- Das Konfigurieren von Load Carriern, Greifern und Regions of Interest ist jetzt nur noch in den globalen Datenbankmodulen möglich, die unter `/nodes` liegen, z.B. `/nodes/rc_load_carrier_db`. Die entsprechenden Services in den Detektionsmodulen wurden entfernt oder sind veraltet.
- Templates befinden sich jetzt unter `/templates`, z.B. `/templates/rc_silhouettematch`.

6.2.3 Verfügbare Ressourcen und Anfragen

Die für die REST-API verfügbaren Ressourcen lassen sich in folgende Teilbereiche gliedern:

- **`/nodes`** Zugriff auf die globalen *Datenbankmodule* (Abschnitt 5.6) des *rc_visard* mit ihren Laufzeitzuständen, Parametern und angebotenen Services, um Daten zu speichern, die in mehreren Modulen genutzt werden, z.B. Load Carrier, Greifer und Regions of Interest.
- **`/pipelines/0/nodes`** Zugriff auf die 3D-Kamera-, Navigations-, Detektions- und Konfigurations-*Softwaremodule* (Abschnitt 5) des *rc_visard* mit ihren jeweiligen Laufzeitzuständen, Parametern und verfügbaren Services.
- **`/pipelines`** Zugriff auf den Status und die Konfiguration der Kamerapipelines. Es gibt immer nur eine Pipeline mit der Nummer 0.
- **`/templates`** Zugriff auf die im *rc_visard* hinterlegten Objekttemplates.
- **`/cad`** Zugriff auf die CAD-Elemente, z.B. für Greifer, im *rc_visard*.
- **`/datastreams`** Zugriff auf die und Verwaltung der Datenströme der *rc_dynamics Schnittstelle* (Abschnitt 6.3) des *rc_visard*.
- **`/system`** Zugriff auf Systemzustand, Netzwerkkonfiguration, und Verwaltung der Lizenzen sowie der Firmware-Updates.
- **`/logs`** Zugriff auf die im *rc_visard* hinterlegten Logdateien.
- **`/generic_robot_interface`** Zugriff auf die Jobs und Hand-Auge-Kalibrierkonfigurationen für das Generic Robot Interface auf dem *rc_visard*.

6.2.3.1 Module, Parameter und Services

Die *Softwaremodule* (Abschnitt 5) des *rc_visard* heißen in der REST-API *Nodes* und vereinen jeweils bestimmte algorithmische Funktionen. Über folgenden Befehl lassen sich alle globalen Datenbankmodule der REST-API mit ihren jeweiligen Services und Parametern auflisten:

```
curl -X GET http://<host>/api/v2/nodes
```

Informationen zu einem bestimmten Modul (z.B. `rc_load_carrier_db`) lassen sich mit folgendem Befehl abrufen:

```
curl -X GET http://<host>/api/v2/nodes/rc_load_carrier_db
```

Alle verfügbaren 3D-Kamera-, Navigations-, Detektions- und Konfigurationsmodule der REST-API lassen sich mit ihren Services und Parametern wie folgt auflisten:

```
curl -X GET http://<host>/api/v2/pipelines/0/nodes
```

Informationen zu einem bestimmten Modul (z.B. `rc_camera`) lassen sich mit folgendem Befehl abrufen:

```
curl -X GET http://<host>/api/v2/pipelines/0/nodes/rc_camera
```

Status: Während der Laufzeit stellt jedes Modul Informationen zu seinem aktuellen Status bereit. Dies umfasst nicht nur den aktuellen **Verarbeitungsstatus** des Moduls (z.B. `running` oder `stale`), sondern die meisten Module melden auch Laufzeitstatistiken oder schreibgeschützte Parameter, sogenannte **Statuswerte**. Die Statuswerte des `rc_camera`-Moduls lassen sich beispielsweise wie folgt abrufen:

```
curl -X GET http://<host>/api/v2/pipelines/0/nodes/rc_camera/status
```

Bemerkung: Die zurückgegebenen **Statuswerte** sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Bemerkung: Statuswerte werden nur gemeldet, wenn sich das jeweilige Modul im Zustand `running` befindet.

Parameter: Die meisten Module stellen Parameter über die REST-API des *rc_visard* zur Verfügung, damit ihr Laufzeitverhalten an den Anwendungskontext oder die Anforderungen angepasst werden kann. Die REST-API ermöglicht es, den Wert eines Parameters zu setzen und abzufragen. Darüber hinaus stellt sie weitere Angaben, wie z.B. den jeweiligen Standardwert und zulässige Minimal- bzw. Maximalwerte von Parametern, zur Verfügung.

Die `rc_stereomatching`-Parameter lassen sich beispielsweise wie folgt abrufen:

```
curl -X GET http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters
```

Der `quality`-Parameter dieses Moduls könnte wie folgt auf den Wert `Full` gesetzt werden:

```
curl -X PUT http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters?quality=Full
```

oder äquivalent

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "value": "Full" }' http://<host>
↪/api/v2/pipelines/0/nodes/rc_stereomatching/parameters/quality
```

Bemerkung: Laufzeitparameter sind modulspezifisch und werden in dem jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Bemerkung: Die meisten Parameter, die die Module über die REST-API anbieten, lassen sich auch über die benutzerfreundliche *Web GUI* (Abschnitt 6.1) des *rc_visard* erkunden und austesten.

Bemerkung: Einige der Parameter, die über die REST-API des *rc_visard* bereitgestellt werden, sind auch über die *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 6.7) zugänglich. Die Einstellung dieser Parameter über die REST-API und die Web GUI ist verboten, solange ein GenICam-Client verbunden ist.

Zudem bietet jedes Modul, das Laufzeitparameter bereitstellt, auch einen Service, um die Werkseinstellungen aller Parameter wiederherzustellen.

Services: Die meisten Module bieten auch Services, die sich über die REST-API aufrufen lassen. Hierzu gehört beispielsweise das oben bereits genannte Wiederherstellen von Parametern oder auch das Starten und Stoppen von Modulen. Die *Services des Moduls zur Hand-Auge-Kalibrierung* (Abschnitt 5.5.1.5) lassen sich beispielsweise wie folgt aufrufen:

```
curl -X GET http://<host>/api/v2/pipelines/0/nodes/rc_hand_eye_calibration/services
```

Um einen Service eines Moduls aufzurufen, wird eine `PUT`-Anfrage mit servicespezifischen Argumenten für die jeweilige Ressource gestellt (siehe das `"args"`-Feld des *Service-Datenmodells*, Abschnitt 6.2.4). Beispielsweise lässt sich folgendermaßen eine Bildaufnahme mit dem Stereo-Matching-Modul auslösen:

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "args": {} }' http://<host>/api/
↪v2/pipelines/0/nodes/rc_stereomatching/services/acquisition_trigger
```

Bemerkung: Die Services und zugehörigen Argumente sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Die folgende Liste enthält alle REST-API-Anfragen zum Status der globalen Datenbankmodule und ihrer Parameter und Services:

GET /nodes

Abruf einer Liste aller verfügbaren globalen Nodes.

Musteranfrage

```
GET /api/v2/nodes HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "name": "rc_roi_db",
    "parameters": [],
    "services": [
      "set_region_of_interest",
      "get_regions_of_interest",
      "delete_regions_of_interest",
      "set_region_of_interest_2d",
      "get_regions_of_interest_2d",
      "delete_regions_of_interest_2d"
    ],
    "status": "running"
  },
  {
    "name": "rc_load_carrier_db",
    "parameters": [],
    "services": [
      "set_load_carrier",
      "get_load_carriers",
      "delete_load_carriers"
    ],
    "status": "running"
  },
  {
    "name": "rc_gripper_db",
    "parameters": [],
    "services": [
      "set_gripper",
      "get_grippers",
      "delete_grippers"
    ],
    "status": "running"
  }
]
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 6.2.4)

GET /nodes/{node}

Abruf von Informationen zu einem einzelnen globalen Modul.

Musteranfrage

```
GET /api/v2/nodes/<node> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "name": "rc_roi_db",
  "parameters": [],
  "services": [
    "set_region_of_interest",
    "get_regions_of_interest",
    "delete_regions_of_interest",
    "set_region_of_interest_2d",
    "get_regions_of_interest_2d",
    "delete_regions_of_interest_2d"
  ],
  "status": "running"
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 6.2.4)

GET /nodes/{node}/services

Abruf von Beschreibungen aller von einem globalen Modul angebotenen Services.

Musteranfrage

```
GET /api/v2/nodes/<node>/services HTTP/1.1
```

Musteranfrage

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "args": {},
    "description": "string",
    "name": "string",
    "response": {}
  }
]
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service-Array)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.2.4)

`GET /nodes/{node}/services/{service}`

Abruf der Beschreibung eines Services eines globalen Moduls.

Musteranfrage

```
GET /api/v2/nodes/<node>/services/<service> HTTP/1.1
```

Musteranfrage

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Parameter

- **`node`** (*`string`*) – Modulname *(obligatorisch)*
- **`service`** (*`string`*) – Name des Service *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service)*
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.2.4)

`PUT /nodes/{node}/services/{service}`

Aufruf des Services eines Moduls: Die benötigten Argumente und die zugehörige Antwort hängt vom Modul und vom Service ab.

Musteranfrage

```
PUT /api/v2/nodes/<node>/services/<service> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Musteranfrage

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

JSON-Objekt zur Anfrage

- **service args** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Serviceaufruf erledigt *(Rückgabe: Service)*
- 403 Forbidden – Service-Aufruf verboten, z.B. weil keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.2.4)

GET /nodes/{node}/status

Abruf des Status eines globalen Datenbankmoduls.

Musteranfrage

```
GET /api/v2/nodes/<node>/status HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "running",
  "timestamp": 1503075030.2335997,
  "values": []
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeStatus)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeStatus* (Abschnitt 6.2.4)

Die folgende Liste enthält alle REST-API-Anfragen zum Status der 3D-Kamera-, Navigations-, Detektions- und Konfigurationsmodule und ihrer Parameter und Services:

`GET /pipelines/{pipeline}/nodes`
Abruf einer Liste aller verfügbaren Module.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "name": "rc_stereocalib",
    "parameters": [
      "grid_width",
      "grid_height",
      "snap"
    ],
    "services": [
      "reset_defaults",
      "change_state"
    ],
    "status": "idle"
  },
  {
    "name": "rc_camera",
    "parameters": [
      "fps",
      "exp_auto",
      "exp_value",
      "exp_max"
    ],
    "services": [
      "reset_defaults"
    ],
    "status": "running"
  },
  {
    "name": "rc_hand_eye_calibration",
    "parameters": [
      "grid_width",
      "grid_height",
      "robot_mounted"
    ],
    "services": [
      "reset_defaults",
      "set_pose",
      "reset",
      "save",
      "calibrate",
      "get_calibration"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    ],
    "status": "idle"
  },
  {
    "name": "rc_stereo_ins",
    "parameters": [],
    "services": [],
    "status": "idle"
  },
  {
    "name": "rc_stereomatching",
    "parameters": [
      "quality",
      "seg",
      "fill",
      "minconf",
      "mindepth",
      "maxdepth",
      "maxdeptherr"
    ],
    "services": [
      "reset_defaults"
    ],
    "status": "running"
  }
]
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`) *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 6.2.4)

GET /pipelines/{pipeline}/nodes/{node}

Abruf von Informationen zu einem einzelnen Modul.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node> HTTP/1.1
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "name": "rc_camera",
  "parameters": [
    "fps",
    "exp_auto",
    "exp_value",
    "exp_max"
  ],
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  "services": [
    "reset_defaults"
  ],
  "status": "running"
}
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 6.2.4)

`GET /pipelines/{pipeline}/nodes/{node}/parameters`

Abruf von Parametern eines Moduls.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/parameters?name=<name> HTTP/1.1
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "default": 25,
    "description": "Frames per second in Hz",
    "max": 25,
    "min": 1,
    "name": "fps",
    "type": "float64",
    "value": 25
  },
  {
    "default": true,
    "description": "Switching between auto and manual exposure",
    "max": true,
    "min": false,
    "name": "exp_auto",
    "type": "bool",
    "value": true
  },
  {
    "default": 0.007,
    "description": "Maximum exposure time in s if exp_auto is true",
    "max": 0.018,
    "min": 6.6e-05,
    "name": "exp_max",
    "type": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "value": 0.007
  }
]
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Anfrageparameter

- **name** (*string*) – Schränkt Ergebnisse auf Parameter mit diesem Namen ein *(optional)*.

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 6.2.4)

PUT /pipelines/{pipeline}/nodes/{node}/parameters

Aktualisierung mehrerer Parameter.

Musteranfrage

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters HTTP/1.1
Accept: application/json application/ubjson

[
  {
    "name": "string",
    "value": {}
  }
]
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "default": 25,
    "description": "Frames per second in Hz",
    "max": 25,
    "min": 1,
    "name": "fps",
    "type": "float64",
    "value": 10
  },
  {
    "default": true,
    "description": "Switching between auto and manual exposure",
    "max": true,
    "min": false,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "name": "exp_auto",
    "type": "bool",
    "value": false
  },
  {
    "default": 0.005,
    "description": "Manual exposure time in s if exp_auto is false",
    "max": 0.018,
    "min": 6.6e-05,
    "name": "exp_value",
    "type": "float64",
    "value": 0.005
  }
]
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

JSON-Objekt-Array zur Anfrage

- **parameters** (*ParameterNameValue*) – Liste von Parametern *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*
- 400 Bad Request – Ungültiger Parameterwert
- 403 Forbidden – Aktualisierung des Parameters verboten, z.B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist oder keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 6.2.4)
- *ParameterNameValue* (Abschnitt 6.2.4)

GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}

Abruf eines bestimmten Parameters eines Moduls.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/parameters/<param> HTTP/1.1
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "default": 25,
  "description": "Frames per second in Hertz",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  "max": 25,
  "min": 1,
  "name": "fps",
  "type": "float64",
  "value": 10
}
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **param** (*string*) – Name des Parameters *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter)*
- 404 Not Found – Modul oder Parameter nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 6.2.4)

PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}

Aktualisierung eines bestimmten Parameters eines Moduls.

Musteranfrage

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters/<param> HTTP/1.1
Accept: application/json application/ubjson

{
  "value": {}
}
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "default": 25,
  "description": "Frames per second in Hertz",
  "max": 25,
  "min": 1,
  "name": "fps",
  "type": "float64",
  "value": 10
}
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **param** (*string*) – Name des Parameters *(obligatorisch)*

JSON-Objekt zur Anfrage

- **parameter** (*ParameterValue*) – zu aktualisierender Parameter als JSON-Objekt *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter)*
- 400 Bad Request – Ungültiger Parameterwert
- 403 Forbidden – Aktualisierung des Parameters verboten, z.B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist oder keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Parameter nicht gefunden

Referenzierte Datenmodelle

- *ParameterValue* (Abschnitt 6.2.4)
- *Parameter* (Abschnitt 6.2.4)

GET /pipelines/{pipeline}/nodes/{node}/services

Abruf von Beschreibungen aller von einem Modul angebotenen Services.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/services HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "args": {},
    "description": "Restarts the module.",
    "name": "restart",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  },
  {
    "args": {},
    "description": "Starts the module.",
    "name": "start",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  },
  {
    "args": {},
    "description": "Stops the module.",
    "name": "stop",
    "response": {
      "accepted": "bool",
      "current_state": "string"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
]
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service-Array)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.2.4)

GET /pipelines/{pipeline}/nodes/{node}/services/{service}

Abruf der Beschreibung eines modulspezifischen Services.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/services/<service> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "slot": "int32"
  },
  "description": "Save a pose (grid or gripper) for later calibration.",
  "name": "set_pose",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service)*
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.2.4)

PUT /pipelines/{pipeline}/nodes/{node}/services/{service}

Aufruf des Services eines Moduls: Die benötigten Argumente und die zugehörige Antwort hängt vom Modul und vom Service ab.

Musteranfrage

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/services/<service> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "name": "set_pose",
  "response": {
    "message": "Grid detected, pose stored.",
    "status": 1,
    "success": true
  }
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

JSON-Objekt zur Anfrage

- **service args** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Serviceaufruf erledigt *(Rückgabe: Service)*

- 403 Forbidden – Service-Aufruf verboten, z.B. weil keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.2.4)

GET /pipelines/{pipeline}/nodes/{node}/status

Abruf des Status eines Moduls.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/status HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "running",
  "timestamp": 1503075030.2335997,
  "values": {
    "baseline": "0.0650542",
    "color": "0",
    "exp": "0.00426667",
    "focal": "0.844893",
    "fps": "25.1352",
    "gain": "12.0412",
    "height": "960",
    "temp_left": "39.6",
    "temp_right": "38.2",
    "time": "0.00406513",
    "width": "1280"
  }
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeStatus)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeStatus* (Abschnitt 6.2.4)

6.2.3.2 Datenströme

Über die folgenden Ressourcen und Anfragen ist es möglich, auf die Datenströme der *rc_dynamics-Schnittstelle* (Abschnitt 6.3) zuzugreifen und diese zu konfigurieren. Mit diesen REST-API-Anfragen ist es möglich,

- die verfügbaren und laufenden Datenströme anzuzeigen, z.B.

```
curl -X GET http://<host>/api/v1/datastreams
```

- einen Datenstrom in Richtung eines Ziels zu starten, z.B.

```
curl -X PUT --header 'Content-Type: application/x-www-form-urlencoded' -d 'destination=
↪<target-ip>:<target-port>' http://<host>/api/v1/datastreams/pose
```

- Datenströme zu stoppen, z.B.

```
curl -X DELETE http://<host>/api/v1/datastreams/pose?destination=<target-ip>:<target-port>
```

Die folgende Liste enthält alle REST-API-Anfragen zu Datenströmen:

`GET /datastreams`
Abruf einer Liste aller verfügbaren Datenströme.

Musteranfrage

```
GET /api/v2/datastreams HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "description": "Pose of left camera at VisualOdometry rate (~10Hz)",
    "destinations": [
      "192.168.1.13:30000"
    ],
    "name": "pose",
    "protobuf": "Frame",
    "protocol": "UDP"
  },
  {
    "description": "Pose of left camera (RealTime 200Hz)",
    "destinations": [
      "192.168.1.100:20000",
      "192.168.1.42:45000"
    ],
    "name": "pose_rt",
    "protobuf": "Frame",
    "protocol": "UDP"
  },
  {
    "description": "Raw IMU (InertialMeasurementUnit) values (RealTime 200Hz)",
    "destinations": [],
    "name": "imu",
    "protobuf": "Imu",
    "protocol": "UDP"
  },
  {
    "description": "Dynamics of sensor (pose, velocity, acceleration) (RealTime 200Hz)",
    "destinations": [
      "192.168.1.100:20001"
    ],
    "name": "dynamics",
    "protobuf": "Dynamics",
    "protocol": "UDP"
  }
]
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Datenstrom-Array)*

Referenzierte Datenmodelle

- *Stream* (Abschnitt 6.2.4)

`GET /datastreams/{stream}`
Abruf der Datenstrom-Konfiguration.

Musteranfrage

```
GET /api/v2/datastreams/<stream> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "description": "Pose of left camera at VisualOdometry rate (~10Hz)",
  "destinations": [
    "192.168.1.13:30000"
  ],
  "name": "pose",
  "protobuf": "Frame",
  "protocol": "UDP"
}
```

Parameter

- **`stream`** (*`string`*) – Name des Datenstroms *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Datenstrom)*
- 404 Not Found – Datenstrom nicht gefunden

Referenzierte Datenmodelle

- *Stream* (Abschnitt 6.2.4)

`PUT /datastreams/{stream}`
Aktualisierung einer Datenstrom-Konfiguration.

Musteranfrage

```
PUT /api/v2/datastreams/<stream> HTTP/1.1
Accept: application/x-www-form-urlencoded
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "description": "Pose of left camera at VisualOdometry rate (~10Hz)",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  "destinations": [
    "192.168.1.13:30000",
    "192.168.1.25:40000"
  ],
  "name": "pose",
  "protobuf": "Frame",
  "protocol": "UDP"
}
```


Parameter

- **stream** (*string*) – Name des Datenstroms *(obligatorisch)*

Formularparameter

- **destination** – Hinzuzufügendes Ziel („IP:Port“) *(obligatorisch)*

Anfrage-Header

- Accept – application/x-www-form-urlencoded

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Datenstrom)*
- 404 Not Found – Datenstrom nicht gefunden

Referenzierte Datenmodelle

- *Stream* (Abschnitt 6.2.4)

DELETE /datastreams/{stream}

Löschen eines Ziels aus der Datenstrom-Konfiguration.

Musteranfrage

```
DELETE /api/v2/datastreams/<stream>?destination=<destination> HTTP/1.1
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "description": "Pose of left camera at VisualOdometry rate (~10Hz)",
  "destinations": [],
  "name": "pose",
  "protobuf": "Frame",
  "protocol": "UDP"
}
```


Parameter

- **stream** (*string*) – Name des Datenstroms *(obligatorisch)*

Anfrageparameter

- **destination** (*string*) – Zu löschendes Ziel („IP:Port“): Fehlt die Angabe, werden alle Ziele gelöscht *(optional)*.

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Datenstrom)*
- 404 Not Found – Datenstrom nicht gefunden

Referenzierte Datenmodelle

- *Stream* (Abschnitt 6.2.4)

6.2.3.3 System und Logs

Die folgenden Ressourcen und Anfragen sind für die System-Level-API des *rc_visard* verfügbar. Sie ermöglichen Folgendes:

- Zugriff auf Logdateien (systemweit oder modulspezifisch),
- Abruf von Informationen zum Gerät und zur Laufzeitstatistik, wie Datum, MAC-Adresse, Uhrzeitsynchronisierungsstatus und verfügbare Ressourcen,
- Verwaltung installierter Softwarelizenzen, und
- Aktualisierung des Firmware-Images des *rc_visard*.

`GET /logs`
Abruf einer Liste aller verfügbaren Logdateien.

Musteranfrage

```
GET /api/v2/logs HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "date": 1503060035.0625782,
    "name": "rcsense-api.log",
    "size": 730
  },
  {
    "date": 1503060035.741574,
    "name": "stereo.log",
    "size": 39024
  },
  {
    "date": 1503060044.0475223,
    "name": "camera.log",
    "size": 1091
  },
  {
    "date": 1503060035.2115774,
    "name": "dynamics.log"
  }
]
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: LogInfo-Array)*

Referenzierte Datenmodelle

- *LogInfo* (Abschnitt 6.2.4)

`GET /logs/{log}`

Abruf einer Logdatei: Die Art des Inhalts der Antwort richtet sich nach dem *format*-Parameter.

Musteranfrage

```
GET /api/v2/logs/<log>?format=<format>&limit=<limit> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "date": 1503060035.2115774,
  "log": [
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Running rc_stereo_ins version 2.4.0",
      "timestamp": 1503060034.083
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Starting up communication interfaces",
      "timestamp": 1503060034.085
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Autostart disabled",
      "timestamp": 1503060034.098
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Initializing realtime communication",
      "timestamp": 1503060034.209
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Startet state machine in state IDLE",
      "timestamp": 1503060034.383
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "Init stereovisodo ...",
      "timestamp": 1503060034.814
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Using standard VO",
      "timestamp": 1503060034.913
    },
    {
      "component": "rc_stereovisodo",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "level": "INFO",
      "message": "rc_stereovisodo: Playback mode: false",
      "timestamp": 1503060035.132
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Ready",
      "timestamp": 1503060035.212
    }
  ],
  "name": "dynamics.log",
  "size": 695
}
```


Parameter

- **log** (*string*) – Name der Logdatei *(obligatorisch)*

Anfrageparameter

- **format** (*string*) – Rückgabe des Logs im JSON- oder Rohdatenformat (mögliche Werte: `json` oder `raw`; Voreinstellung: `json`) *(optional)*
- **limit** (*integer*) – Beschränkung auf die letzten x Zeilen im JSON-Format (Voreinstellung: `100`) *(optional)*

Antwort-Headers

- Content-Type – text/plain application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Log)*
- 404 Not Found – Log nicht gefunden

Referenzierte Datenmodelle

- *Log* (Abschnitt 6.2.4)

GET /system

Abruf von Systeminformationen zum Sensor.

Musteranfrage

```
GET /api/v2/system HTTP/1.1
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dns": {
    "dns_servers": [
      "10.0.0.1",
      "1.1.1.1"
    ],
    "manual_dns_servers": [
      "1.1.1.1"
    ]
  },
  "firmware": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "active_image": {
      "image_version": "1.1.0"
    },
    "fallback_booted": true,
    "inactive_image": {
      "image_version": "1.0.0"
    },
    "next_boot_image": "active_image"
  },
  "hostname": "rc-visard-02873515",
  "link_speed": 1000,
  "mac": "00:14:2D:2B:D8:AB",
  "ntp": {
    "enabled": true,
    "manual_ntp_servers": [
      "10.0.0.1"
    ],
    "offset": -3.2666e-05,
    "selected_ntp_servers": [
      "10.0.0.1"
    ],
    "synchronized": true
  },
  "ptp_status": {
    "master_ip": "",
    "offset": 0,
    "offset_dev": 0,
    "offset_mean": 0,
    "state": "off"
  },
  "ready": true,
  "serial": "02873515",
  "time": 1504080462.641875,
  "uptime": 65457.42
}
```


Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: SysInfo)*

Referenzierte Datenmodelle

- *SysInfo* (Abschnitt 6.2.4)

GET /system/backup

Abruf eines Backups der Einstellungen.

Musteranfrage

```
GET /api/v2/system/backup?pipelines=<pipelines>&load_carriers=<load_carriers>&regions_of_
↪interest=<regions_of_interest>&grippers=<grippers> HTTP/1.1
```

Anfrageparameter

- **pipelines** (*boolean*) – Backup der Pipelines mit Moduleinstellungen, d.h. Parameter und bevorzugte TCP-Orientierung (Standardwert: `True`) *(optional)*
- **load_carriers** (*boolean*) – Backup der Load Carrier (Standardwert: `True`) *(optional)*

- **regions_of_interest** (*boolean*) – Backup der Regions of Interest (Standardwert: `True`) *(optional)*
- **grippers** (*boolean*) – Backup der Greifer (Standardwert: `True`) *(optional)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

POST /system/backup

Backup einspielen.

Musteranfrage

```
POST /api/v2/system/backup HTTP/1.1
Accept: application/json application/ubjson

{}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "return_code": {
    "message": "backup restored",
    "value": 0
  },
  "warnings": []
}
```

Request JSON Object

- **backup** (*object*) – Backup-Daten als json-Objekt *(erforderlich)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/dns

Abfragen der DNS-Server Einstellungen.

Musteranfrage

```
GET /api/v2/system/dns HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dns": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "dns_servers": [
      "10.0.0.1",
      "1.1.1.1"
    ],
    "manual_dns_servers": [
      "1.1.1.1"
    ]
  }
}
```


Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: DNS)*

Referenzierte Datenmodelle

- *DNS* (Abschnitt 6.2.4)

`PUT /system/dns`

Setze manuelle DNS-Server.

Musteranfrage


```
PUT /api/v2/system/dns HTTP/1.1
Accept: application/json application/ubjson

{}
```


Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dns": {
    "dns_servers": [
      "10.0.0.1",
      "1.1.1.1"
    ],
    "manual_dns_servers": [
      "1.1.1.1"
    ]
  }
}
```


Request JSON Object

- **`manual_dns_servers`** (*`ManualDNSServers`*) – Manuelle DNS-Server *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: DNS)*

- 400 Bad Request – ungültige/fehlende Argumente

Referenzierte Datenmodelle

- *DNS* (Abschnitt 6.2.4)
- *ManualDNSServers* (Abschnitt 6.2.4)

`GET /system/license`
Abruf von Informationen zu den auf dem Sensor installierten Lizenzen.

Musteranfrage

```
GET /api/v2/system/license HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "components": {
    "calibration": true,
    "fusion": true,
    "hand_eye_calibration": true,
    "rectification": true,
    "self_calibration": true,
    "slam": false,
    "stereo": true,
    "svo": true
  },
  "valid": true
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: LicenseInfo)*

Referenzierte Datenmodelle

- *LicenseInfo* (Abschnitt 6.2.4)

`POST /system/license`
Aktualisierung der auf dem Sensor installierten Lizenz mithilfe einer Lizenzdatei.

Musteranfrage

```
POST /api/v2/system/license HTTP/1.1
Accept: multipart/form-data
```

Formularparameter

- **`file`** – Lizenzdatei *(obligatorisch)*

Anfrage-Header

- Accept – Multipart/Formulardaten

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – Keine gültige Lizenz

GET /system/max_power_test
Abruf des letzten Maximalleistungstests.

Musteranfrage

```
GET /api/v2/system/max_power_test HTTP/1.1
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

POST /system/max_power_test
Ausführen des Maximalleistungstests. Führt eine maximale Belastung der GPU (und CPU) durch, damit 10 Sekunden lang maximale Leistung verbraucht wird, um die Stromversorgung zu testen. WARNUNG: Das System gibt aufgrund eines sofortigen Neustarts möglicherweise keine Antwort zurück, wenn die Stromversorgung nicht ausreicht.

Musteranfrage

```
POST /api/v2/system/max_power_test?nocpu=<nocpu> HTTP/1.1
```

Anfrageparameter

- **nocpu** (*boolean*) – Nur die GPU belasten, keine CPU Worker ausführen *(optional)*

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Test abgeschlossen. Siehe return_code für das Ergebnis.
- 400 Bad Request – Test läuft bereits.

GET /system/network
Abruf der aktuellen Netzwerk Konfiguration.

Musteranfrage

```
GET /api/v2/system/network HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "current_method": "DHCP",
  "default_gateway": "10.0.3.254",
  "ip_address": "10.0.1.41",
  "settings": {
    "dhcp_enabled": true,
    "persistent_default_gateway": "",
    "persistent_ip_address": "192.168.0.10",
    "persistent_ip_enabled": false,
    "persistent_subnet_mask": "255.255.255.0"
  },
  "subnet_mask": "255.255.252.0"
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NetworkInfo)*

Referenzierte Datenmodelle

- *NetworkInfo* (Abschnitt 6.2.4)

`GET /system/network/settings`

Abruf der aktuellen Netzwerkeinstellungen.

Musteranfrage

```
GET /api/v2/system/network/settings HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dhcp_enabled": true,
  "persistent_default_gateway": "",
  "persistent_ip_address": "192.168.0.10",
  "persistent_ip_enabled": false,
  "persistent_subnet_mask": "255.255.255.0"
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NetworkSettings)*

Referenzierte Datenmodelle

- *NetworkSettings* (Abschnitt 6.2.4)

`PUT /system/network/settings`

Setzen der aktuellen Netzwerkeinstellungen.

Musteranfrage

```
PUT /api/v2/system/network/settings HTTP/1.1
Accept: application/json application/ubjson

{}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dhcp_enabled": true,
  "persistent_default_gateway": "",
  "persistent_ip_address": "192.168.0.10",
  "persistent_ip_enabled": false,
  "persistent_subnet_mask": "255.255.255.0"
}
```

Request JSON Object

- **settings** (*NetworkSettings*) – Anzuwendende Netzwerkeinstellungen *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NetworkSettings)*
- 400 Bad Request – ungültige/fehlende Argumente
- 403 Forbidden – Das Ändern der Netzwerkeinstellungen ist nicht erlaubt, da eine laufende GigE Vision-Applikation diese sperrt.

Referenzierte Datenmodelle

- *NetworkSettings* (Abschnitt 6.2.4)

GET /system/ntp

Abfragen der NTP Einstellungen.

Musteranfrage

```
GET /api/v2/system/ntp HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "ntp": {
    "enabled": true,
    "manual_ntp_servers": [
      "10.0.0.1"
    ],
    "offset": -3.2666e-05,
    "selected_ntp_servers": [
      "10.0.0.1"
    ],
    "synchronized": true
  }
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NTP)*

Referenzierte Datenmodelle

- *NTP* (Abschnitt 6.2.4)

PUT /system/ntp

Setze manuelle NTP-Server.

Musteranfrage

```
PUT /api/v2/system/ntp HTTP/1.1
Accept: application/json application/ubjson

{}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "ntp": {
    "enabled": true,
    "manual_ntp_servers": [
      "10.0.0.1"
    ],
    "offset": -3.2666e-05,
    "selected_ntp_servers": [
      "10.0.0.1"
    ],
    "synchronized": true
  }
}
```

Request JSON Object

- **manual_ntp_servers** (*ManualNTPServers*) – Manuelle NTP-Server *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NTP)*
- 400 Bad Request – ungültige/fehlende Argumente

Referenzierte Datenmodelle

- *NTP* (Abschnitt 6.2.4)
- *ManualNTPServers* (Abschnitt 6.2.4)

PUT /system/reboot

Neustart des Sensors.

Musteranfrage

```
PUT /api/v2/system/reboot HTTP/1.1
```

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/rollback

Abruf von Informationen zu Firmware/System-Images, die aktuell auf dem Sensor aktiv oder inaktiv sind.

Musteranfrage

```
GET /api/v2/system/rollback HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "active_image": {
    "image_version": "1.1.0"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "1.0.0"
  },
  "next_boot_image": "active_image"
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: FirmwareInfo)*

Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 6.2.4)

PUT /system/rollback

Rollback auf vorherige Firmware-Version (inaktives System-Image).

Musteranfrage

```
PUT /api/v2/system/rollback HTTP/1.1
```

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – Bereits auf die Verwendung der inaktiven Partition beim nächsten Boot-Vorgang gesetzt.
- 500 Internal Server Error – Interner Fehler

GET /system/time

Abfrage der Systemzeit in UTC als String mit dem Format „YYYY-MM-DD hh:mm:ss“

Musteranfrage

```
GET /api/v2/system/time HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "utc": "2023-10-05 08:35:26"
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

PUT /system/time

Setzen der Systemzeit in UTC als String mit dem Format „YYYY-MM-DD hh:mm:ss“

Musteranfrage

```
PUT /api/v2/system/time?utc=<utc> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "utc": "2023-10-05 08:35:26"
}
```

Anfrageparameter

- **utc** (*string*) – Zeit in UTC als String mit dem Format „YYYY-MM-DD hh:mm:ss“ (obligatorisch)

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – ungültige/fehlende Argumente
- 403 Forbidden – Ändern der Zeit nicht erlaubt, da Zeitsynchronisation via NTP oder PTP aktiv ist.

GET /system/ui_lock

Abruf des UI Lock Status

Musteranfrage

```
GET /api/v2/system/ui_lock HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": false
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: UILock)*

Referenzierte Datenmodelle

- *UILock* (Abschnitt 6.2.4)

DELETE /system/ui_lock
UI Lock entfernen.

Musteranfrage

```
DELETE /api/v2/system/ui_lock HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": false,
  "valid": false
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

POST /system/ui_lock
Verifizieren oder Setzen des UI Locks.

Musteranfrage

```
POST /api/v2/system/ui_lock?hash=<hash>&set=<set> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "valid": true
}
```

Anfrageparameter

- **hash** (*string*) – Hash des UI Lock Passworts *(obligatorisch)*
- **set** (*boolean*) – neuen Hash setzen anstatt zu verifizieren *(optional)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/update
Abruf von Informationen zu Firmware/System-Images, die aktuell auf dem Sensor aktiv oder inaktiv sind.

Musteranfrage

```
GET /api/v2/system/update HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "active_image": {
    "image_version": "1.1.0"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "1.0.0"
  },
  "next_boot_image": "active_image"
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: FirmwareInfo)*

Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 6.2.4)

`POST /system/update`
Aktualisierung des Firmware/System-Images mit einer Mender-Artefakt-Datei: Um die aktualisierte Firmware zu aktivieren, ist anschließend ein Neustart erforderlich.

Musteranfrage

```
POST /api/v2/system/update HTTP/1.1
Accept: multipart/form-data
```

Formularparameter

- **`file`** – Mender-Artefakt-Datei *(obligatorisch)*

Anfrage-Header

- Accept – Multipart/Formulardaten

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – Client-Fehler, z.B. kein gültiges Mender-Artefakt

6.2.4 Datentyp-Definitionen

Die REST-API definiert folgende Datenmodelle, die verwendet werden, um auf die *verfügbaren Ressourcen* (Abschnitt 6.2.3) zuzugreifen oder diese zu ändern, entweder als benötigte Attribute/Parameter oder als Rückgabewerte.

DNS: DNS-Server Einstellungen.

Ein Objekt des Typs DNS besitzt folgende Eigenschaften:

- **dns_servers** (string-Array)
- **manual_dns_servers** (string-Array)

Musterobjekt

```
{
  "dns_servers": [
    "string",
    "string"
  ],
  "manual_dns_servers": [
    "string",
    "string"
  ]
}
```


DNS-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/dns*
- *PUT /system/dns*

FirmwareInfo: Informationen zu aktuell aktiven und inaktiven Firmware-Images und dazu, welches Image für den Boot-Vorgang verwendet wird.

Ein Objekt des Typs FirmwareInfo besitzt folgende Eigenschaften:

- **active_image** (*ImageInfo*): siehe Beschreibung von *ImageInfo*.
- **fallback_booted** (boolean): TRUE, wenn das gewünschte Image nicht hochgefahren werden konnte und ein Fallback auf das zuvor genutzte Image vorgenommen wurde.
- **inactive_image** (*ImageInfo*): siehe Beschreibung von *ImageInfo*.
- **next_boot_image** (string): Firmware-Image, das beim nächsten Neustart geladen wird (entweder `active_image` oder `inactive_image`).

Musterobjekt


```
{
  "active_image": {
    "image_version": "string"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "string"
  },
  "next_boot_image": "string"
}
```


FirmwareInfo-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/rollback*
- *GET /system/update*

GripperElement: CAD-Greiferelement

Ein Objekt des Typs GripperElement besitzt folgende Eigenschaften:

- **id** (string): Eindeutiger Name des Elements

Musterobjekt


```
{
  "id": "string"
}
```


GripperElement-Objekte werden in folgenden Anfragen verwendet:

- *GET /cad/gripper_elements*
- *GET /cad/gripper_elements/{id}*

- *PUT /cad/gripper_elements/{id}*

ImageInfo: Informationen zu einem bestimmten Firmware-Image.

Ein Objekt des Typs ImageInfo besitzt folgende Eigenschaften:

- **image_version** (string): Image-Version.

Musterobjekt

```
{
  "image_version": "string"
}
```

ImageInfo-Objekte sind in *FirmwareInfo* enthalten.

LicenseComponentConstraint: Einschränkungen für die Modul-Version.

Ein Objekt des Typs LicenseComponentConstraint besitzt folgende Eigenschaften:

- **max_version** (string) - optionale höchste unterstützte Version (exclusive)
- **min_version** (string) - optionale minimale unterstützte Version (inclusive)

Musterobjekt

```
{
  "max_version": "string",
  "min_version": "string"
}
```

LicenseComponentConstraint-Objekte sind in *LicenseConstraints* enthalten.

LicenseComponents: Liste der Lizenzstatus-Angaben der einzelnen Softwaremodule: Der zugehörige Statusindikator ist auf TRUE gesetzt, wenn das entsprechende Modul mit einer installierten Softwarelizenz entsperrt ist.

Ein Objekt des Typs LicenseComponents besitzt folgende Eigenschaften:

- **calibration** (boolean): Modul zur Kamerakalibrierung.
- **fusion** (boolean): Modul zur Stereo-INS/Datenfusion.
- **hand_eye_calibration** (boolean): Modul zur Hand-Auge-Kalibrierung.
- **rectification** (boolean): Modul zur Bildrektifizierung.
- **self_calibration** (boolean): Modul zur Selbstkalibrierung der Kamera.
- **slam** (boolean): SLAM-Modul.
- **stereo** (boolean): Stereo-Matching-Modul.
- **svo** (boolean): visuelle Odometrie-Modul.

Musterobjekt

```
{
  "calibration": false,
  "fusion": false,
  "hand_eye_calibration": false,
  "rectification": false,
  "self_calibration": false,
  "slam": false,
  "stereo": false,
  "svo": false
}
```

LicenseComponents-Objekte sind in *LicenseInfo* enthalten.

LicenseConstraints: Versionseinschränkungen für Module.

Ein Objekt des Typs LicenseConstraints besitzt folgende Eigenschaften:

- **image_version** (*LicenseComponentConstraint*) - siehe Beschreibung von *LicenseComponentConstraint*

Musterobjekt

```
{
  "image_version": {
    "max_version": "string",
    "min_version": "string"
  }
}
```

LicenseConstraints-Objekte sind in *LicenseInfo* enthalten.

LicenseInfo: Informationen zur aktuell auf dem Sensor angewandten Softwarelizenz.

Ein Objekt des Typs LicenseInfo besitzt folgende Eigenschaften:

- **components** (*LicenseComponents*): siehe Beschreibung von *LicenseComponents*.
- **components_constraints** (*LicenseConstraints*) - siehe Beschreibung von *LicenseConstraints*
- **valid** (boolean): Angabe, ob eine Lizenz gültig ist oder nicht.

Musterobjekt

```
{
  "components": {
    "calibration": false,
    "fusion": false,
    "hand_eye_calibration": false,
    "rectification": false,
    "self_calibration": false,
    "slam": false,
    "stereo": false,
    "svo": false
  },
  "components_constraints": {
    "image_version": {
      "max_version": "string",
      "min_version": "string"
    }
  },
  "valid": false
}
```

LicenseInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /system/license*

Log: Inhalt einer bestimmten Logdatei im JSON-Format.

Ein Objekt des Typs Log besitzt folgende Eigenschaften:

- **date** (float): UNIX-Uhrzeit, zu der das Log zuletzt geändert wurde.
- **log** (*LogEntry*-Array): die eigentlichen Logeinträge.
- **name** (string): Name der Logdatei.
- **size** (Integer): Größe der Logdatei in Bytes.

Musterobjekt

```
{
  "date": 0,
  "log": [
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    },
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    }
  ],
  "name": "string",
  "size": 0
}
```


Log-Objekte werden in folgenden Anfragen verwendet:

- *GET /logs/{log}*

LogEntry: Darstellung eines einzelnen Logeintrags in einer Logdatei.

Ein Objekt des Typs LogEntry besitzt folgende Eigenschaften:

- **component** (string): Name des Moduls, das diesen Eintrag angelegt hat.
- **level** (string): Logstufe (mögliche Werte: `DEBUG`, `INFO`, `WARN`, `ERROR` oder `FATAL`)
- **message** (string): eigentliche Lognachricht.
- **timestamp** (float): UNIX-Uhrzeit des Logeintrags.

Musterobjekt


```
{
  "component": "string",
  "level": "string",
  "message": "string",
  "timestamp": 0
}
```


LogEntry-Objekte sind in *Log* enthalten.

LogInfo: Informationen zu einer bestimmten Logdatei.

Ein Objekt des Typs LogInfo besitzt folgende Eigenschaften:

- **date** (float): UNIX-Uhrzeit, zu der das Log zuletzt geändert wurde.
- **name** (string): Name der Logdatei.
- **size** (Integer): Größe der Logdatei in Bytes.

Musterobjekt


```
{
  "date": 0,
  "name": "string",
  "size": 0
}
```


LogInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /logs*

ManualDNSServers: Liste der manuellen DNS-Server.

Ein Objekt des Typs ManualDNSServers besitzt folgende Eigenschaften:

- **manual_dns_servers** (string-Array)

Musterobjekt

```
{
  "manual_dns_servers": [
    "string",
    "string"
  ]
}
```

ManualDNSServers-Objekte werden in folgenden Anfragen verwendet:

- *PUT /system/dns*

ManualNTPServers: Liste der manuellen NTP-Server.

Ein Objekt des Typs ManualNTPServers besitzt folgende Eigenschaften:

- **manual_ntp_servers** (string-Array)

Musterobjekt

```
{
  "manual_ntp_servers": [
    "string",
    "string"
  ]
}
```

ManualNTPServers-Objekte werden in folgenden Anfragen verwendet:

- *PUT /system/ntp*

NTP: Status der NTP-Zeitsynchronisierung.

Ein Objekt des Typs NTP besitzt folgende Eigenschaften:

- **enabled** (boolean) - NTP ist eingeschaltet
- **manual_ntp_servers** (string-Array)
- **offset** (string): von NTP gemeldeter Zeit-Offset
- **selected_ntp_servers** (string-Array)
- **synchronized** (boolean): synchronisiert mit dem NTP-Server.

Musterobjekt

```
{
  "enabled": false,
  "manual_ntp_servers": [
    "string",
    "string"
  ],
  "offset": "string",
  "selected_ntp_servers": [
    "string",
    "string"
  ],
  "synchronized": false
}
```

NTP-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/ntp*
- *PUT /system/ntp*

NetworkInfo: Aktuelle Netzwerk Konfiguration.

Ein Objekt des Typs NetworkInfo besitzt folgende Eigenschaften:

- **current_method** (string) - Methode mit der die aktuellen Einstellungen gesetzt wurden (mögliche Werte: `INIT`, `LinkLocal`, `DHCP`, `PersistentIP`, `TemporaryIP`)
- **default_gateway** (string) - aktueller Default Gateway
- **ip_address** (string) - aktuelle IP-Adresse
- **settings** (*NetworkSettings*) - siehe Beschreibung von *NetworkSettings*
- **subnet_mask** (string) - aktuelle Subnetzmaske

Musterobjekt

```
{
  "current_method": "string",
  "default_gateway": "string",
  "ip_address": "string",
  "settings": {
    "dhcp_enabled": false,
    "persistent_default_gateway": "string",
    "persistent_ip_address": "string",
    "persistent_ip_enabled": false,
    "persistent_subnet_mask": "string"
  },
  "subnet_mask": "string"
}
```

NetworkInfo-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/network*

NetworkSettings: Aktuelle Netzwerk Einstellungen.

Ein Objekt des Typs NetworkSettings besitzt folgende Eigenschaften:

- **dhcp_enabled** (boolean) - DHCP eingeschaltet
- **persistent_default_gateway** (string) - Persistenter Default Gateway
- **persistent_ip_address** (string) - Persistente IP-Adresse
- **persistent_ip_enabled** (boolean) - Persistente IP aktiviert
- **persistent_subnet_mask** (string) - Persistente Subnetzmaske

Musterobjekt

```
{
  "dhcp_enabled": false,
  "persistent_default_gateway": "string",
  "persistent_ip_address": "string",
  "persistent_ip_enabled": false,
  "persistent_subnet_mask": "string"
}
```

NetworkSettings-Objekte sind in *NetworkInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/network/settings*
- *PUT /system/network/settings*

NodeInfo: Beschreibung eines auf dem Sensor laufenden Softwaremoduls.

Ein Objekt des Typs NodeInfo besitzt folgende Eigenschaften:

- **name** (string): Name des Moduls.
- **parameters** (string-Array): Liste der Laufzeitparameter des Moduls.
- **services** (string-Array): Liste der von diesem Modul angebotenen Services.
- **status** (string): Status des Moduls (mögliche Werte: `unknown`, `down`, `idle` oder `running`).

Musterobjekt

```
{
  "name": "string",
  "parameters": [
    "string",
    "string"
  ],
  "services": [
    "string",
    "string"
  ],
  "status": "string"
}
```

NodeInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes*
- *GET /nodes/{node}*
- *GET /pipelines/{pipeline}/nodes*
- *GET /pipelines/{pipeline}/nodes/{node}*

NodeStatus: Detaillierter aktueller Status des Moduls, einschließlich Laufzeitstatistik.

Ein Objekt des Typs NodeStatus besitzt folgende Eigenschaften:

- **status** (string): Status des Moduls (mögliche Werte: `unknown`, `down`, `idle` oder `running`).
- **timestamp** (float): UNIX-Uhrzeit, zu der die Werte zuletzt aktualisiert wurden.
- **values** (object): Dictionary (Schlüssel-Werte-Auflistung) mit den aktuellen Statuswerten/Statistiken des Moduls.

Musterobjekt

```
{
  "status": "string",
  "timestamp": 0,
  "values": {}
}
```

NodeStatus-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes/{node}/status*
- *GET /pipelines/{pipeline}/nodes/{node}/status*

Parameter: Darstellung der Laufzeitparameter eines Moduls: Der Datentyp des Werts („value“) eines Parameters (und damit der Datentyp der Felder „min“, „max“ und „default“) lässt sich vom Feld „type“ ableiten und kann ein primitiver Datentyp sein.

Ein Objekt des Typs Parameter besitzt folgende Eigenschaften:

- **default** (Typ nicht definiert): ab Werk voreingestellter Wert des Parameters.
- **description** (string): Beschreibung des Parameters.

- **max** (Typ nicht definiert): Höchstwert, der diesem Parameter zugewiesen werden kann.
- **min** (Typ nicht definiert): Mindestwert, der diesem Parameter zugewiesen werden kann.
- **name** (string): Name des Parameters.
- **type** (string): als Zeichenfolge dargestellter primitiver Datentyp des Parameters (mögliche Werte: `bool`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `float32`, `float64` oder `string`).
- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt

```
{
  "default": {},
  "description": "string",
  "max": {},
  "min": {},
  "name": "string",
  "type": "string",
  "value": {}
}
```

Parameter-Objekte werden in folgenden Anfragen verwendet:

- *GET /pipelines/{pipeline}/nodes/{node}/parameters*
- *PUT /pipelines/{pipeline}/nodes/{node}/parameters*
- *GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}*
- *PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}*

ParameterNameValue: Parametername und -wert. Der Typ des Parameterwerts (Felder ‚value' und ‚min', ‚max', ‚default') ist durch das Feld ‚type' angegeben und kann einer der eingebauten primitiven Datentypen sein.

Ein Objekt des Typs ParameterNameValue besitzt folgende Eigenschaften:

- **name** (string): Name des Parameters.
- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt

```
{
  "name": "string",
  "value": {}
}
```

ParameterNameValue-Objekte werden in folgenden Anfragen verwendet:

- *PUT /pipelines/{pipeline}/nodes/{node}/parameters*

ParameterValue: Parameterwert. Der Typ des Parameterwerts (Felder ‚value' und ‚min', ‚max', ‚default') ist durch das Feld ‚type' angegeben und kann einer der eingebauten primitiven Datentypen sein.

Ein Objekt des Typs ParameterValue besitzt folgende Eigenschaften:

- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt

```
{
  "value": {}
}
```

ParameterValue-Objekte werden in folgenden Anfragen verwendet:

- *PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}*

PtpStatus: Status der PTP-Zeitsynchronisierung gemäß IEEE 1588.

Ein Objekt des Typs PtpStatus besitzt folgende Eigenschaften:

- **master_ip** (string): IP-Adresse des Haupttaktgebers.
- **offset** (float): zeitlicher Versatz zum Haupttaktgeber in Sekunden.
- **offset_dev** (float): Standardabweichung des zeitlichen Versatzes zum Haupttaktgeber in Sekunden.
- **offset_mean** (float): mittlere Zeitverschiebung in Sekunden zum Haupttaktgeber.
- **state** (string): PTP-Zustand (mögliche Werte: `off`, `unknown`, `INITIALIZING`, `FAULTY`, `DISABLED`, `LISTENING`, `PASSIVE`, `UNCALIBRATED` oder `SLAVE`).

Musterobjekt

```
{
  "master_ip": "string",
  "offset": 0,
  "offset_dev": 0,
  "offset_mean": 0,
  "state": "string"
}
```

PtpStatus-Objekte sind in *SysInfo* enthalten.

Service: Darstellung eines von einem Modul angebotenen Services.

Ein Objekt des Typs Service besitzt folgende Eigenschaften:

- **args** (*ServiceArgs*): siehe Beschreibung von *ServiceArgs*.
- **description** (string): Kurzbeschreibung des Services.
- **name** (string): Name des Services.
- **response** (*ServiceResponse*): siehe Beschreibung von *ServiceResponse*.

Musterobjekt

```
{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Service-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes/{node}/services*
- *GET /nodes/{node}/services/{service}*
- *PUT /nodes/{node}/services/{service}*
- *GET /pipelines/{pipeline}/nodes/{node}/services*
- *GET /pipelines/{pipeline}/nodes/{node}/services/{service}*
- *PUT /pipelines/{pipeline}/nodes/{node}/services/{service}*

ServiceArgs: Argumente, die für den Aufruf eines Services benötigt werden: Diese Argumente werden in der Regel in einem (verschachtelten) Dictionary (Schlüssel-Werte-Auflistung) dargestellt. Der genaue Inhalt dieses Dictionarys hängt vom jeweiligen Modul und vom Serviceaufruf ab.

ServiceArg-Objekte sind in *Service* enthalten.

ServiceResponse: Die von dem Serviceaufruf zurückgegebene Antwort: Die Antwort wird in der Regel in einem (verschachtelten) Dictionary (Schlüssel-Werte-Auflistung) dargestellt. Der genaue Inhalt dieses Dictionarys hängt vom jeweiligen Modul und von dem Serviceaufruf ab.

ServiceResponse-Objekte sind in *Service* enthalten.

Stream: Darstellung eines von der rc_dynamics-Schnittstelle bereitgestellten Datenstroms.

Ein Objekt des Typs Stream besitzt folgende Eigenschaften:

- **destinations** (*StreamDestination*-Array): Liste der Ziele, an welche diese Daten aktuell gestreamt werden.
- **name** (string): Name des Datenstroms, der angibt, welche rc_dynamics-Daten gestreamt werden.
- **type** (*StreamType*): siehe Beschreibung von *StreamType*.

Musterobjekt

```
{
  "destinations": [
    "string",
    "string"
  ],
  "name": "string",
  "type": {
    "protobuf": "string",
    "protocol": "string"
  }
}
```

Stream-Objekte werden in folgenden Anfragen verwendet:

- *GET /datastreams*
- *GET /datastreams/{stream}*
- *PUT /datastreams/{stream}*
- *DELETE /datastreams/{stream}*

StreamDestination: Ein Ziel eines rc_dynamics-Datenstroms, dargestellt als Zeichenfolge wie z.B. ‚IP:Port'.

Ein Objekt des Typs StreamDestination ist eine Zeichenfolge.

StreamDestination-Objekte sind in *Stream* enthalten.

StreamType: Beschreibung eines Datenstromprotokolls.

Ein Objekt des Typs StreamType besitzt folgende Eigenschaften:

- **protobuf** (string): Datenformat zur Serialisierung, d.h. Name der ProtoBuf-Nachrichtendefinition.
- **protocol** (string): Netzwerkprotokoll des Streams (UDP).

Musterobjekt

```
{
  "protobuf": "string",
  "protocol": "string"
}
```

StreamType-Objekte sind in *Stream* enthalten.

SysInfo: Systeminformation zum Sensor.

Ein Objekt des Typs SysInfo besitzt folgende Eigenschaften:

- **dns** (*DNS*): siehe Beschreibung von *DNS*.
- **firmware** (*FirmwareInfo*): siehe Beschreibung von *FirmwareInfo*.
- **hostname** (string): Host-Name.
- **link_speed** (Integer): Ethernet-Verbindungsgeschwindigkeit in Mb/Sekunde.
- **mac** (string): MAC-Adresse.
- **network** (*NetworkInfo*): siehe Beschreibung von *NetworkInfo*
- **dns** (*NTP*): siehe Beschreibung von *NTP*.
- **ptp_status** (*PtpStatus*): siehe Beschreibung von *PtpStatus*.
- **ready** (boolean): Das System ist vollständig hochgefahren und betriebsbereit.
- **serial** (string): Seriennummer des Sensors.
- **time** (float): Systemzeit als UNIX-Zeitstempel.
- **ui_lock** (*UILock*): siehe Beschreibung von *UILock*
- **uptime** (float): Betriebszeit in Sekunden.

Musterobjekt


```
{
  "dns": {
    "dns_servers": [
      "string",
      "string"
    ],
    "manual_dns_servers": [
      "string",
      "string"
    ]
  },
  "firmware": {
    "active_image": {
      "image_version": "string"
    },
    "fallback_booted": false,
    "inactive_image": {
      "image_version": "string"
    },
    "next_boot_image": "string"
  },
  "hostname": "string",
  "link_speed": 0,
  "mac": "string",
  "network": {
    "current_method": "string",
    "default_gateway": "string",
    "ip_address": "string",
    "settings": {
      "dhcp_enabled": false,
      "persistent_default_gateway": "string",
      "persistent_ip_address": "string",
      "persistent_ip_enabled": false,
      "persistent_subnet_mask": "string"
    },
    "subnet_mask": "string"
  },
  "ntp": {
    "enabled": false,
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "manual_ntp_servers": [
      "string",
      "string"
    ],
    "offset": "string",
    "selected_ntp_servers": [
      "string",
      "string"
    ],
    "synchronized": false
  },
  "ptp_status": {
    "master_ip": "string",
    "offset": 0,
    "offset_dev": 0,
    "offset_mean": 0,
    "state": "string"
  },
  "ready": false,
  "serial": "string",
  "time": 0,
  "ui_lock": {
    "enabled": false
  },
  "uptime": 0
}
```


SysInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /system*

Template: Template für die Erkennung

Ein Objekt des Typs Template besitzt folgende Eigenschaften:

- **id** (string): Eindeutiger Name des Templates

Musterobjekt


```
{
  "id": "string"
}
```


Template-Objekte werden in folgenden Anfragen verwendet:

- *GET /templates/rc_boxpick*
- *GET /templates/rc_boxpick/{id}*
- *PUT /templates/rc_boxpick/{id}*
- *GET /templates/rc_silhouettematch*
- *GET /templates/rc_silhouettematch/{id}*
- *PUT /templates/rc_silhouettematch/{id}*

UILock: UI Lock Status.

Ein Objekt des Typs UILock besitzt folgende Eigenschaften:

- **enabled** (boolean)

Musterobjekt

```
{
  "enabled": false
}
```


UILock-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/ui_lock*

6.2.5 Swagger UI

Die Swagger UI des *rc_visard* ermöglicht es Entwicklern, die REST-API – beispielsweise zu Entwicklungs- und Testzwecken – leicht darzustellen und zu verwenden. Der Zugriff auf `http://<host>/api/` oder auf `http://<host>/api/swagger` (der erste Link leitet automatisch auf den zweiten Link weiter) öffnet eine Vorschau der allgemeinen API-Struktur des *rc_visard*, einschließlich aller *verfügbaren Ressourcen und Anfragen* (Abschnitt 6.2.3). Auf dieser vereinfachten Benutzeroberfläche lassen sich alle Funktionen erkunden und austesten.

Bemerkung: Der Benutzer muss bedenken, dass die Swagger UI des *rc_visard*, auch wenn sie zur Erprobung der REST-API bestimmt ist, eine voll funktionstüchtige Schnittstelle ist. Das bedeutet, dass alle ausgelösten Anfragen tatsächlich bearbeitet werden und den Zustand und/oder das Verhalten des Geräts beeinflussen. Dies gilt insbesondere für Anfragen des Typs `PUT`, `POST` und `DELETE`.

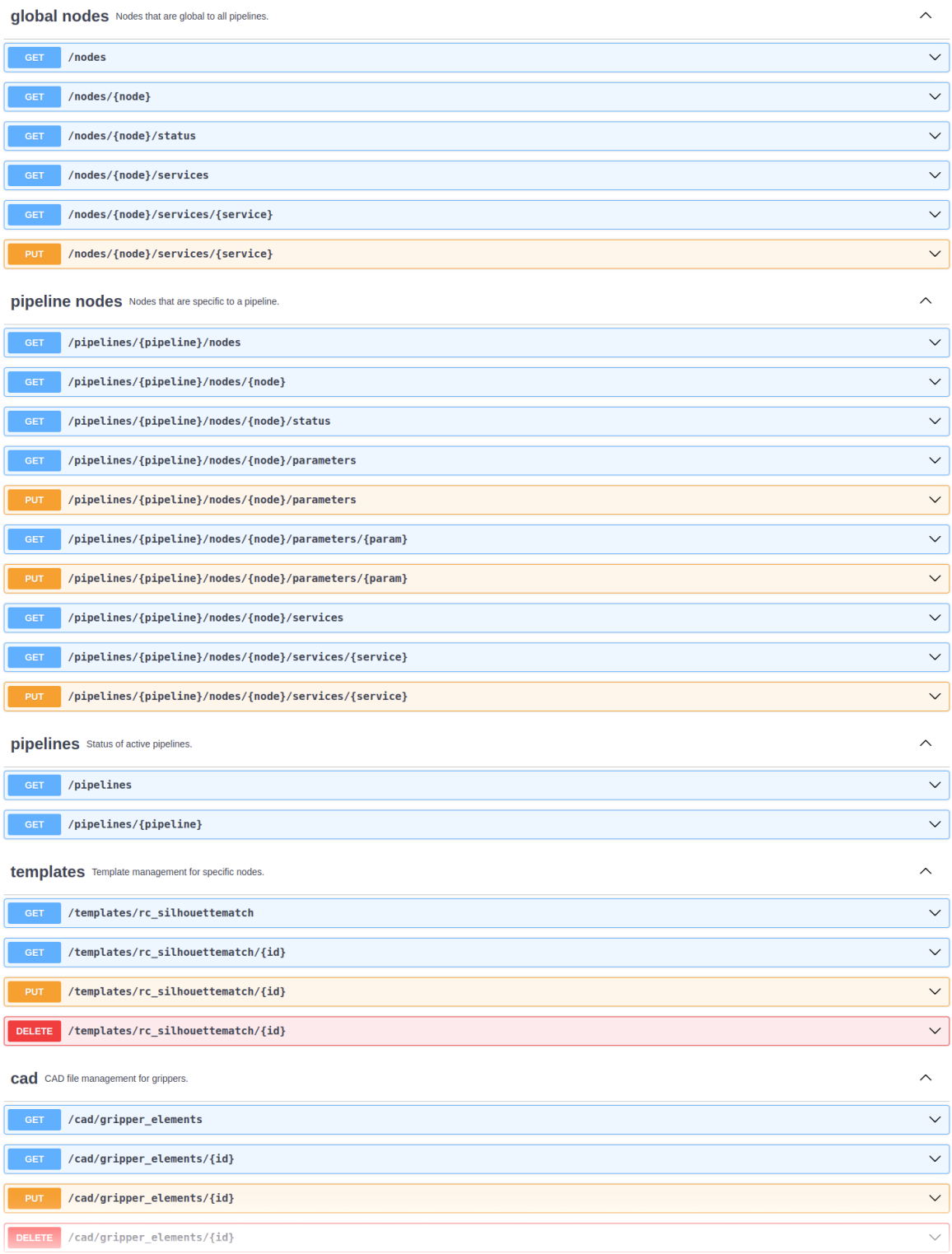


Abb. 6.2: Startansicht der Swagger UI des *rc_visard* mit den Ressourcen und Anfragen

Mithilfe dieser Schnittstelle können alle verfügbaren Ressourcen und Anfragen erprobt werden, indem diese durch Klick auf- und zugeklappt werden. Die folgende Abbildung zeigt ein Beispiel dafür, wie sich der aktuelle Zustand eines Moduls abrufen lässt, indem die erforderlichen Parameter (`pipeline`-Nummer und `node`-Name) ausgefüllt werden und anschließend *Execute* geklickt wird. Daraufhin zeigt die Swagger UI unter anderem den `curl`-Befehl an, der bei Auslösung der Anfrage ausgeführt wurde, sowie den Antworttext, in dem der aktuelle Status des angefragten Moduls in einer Zeichenfolge im

JSON-Format enthalten ist.

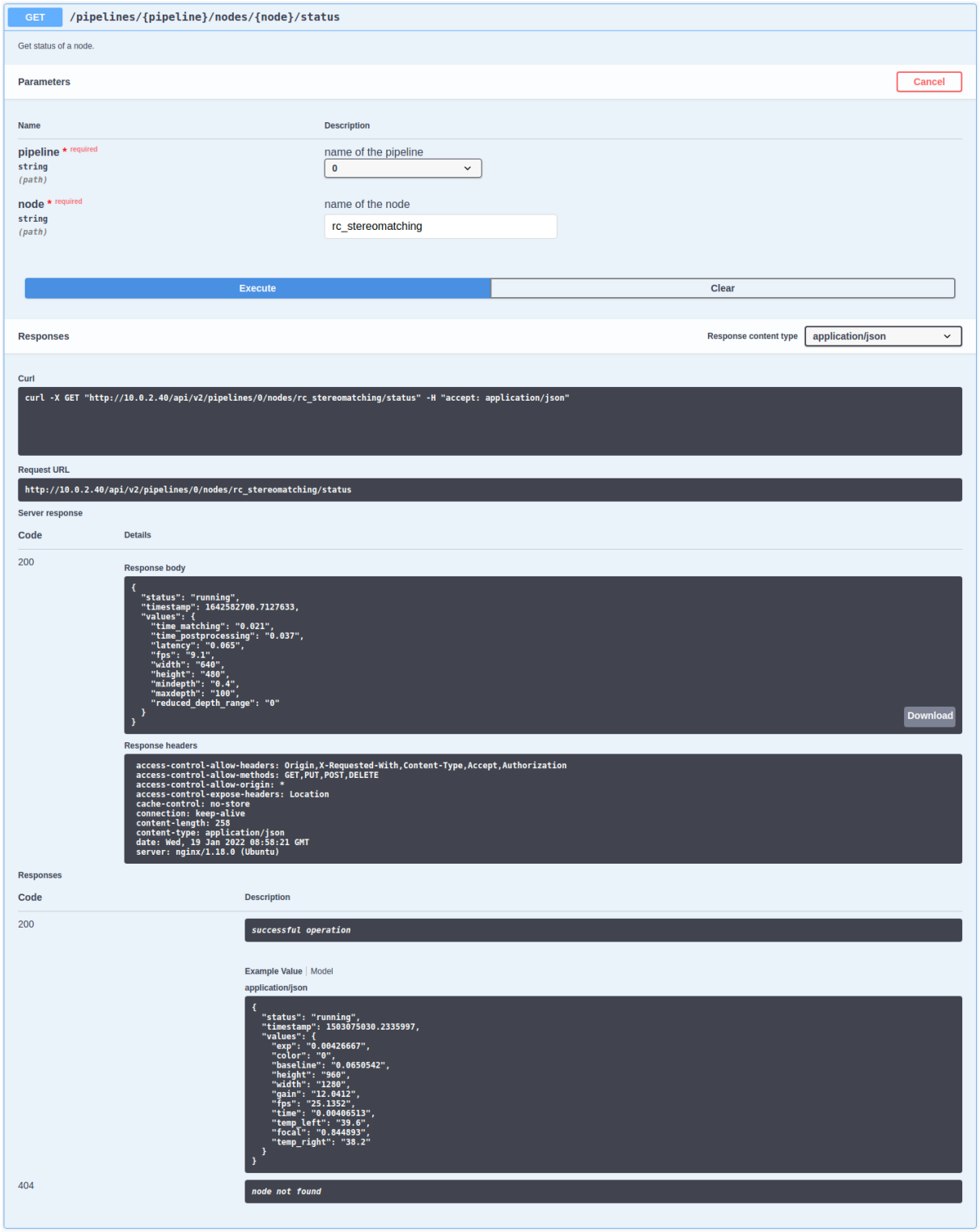


Abb. 6.3: Ergebnis nach Abfrage des Status des `rc_stereomatching`-Moduls

Einige Aktionen, wie das Setzen von Parametern oder der Aufruf von Services, bedürfen komplexerer Parameter als eine HTTP-Anfrage. Die Swagger UI erlaubt es Entwicklern, die für diese Aktionen benötigten Attribute, wie im nächsten Beispiel gezeigt, während der Laufzeit zu erkunden. In der folgenden Abbildung werden die Attribute, die für den `set_pose`-Service des `rc_hand_eye_calibration`-Moduls benötigt werden, erkundet, indem eine `GET`-Anfrage zu dieser Ressource durchgeführt wird. Die Ant-

wort enthält eine vollständige Beschreibung des angebotenen Services, einschließlich aller erforderlichen Argumente mit ihren Namen und Typen in einer Zeichenfolge im JSON-Format.

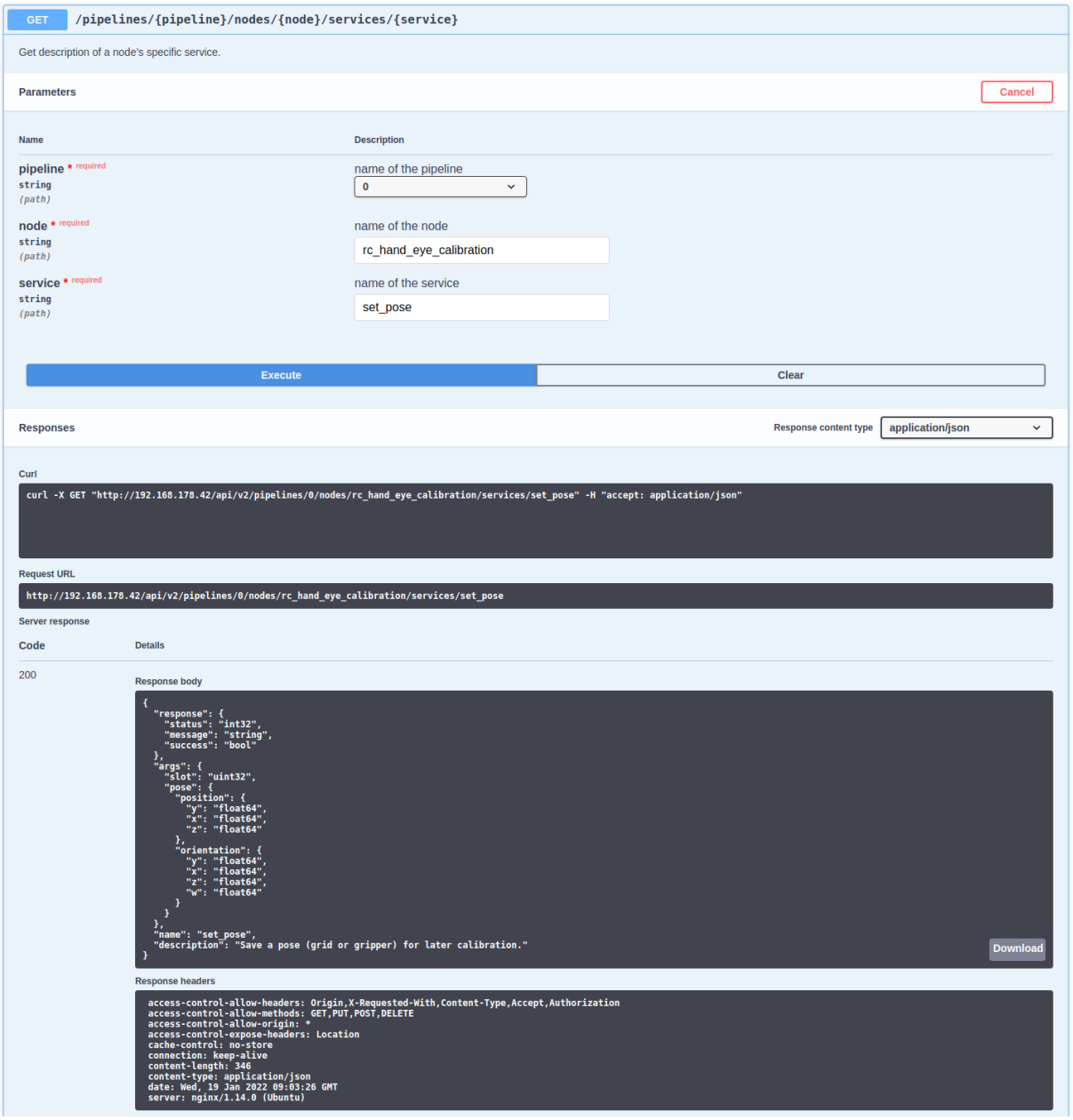


Abb. 6.4: Ergebnis der `GET`-Anfrage zum `set_pose`-Service zeigt die für diesen Service benötigten Argumente

Der Benutzer kann diesen vorformatierten JSON-Text als Muster für die Argumente nutzen, um damit den Service tatsächlich aufzurufen:

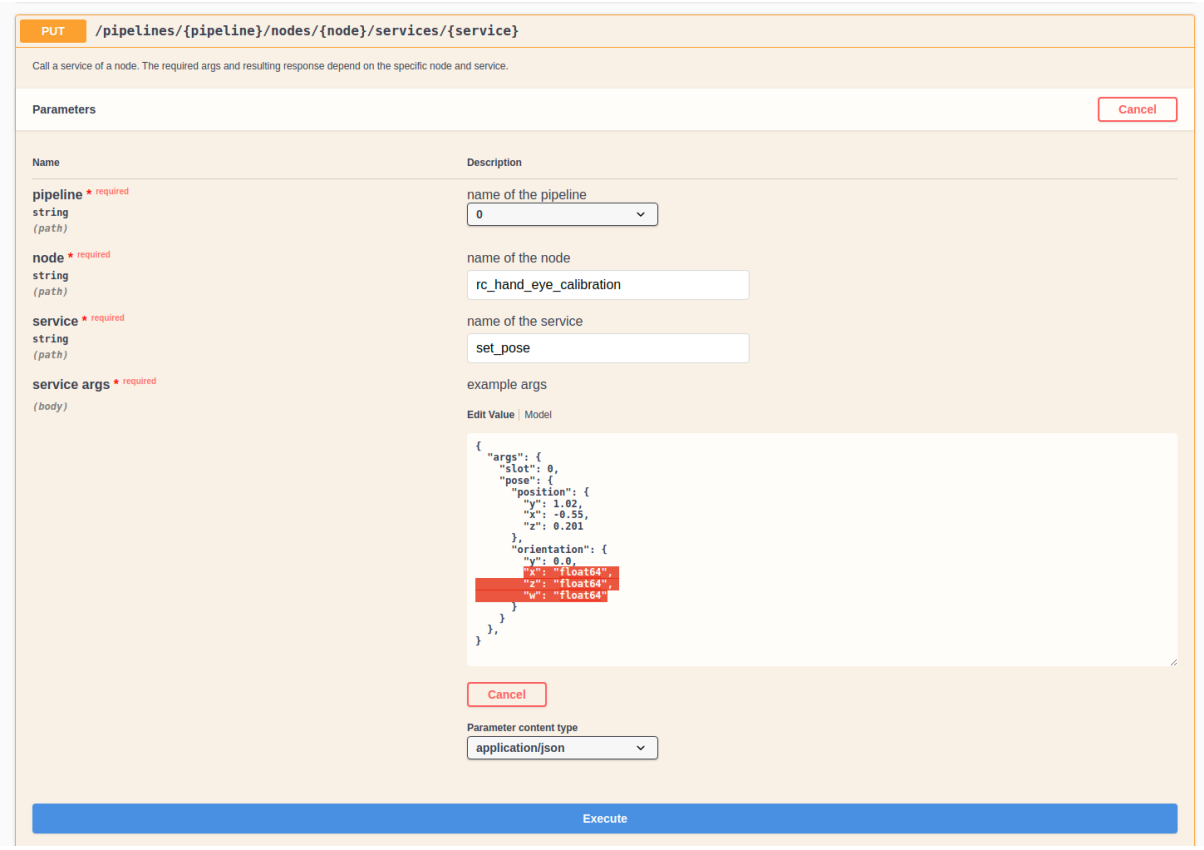


Abb. 6.5: Ausfüllen der Argumente des `set_pose`-Services

6.3 Die rc_dynamics-Schnittstelle

Die rc_dynamics-Schnittstelle bietet über Echtzeit-Datenströme kontinuierlichen Zugang zu verschiedenen *Dynamik-Zustandsschätzungen* (Abschnitt 5.3.1.2). Die Schnittstelle ermöglicht es, Zustandsschätzungen aller Art so zu konfigurieren, dass sie an einen beliebigen Host im Netzwerk gestreamt werden. Das dafür eingesetzte *Datenstromprotokoll* (Abschnitt 6.3.3) unterstützt alle gängigen Betriebssysteme und Programmiersprachen.

6.3.1 Starten/Stoppen der Dynamik-Zustandsschätzungen

Die Dynamik-Zustandsschätzungen des *rc_visard* sind nur verfügbar, wenn die zugehörige Komponente, d.h. das *Dynamik-Modul* (Abschnitt 5.3.1), eingeschaltet ist. Dies lässt sich sowohl über die Web GUI – eine entsprechende Schaltfläche ist auf der Seite *Dynamik* vorgesehen – oder über die REST-API mittels eines Serviceaufrufs vornehmen. Eine Muster-Curl-Anfrage zum Starten der Dynamik-Zustandsschätzung würde wie folgt aussehen:

```
curl -X PUT --header 'Content-Type: application/json' -d '{}' 'http://<host>/api/v1/nodes/rc_
↪dynamics/services/start'
```

Bemerkung: Um Rechenressourcen zu sparen, wird empfohlen, die Dynamik-Zustandsschätzungen zu stoppen, wenn sie nicht länger benötigt werden.

6.3.2 Konfiguration von Datenströmen

Verfügbare Datenströme, d.h. Dynamik-Zustandsschätzungen, lassen sich über die *REST-API* (Abschnitt 6.2.3.2) des *rc_visard* auflisten und konfigurieren. So lässt sich beispielsweise mit dem Befehl `GET /datastreams` eine Liste aller verfügbaren Datenströme abrufen. Für eine detaillierte Beschreibung der im Folgenden benannten Datenströme siehe *Verfügbare Zustandsschätzungen* (Abschnitt 5.3.1.2).

Tab. 6.1: Datenströme, die über die rc_dynamics-Schnittstelle verfügbar sind

Name	Protokoll	ProtoBuf	Beschreibung
`dynamics`	UDP	*Dynamics*	Dynamik des *rc_visard* (Pose, Geschwindigkeit, Beschleunigung), in Echtzeit (IMU-Frequenz) bereitgestellt vom INS- oder SLAM-Modul (Best-Effort-Prinzip)
`dynamics_ins`	UDP	*Dynamics*	Dynamik des *rc_visard* (Pose, Geschwindigkeit, Beschleunigung), in Echtzeit (IMU-Frequenz) bereitgestellt vom Stereo-INS-Modul
`imu`	UDP	*Imu*	Rohdaten der inertialen Messeinheit (IMU), in Echtzeit (IMU-Frequenz) bereitgestellt
`pose`	UDP	*Frame*	Pose der linken Kamera, mit maximaler Kamerafrequenz bereitgestellt vom INS- oder SLAM-Modul (Best-Effort-Prinzip)
`pose_ins`	UDP	*Frame*	Pose der linken Kamera, mit maximaler Kamerafrequenz bereitgestellt vom INS-Modul
`pose_rt`	UDP	*Frame*	Pose der linken Kamera, in Echtzeit (IMU-Frequenz) bereitgestellt vom INS- oder SLAM-Modul (Best-Effort-Prinzip)
`pose_rt_ins`	UDP	*Frame*	Pose der linken Kamera, in Echtzeit (IMU-Frequenz) bereitgestellt vom INS-Modul

Das allgemeine Verfahren für die Arbeit mit der rc_dynamics-Schnittstelle gestaltet sich wie folgt:

1. **Abfrage eines Datenstroms über die REST-API:** Der folgende Beispiel-`curl`-Befehl löst eine `PUT /datastreams/{stream}`-Anfrage aus, mit der die Übertragung eines Datenstroms des Typs `pose_rt` vom *rc_visard* an den Client-Host `10.0.1.14` an Port `30000` ausgelöst werden soll:

   ```
   curl -X PUT --header 'Content-Type: application/x-www-form-urlencoded' --header
   ↪'Accept: application/json' -d 'destination=10.0.1.14:30000' 'http://<host>/api/v1/
   ↪datastreams/pose_rt'
   ```

2. **Empfang und Deserialisierung der Daten:** Wird die Anfrage erfolgreich verarbeitet, wird ein Datenstrom initialisiert und die Daten des angegebenen Datenstrom-Typs werden kontinuierlich an den Client-Host gesandt. Der Client muss die Daten dem *Datenstromprotokoll* (Abschnitt 6.3.3) zufolge empfangen, deserialisieren und verarbeiten.

3. **Stoppen eines Datenstroms über die REST-API:** Der folgende Beispiel-`curl`-Befehl löst eine `DELETE /datastreams/{stream}`-Anfrage aus, mit der die zuvor beantragte Übertragung eines Datenstroms des Typs `pose_rt` mit dem Ziel `10.0.1.14:30000` gelöscht, d.h. gestoppt, wird:

   ```
   curl -X DELETE --header 'Accept: application/json' 'http://<host>/api/v1/datastreams/
   ↪pose_rt?destination=10.0.1.14:30000'
   ```

   Sollen alle Ziele für einen Datenstrom entfernt werden, ist lediglich der Zielparameter wegzulassen.

Warnung: Datenströme können nicht automatisch gelöscht werden. Dies bedeutet, dass der *rc_visard* weiterhin Daten sendet, auch wenn der Client getrennt wird oder die gesandten Daten nicht länger verwendet. Maximal 10 Ziele pro Datenstrom sind erlaubt. Es wird daher dringend empfohlen, Datenströme über die REST-API zu stoppen, wenn sie nicht länger verwendet werden.

6.3.3 Datenstromprotokoll

Sobald ein Datenstrom eingerichtet ist, werden die Daten über das folgende Protokoll kontinuierlich an den angegebenen Client-Host und Port (`destination`) gesandt:

Netzwerkprotokoll: Derzeit wird ausschließlich das Netzwerkprotokoll *UDP* unterstützt, was bedeutet, dass Daten als UDP-Datagramme versandt werden.

Datenserialisierung: Die gesandten Daten werden über Google Protocol Buffers serialisiert. Dabei werden folgende Nachrichtentyp-Definitionen verwendet.

- Die *Kameraposen-Datenströme* und *Echtzeit-Datenströme der Kamerapose* (Abschnitt 5.3.1.2) werden mithilfe des Nachrichtentyps `Frame` serialisiert:

```
message Frame
{
  optional PoseStamped pose  = 1;
  optional string parent     = 2; // Name of the parent frame
  optional string name       = 3; // Name of the frame
  optional string producer   = 4; // Name of the producer of this data
}
```

  Das Feld `producer` kann die Werte `ins`, `slam`, `rt_ins` und `rt_slam` annehmen. Diese geben an, ob die Daten von SLAM oder Stereo-INS berechnet wurden und ob es Echtzeit Daten (rt) sind oder nicht.

- Der *Echtzeit-Dynamik-Datenstrom* (Abschnitt 5.3.1.2) wird mithilfe des Nachrichtentyps `Dynamics` serialisiert:

```
message Dynamics
{
  optional Time timestamp                    = 1; // Time when the data was␣
↪captured
  optional Pose pose                         = 2;
  optional string pose_frame                 = 3; // Name of the frame that␣
↪the pose is given in
  optional Vector3d linear_velocity          = 4; // Linear velocity in m/s
  optional string linear_velocity_frame      = 5; // Name of the frame that␣
↪the linear_velocity is given in
  optional Vector3d angular_velocity         = 6; // Angular velocity in rad/s
  optional string angular_velocity_frame     = 7; // Name of the frame that␣
↪the angular_velocity is given in
  optional Vector3d linear_acceleration      = 8; // Gravity compensated␣
↪linear acceleration in m/s^2
  optional string linear_acceleration_frame  = 9; // Name of the frame that␣
↪the acceleration is given in
  repeated double covariance                 = 10 [packed=true]; // Row-major␣
↪representation of the 15x15 covariance matrix
  optional Frame cam2imu_transform           = 11; // pose of the left camera␣
↪wrt. the IMU frame
  optional bool possible_jump                = 12; // True if there possibly␣
↪was a jump in the pose estimation
  optional string producer                   = 13; // Name of the producer of␣
↪this data
}
```

Das Feld `producer` kann die Werte `rt_ins` und `rt_slam` annehmen. Diese geben an, ob die Daten von SLAM oder Stereo-INS berechnet wurden.

- Der *IMU-Datenstrom* (Abschnitt 5.3.1.2) wird mithilfe des Nachrichtentyps `Imu` serialisiert:

```
message Imu
{
  optional Time timestamp                    = 1; // Time when the data was␣
↪captured
  optional Vector3d linear_acceleration      = 2; // Linear acceleration in m/
↪s^2 measured by the IMU
  optional Vector3d angular_velocity         = 3; // Angular velocity in rad/
↪s measured by the IMU
}
```

- Die enthaltenen Nachrichtentypen `PoseStamped`, `Pose`, `Time`, `Quaternion` und `Vector3D` werden wie folgt definiert:

```
message PoseStamped
{
  optional Time timestamp  = 1; // Time when the data was captured
  optional Pose pose       = 2;
}
```

```
message Pose
{
  optional Vector3d position      = 1; // Position in meters
  optional Quaternion orientation = 2; // Orientation as unit quaternion
  repeated double covariance      = 3 [packed=true]; // Row-major␣
↪representation of the 6x6 covariance matrix (x, y, z, rotation about X axis,␣
↪rotation about Y axis, rotation about Z axis)
}
```

```
message Time
{
  /// \brief Seconds
  optional int64 sec = 1;

  /// \brief Nanoseconds
  optional int32 nsec = 2;
}
```

```
message Quaternion
{
  optional double x = 2;
  optional double y = 3;
  optional double z = 4;
  optional double w = 5;
}
```

```
message Vector3d
{
  optional double x = 1;
  optional double y = 2;
  optional double z = 3;
}
```

6.3.4 Die rc_dynamics-Programmierschnittstelle

Das Open-Source-`rc_dynamics_api`-Paket bietet einen einfachen, benutzerfreundlichen C++-Wrapper, mit dem rc_dynamics-Datenströme angefragt und geparst werden können. Siehe http://www.roboception.com/download.

6.4 Generic Robot Interface

Das Generic Robot Interface (GRI) ist ein Integrationslayer, der die *REST-API v2* (Abschnitt 6.2) überbrückt und eine standardisierte Kommunikationsschnittstelle zu den Softwaremodulen über eine einfache TCP-Socket-Kommunikation auf Port 7100 bietet. Es kann über eine separate *Lizenz*(Abschnitt 7.4) aktiviert werden.

6.4.1 Überblick

Das GRI wird in zwei Phasen verwendet. In der Konfigurationsphase werden *Jobs* erstellt und unter eindeutigen numerischen IDs gespeichert. Ein Job enthält alles, was für eine bestimmte Operation benötigt wird: das aufzurufende Modul und den aufzurufenden Service, die zu verwendenden Argumente und die an den Roboter zurückzugebenden Daten. Jobs und optional *Hand-Auge-Kalibrierungskonfigurationen* werden einmalig vorbereitet und über die Web GUI oder die REST-API definiert (siehe *Job Arten*).

Zur Laufzeit triggert der Roboter diese Jobs über ihre ID mittels eines TCP-Sockets auf Port 7100 und tauscht dabei Binärnachrichten fester Größe aus (siehe *Spezifikation des Binären GRI Protokolls*). Das GRI übernimmt intern die REST-API Kommunikation und gibt die ausgewählten Posenergebnisse in einem roboterspezifisch wählbaren Format zurück. Ein typischer Job-Trigger läuft wie folgt ab:

1. Der Roboter sendet eine Trigger-Aktion (*Action*) (`TRIGGER_JOB_SYNC` oder `TRIGGER_JOB_ASYNC`, siehe *Aktionen*) zusammen mit der Job-ID und übergibt optional dynamische Werte in den `data`-Feldern der Anfrage.
2. Das GRI sucht den Job und übersetzt ihn in die entsprechenden REST-API Aufrufe an das ausgewählte Modul.
3. Die Ergebnisse des Moduls, etwa Greifposen, werden vom GRI zwischengespeichert, und die erste Pose wird im gewählten *Posenformat* an den Roboter zurückgegeben.
4. Der Roboter ruft alle weiteren Ergebnisse einzeln mit `GET_NEXT_POSE` und `GET_RELATED_POSE` ab.

Eine Job-Definition dient somit als wiederverwendbare Vorlage, die der Roboter zur Laufzeit über ihre ID triggert. Der folgende Abschnitt beschreibt, wie Jobs definiert werden; das Binärprotokoll und die Details der Roboterintegration werden weiter unten beschrieben.

6.4.2 Job Definition

Jobs sind vorkonfigurierte Aufgaben, die von der Roboteranwendung getriggert werden können. Jeder Job hat eine eindeutige ID und enthält alle notwendigen Informationen für eine bestimmte Operation, z.B. das Berechnen von Greifpunkten für das Bin Picking oder das Ändern von Laufzeitparametern eines Moduls. Einmal konfiguriert, kann der Roboter diese Jobs mit einfachen Socket-Befehlen ausführen und gegebenenfalls die zurückgegebenen Posen empfangen.

Die Jobs können über die Web GUI oder über die REST-API definiert werden (siehe *Job und HEC_config API*).

In der Web GUI wird ein Pipeline-Service-Job mit *Job erstellen* im Bereich *Generic Robot Interface* des *Ausprobieren*-Abschnitts des jeweiligen Moduls angelegt, wobei die dort aktuell konfigurierten Einstellungen verwendet werden. Mit *Job laden* wird ein gespeicherter Job zur Bearbeitung wieder in den *Ausprobieren*-Abschnitt geladen. Beim Speichern des Jobs können seine zusätzlichen Rückgabewerte, dynamischen Argumente und Zuordnungstabellen (siehe *Rückgabe von Posen und Zusatzdaten*)

mit *Zusatzdaten konfigurieren* definiert und bearbeitet werden. Parameter-Jobs können auf der Seite *Generic Robot Interface* der Web GUI erstellt und bearbeitet werden.

6.4.2.1 Job Arten

Das Generic Robot Interface unterstützt drei Arten von Jobs, die über das Feld `job_type` ausgewählt werden:

- `CALL_PIPELINE_SERVICE` – ruft einen Service auf einer bestimmten Kamera-Pipeline auf (z.B. um Objekte zu erkennen oder Greifpunkte zu berechnen) und gibt Posendaten an den Roboter zurück.
- `CALL_GLOBAL_SERVICE` – ruft einen Service auf, der nicht an eine Pipeline gebunden ist (z.B. Datenbank-Services für Regions of Interest, Load Carrier oder Greifer). Gibt nur dann eine Pose zurück, wenn dies explizit konfiguriert ist.
- `SET_PIPELINE_PARAMETERS` – setzt Laufzeitparameter in einer Pipeline-Node. Gibt keine Pose zurück.

Die Felder `selected_return`, `pose_path`, `additional_return` und `dynamic_args` steuern, *welche* Pose ein Job zurückgibt und wie der Roboter den Aufruf beeinflussen kann. Sie werden ausführlich in *Rückgabe von Posen und Zusatzdaten* beschrieben und an vollständigen Jobs in *Ausführliche Beispiele* veranschaulicht.

Pipeline-Service-Job (`CALL_PIPELINE_SERVICE`)

Dieser Job ruft einen Service auf einer bestimmten Kamera-Pipeline auf, um beispielsweise Objekte zu erkennen oder Greifpunkte zu berechnen, und gibt Posendaten an den Roboter zurück (z.B. Greifposen).

Feld	Angabe	Beschreibung
`job_type`	erforderlich	die Art des Jobs `CALL_PIPELINE_SERVICE`
`name`	erforderlich	beschreibender Name zur Unterscheidung von Jobs
`pipeline`	erforderlich	die für den Job zu verwendende Kamera-Pipeline (z.B. `"0"`)
`node`	erforderlich	der REST-API Name der Pipeline-Node (z.B. `rc_itempick`)
`service`	erforderlich	der REST-API Name des aufzurufenden Services (z.B. `compute_grasps`)
`args`	erforderlich	die REST-API JSON-Argumente, die dem Service übergeben werden
`selected_return`	erforderlich	das REST-API Antwortfeld, das die zurückzugebenden Posen enthält (z.B. `grasps` oder `matches`); siehe *Auswahl der zurückgegebenen Pose: selected_return und pose_path*
`pose_path`	optional	das Feld innerhalb des `selected_return` Objekts, das die Pose enthält (Standardwert `pose`); siehe *Auswahl der zurückgegebenen Pose: selected_return und pose_path*
`additional_return`	optional	zusätzliche Antwortfelder, die in `data_4` bis `data_10` zurückgegeben werden; siehe *additional_return (Modul zu Roboter)*
`dynamic_args`	optional	füllt `args` aus den Anfragefeldern `data_1` bis `data_4` des Roboters; siehe *dynamic_args (Roboter zu Modul)*

Eine minimale Definition eines Pipeline-Service-Jobs ist:

```
{
   "args": {
      "pose_frame": "external",
      "suction_surface_length": 0.02,
      "suction_surface_width": 0.02
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  },
  "job_type": "CALL_PIPELINE_SERVICE",
  "name": "Compute Grasps",
  "node": "rc_itempick",
  "pipeline": "0",
  "selected_return": "grasps",
  "service": "compute_grasps"
}
```

Bei Pipeline-Service-Jobs fügt das GRI die mit der Trigger-Anfrage gesendete Roboterpose als `robot_pose` (Position in Metern, Orientierung als Quaternion) zu den Service-Argumenten hinzu. Eine in `args` angegebene `robot_pose` wird überschrieben. Globale Service-Jobs erhalten die Roboterpose nicht.

Die verfügbaren Werte für `selected_return` hängen von der gewählten Node (Modul) ab und können z.B. `grasps` oder `matches` sein. Details zu `node`, `service`, `args` und `selected_return` sind in den Servicedefinitionen des entsprechenden Moduls beschrieben.

Globaler Service-Job (`CALL_GLOBAL_SERVICE`)

Dieser Job ruft einen Service auf, der nicht an eine bestimmte Pipeline gebunden ist, z.B. Datenbank-Services zum Festlegen oder Abrufen von Regions of Interest, Load Carriern oder Greifern. Standardmäßig geben globale Service-Jobs keine Pose zurück; sie geben nur dann eine Pose zurück, wenn `selected_return` (und typischerweise `pose_path`) konfiguriert ist.

Feld	Angabe	Beschreibung
`job_type`	erforderlich	die Art des Jobs `CALL_GLOBAL_SERVICE`
`name`	erforderlich	beschreibender Name zur Unterscheidung von Jobs
`node`	erforderlich	der REST-API Name der globalen Node (z.B. `rc_load_carrier_db`)
`service`	erforderlich	der REST-API Name des aufzurufenden Services
`args`	erforderlich	die REST-API JSON-Argumente, die dem Service übergeben werden
`selected_return`	optional	das REST-API Antwortfeld, das die zurückzugebende(n) Pose(n) enthält (z.B. zum Abrufen von Greifer- oder TCP-Definitionen); ohne dieses Feld gibt der Job keine Pose zurück; siehe *Auswahl der zurückgegebenen Pose: selected_return und pose_path*
`pose_path`	optional	das Feld innerhalb des `selected_return` Objekts, das die Pose enthält (Standardwert `pose`); siehe *Auswahl der zurückgegebenen Pose: selected_return und pose_path*
`additional_return`	optional	zusätzliche Antwortfelder, die in `data_4` bis `data_10` zurückgegeben werden; siehe *additional_return (Modul zu Roboter)*
`dynamic_args`	optional	füllt `args` aus den Anfragefeldern `data_1` bis `data_4` des Roboters; siehe *dynamic_args (Roboter zu Modul)*

Das Feld `pipeline` darf für globale Service-Jobs nicht gesetzt werden.

Eine minimale Definition eines globalen Service-Jobs ist:

```
{
  "args": {
    "region_of_interest_2d": {
      "id": "2d_roi",
      "width": 526,
      "height": 501,
      "offset_x": 558,
      "offset_y": 307
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      }
   },
   "job_type": "CALL_GLOBAL_SERVICE",
   "name": "Set 2D ROI",
   "node": "rc_roi_db",
   "service": "set_region_of_interest_2d"
}
```

Details zu `node`, `service` und `args` sind in den Servicedefinitionen des entsprechenden Moduls beschrieben.

Parameter-Job (`SET_PIPELINE_PARAMETERS`)

Dieser Job setzt Laufzeitparameter in einer Pipeline-Node, z.B. zum Anpassen von Kamera- oder Detektionsmoduleinstellungen. Parameter-Jobs geben keine Pose zurück.

Feld	Angabe	Beschreibung
`job_type`	erforderlich	die Art des Jobs `SET_PIPELINE_PARAMETERS`
`name`	erforderlich	beschreibender Name zur Unterscheidung von Jobs
`pipeline`	erforderlich	die für den Job zu verwendende Kamera-Pipeline (z.B. `"0"`)
`node`	erforderlich	der REST-API Name der Pipeline-Node (z.B. `rc_stereomatching`)
`parameters`	erforderlich	die zu setzenden Laufzeitparameter als Key-Value-Paare

Eine Beispieldefinition für einen Parameter-Job ist:

```
{
   "job_type": "SET_PIPELINE_PARAMETERS",
   "name": "Set Stereo Parameters",
   "node": "rc_stereomatching",
   "parameters": {
      "maxdepth": 2,
      "quality": "High"
   },
   "pipeline": "0"
}
```

Wenn ein Parameter-Job ausgelöst wird, ist `data_1` der Antwort `0`, wenn alle Parameter gesetzt wurden, und andernfalls `-1`; im letzteren Fall ist `error_code` gleich `API_RESPONSE_ERROR`. Anders als bei den anderen Job-Typen ist `data_1` hier nicht der `return_code` einer Node. `GET_JOB_STATUS` meldet `data_1` für diese Jobs als `0`.

Details zu `node` und `parameters` sind in den Laufzeitparameterdefinitionen des entsprechenden Moduls beschrieben.

6.4.2.2 Primäre und zugehörige Objekte

Detektionsergebnisse bestehen oft aus zwei miteinander verknüpften Arten von Objekten. Die *primären Objekte* (*primary*) sind die Objekte, die durch `selected_return` des Jobs benannt werden, z.B. `grasps`. Die *zugehörigen Objekte* (*related*) sind die Objekte, denen ein primäres Objekt entspricht, z.B. die `items` oder `matches`, für die ein Greifpunkt berechnet wurde.

Der Roboter iteriert mit `GET_NEXT_POSE` einzeln über die primären Objekte. Nach jedem primären Objekt kann er das zugehörige Objekt mit `GET_RELATED_POSE` abfragen (siehe *Aktionen*). Die Verknüpfung zwischen beiden wird durch die REST-API Antwort selbst hergestellt: Zum Beispiel enthält jeder `grasp` eine `item_uuid`, die mit der `uuid` genau eines `item` übereinstimmt.

Ein primäres Objekt *grasp* hat daher genau ein zugehöriges Objekt, ein *item* oder ein *match*. Umgekehrt kann ein primäres Objekt *match* mehrere zugehörige *grasps* haben. Da die Zuordnung pro primärem Objekt aufgelöst wird, geben zwei Greifpunkte, die zum selben Item gehören, bei `GET_RELATED_POSE` jeweils dasselbe Item zurück; das GRI filtert diese wiederholten zugehörigen Objekte nicht heraus.

6.4.2.3 Rückgabe von Posen und Zusatzdaten

Jedes Ergebnis, das ein Job liefert, enthält eine Pose in den Posenfeldern der Antwort (`pos_x` bis `rot_4`). Darüber hinaus lässt Protokoll V1 mehrere Anfrage- und Antwortfelder ungenutzt, und die Job-Konfiguration verwendet diese, um zusätzliche Informationen zwischen dem Roboter und der REST-API auszutauschen:

- `selected_return` und `pose_path` bestimmen, *welche* Pose zurückgegeben wird.
- `additional_return` gibt zusätzliche Werte aus der REST-API Antwort in den Antwortfeldern `data_4` bis `data_10` zurück (Modul zu Roboter, siehe *Antwort-Body Definition*).
- `dynamic_args` fügt vom Roboter bereitgestellte Werte aus den Anfragefeldern `data_1` bis `data_4` in die `args` des REST-API Aufrufs ein (Roboter zu Modul, siehe *Anfrage-Body Definition*).
- `maps` übersetzt zwischen den Ganzzahlen, die der Roboter austauscht, und den Zeichenkettenwerten, die die REST-API verwendet.

`additional_return`, `dynamic_args` und `maps` können beim Speichern eines Jobs auch in der Web GUI mit *Zusatzdaten konfigurieren* definiert und bearbeitet werden.

Alle diese Optionen lokalisieren Werte innerhalb der REST-API Antwort oder Anfrage über einen `json_path`, der zuerst erläutert wird.

Werte mit `json_path` lokalisieren

Ein `json_path` adressiert einen einzelnen Wert innerhalb der REST-API Antwort (oder, bei `dynamic_args`, innerhalb der Argumente der Anfrage). Es gibt zwei Formen:

- **Relativ zum Objekt** (kein führender Punkt, z.B. `quality`): wird *innerhalb* des primären oder zugehörigen Objekts aufgelöst, das gerade von `GET_NEXT_POSE` / `GET_RELATED_POSE` zurückgegeben wird.
- **Relativ zu Root** (führender Punkt, z.B. `.timestamp.sec`): wird von der Root (Wurzel) der gesamten REST-API Antwort aus aufgelöst, was für globale Metadaten oder für eine übergeordnete Eigenschaft nützlich ist.

Um ein bestimmtes Element einer Liste zu erreichen, wird die Klammernotation mit einem nullbasierten Index verwendet, z.B. `item_models[0].type`.

Die folgende kommentierte Antwort zeigt, wie sich die einzelnen Formen auflösen, unter der Annahme, dass der Job `grasps` als primäre Objekte und `items` als zugehörige Objekte zurückgibt:

```
{
   "timestamp": {
      "sec": 1719900000
   },
   "grasps": [
      {
         "quality": 0.85,
         "item_uuid": "abc-123"
      }
   ],
   "items": [
      {
         "uuid": "abc-123",
         "bounding_box": { "x": 0.15 }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      }
   ]
}
```

- `quality` (relativ zum Objekt, Geltungsbereich `primary`) wird zu `0.85` aufgelöst.
- `bounding_box.x` (relativ zum Objekt, Geltungsbereich `related`) wird zu `0.15` aufgelöst.
- `.timestamp.sec` (relativ zu Root) wird zu `1719900000` aufgelöst.

Bemerkung: Für den Zugriff auf ein Listenelement ist immer die Klammernotation (`item_models[0]`) zu verwenden. Bei `item_models.0` sucht das GRI stattdessen nach einem Dictionary-Schlüssel, der wörtlich `"0"` heißt, anstatt nach dem ersten Listenelement.

Da `GET_NEXT_POSE` und `GET_RELATED_POSE` jeweils immer nur ein Objekt zurückgeben, wird der `json_path` relativ zum Objekt auch nur für ein *einzelnes* Objekt gesetzt: Es muss `quality` verwendet werden, nicht `grasps[0].quality`.

Auswahl der zurückgegebenen Pose: `selected_return` und `pose_path`

`selected_return` benennt das Antwortfeld, das die primären Objekte enthält, über die iteriert werden soll, z.B. `grasps`, `matches`, `items` oder `load_carriers`. Es kann mittels Klammernotation auch in eine tiefer in der Antwort verschachtelte Liste indizieren, um beispielsweise über die TCPs eines bestimmten Greifers zu iterieren:

```
{
   "selected_return": "grippers[0].tcps",
   "pose_path": "pose_flange"
}
```

`pose_path` teilt dem GRI mit, wo *innerhalb* jedes ausgewählten `selected_return` Objekts die an den Roboter zurückzugebende Pose (ein Objekt mit `position` und `orientation`) zu finden ist. Es folgt denselben Pfadregeln wie `json_path` und hat den Standardwert `pose`, weshalb es normalerweise weggelassen wird: Alle integrierten Pipeline-Rückgabetypen (`grasps`, `matches`, `items`, `load_carriers`) stellen ihre Pose in einem Feld namens `pose` bereit.

`pose_path` ist explizit zu setzen, wenn das ausgewählte Objekt kein `pose`-Feld hat oder wenn es mehr als eine mögliche Pose bereitstellt und eine bestimmte ausgewählt werden muss. Dies ist im obigen Greifer-Beispiel der Fall, wo jeder TCP separate Felder `pose_flange` und `pose_parent` anstelle eines einzelnen `pose` hat. Derselbe Wert wird sowohl auf das primäre als auch auf das zugehörige Objekt angewendet, sodass er nur dann vom Standardwert abweichen muss, wenn das betreffende Feld nicht `pose` heißt. Wenn der aufgelöste Wert fehlt oder kein Objekt mit sowohl `position` als auch `orientation` ist, wird der Job als fehlkonfiguriert abgelehnt (`MISCONFIGURED_JOB`).

`additional_return` (Modul zu Roboter)

`additional_return` bildet zusätzliche Werte aus der REST-API Antwort auf die Antwortfelder `data_4` bis `data_10` ab. Seine Zuordnungen sind nach dem Objekt gruppiert, auf das sie angewendet werden:

- `primary`: wird auf das von `GET_NEXT_POSE` zurückgegebene primäre Objekt angewendet (die durch `selected_return` benannte Objektart, z.B. `grasps`).
- `related`: wird auf das entsprechende zugehörige Objekt angewendet, das von `GET_RELATED_POSE` zurückgegeben wird (siehe *Primäre und zugehörige Objekte*).

Jede Zuordnung ist ein Objekt mit den folgenden Feldern:

- `target_field`: das Antwortfeld, in das geschrieben wird (`data_4` bis `data_10`)

- `json_path`: wo der Wert in der REST-API Antwort gelesen wird (siehe *Werte mit json_path lokalisieren*)
- `type`: wie der Wert in das `int32` Antwortfeld konvertiert wird:
  - `int`: wird als Ganzzahl geschrieben (Standard, wenn `type` nicht angegeben ist); ein Wert mit Nachkommastellen wird mit `TYPE_CONVERSION_ERROR` abgelehnt
  - `float`: wird mit `1,000,000` multipliziert (analog zur Posenskalierung in *Posenformate*)
  - `bool`: `1` oder `0`
  - `map`: wird über eine `maps` Nachschlagetabelle in eine Ganzzahl übersetzt (siehe *Zuordnungstabellen (maps)*)

Wenn ein `json_path` zu keinem Wert aufgelöst wird, wird dieses Feld einfach übersprungen (Warnung `MISSING_ADDITIONAL_DATA`, siehe *Fehlercodes und Bedeutung*). Eine fehlgeschlagene `map` Suche oder Typkonvertierung bricht dagegen die gesamte Antwort mit einem Fehler ab (`UNMAPPED_VALUE_ERROR` / `TYPE_CONVERSION_ERROR`).

Wenn ein Job kein `selected_return` hat, gibt es keine primären Objekte, über die iteriert werden kann. `primary` Zuordnungen werden dann direkt auf die `TRIGGER_JOB_SYNC` Antwort angewendet, und ihre `json_path` Werte werden ausgehend von der Wurzel der REST-API Antwort aufgelöst. Dadurch können auch globale Service-Jobs, die keine Pose zurückgeben, Werte zurückliefern.

`dynamic_args` (Roboter zu Modul)

`dynamic_args` bildet Werte aus der Roboteranfrage auf die `args` des REST-API Serviceaufrufs ab, sodass der Roboter Detektionsargumente zum Zeitpunkt des Triggerns anpassen kann, ohne den Job neu zu konfigurieren. Es gilt für `CALL_PIPELINE_SERVICE` und `CALL_GLOBAL_SERVICE` Jobs; `SET_PIPELINE_PARAMETERS` Jobs verwenden es nicht. Es ist eine Liste von Einträgen mit jeweils:

- `source_field`: das Anfragefeld, aus dem gelesen wird (`data_1` bis `data_4`, siehe *Anfrage-Body Definition*)
- `json_path`: der Ziel-Schlüsselpfad innerhalb der `args` der REST-API (dieselben Pfadregeln wie *Werte mit json_path lokalisieren*)
- `type`: wie der Wert vor dem Einfügen konvertiert wird:
  - `int`: wird unverändert eingefügt (Standardwert, wenn `type` weggelassen wird)
  - `float`: wird durch `1,000,000` geteilt
  - `bool`: wird als `1` / `0` interpretiert
  - `map`: die vom Roboter gesendete Ganzzahl wird in einer `maps` Tabelle nachgeschlagen, um den entsprechenden Wert einzufügen

Die Werte werden beim Triggern des Jobs zusätzlich zu den statischen `args` angewendet, sodass ein `json_path`, der auf ein vorhandenes Argument zeigt, dieses überschreibt. Eine fehlgeschlagene `map` Suche oder Typkonvertierung wird mit einem Fehler abgelehnt (`UNMAPPED_VALUE_ERROR` / `TYPE_CONVERSION_ERROR`, siehe *Fehlercodes und Bedeutung*).

Zuordnungstabellen (`maps`)

Manche Argumente und Rückgabewerte sind Zeichenketten (etwa IDs) statt Zahlen. `maps` übersetzt zwischen den Ganzzahlen, die der Roboter austauscht, und diesen Zeichenkettenwerten auf dem GRI-Server. Dieselbe Tabelle wird in beiden Richtungen verwendet – `dynamic_args` schlägt *Roboter-ID zu Wert* nach, `additional_return` schlägt *Wert zu Roboter-ID* nach – daher muss jede Tabelle eine 1:1-Zuordnung (bijektiv) sein. Ein `dynamic_args` oder `additional_return` Eintrag vom `type` `map` wählt die zu verwendende Tabelle über ein `map` Feld aus:

```
{
   "dynamic_args": [
      { "source_field": "data_2", "json_path": "template_id", "type": "map", "map": "models" }
   ],
   "maps": {
      "models": {
         "1": "part_A_large",
         "2": "part_B_small"
      }
   }
}
```

Wenn der Roboter `data_2 = 2` sendet, wird die Template-ID `"part_B_small"` in den REST-API Aufruf eingefügt.

6.4.2.4 Ausführliche Beispiele

Die folgenden vollständigen Job-Definitionen kombinieren die oben genannten Optionen. Für jede wird zunächst die Konfiguration gezeigt, gefolgt von dem, was der Roboter zur Laufzeit sendet oder empfängt.

Beispiel 1: Pipeline-Service-Job mit zusätzlicher Rückgabe

Dieser ItemPick-Job gibt Greifposen zurück und, neben jeder Pose, zusätzliche Informationen über den Greifpunkt (Geltungsbereich `primary`) und über das Item, für das der Greifpunkt berechnet wurde (Geltungsbereich `related`).

```
{
   "name": "ItemPickAIJob",
   "job_type": "CALL_PIPELINE_SERVICE",
   "pipeline": "0",
   "node": "rc_itempick",
   "service": "compute_grasps",
   "selected_return": "grasps",
   "args": {
      "collision_detection": {
         "gripper_id": "MultiSuctionGripper",
         "pre_grasp_offset": { "x": 0, "y": 0, "z": -0.1 }
      },
      "item_models": [ { "type": "BAG" } ],
      "pose_frame": "external",
      "region_of_interest_id": "ROI",
      "suction_surface_length": 0.05,
      "suction_surface_width": 0.05
   },
   "additional_return": {
      "primary": [
         { "json_path": "quality", "target_field": "data_4", "type": "float" },
         { "json_path": "tcp_id",  "target_field": "data_5", "type": "map", "map": "gripper_tcps"␣
↪}
      ],
      "related": [
         { "json_path": "bounding_box.x", "target_field": "data_6", "type": "float" },
         { "json_path": "bounding_box.y", "target_field": "data_7", "type": "float" },
         { "json_path": "bounding_box.z", "target_field": "data_8", "type": "float" }
      ]
   },
   "maps": {
      "gripper_tcps": { "1": "TCP1", "2": "TCP2", "3": "TCP3" }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
   }
}
```

- `selected_return` ist `grasps`, sodass jeder `GET_NEXT_POSE` eine Greifpose zurückgibt.
- `additional_return.primary` liest zwei Werte aus diesem Greifpunkt: seine `quality` in `data_4` (als Float) und seine `tcp_id` in `data_5` (über die Tabelle `gripper_tcps` in eine Ganzzahl übersetzt).
- `additional_return.related` liest die Bounding Box des zugehörigen Items in `data_6` bis `data_8`, verfügbar nach `GET_RELATED_POSE`.

Zur Laufzeit ergibt sich für einen Greifpunkt mit `quality = 0.85` und `tcp_id = "TCP2"` an einem Item mit der Bounding Box `(0.15, 0.10, 0.05)` m:

- `GET_NEXT_POSE` gibt die Greifpose zurück, `data_4 = 850000` (`0.85 x 1,000,000`) und `data_5 = 2` (der in `gripper_tcps` auf `"TCP2"` abgebildete Schlüssel).
- `GET_RELATED_POSE` gibt die Pose des Items zurück, `data_6 = 150000`, `data_7 = 100000` und `data_8 = 50000` (jede Bounding-Box-Dimension x `1,000,000`).

Beispiel 2: Globaler Service-Job, der eine Pose zurückgibt

Globale Service-Jobs geben standardmäßig keine Pose zurück. Dieser Job setzt `selected_return` und `pose_path` so, dass er die für einen Greifer in der Greifer-Datenbank gespeicherten TCP-Posen zurückgibt.

```
{
   "name": "GetMultiSuctionGripperTCPs",
   "job_type": "CALL_GLOBAL_SERVICE",
   "node": "rc_gripper_db",
   "service": "get_grippers",
   "selected_return": "grippers[0].tcps",
   "pose_path": "pose_flange",
   "args": {
      "gripper_ids": [ "MultiSuctionGripper" ]
   },
   "additional_return": {
      "primary": [
         { "json_path": "id", "target_field": "data_4", "type": "map", "map": "tcp_names" },
         { "json_path": ".grippers[0].flange_radius", "target_field": "data_5", "type": "float" }
      ]
   },
   "maps": {
      "tcp_names": { "1": "TCP1", "2": "TCP2", "3": "TCP3" }
   }
}
```

- `selected_return` verwendet die Indexadressierung `grippers[0].tcps`, um über die TCPs des ersten zurückgegebenen Greifers als primäre Objekte zu iterieren.
- `pose_path` wird auf `pose_flange` gesetzt, weil ein TCP kein einfaches `pose`-Feld hat, sondern separate Posen `pose_flange` und `pose_parent`; zurückgegeben wird `pose_flange`.
- `additional_return.primary` kombiniert beide Formen: die `id` (die ID des aktuellen TCP, über `tcp_names` in eine Ganzzahl abgebildet), die relativ zum Objekt gewählt ist, und den `.grippers[0].flange_radius` (eine Eigenschaft des übergeordneten Greifers, für jeden TCP gleich), der relativ zu Root (Wurzel) adressiert ist.

Zur Laufzeit gibt `GET_NEXT_POSE` für einen TCP mit `id = "TCP1"` an einem Greifer mit `flange_radius = 0.04` m die `pose_flange` des TCP zurück, `data_4 = 1` (`"TCP1"` über `tcp_names`) und `data_5 = 40000` (`0.04 x 1,000,000`).

Beispiel 3: Pipeline-Service-Job mit dynamischen Argumenten

Dieser BoxPick-Job lässt den Roboter die zu erkennenden Rechteckabmessungen zum Zeitpunkt des Triggerns wählen, sodass derselbe Job unterschiedliche Boxgrößen greifen kann, ohne neu konfiguriert zu werden. Die vier Abmessungen des `RECTANGLE` Item-Modells werden aus den Anfragefeldern `data_1` bis `data_4` gefüllt.

```
{
   "name": "BoxPickDynamicRectangle",
   "job_type": "CALL_PIPELINE_SERVICE",
   "pipeline": "0",
   "node": "rc_boxpick",
   "service": "compute_grasps",
   "selected_return": "grasps",
   "args": {
      "pose_frame": "external",
      "item_models": [
         {
            "type": "RECTANGLE",
            "rectangle": {
               "min_dimensions": { "x": 0.05, "y": 0.05 },
               "max_dimensions": { "x": 0.10, "y": 0.10 }
            }
         }
      ],
      "suction_surface_length": 0.05,
      "suction_surface_width": 0.05
   },
   "dynamic_args": [
      { "source_field": "data_1", "json_path": "item_models[0].rectangle.min_dimensions.x", "type
↪": "float" },
      { "source_field": "data_2", "json_path": "item_models[0].rectangle.min_dimensions.y", "type
↪": "float" },
      { "source_field": "data_3", "json_path": "item_models[0].rectangle.max_dimensions.x", "type
↪": "float" },
      { "source_field": "data_4", "json_path": "item_models[0].rectangle.max_dimensions.y", "type
↪": "float" }
   ],
   "additional_return": {
      "primary": [
         { "json_path": "quality", "target_field": "data_4", "type": "float" }
      ]
   }
}
```

- `dynamic_args` verwendet dieselbe `json_path` Grammatik wie `additional_return` und indiziert hier in `item_models[0]`, um die Rechteckabmessungen innerhalb von `args` zu erreichen.
- Die in `args` angegebenen Abmessungen sind Platzhalter: Die aus der Anfrage eingefügten Werte (`dynamic_args`) überschreiben sie, wenn der Job getriggert wird.
- Die Anfragefelder `data_1` bis `data_4` (gefüllt durch `dynamic_args`) und die Antwortfelder `data_4` bis `data_10` (gefüllt durch `additional_return`) gehören zu getrennten Nachrichten, sodass die doppelte Verwendung von `data_4` sowohl für die gesendete minimale x-Abmessung als auch für die zurückgegebene `quality` des Greifpunkts kein Konflikt ist.

Wenn der Roboter den Job zur Laufzeit mit `data_1 = 50000`, `data_2 = 50000`, `data_3 = 100000` und `data_4 = 100000` triggert, teilt das GRI jeden Wert durch `1,000,000` und ruft `rc_boxpick/compute_grasps` mit `min_dimensions (0.05, 0.05)` m und `max_dimensions (0.10, 0.10)` m auf. Jeder folgende `GET_NEXT_POSE` gibt dann eine Greifpose zurück, wobei `data_4` auf `quality x 1,000,000` dieses Greifpunkts gesetzt ist.

6.4.2.5 Ausführungsmodi

Das Generic Robot Interface unterstützt zwei Ausführungsmodi zur Optimierung der Roboterzykluszeit:

- Synchrone Ausführung: Der Roboter startet einen Job und wartet auf das erste Ergebnis. Dieser Modus empfiehlt sich, wenn die Ergebnisse sofort benötigt werden.
- Asynchrone Ausführung: Der Roboter startet einen Job und kann mit anderen Vorgängen fortfahren, während der Job im Hintergrund läuft. Der Jobstatus kann abgefragt und Ergebnisse abgerufen werden, sobald diese vorliegen. Dieser Modus maximiert die Effizienz bei langen Erkennungszeiten.

6.4.3 Hand-Auge-Kalibrierung

Für jede Kamera-Pipeline kann eine Hand-Auge-Kalibrierkonfiguration definiert werden, um eine programmgesteuerte Hand-Auge-Kalibrierung mithilfe des GRI zu ermöglichen. Jede Hand-Auge-Kalibrierkonfiguration besteht aus den folgenden Informationen:

- `grid_height`: Höhe des Kalibriermusters in Metern
- `grid_width`: Breite des Kalibriermusters in Metern
- `robot_mounted`: Boolean, das festlegt, ob die Kamera am Roboter montiert ist
- `tcp_offset`: 0 für 6DOF-Roboter. Für 4DOF-Roboter: der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem (am Roboter montierter Sensor) oder zur sichtbaren Oberfläche des Kalibriermusters (statisch montierter Sensor) entlang der TCP-Rotationsachse in Metern.
- `tcp_rotation_axis`: -1 für 6DOF-Roboter. Für 4DOF-Roboter: Bestimmt die Achse des Roboterkoordinatensystems, um die der Roboter seinen TCP drehen kann (0 wird für X, 1 für Y und 2 für die Z-Achse verwendet).

Nähere Informationen zu diesen Einstellungen und zur Hand-Auge-Kalibrierung im Allgemeinen sind in *Hand-Auge-Kalibrierung* beschrieben.

Die Hand-Auge-Kalibrierkonfigurationen können über die Web GUI oder über die REST-API (siehe *Job und HEC_config API*) gesetzt werden.

6.4.4 Spezifikation des Binären GRI Protokolls

Diese Spezifikation definiert das genaue On-Wire-Format für Client-Server-Nachrichten. Eine Nachricht besteht aus einem festen 8-Byte-Header und einem Body, dessen Layout von der Protokollversion abhängt. Derzeit gibt es nur die Protokollversion 1.

Bemerkung: Alle Mehrbyte-Ganzzahlen sind **Little-Endian**. Die Typen sind `uint8` (8 Bit ohne Vorzeichen), `int16` (16 Bit mit Vorzeichen) und `int32` (32 Bit mit Vorzeichen).

6.4.4.1 Header (8 Bytes)

Tab. 6.2: Header Definition

Feld	Typ	Größe	Beschreibung
Magic Number	uint32	4	ASCII tag *„GRI0“*, Bytes *47 52 49 00* (Little-Endian)
protocol_version	uint8	1	Protokollversion: derzeit *1*
message_length	uint8	1	Gesamte Nachrichtengröße (Bytes), inkl. Header + Body
pose_format	uint8	1	Datenformat für Posen (siehe *Posenformate*)
Action	uint8	1	Kommando/Aktion (siehe *Aktionen*)

6.4.4.2 Posenformate

Das GRI verwendet zur Positionsdarstellung immer **Millimeter**. Die folgenden Tabellen zeigen verschiedene Rotationsformate, die passend zur Rotationsdarstellung des verwendeten Roboters ausgewählt werden können. Die Formate sind in Nicht-Euler-Rotationsformate, Tait-Bryan-Euler-Rotationsformate (alle drei Achsen werden verwendet) und reine Euler-Rotationsformate (erste und letzte Rotationsachse sind identisch) unterteilt.

Tab. 6.3: Nicht-Euler Rotationsformate

Name	Wert	rot_1	rot_2	rot_3	rot_4	Einheit	Beispielroboter
QUAT_WXYZ	1	w	x	y	z	–	ABB
QUAT_XYZW	2	x	y	z	w	–	Fruitcore HORST
AXIS_ANGLE_RAD	3	rx	ry	rz	–	rad	Universal Robots

In der folgenden Notation kennzeichnen Hochstriche aufeinanderfolgende Rotationen im intrinsischen Bezugssystem (z. B. Y‘ = Rotation um die neue Y-Achse nach der ersten Rotation). `_B` und `_F` bestimmen die Reihenfolge der Rotationskomponenten. `F` steht für vorwärts (forward), d.h. die Rotationskomponenten werden in derselben Reihenfolge angegeben, in der die Rotation angewendet wird, und `B` steht für rückwärts (backward), d.h. die Rotationskomponenten werden in umgekehrter Reihenfolge angegeben. `_RAD` und `_DEG` bestimmen, ob die Rotationskomponenten gegebenenfalls in Radian oder Grad angegeben werden. Das Format `EULER_ZYX_B_DEG` bedeutet also, dass die intrinsische Rotationsreihenfolge z-y‘-x‘ ist (zuerst Rotation um die z-Achse, dann Rotation um die neue y-Achse, dann Rotation um die neue x-Achse), die Reihenfolge der Rotationskomponenten rückwärts ist (das erste Rotationselement ist also der Winkel um die x-Achse), und die Winkel in Grad angegeben werden.

Tab. 6.4: Tait-Bryan-Euler-Rotationsformate. Hochstriche zeigen aufeinanderfolgende Rotationen im intrinsischen Koordinatensystem an (z.B. ist Y‘ eine Rotation um die neue Y-Achse nach der ersten Rotation). **_F (Forward)**: *[1., 2., 3.]* | **_B (Backward)**: *[3., 2., 1.]*, **_DEG (degrees): Grad** | **_RAD (radian): Radian**.

Name	Wert	rot_1	rot_2	rot_3	rot_4	Einheit	Beispielroboter
EULER_XYZ_F_DEG	4	X	Y‘	Z“	–	deg	
EULER_XYZ_F_RAD	5	X	Y‘	Z“	–	rad	
EULER_XYZ_B_DEG	6	Z“	Y‘	X	–	deg	
EULER_XYZ_B_RAD	7	Z“	Y‘	X	–	rad	
EULER_XZY_F_DEG	8	X	Z‘	Y“	–	deg	
EULER_XZY_F_RAD	9	X	Z‘	Y“	–	rad	
EULER_XZY_B_DEG	10	Y“	Z‘	X	–	deg	
EULER_XZY_B_RAD	11	Y“	Z‘	X	–	rad	
EULER_YXZ_F_DEG	12	Y	X‘	Z“	–	deg	
EULER_YXZ_F_RAD	13	Y	X‘	Z“	–	rad	
EULER_YXZ_B_DEG	14	Z“	X‘	Y	–	deg	
EULER_YXZ_B_RAD	15	Z“	X‘	Y	–	rad	
EULER_YZX_F_DEG	16	Y	Z‘	X“	–	deg	
EULER_YZX_F_RAD	17	Y	Z‘	X“	–	rad	
EULER_YZX_B_DEG	18	X“	Z‘	Y	–	deg	
EULER_YZX_B_RAD	19	X“	Z‘	Y	–	rad	
EULER_ZXY_F_DEG	20	Z	X‘	Y“	–	deg	
EULER_ZXY_F_RAD	21	Z	X‘	Y“	–	rad	
EULER_ZXY_B_DEG	22	Y“	X‘	Z	–	deg	
EULER_ZXY_B_RAD	23	Y“	X‘	Z	–	rad	
EULER_ZYX_F_DEG	24	Z	Y‘	X“	–	deg	KUKA
EULER_ZYX_F_RAD	25	Z	Y‘	X“	–	rad	
EULER_ZYX_B_DEG	26	X“	Y‘	Z	–	deg	FANUC, Mitsubishi, Yaskawa
EULER_ZYX_B_RAD	27	X“	Y‘	Z	–	rad	

Tab. 6.5: Euler-Rotationsformate. Hochstriche zeigen aufeinanderfolgende Rotationen im intrinsischen Koordinatensystem an (z.B. ist Y‘ eine Rotation um die neue Y-Achse nach der ersten Rotation). **_F (Forward)**: *[1., 2., 3.]* | **_B (Backward)**: *[3., 2., 1.]*, **_DEG (degrees): Grad** | **_RAD (radian): Radian**.

Name	Wert	rot_1	rot_2	rot_3	rot_4	Einheit	Beispielroboter
EULER_XYX_F_DEG	28	X	Y‘	X“	–	deg	
EULER_XYX_F_RAD	29	X	Y‘	X“	–	rad	
EULER_XYX_B_DEG	30	X“	Y‘	X	–	deg	
EULER_XYX_B_RAD	31	X“	Y‘	X	–	rad	
EULER_XZX_F_DEG	32	X	Z‘	X“	–	deg	
EULER_XZX_F_RAD	33	X	Z‘	X“	–	rad	
EULER_XZX_B_DEG	34	X“	Z‘	X	–	deg	
EULER_XZX_B_RAD	35	X“	Z‘	X	–	rad	
EULER_YXY_F_DEG	36	Y	X‘	Y“	–	deg	
EULER_YXY_F_RAD	37	Y	X‘	Y“	–	rad	
EULER_YXY_B_DEG	38	Y“	X‘	Y	–	deg	
EULER_YXY_B_RAD	39	Y“	X‘	Y	–	rad	
EULER_YZY_F_DEG	40	Y	Z‘	Y“	–	deg	
EULER_YZY_F_RAD	41	Y	Z‘	Y“	–	rad	
EULER_YZY_B_DEG	42	Y“	Z‘	Y	–	deg	
EULER_YZY_B_RAD	43	Y“	Z‘	Y	–	rad	
EULER_ZXZ_F_DEG	44	Z	X‘	Z“	–	deg	
EULER_ZXZ_F_RAD	45	Z	X‘	Z“	–	rad	
EULER_ZXZ_B_DEG	46	Z“	X‘	Z	–	deg	
EULER_ZXZ_B_RAD	47	Z“	X‘	Z	–	rad	
EULER_ZYZ_F_DEG	88	Z	Y‘	Z“	–	deg	Kawasaki
EULER_ZYZ_F_RAD	49	Z	Y‘	Z“	–	rad	
EULER_ZYZ_B_DEG	50	Z“	Y‘	Z	–	deg	
EULER_ZYZ_B_RAD	51	Z“	Y‘	Z	–	rad	

Alle Posenkomponenten (Position **und** Rotation) sind `int32` mit 1.000.000 skaliert.

- Float zu Int: `int = round(float * 1000000)`
- Int zu Float: `float = int / 1000000.0`
- Positionen werden **vor der Skalierung** in Millimetern erwartet.
- Winkel werden vor der Skalierung in Grad/Radian (je nach Format) erwartet.
- Quaternion-Komponenten haben keine Einheit, verwenden aber dieselbe Skalierung.
- `rot_4` ist bei Euler oder Axis-Angle format ungenutzt (auf 0 gesetzt)
- Wertebereich: `int32` begrenzt jede skalierte Komponente auf ±2147,483647, d.h. Positionen auf ±2147,48 mm pro Achse. Werte außerhalb dieses Bereichs können nicht übertragen werden; das GRI sendet dann keine Antwort.

6.4.4.3 Aktionen

Die folgenden Aktionen können gesendet werden.

Tab. 6.6: GRI Aktionen

Name	Wert	Beschreibung
STATUS	1	Ready-Zustand des Systems abfragen; schreibt den Ready-Zustand auf `data_2` (1 oder 0)
TRIGGER_JOB_SYNC	2	Führt einen Job synchron aus
TRIGGER_JOB_ASYNC	3	Startet einen Job asynchron
GET_JOB_STATUS	4	Abfrage des Jobstatus (siehe *Jobstatus*)
GET_NEXT_POSE	5	Abfrage des nächsten verfügbaren Ergebnisses
GET_RELATED_POSE	6	Abtrage der nächsten zugehörigen Pose
HEC_INIT	7	Hand-Auge-Kalibrierung initialisieren
HEC_SET_POSE	8	Kalibrierpose abspeichern
HEC_CALIBRATE	9	Kalibrierung durchführen und Ergebnis speichern

STATUS (1)

Liefert den Ready-Zustand des *rc_visard* in `data_2` (1 wenn ready, 0 wenn nicht).

TRIGGER_JOB_SYNC (2)

Führt den Job aus und gibt sofort das **erste** Ergebnis zurück. Weitere Ergebnisse werden für einen späteren Abruf gespeichert. Wenn der Job erfolgreich ist und Ergebnisse zurückliefert, ist der `error_code` Null und die Pose befüllt. Wenn keine Ergebnisse zurückgeliefert werden, liefert der `error_code` den Wert `NO_POSES_FOUND` (positiver Wert als Warnung). Außerdem wird Folgendes gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Anzahl der verbleibenden *primären Objekte* (siehe *Primäre und zugehörige Objekte*)
- `data_3` = Anzahl der verbleibenden *zugehörigen Objekte* (siehe *Primäre und zugehörige Objekte*)

Wird der Job nicht innerhalb des Verarbeitungs-Timeouts fertig oder antwortet der Sensor nicht innerhalb des API-Timeouts, ist `error_code` `PROCESSING_TIMEOUT` und der Job wird als `FAILED` markiert.

TRIGGER_JOB_ASYNC (3)

Startet den Job und endet sofort. Der Status des Jobs kann mit `GET_JOB_STATUS` (4) (siehe *Jobstatus*) abgefragt und die Ergebnisse mit `GET_NEXT_POSE` (5) abgerufen werden, sobald der Job abgeschlossen (`DONE`) ist

GET_JOB_STATUS (4)

Liefert den Jobstatus. Es wird gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Jobstatus (siehe Tabelle *Job Statuswerte*)

Fehlerdetails sind in `error_code` enthalten. Ein Job mit Timeout meldet `FAILED` mit `error_code` `PROCESSING_TIMEOUT`; bei `TRIGGER_JOB_ASYNC` ist das der Fall, wenn der Sensor nicht innerhalb des API-Timeouts antwortet.

GET_NEXT_POSE (5)

Gibt das nächste Ergebnis des *primären Objekts* zurück. Außerdem wird Folgendes gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Anzahl der verbleibenden *primären Objekte* (siehe *Primäre und zugehörige Objekte*)
- `data_3` = Anzahl der verbleibenden *zugehörigen Objekte* (siehe *Primäre und zugehörige Objekte*)

Wenn keine weiteren primären Objekte verfügbar sind, wird `NO_POSES_FOUND` zurückgegeben. Der Job bleibt im Zustand `DONE`, sodass die zugehörigen Objekte des letzten primären Objekts weiterhin mit `GET_RELATED_POSE` abgerufen werden können. Wenn der Job noch läuft, gibt `GET_NEXT_POSE`

`JOB_STILL_RUNNING` zurück; wenn er fehlgeschlagen ist oder noch nicht ausgelöst wurde, wird `NO_POSES_FOUND` zurückgegeben.

GET_RELATED_POSE (6)

Gibt die nächste Pose des *zugehörigen Objekts* zum aktuellen *primären Objekt* zurück. Außerdem wird Folgendes gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Anzahl der verbleibenden *primären Objekte* (siehe *Primäre und zugehörige Objekte*)
- `data_3` = Anzahl der verbleibenden *zugehörigen Objekte* (siehe *Primäre und zugehörige Objekte*)

Wenn keine zugehörigen Posen gefunden wurden, wird `NO_RELATED_POSES` zurückgeliefert.

HEC_INIT (7)

Diese Aktion initialisiert die Hand-Auge-Kalibrierung. Sie löscht existierende Kalibrierdaten, wendet die Hand-Auge-Kalibrierkonfiguration der Pipeline an und bereitet das System zum Aufnehmen von Kalibrierposen vor. Der Wert in `data_1` gibt die Zielpipeline für die Kalibrierung an.

HEC_SET_POSE (8)

Diese Aktion wird achtmal verwendet, um unterschiedliche Roboterposen mit sichtbarem Kalibriermuster aufzuzeichnen. Das Feld `data_2` dient zur Angabe des Bildspeicherplatzes (Slot) (0-7). Eine vorherige Pose in einem Slot wird überschrieben, wenn dieser wiederverwendet wird. Jede Pose muss eine andere Ansicht des Kalibriermusters liefern, wie in *Hand-Auge-Kalibrierung* beschrieben. Der Inhalt von `data_1` gibt die Zielpipeline an.

HEC_CALIBRATE (9)

Diese Aktion verarbeitet alle aufgezeichneten Posen und berechnet die Transformation zwischen Kamera und Roboter. Erfolgreiche Kalibrierergebnisse werden automatisch gespeichert. Der Inhalt von `data_1` gibt die Zielpipeline an.

6.4.4.4 Jobstatus

Die folgenden Statuswerte für Jobs können zurückgeliefert werden.

Tab. 6.7: Job Statuswerte

Name	Wert
INACTIVE	1
RUNNING	2
DONE	3
FAILED	4

6.4.4.5 Body Definitionen

Es gibt unterschiedliche Body-Definitionen, je nachdem, ob eine Anfrage gesendet oder eine Antwort empfangen wird. Der Anfrage-Body besteht aus insgesamt 54 Bytes und seine Definition ist in der Tabelle *Anfrage-Body Definition* angegeben.

Tab. 6.8: Anfrage-Body Definition

Feld	Typ	Größe	Beschreibung
Header	struct	8	Nachrichtenheader (siehe *Header (8 Bytes)*)
job_id	uint16	2	Eindeutige Job ID aus der Job Konfiguration
pos_x	int32	4	Position X (skaliert imt 10^6)
pos_y	int32	4	Position Y (skaliert imt 10^6)
pos_z	int32	4	Position Z (skaliert imt 10^6)
rot_1	int32	4	Rotationskomponente 1 (skaliert mit 10^6)
rot_2	int32	4	Rotationskomponente 2 (skaliert mit 10^6)
rot_3	int32	4	Rotationskomponente 3 (skaliert mit 10^6)
rot_4	int32	4	Rotationskomponente 4 (skaliert mit 10^6)
data_1	int32	4	Zusätzlicher Parameter 1
data_2	int32	4	Zusätzlicher Parameter 2
data_3	int32	4	Zusätzlicher Parameter 3
data_4	int32	4	Zusätzlicher Parameter 4

Die Job-ID ist der eindeutige Bezeichner aus der Job-Konfiguration. Die Verwendung der Felder `data_1. ..data_4` hängt von der Aktion und dem Job ab. Sie werden auf *0* gesetzt, wenn sie nicht verwendet werden. Für `TRIGGER_JOB_SYNC`/`TRIGGER_JOB_ASYNC` können diese Felder über die `dynamic_args` Konfiguration des Jobs mit vom Roboter bereitgestellten Werten gefüllt werden, siehe *Rückgabe von Posen und Zusatzdaten*.

Der Antwort-Body besteht aus insgesamt 80 Bytes, seine Definition ist in Tabelle *Antwort-Body Definition* angegeben.

Tab. 6.9: Antwort-Body Definition

Feld	Typ	Größe	Beschreibung
Header	struct	8	Protokollheader
job_id	uint16	2	Verarbeitete Job Nummer
error_code	int16	2	GRI Ergebnisstatus (Schweregrad nach Vorzeichen)
pos_x	int32	4	Position X (skaliert imt 10^6)
pos_y	int32	4	Position Y (skaliert imt 10^6)
pos_z	int32	4	Position Z (skaliert imt 10^6)
rot_1	int32	4	Rotationskomponente 1 (skaliert mit 10^6)
rot_2	int32	4	Rotationskomponente 2 (skaliert mit 10^6)
rot_3	int32	4	Rotationskomponente 3 (skaliert mit 10^6)
rot_4	int32	4	Rotationskomponente 4 (skaliert mit 10^6)
data_1	int32	4	Rückgabecode der Node (0 wenn keiner)
data_2	int32	4	Zusätzliches Ergebnis 2
data_3	int32	4	Zusätzliches Ergebnis 3
data_4	int32	4	Zusätzliches Ergebnis 4
data_5	int32	4	Zusätzliches Ergebnis 5
data_6	int32	4	Zusätzliches Ergebnis 6
data_7	int32	4	Zusätzliches Ergebnis 7
data_8	int32	4	Zusätzliches Ergebnis 8
data_9	int32	4	Zusätzliches Ergebnis 9
data_10	int32	4	Zusätzliches Ergebnis 10

Bemerkung: Für `rc_measure` wird `mean_z` auf pos_x/pos_y/pos_z ausgegeben.

Die Felder `data_4` bis `data_10` werden gemäß der `additional_return` Konfiguration des Jobs gefüllt, siehe *Rückgabe von Posen und Zusatzdaten*.

6.4.4.6 Fehlercodes und Bedeutung

Der Fehlercode `error_code` ist ein `int16` und kodiert Fehler/Warnungen durch Vorzeichen:

- Negativ *< 0* = **error** (Fehler)
- Null = *0* = **success** (Erfolg)
- Positiv *> 0* = **warning** (Erfolg mit Warnung)

Die folgenden Tabellen geben die verschiedenen Fehlercodes an und sind nach Vorzeichen aufgeteilt und sortiert.

Erfolg

Name	Wert	Beschreibung
NO_ERROR	0	Verarbeitung erfolgreich

Negative Fehlercodes

Name	Wert	Beschreibung
UNKNOWN_ERROR	-1	GRI intern, nicht spezifiziert
INTERNAL_ERROR	-2	GRI interner Systemfehler
API_NOT_REACHABLE	-3	API nicht erreichbar
API_RESPONSE_ERROR	-4	API-Aufruf fehlgeschlagen oder hat negativen Code zurückgeliefert
PIPELINE_NOT_AVAILABLE	-5	Reserviert, derzeit nicht verwendet
INVALID_REQUEST_ERROR	-6	Fehlerhafte Anfrage
INVALID_REQUEST_LENGTH	-7	Falsche Nachrichtenlänge
INVALID_ACTION	-8	Nicht unterstützte Aktion
PROCESSING_TIMEOUT	-9	Timeout während der Verarbeitung
UNKNOWN_PROTOCOL_VERSION	-10	Protokollversion nicht unterstützt
WRONG_PROTOCOL_FOR_JOB	-11	Job für nicht unterstützte Protokollversion konfiguriert
JOB_DOES_NOT_EXIST	-12	Invalid job ID
MISCONFIGURED_JOB	-13	Ungültige Job Konfiguration
HEC_CONFIG_ERROR	-14	Ungültige Konfigurationsparameter
HEC_INIT_ERROR	-15	Initialisierung der Kalibrierung fehlgeschlagen
HEC_SET_POSE_ERROR	-16	Pose konnte nicht in angegebenem Slot aufgenommen werden
HEC_CALIBRATE_ERROR	-17	Kalibrierung konnte aus aufgenommenen Posen nicht berechnet werden
HEC_INSUFFICIENT_DETECTION	-18	Kalibriermuster nicht sichtbar oder nicht erkannt
TYPE_CONVERSION_ERROR	-19	Wert nicht in den konfigurierten Typ konvertierbar
UNMAPPED_VALUE_ERROR	-20	Wert hat keinen Eintrag in der referenzierten maps-Tabelle

Positive Codes

Name	Wert	Beschreibung
NO_POSES_FOUND	1	Keine Ergebnisse verfügbar
NO_RELATED_POSES	2	Keine zugehörigen Objekte gefunden
NO_RETURN_SPECIFIED	3	Job ohne Rückgabewerte konfiguriert
JOB_STILL_RUNNING	4	Asynchroner Job nicht beendet
MISSING_ADDITIONAL_DATA	5	additional_return-Pfad nicht gefunden

Node Rückgabecode Bedeutung

Die Module/Nodes können einen `return_code` zurückgeben. Dieser Node-Rückgabecode wird im Antwortfeld `data_1` platziert (standardmäßig 0, wenn kein Code vorhanden ist). Der primäre Status des GRI wird in `error_code` zurückgegeben (vorzeichenbasierte Bedeutung).

`API_RESPONSE_ERROR` wird gemeldet, wenn die Node einen negativen `return_code` zurückgibt, wenn die REST-API mit einem HTTP-Fehler oder ohne `response` antwortet und wenn ein Parameter-Job nicht alle Parameter setzen konnte. Enthält der Fehler der REST-API einen `return_code` (z.B. für ein Modul oder eine Pipeline, die nicht bereit ist), wird dieser Wert in `data_1` gemeldet, außer bei Parameter-Jobs.

Fehlerhafte Nachrichten

Das GRI beantwortet eine fehlerhafte Nachricht mit einem Fehler und schließt danach die Verbindung, wenn dem Bytestrom nicht mehr vertraut werden kann. Der Roboter muss sich dann neu verbinden.

Nachricht	`error_code`	Verbindung
falsche `magic_number`	`INVALID_REQUEST_ERROR`	geschlossen
`message_length` passt nicht zur Protokollversion	`INVALID_REQUEST_LENGTH`	geschlossen
Header empfangen, aber der Body ist unvollständig oder kommt nicht rechtzeitig an	`INVALID_REQUEST_ERROR`	geschlossen
Header unvollständig (weniger als 8 Bytes kommen rechtzeitig an)	keine Antwort	geschlossen
nicht unterstützte `protocol_version`, bei einer `message_length`, die zu den gesendeten Bytes passt	`UNKNOWN_PROTOCOL_VERSION`	bleibt offen
vollständige Nachricht mit ungültigem Inhalt (z.B. `HEC_SET_POSE` ohne Pose)	`INVALID_REQUEST_ERROR`	bleibt offen

6.4.5 Integration mit einem Roboter

Die Generic Robot Interface bietet die Kommunikation auf Port 7100 an.

Für die Integration der GRI-Kommunikation mit einem Roboter werden Beispiele für verschiedene Robotersprachen unter https://github.com/roboception/rc_generic_robot_interface_robot angeboten.

Unterschiedliche Roboterplattformen können durch die Implementierung eines TCP-Socket-Clients unterstützt werden, der dem GRI-Binärprotokoll folgt (siehe *Spezifikation des Binären GRI Protokolls*). Dies erfordert einen Robotercontroller mit TCP/IP-Unterstützung und der Fähigkeit, Roboterposen in Binärnachrichten zu packen und Binärnachrichten in Roboterposen zu parsen.

Die Implementierungsschritte sind wie folgt:

1. TCP Socketverbindung aufbauen
2. Anfragenachricht zusammenstellen:
   - Nachrichtenheader setzen (8 Bytes)
   - Job ID setzen (2 Bytes)
   - Position verpacken (12 Bytes, 3x int32)
   - Rotation verpacken (16 Bytes, 4x int32)
   - Zusätzliche Daten verpacken (16 Bytes, 4x int32)
3. Anfrage senden (54 Bytes insgesamt)
4. Antwort empfangen (80 Bytes insgesamt)
5. Antwort parsen:
   - Header (8 Bytes)
   - Job ID (2 Bytes)

- Fehlercode (2 Bytes)
- Position (12 Bytes, 3x int32)
- Rotation (16 Bytes, 4x int32)
- Zusätzliche Daten (40 bytes, 10x int32)

6.4.5.1 Byte-Interpretation in der Socket-Kommunikation

Einige Skriptsprachen für Roboter interpretieren einzelne Socket-Bytes als vorzeichenbehaftete Werte im Bereich `[-128, 127]` anstatt als vorzeichenlose Werte im Bereich `[0, 255]`. Falls dies der Fall ist, muss jedes Byte **vor** der Rekonstruktion von `int32`-Werten in einen vorzeichenlosen Wert konvertiert werden.

```
# Convert signed byte to unsigned
if byte_value < 0:
    byte_value = byte_value + 256
```

Nach der Konvertierung muss der `int32` Wert in Little-Endian-Byte-Reihenfolge rekonstruiert werden. Anschließend wird die Vorzeicheninterpretation auf das höchstwertige Byte (most significant byte, MSB) angewendet, um festzustellen, ob der Gesamtwert des `int32` negativ ist.

Bemerkung: Alle Posenkomponenten verwenden die in *Posenformate* beschriebene Skalierung.

6.4.6 Job und HEC_config API

Die Jobdefinitionen und die Definitionen von HEC_configs für die Hand-Auge-Kalibrierung können über die folgenden REST-API-Endpunkte gesetzt, abgerufen und gelöscht werden.

`GET /generic_robot_interface/hec_configs`
Liefert die definierten Hand-Auge-Kalibrierkonfigurationen zurück

Musteranfrage

```
GET /api/v2/generic_robot_interface/hec_configs HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "grid_height": 0.18,
    "grid_width": 0.26,
    "robot_mounted": true,
    "tcp_offset": 0,
    "tcp_rotation_axis": -1
  }
}
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /generic_robot_interface/hec_configs/{pipeline}
Liefert die Hand-Auge-Kalibrierkonfiguration für die ausgewählte Pipeline

Musteranfrage

```
GET /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "grid_height": 0.18,
  "grid_width": 0.26,
  "robot_mounted": true,
  "tcp_offset": 0,
  "tcp_rotation_axis": -1
}
```

Parameter

- **pipeline** (*string*) – Pipeline der Hand-Auge-Kalibrierkonfiguration *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

PUT /generic_robot_interface/hec_configs/{pipeline}
Setzt eine Hand-Auge-Kalibrierkonfiguration für die ausgewählte Pipeline.

Musteranfrage

```
PUT /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "return_code": {
    "message": "HEC configuration saved successfully",
    "value": 0
  }
}
```

Parameter

- **pipeline** (*string*) – Pipeline der Hand-Auge-Kalibrierkonfiguration *(obligatorisch)*

JSON-Objekt zur Anfrage

- **hand-eye calibration configuration** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

DELETE /generic_robot_interface/hec_configs/{pipeline}

Entfernt eine Hand-Auge-Kalibrierkonfiguration.

Musteranfrage

```
DELETE /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **pipeline** (*string*) – Pipeline der Hand-Auge-Kalibrierkonfiguration *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Konfiguration für die angegebene Pipeline nicht gefunden

GET /generic_robot_interface/jobs

Liefert die definierten Jobs zurück

Musteranfrage

```
GET /api/v2/generic_robot_interface/jobs HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "args": {
      "pose_frame": "external",
      "tags": []
    },
    "job_type": "CALL_PIPELINE_SERVICE",
    "name": "detect_qr_code",
    "node": "rc_qr_code_detect",
    "pipeline": "0",
    "selected_return": "tags",
    "service": "detect"
  },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "1": {
    "job_type": "SET_PARAMETERS_SERVICE",
    "name": "set_depth_full_quality",
    "node": "rc_stereomatching",
    "parameters": {
      "double_shot": true,
      "quality": "Full"
    },
    "pipeline": "0"
  }
}
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`GET /generic_robot_interface/jobs/{job_id}`

Liefert die Definition des ausgewählten Jobs zurück

Musteranfrage

```
GET /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {
    "pose_frame": "camera",
    "tags": []
  },
  "job_type": "CALL_PIPELINE_SERVICE",
  "name": "detect_qr_code",
  "node": "rc_qr_code_detect",
  "pipeline": "0",
  "selected_return": "tags",
  "service": "detect"
}
```

Parameter

- **`job_id`** (*`string`*) – ID des Jobs *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`PUT /generic_robot_interface/jobs/{job_id}`

Legt eine Jobdefinition für die ausgewählte Job-Art fest. Die erforderlichen Felder hängen vom gewählten Job-Art ab.

Musteranfrage

```
PUT /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "job_id": "1",
  "return_code": {
    "message": "Job configuration updated successfully",
    "value": 0
  }
}
```

Parameter

- **job_id** (*string*) – ID des Jobs *(obligatorisch)*

JSON-Objekt zur Anfrage

- **job definition** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

DELETE /generic_robot_interface/jobs/{job_id}

Entfernt eine Jobdefinition

Musteranfrage

```
DELETE /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **job_id** (*string*) – ID des Jobs *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Job mit angegebener ID nicht gefunden

6.5 OPC UA Interface

Der *rc_visard* bietet auch ein optionales OPC UA Interface auf dem TCP Port 4840 an. Der OPC UA Server kann über eine separate *Lizenz* (Abschnitt 7.4) aktiviert werden.

Der OPC UA Server ermöglicht Zugriff auf die Parameter und Services aller verfügbaren Softwaremodule analog zur REST-API. Um den OPC UA Adressraum zu durchsuchen kann z.B. der frei erhältliche UAExpert GUI Client verwendet werden.

Der OPC UA Server nutzt das `DataTypeDefinition` Attribut (verfügbar in OPC UA Version 1.04) für benutzerdefinierte Datentypen und verwendet auch Methoden und Arrays variabler Länge. Bitte überprüfen Sie, ob Ihr OPC UA Client dies unterstützt.

Bitte kontaktieren Sie support@roboception.de wenn Sie Interesse haben den OPC UA Server zu nutzen.

6.6 KUKA Ethernet KRL Schnittstelle

Der *rc_visard* stellt ein Ethernet KRL Interface (EKI-Bridge) zur Verfügung, welches eine Kommunikation von KUKA KRL via KUKA.EthernetKRL XML mit dem *rc_visard* erlaubt.

Bemerkung: Dieses Modul ist optional und benötigt eine gesonderte EKIBridge-*Lizenz* (Abschnitt 7.4).

Bemerkung: Das KUKA.EthernetKRL add-on Software-Paket Version 2.2 bis maximal 5.x muss auf der Robotersteuerung aktiviert sein, um dieses Modul zu benutzen.

Die EKI-Bridge kann benutzt werden, um programmatisch

- Serviceanfragen auszuführen, z.B. um individuelle Module zu starten und stoppen, oder um angebotene Services wie z.B. die Hand-Auge-Kalibrierung oder Berechnung von Greifposen zu nutzen,
- Laufzeitparameter abzufragen und zu ändern, z.B. der Kamera oder Disparitätsberechnung.

Bemerkung: Eine bekannte Einschränkung der EKI Bridge ist, dass Strings, die valide Zahlen darstellen, nach int/float konvertiert werden. Daher sollten benutzerdefinierte Namen (wie ROI IDs, etc.) immer mindestens einen Buchstaben enthalten, sodass diese als Serviceargumente benutzt werden können.

6.6.1 Konfiguration der Ethernet-Verbindung

Die EKI-Bridge hört auf Port 7000 auf EKI-XML-Nachrichten und übersetzt diese transparent zur *rc_visard REST-API v2* (Abschnitt 6.2). Die empfangenen EKI-Nachrichten werden in JSON umgewandelt und an die *rc_visard* REST-API weitergeleitet. Die Antwort der REST-API wird anschließend zurück in EKI-XML gewandelt.

Die EKI-Bridge erlaubt den Zugriff auf Laufzeitparameter und Services aller Module, die in *Softwaremodule* (Abschnitt 5) beschrieben sind.

Die Ethernet-Verbindung zum *rc_visard* wird auf der Robotersteuerung mit XML-Dateien konfiguriert.

Die Ethernet-Verbindung zum *rc_visard* wird auf der Robotersteuerung mit XML-Dateien konfiguriert. Die EKI-XML-Konfigurationsdateien aller Module auf dem *rc_visard* können hier heruntergeladen werden:

https://doc.rc-visard.com/latest/de/eki.html#eki-xml-configuration-files

Für jedes Softwaremodul, das Laufzeitparameter anbietet, gibt es eine XML-Konfigurationsdatei, um die Parameter abzufragen und zu setzen. Diese sind nach dem Schema `<node_name>-parameters.xml` benannt. Für jeden Service eines Softwaremoduls gibt eine eigene XML-Konfigurationsdatei. Diese ist nach dem Schema `<node_name>-<service_name>.xml` benannt.

Die IP des *rc_visard* im Netzwerk muss in der XML Datei eingetragen werden.

Diese Konfigurationsdateien müssen im Verzeichnis `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL` auf der Robotersteuerung abgelegt werden. Sie werden gelesen, sobald eine Verbindung initialisiert wird.

Um z.B. eine Ethernet-Verbindung mit dem Ziel aufzubauen, um die `rc_stereomatching`-Parameter zu konfigurieren, ist der folgende KRL-Code notwendig.

```
DECL EKI_Status RET
RET = EKI_INIT("rc_stereomatching-parameters")
RET = EKI_Open("rc_stereomatching-parameters")

; ----------- Desired operation -----------

RET = EKI_Close("rc_stereomatching-parameters")
```

Bemerkung: Die EKI-Bridge terminiert automatisch die Verbindung zum Client, wenn eine empfangene XML-Nachricht ungültig ist.

6.6.2 Allgemeine XML-Struktur

Für die Datenanfrage nutzt die EKI-Bridge `<req>` als Wurzelelement (kurz für „Request“).

Das Wurzelelement enthält immer die folgenden Elemente.

- `<node>`: Dieses enthält ein Unterelement, über das die EKI-Bridge das Ziel-Softwaremodul identifiziert. Der Modulname ist bereits in der XML-Konfigurationsdatei vorausgefüllt.
- `<end_of_request>`: „End-of-Request“ Flag, das das Ende der Anfrage markiert und diese auslöst.

Die generische XML-Struktur sieht wie folgt aus.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Für den Datenempfang nutzt die EKI-Bridge `<res>` als Wurzelelement (kurz für „Response“). Das Wurzelelement enthält immer ein `<return_code>` Unterelement.

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Bemerkung: Standardmäßig ist in den Konfigurationsdateien 998 als Flag angegeben, über welches KRL benachrichtigt wird, sobald eine Antwortnachricht empfangen wurde. Falls dieser Wert bereits in Benutzung ist, sollte dieser in der entsprechenden Konfigurationsdatei geändert werden.

6.6.2.1 Rückgabecode

Das `<return_code>`-Element enthält die Attribute `value` und `message`.

Wie für alle anderen Softwaremodule gibt eine erfolgreiche Anfrage ein `res/return_code/@value` mit dem Wert 0 zurück. Negative Werte geben an, dass die Anfrage fehlgeschlagen ist. Die Fehlermeldung ist in `res/return_code/@message` enthalten. Positive Werte geben an, dass die Anfrage erfolgreich war, aber weitere Informationen in `res/return_code/@message` enthalten sind.

Die folgenden Rückgabecodes können von der EKI-Bridge zurückgegeben werden:

Tab. 6.10: Rückgabecodes der EKI-Bridge

Code	Beschreibung
0	Erfolgreich
-1	Parsing-Fehler in der Konvertierung von XML zu JSON
-2	Interner Fehler
-5	Verbindungsfehler von der REST-API
-9	Fehlende oder ungültige Lizenz für das EKI-Bridge-Modul

Bemerkung: Die EKI-Bridge liefert auch Rückgabecodes spezifisch zu den individuellen Softwaremodulen zurück. Diese sind im jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Bemerkung: Aufgrund von Limitierungen in KRL ist die maximale Länge eines Strings, der von der EKI-Bridge zurückgegeben wird, auf 512 Zeichen begrenzt. Alle längeren Strings werden gekürzt.

6.6.3 Services

Das XML-Schema für die Services der Softwaremodule wird aus den Argumenten und der Antwort in *JavaScript Object Notation (JSON)* generiert, wie in *Softwaremodule* (Abschnitt 5) beschrieben. Diese Umwandlung ist bis auf die unten beschriebenen Regeln transparent.

Konvertierung von Posen:

Eine Pose ist ein JSON-Objekt, das die Schlüssel `position` und `orientation` enthält.

```
{
  "pose": {
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
    },
    "orientation": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
      "w": "float64",
    }
  }
}
```

Dieses JSON-Objekt wird zu einem KRL `FRAME` in der XML-Nachricht konvertiert.

```
<pose X="..." Y="..." Z="..." A="..." B="..." C="..."></pose>
```

Positionen werden von Metern in Millimetern umgerechnet und Orientierungen von Quaternionen in das KUKA-ABC-Format (in Grad).

Bemerkung: Es werden in der EKI-Bridge keine anderen Größenumrechnungen vorgenommen. Alle Abmessungen und 3D-Koordinaten, die nicht zu einer Pose gehören, werden in Metern erwartet und zurückgegeben.

Arrays:

Arrays enthalten die Unterelemente `<le>` (kurz für „List Element“). Als Beispiel wird das JSON-Objekt

```
{
  "rectangles": [
    {
      "x": "float64",
      "y": "float64"
    }
  ]
}
```

in das folgende XML-Fragment konvertiert

```
<rectangles>
  <le>
    <x>...</x>
    <y>...</y>
  </le>
</rectangles>
```

XML-Attribute:

Alle JSON-Schlüssel, deren Wert ein primitiver Datentyp ist und die nicht zu einem Array gehören, werden in XML-Attributen gespeichert. Als Beispiel wird das JSON-Objekt

```
{
  "item": {
    "uuid": "string",
    "confidence": "float64",
    "rectangle": {
      "x": "float64",
      "y": "float64"
    }
  }
}
```

in das folgende XML-Fragment konvertiert

```
<item uuid="..." confidence="...">
  <rectangle x="..." y="...">
  </rectangle>
</item>
```

6.6.3.1 Anfrage-XML-Struktur

Das `<SEND>`-Element in der XML-Konfigurationsdatei für einen generischen Service folgt der folgenden Spezifikation:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/service/<service_name>" Type="STRING"/>
    <ELEMENT Tag="req/args/<argX>" Type="<argX_type>"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Das <service>-Element hat ein XML-Unterelement, über das die EKI-Bridge den angefragten Service identifiziert. Es ist bereits vorausgefüllt in der Konfigurationsdatei enthalten.

Das <args> Element beinhaltet die Service-Argumente. Diese können jeweils mit der KRL-Instruktion EKI_Set<Type> gesetzt werden.

Beispielsweise sieht das <SEND>-Element des rc_load_carrier_db get_load_carriers Services (siehe *LoadCarrierDB*, Abschnitt 5.6.1) wie folgt aus.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/rc_load_carrier_db" Type="STRING"/>
    <ELEMENT Tag="req/service/get_load_carriers" Type="STRING"/>
    <ELEMENT Tag="req/args/load_carrier_ids/le" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Das <end_of_request>-Element erlaubt es, Anfragen mit Arrays zu übermitteln. Um ein Array zu senden, wird die Anfrage in so viele Nachrichten wie Array-Elemente aufgeteilt. Die letzte Nachricht beinhaltet alle XML-Tags inklusive dem <end_of_request>-Flag, während alle anderen Nachrichten jeweils nur ein Array-Element enthalten.

Um z.B. zwei Load-Carrier-Modelle mit dem get_load_carriers Service vom rc_load_carrier_db abzufragen, muss der Nutzer zwei XML-Nachrichten senden. Die erste XML-Nachricht lautet:

```
<req>
  <args>
    <load_carrier_ids>
      <le>load_carrier1</le>
    </load_carrier_ids>
  </args>
</req>
```

Diese Nachricht kann über KRL mit dem EKI_Send Kommando gesendet werden, indem das Listenelement als Pfad angegeben wird.

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_load_carrier_db-get_load_carriers", "req/args/load_carrier_ids/
↪le", "load_carrier1")
RET = EKI_Send("rc_load_carrier_db-get_load_carriers", "req/args/load_carrier_ids/le")
```

Die zweite Nachricht beinhaltet alle XML-Tags und löst die Anfrage beim rc_load_carrier_db Softwaremodul aus.

```
<req>
  <node>
    <rc_load_carrier_db></rc_load_carrier_db>
  </node>
  <service>
    <get_load_carriers></get_load_carriers>
  </service>
  <args>
    <load_carrier_ids>
      <le>load_carrier2</le>
    </load_carrier_ids>
  </args>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  <end_of_request></end_of_request>
</req>
```

Diese Nachricht kann über KRL gesendet werden, indem `req` als Pfad für `EKI_Send` angegeben wird:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_load_carrier_db-get_load_carriers", "req/args/load_carrier_ids/
↪le", "load_carrier2")
RET = EKI_Send("rc_load_carrier_db-get_load_carriers", "req")
```

6.6.3.2 Antwort-XML-Struktur

Das `<SEND>`-Element in der XML-Konfigurationsdatei für einen generischen Service folgt der folgenden Spezifikation:

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/<resX>" Type="<resX_type>"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Beispielsweise sieht das `<RECEIVE>`-Element des `rc_april_tag_detect detect` Services (siehe *TagDetect*, Abschnitt 5.4.3) wie folgt aus.

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose_frame" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose/@X" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Y" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Z" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@A" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@B" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@C" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/instance_id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/size" Type="REAL"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Bei Arrays beinhaltet die Antwort mehrere Instanzen des gleichen XML-Elements. Jedes Element wird in einen separaten Puffer in EKI geschrieben und kann daraus mit KRL-Instruktionen ausgelesen werden. Die Anzahl an Instanzen (Array-Elementen) kann über `EKI_CheckBuffer` abgefragt werden und jede Instanz mit `EKI_Get<Type>` ausgelesen werden.

Beispielsweise können die Ergebnisposen aus einer Antwort des `rc_april_tag_detect detect` Services in KRL wie folgt ausgelesen werden:

```
DECL EKI_STATUS RET
DECL INT i
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
DECL INT num_instances
DECL FRAME poses[32]

DECL FRAME pose = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}

RET = EKI_CheckBuffer("rc_april_tag_detect-detect", "res/tags/le/pose")
num_instances = RET.Buff
for i=1 to num_instances
  RET = EKI_GetFrame("rc_april_tag_detect-detect", "res/tags/le/pose", pose)
  poses[i] = pose
endfor
RET = EKI_ClearBuffer("rc_april_tag_detect-detect", "res")
```

Bemerkung: Vor jeder Anfrage über EKI zum *rc_visard* sollten alle Puffer geleert werden, um sicherzustellen, dass nur die aktuelle Antwort in den EKI-Puffern enthalten ist.

6.6.3.3 Speichergrenzen und Array-Trimmung

Beim Zurückgeben großer Arrays erzwingt KUKA EKI zwei Speicherbeschränkungen:

- `BUFFERING` Limit (Warteschlangenlänge): Die maximale Anzahl von Elementen, die in einem einzelnen Array-Bucket gespeichert werden. Eine einzelne Antwort enthält oft mehrere unabhängige Listen (z. B. `res/grasps/le/...` und `res/matches/le/...`), die in vollständig getrennten Buckets im Robotercontroller gespeichert werden. Kein einzelner Bucket darf das konfigurierte `BUFFERING` Limit überschreiten, andernfalls wird die Verbindung getrennt.
- `BUFFSIZE` Limit (Nachrichtengröße): Die maximale Gesamtgröße eines einzelnen XML-Telegramms im Netzwerk in Bytes. Das absolute KUKA-Maximum beträgt 65534 Byte (~64 kB).

Unabhängige Listen innerhalb derselben Response (z.B. `grasps` und `matches`) sind konzeptionell unterschiedlich und haben naturgemäß unterschiedliche Längen. Wenn eine Payload die 64kB-Grenze von `BUFFSIZE` überschreitet, löscht die EKI Bridge automatisch Elemente am Ende der Antwort. Dies bedeutet, dass die EKI Bridge Elemente aus einer Liste trimmen kann, während eine andere Liste unverändert bleibt.

Bemerkung: Alle Kindelemente eines Listenelements (nach le/*) gehören zum selben Array einer REST-API-Response und haben garantiert auch nach dem Trimmen die gleiche Länge.

Es sollte immer `EKI_CheckBuffer` für das jeweilige Array, das gelesen werden soll, verwendet werden, um dessen tatsächliche Länge zu ermitteln. Man kann nicht davon ausgehen, dass die Länge mit der eines anderen Arrays übereinstimmt.

6.6.4 Parameter

Die Parameter aller Softwaremodule können über die EKI-Bridge ausgelesen und gesetzt werden. Die XML-Konfigurationsdatei für ein generisches Softwaremodul folgt dieser Spezifikation:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="req/parameters/<parameter_y>/@value" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
<RECEIVE>
  <XML>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <ELEMENT Tag="res/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@default" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@min" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@max" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@value" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@default" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@min" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@max" Type="REAL"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Die Anfrage wird als Anfrage zum *Lesen* von Parametern interpretiert, wenn die `value`-Attribute aller Parameter leer sind. Falls mindestens ein `value`-Attribut befüllt ist, wird die Anfrage als Anfrage zum *Setzen* von Parametern interpretiert und die befüllten Parameter gesetzt.

Beispielsweise können die aktuellen Werte aller Parameter von `rc_stereomatching` mit der folgenden XML-Nachricht abgefragt werden:

```
<req>
  <node>
    <rc_stereomatching></rc_stereomatching>
  </node>
  <parameters></parameters>
  <end_of_request></end_of_request>
</req>
```

Diese XML-Nachricht kann folgendermaßen über KRL gesendet werden:

```
DECL EKI_STATUS RET
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

Die Antwort der EKI-Bridge enthält alle Parameter:

```
<res>
  <parameters>
    <acquisition_mode default="Continuous" max="" min="" value="Continuous"/>
    <quality default="High" max="" min="" value="High"/>
    <static_scene default="0" max="1" min="0" value="0"/>
    <seg default="200" max="4000" min="0" value="200"/>
    <smooth default="1" max="1" min="0" value="1"/>
    <fill default="3" max="4" min="0" value="3"/>
    <minconf default="0.5" max="1.0" min="0.5" value="0.5"/>
    <mindepth default="0.1" max="100.0" min="0.1" value="0.1"/>
    <maxdepth default="100.0" max="100.0" min="0.1" value="100.0"/>
    <maxdeptherr default="100.0" max="100.0" min="0.01" value="100.0"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

Der `quality`-Parameter von `rc_stereomatching` kann mit folgender XML-Nachricht auf `Low` gesetzt werden:

```
<req>
    <node>
      <rc_stereomatching></rc_stereomatching>
    </node>
    <parameters>
      <quality value="Low"></quality>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    </parameters>
    <end_of_request></end_of_request>
</req>
```

Diese XML-Nachricht kann folgendermaßen über KRL gesendet werden:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_stereomatching-parameters", "req/parameters/quality/@value",
↪"Low")
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

In diesem Fall wird nur der gesetzte Wert von `quality` zurückgegeben:

```
<res>
  <parameters>
    <quality default="High" max="" min="" value="Low"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

6.6.5 Migration zu Firmware Version 22.01

Von Firmware Version 22.01 an bildet die EKI Bridge die *rc_visard REST-API v2* (Abschnitt 6.2) ab.

Dies macht folgende Änderungen nötig:

- Das Konfigurieren von Load Carriern, Greifern und Regions of Interest ist jetzt nur noch in den globalen Datenbankmodulen möglich:
  - Verwendung der `rc_load_carrier_db` XML Dateien um Load Carrier abzurufen, zu erstellen oder zu löschen.
  - Verwendung der `rc_gripper_db` XML Dateien um Greifer abzurufen, zu erstellen oder zu löschen.
  - Verwendung der `rc_roi_db` XML Dateien um Regions of Interest abzurufen, zu erstellen oder zu löschen.
- Load Carrier Erkennung und Füllstandserkennung ist jetzt nur noch über die `rc_load_carrier` Node möglich.
  - Verwendung der `rc_load_carrier` XML Dateien für `detect_load_carriers` und `detect_filling_level` Services.

6.6.6 Beispielanwendungen

Ausführlichere Beispielanwendungen können unter https://github.com/roboception/eki_examples abgerufen werden.

6.6.7 Fehlerbehebung

SmartPad Fehlermeldung: Limit of element memory reached

Dieser Fehler tritt auf, wenn die Anzahl der empfangenen Elemente in einem Array den in der EKI-XML-Konfigurationsdatei festgelegten `BUFFERING` Grenzwert überschreitet. Die Verbindung wird unterbrochen, bevor das KRL-Programm die Daten lesen kann.

- Erhöhen Sie den Wert für `BUFFERING` in der EKI-XML-Konfigurationsdatei auf dem Robotercontroller, um ihn an die maximale Anzahl erwarteter Elemente anzupassen.

- Verringern Sie den Parameter ‚Maximale Matches' (oder ähnlich) im Detektionsmodul des *rc_visard* um sicherzustellen, dass weniger Elemente zurückgeliefert werden.

Der für die Speicherung von EKI-Werten auf dem Roboter verfügbare Gesamtspeicher ist standardmäßig auf 1 MB über alle Verbindungen hinweg begrenzt. Wenn die Grenzwerte zu stark erhöht oder zu viele parallele Arrays verwendet werden, kann der Roboter die Verbindung möglicherweise nicht initialisieren. Informationen zur Vergrößerung dieses Systemspeicherpools finden Sie in der offiziellen KUKA-Dokumentation (KST_EthernetKRL).

SmartPad Fehlermeldung: Reading from empty buffer

Dieser Fehler tritt auf, wenn das KRL-Programm versucht, ein Array-Element zu lesen, das nie empfangen wurde.

- Payload-Kürzung: Die EKI Bridge hat möglicherweise Elemente vom Ende der Antwort getrimmt, um sie an das 64kB-Limit `BUFFSIZE` anzupassen. Dies kann dazu führen, dass unabhängige Arrays unterschiedliche Längen aufweisen (siehe *Speichergrenzen und Array-Trimmung*). Überprüfen Sie die EKI Bridge-Logs auf Warnungen wie „Trimmed XML tree" oder „Pruned response items".
- Nicht übereinstimmende Schleifenanzahl: Das KRL-Programm durchläuft mehr Schleifen, als tatsächlich empfangen wurden. Stellen Sie sicher, dass Sie `EKI_CheckBuffer` für das jeweilige Array verwenden, das Sie lesen.
- KRL-Arraygröße: Stellen Sie sicher, dass die in den KRL-Dateien `.dat` und `.src` definierten Arrays groß genug sind, um alle eingehenden Daten aufzunehmen.

6.7 GigE Vision 2.0/GenICam-Schnittstelle

Gigabit Ethernet for Machine Vision (oder kurz „GigE Vision®") ist ein industrieller Standard für Kameraschnittstellen, der auf UDP/IP basiert (siehe http://www.gigevision.com). Der *rc_visard* nutzt den GigE Vision®-Standard der Version 2.0 und ist damit mit allen GigE Vision®-2.0-Standard-konformen Frameworks und Bibliotheken kompatibel.

GigE Vision® verwendet GenICam (*Generic Interface for Cameras*), um die Eigenschaften der Kamera bzw. des Geräts zu beschreiben. Für nähere Informationen zu dieser generischen Programmierschnittstelle für Kameras siehe http://www.genicam.org/.

Über diese Schnittstelle stellt der *rc_visard* folgende Funktionen zur Verfügung:

- Discovery-Mechanismus,
- IP-Konfiguration,
- Konfiguration kamerabezogener Parameter,
- Bildaufnahme und
- Zeitsynchronisierung über das im Standard IEEE 1588-2008 definierte *Precision Time Protocol* (PTPv2).

Bemerkung: Der *rc_visard* unterstützt Jumbo-Frames mit einer Größe bis 9000 Byte. Für höchste Leistung wird empfohlen, die maximale Übertragungseinheit (MTU) des GigE-Vision-Clients auf 9000 zu stellen.

Bemerkung: Über seine Homepage stellt Roboception Tools und eine C++-Programmierschnittstelle mit Beispielen zum Discovery-Mechanismus, zur Konfiguration und zum Bild-Streaming über die GigE Vision/GenICam-Schnittstelle zur Verfügung (http://www.roboception.com/download).

6.7.1 GigE Vision Ports

GigE Vision ist ein UDP basiertes Protokoll. Auf dem *rc_visard* sind die UDP Ports statisch und bekannt:

- UDP Port 3956: GigE Vision Control Protocol (GVCP). Zum Auffinden, steuern und konfigurieren des Geräts.
- UDP Port 50010: Stream channel source port. Nutzt das GigE Vision Stream Protocol (GVSP) zum transferieren der Bilder.

6.7.2 Wichtige Parameter der GenICam-Schnittstelle

Die folgende Liste enthält einen Überblick über relevante GenICam-Parameter des *rc_visard*, die über die GenICam-Schnittstelle abgerufen und/oder geändert werden können. Neben den Standardparametern, die in der *Standard Feature Naming Convention* (SFNC, siehe http://www.emva.org/standards-technology/genicam/genicam-downloads/) definiert werden, bietet der *rc_visard* zudem eigene Parameter, die sich auf spezielle Eigenschaften des *Kamera Modul* (Abschnitt 5.1) und des *Stereo-Matching Modul* (Abschnitt 5.2) beziehen.

6.7.3 Wichtige Standardparameter der GenICam-Schnittstelle

6.7.3.1 Kategorie: ImageFormatControl

`ComponentSelector`

- Typ: Aufzählung, mögliche Werte: `Intensity`, `IntensityCombined`, `Disparity`, `Confidence` oder `Error`
- Voreinstellung: -
- Beschreibung: Erlaubt dem Benutzer, einen der fünf Bild-Streams zur Konfiguration auszuwählen (siehe *Verfügbare Bild-Streams*, Abschnitt 6.7.6).

`ComponentIDValue` (schreibgeschützt)

- Typ: Integer
- Beschreibung: ID des vom `ComponentSelector` ausgewählten Bild-Streams.

`ComponentEnable`

- Typ: Boolean
- Voreinstellung: -
- Beschreibung: Ist der Parameter auf `true` gesetzt, aktiviert er den im `ComponentSelector` ausgewählten Bild-Stream. Anderenfalls deaktiviert er diesen Stream. Über `ComponentSelector` und `ComponentEnable` lassen sich einzelne Bild-Streams ein- und ausschalten.

`Width` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Bildbreite des vom `ComponentSelector` ausgewählten Bild-Streams in Pixeln.

`Height` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Bildhöhe des vom `ComponentSelector` ausgewählten Bild-Streams in Pixeln.

`WidthMax` (schreibgeschützt)

- Typ: Integer

- Beschreibung: Maximale Breite eines Bildes.

`HeightMax` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Maximale Höhe eines Bildes im Stream. Der Wert beträgt aufgrund der gestapelten Bilder der linken und rechten Kamera im `IntensityCombined`-Stream immer 1920 Pixel (siehe *Verfügbare Bild-Streams*, Abschnitt 6.7.6).

`PixelFormat`

- Typ: Aufzählung, mögliche Werte: `Mono8` oder `YCbCr411_8` (nur bei Farbkameras), `Coord3D_C16`, `Confidence8` und `Error8`
- Beschreibung: Pixelformat der selektierten Komponente. Die Aufzählung erlaubt nur aus Pixelformaten auszuwählen, die für die ausgewählte Komponente möglich sind. Bei einer Farbkamera kann man bei den Komponenten `Intensity` und `IntensityCombined` zwischen den Pixelformaten `Mono8` oder `YCbCr411_8` wählen.

6.7.3.2 Kategorie: AcquisitionControl

`AcquisitionFrameRate`

- Typ: Float, Wertebereich: 1–25 Hz
- Voreinstellung: 25 Hz
- Beschreibung: Bildwiederholrate der Kamera

`ExposureAuto`

- Typ: Aufzählung, mögliche Werte: `Continuous`, `Out1High`, `AdaptiveOut1`, `HDR` oder `Off`
- Voreinstellung: `Continuous`
- Beschreibung: Kombiniert `exp_control` und `exp_auto_mode` Belichtungszeitautomatik Modus. `Off` entspricht *Manual* Belichtungsregelung. `Continuous`, `Out1High` oder `AdaptiveOut1` aktivieren *Auto* Belichtungsregelung mit dem entsprechenden Belichtungszeitautomatik Modus wobei `Continuous` dem *Normal* `exp_auto_mode` entspricht. `HDR` aktiviert die HDR Belichtungsregelung.

`ExposureTime`

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 5000 µs
- Beschreibung: Belichtungszeit der Kameras für den manuellen Belichtungsmodus, ausgedrückt in Mikrosekunden.

6.7.3.3 Kategorie: AnalogControl

`GainSelector` (schreibgeschützt)

- Typ: Aufzählung, Wert: ist immer `All`
- Voreinstellung: `All`
- Beschreibung: Der *rc_visard* unterstützt aktuell nur einen globalen Verstärkungsfaktor.

`Gain`

- Typ: Float, Wertebereich: 0–18 dB
- Voreinstellung: 0 dB
- Beschreibung: Verstärkungsfaktor der Kameras für den manuellen Belichtungsmodus, ausgedrückt in Dezibel.

`BalanceWhiteAuto` (nur für Farbkameras)

- Typ: Aufzählung, mögliche Werte: `Continuous` oder `Off`
- Voreinstellung: `Continuous`
- Beschreibung: Lässt sich für den manuellen Weißabgleich auf `Off` bzw. für den automatischen Weißabgleich auf `Continuous` setzen. Dieser Parameter ist nur für Farbkameras verfügbar.

`BalanceRatioSelector` (nur für Farbkameras)

- Typ: Aufzählung, mögliche Werte: `Red` oder `Blue`
- Voreinstellung: `Red`
- Beschreibung: Auswahl des Verhältnisses welches mit `BalanceRatio` einstellbar ist. `Red` bedeutet Verhältnis von Rot zu Grün, und `Blue` bedeutet Verhältnis von Blau zu Grün. Diese Einstellung ist nur für Farbkameras verfügbar.

`BalanceRatio` (nur für Farbkameras)

- Typ: Float, Wertebereich: 0.125 – 8
- Voreinstellung: 1.2 wenn `Red` und 2.4 wenn `Blue` im `BalanceRatioSelector` eingestellt sind
- Beschreibung: Gewichtung vom roten oder blauen zum grünen Farbkanal. Diese Einstellung ist nur für Farbkameras verfügbar.

6.7.3.4 Kategorie: DigitalIOControl

`LineSelector`

- Typ: Aufzählung, mögliche Werte: `Out1`, `Out2`, `In1` oder `In2`
- Voreinstellung: `Out1`
- Beschreibung: Wählt den Ein- oder Ausgang, um den aktuellen Zustand abzufragen oder die Betriebsart zu setzen.

`LineStatus` (schreibgeschützt)

- Typ: Boolean
- Beschreibung: Aktueller Zustand des mit `LineSelector` ausgewählten Ein- oder Ausgangs.

`LineStatusAll` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Aktueller Zustand der Ein- und Ausgänge, welche in den unteren vier Bits angegeben werden.

Tab. 6.11: Bedeutung der Bits des `LineStatusAll` Parameters.

Bit	4	3	2	1
GPIO	Eingang 2	Eingang 1	Ausgang 2	Ausgang 1

`LineSource`

- Typ: Aufzählung, mögliche Werte: `ExposureActive`, `ExposureAlternateActive`, `Low` oder `High`
- Voreinstellung: `Low`
- Beschreibung: Betriebszustand des mit `LineSelector` gewählten Ausgangs, wie im Abschnitt zum IOControl Modul beschrieben (*out1_mode und out2_mode*, Abschnitt 5.5.4.1). Siehe auch den Parameter `AcquisitionAlternateFilter` zum Filtern von Bildern im Betriebszustand `ExposureAlternateActive`.

6.7.3.5 Kategorie: TransportLayerControl / PtpControl

`PtpEnable`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: Schaltet die PTP-Synchronisierung ein und aus.

6.7.3.6 Kategorie: Scan3dControl

`Scan3dDistanceUnit` (schreibgeschützt)

- Typ: Aufzählung, Wert: ist immer `Pixel`
- Beschreibung: Einheit für die Disparitätsmessungen, ist immer `Pixel`.

`Scan3dOutputMode` (schreibgeschützt)

- Typ: Aufzählung, Wert: ist immer `DisparityC`
- Beschreibung: Modus für die Tiefenmessungen, ist immer `DisparityC`.

`Scan3dFocalLength` (schreibgeschützt)

- Typ: Float
- Beschreibung: Brennweite des mit `ComponentSelector` ausgewählten Bild-Streams in Pixeln. Im Fall der Komponenten `Disparity`, `Confidence` und `Error` hängt der Wert auch von der Auflösung ab, die implizit über `DepthQuality` eingestellt wurde.

`Scan3dBaseline` (schreibgeschützt)

- Typ: Float
- Beschreibung: Basisabstand der Stereokamera in Metern.

`Scan3dPrinciplePointU` (schreibgeschützt)

- Typ: Float
- Beschreibung: Horizontale Position des Bildhauptpunktes des mit `ComponentSelector` ausgewählten Bild-Streams. Im Fall der Komponenten `Disparity`, `Confidence` und `Error` hängt der Wert auch von der Auflösung ab, die implizit über `DepthQuality` eingestellt wurde.

`Scan3dPrinciplePointV` (schreibgeschützt)

- Typ: Float
- Beschreibung: Vertikale Position des Bildhauptpunktes des mit `ComponentSelector` ausgewählten Bild-Streams. Im Fall der Komponenten `Disparity`, `Confidence` und `Error` hängt der Wert auch von der Auflösung ab, die implizit über `DepthQuality` eingestellt wurde.

`Scan3dCoordinateScale` (schreibgeschützt)

- Typ: Float
- Beschreibung: Der Skalierungsfaktor, der mit den Disparitätswerten im Disparitätsbild-Stream zu multiplizieren ist, um die tatsächlichen Disparitätswerte zu erhalten. Der Wert beträgt immer 0,0625.

`Scan3dCoordinateOffset` (schreibgeschützt)

- Typ: Float
- Beschreibung: Der Versatz, der zu den Disparitätswerten im Disparitätsbild-Stream addiert werden muss, um die tatsächlichen Disparitätswerte zu erhalten. Für den *rc_visard* beträgt der Wert immer 0 und kann daher ignoriert werden.

`Scan3dInvalidDataFlag` (schreibgeschützt)

- Typ: Boolean
- Beschreibung: Ist immer `true`, was bedeutet, dass ungültige Daten im Disparitätsbild mit einem spezifischen Wert markiert werden, der durch den Parameter `Scan3dInvalidDataValue` definiert wird.

`Scan3dInvalidDataValue` (schreibgeschützt)

- Typ: Float
- Beschreibung: Ist der Wert, der für ungültige Disparität steht. Der Wert ist immer 0, was bedeutet, dass Disparitätswerte von 0 immer ungültigen Messungen entsprechen. Um zwischen ungültigen Disparitätsmessungen und Messungen, bei denen die Disparität aufgrund der unendlich weit entfernten Objekte 0 beträgt, unterscheiden zu können, wird der Disparitätswert für den letztgenannten Fall auf 0,0625 gesetzt. Dies entspricht noch immer einer Objektentfernung von mehreren hundert Metern.

6.7.3.7 Kategorie: ChunkDataControl

`ChunkModeActive`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: Schaltet Chunk-Daten an, die mit jedem Bild mitgeliefert werden.

6.7.4 Besondere Parameter der GenICam-Schnittstelle des *rc_visard*

6.7.4.1 Kategorie: DeviceControl

`RcSystemReady` (schreibgeschützt)

- Typ: Boolean
- Beschreibung: Gibt an, ob der Bootprozess des Geräts abgeschlossen ist und alle Modules laufen.

`RcParamLockDisable`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: Wenn dieser Wert auf true gesetzt ist, werden die Kamera- und Tiefenbildparameter nicht gesperrt, wenn ein GigE Vision Client mit dem Gerät verbunden wird. Es ist zu beachten, dass – abhängig vom verbundenen GigE Vision Client – Parameteränderungen durch andere Anwendungen (z.B. die Web GUI) möglicherweise nicht durch den GigE Vision Client bemerkt werden, was zu ungewolltem Verhalten führen kann.

6.7.4.2 Kategorie: AcquisitionControl

`AcquisitionAlternateFilter`

- Typ: Aufzählung, mögliche Werte: `Off`, `OnlyHigh` oder `OnlyLow`
- Voreinstellung: `Off`
- Beschreibung: Falls dieser Parameter auf `OnlyHigh` (oder entsprechend `OnlyLow`) und die `LineSource` für mindestens einen Ausgang auf `ExposureAlternateActive` eingestellt wird, dann werden nur die Kamerabilder übertragen, welche aufgenommen wurden, während der konfigurierte Ausgang an war, d.h. ein potentiell angeschlossener Projektor war an (oder bei `OnlyLow` entsprechend aus). Dieser Parameter ist ein einfaches Mittel um nur Bilder ohne ein

projiziertes Muster zu bekommen. Der minimale Zeitunterschied zwischen einem Kamera- und einem Disparitätsbild ist in diesem Fall etwa 40 ms (siehe *IOControl*, Abschnitt 5.5.4.1).

`AcquisitionMultiPartMode`

- Typ: Aufzählung, mögliche Werte: `SingleComponent` oder `SynchronizedComponents`
- Voreinstellung: `SingleComponent`
- Beschreibung: Nur wirksam im MultiPart-Modus. Ist dieser Parameter auf `SingleComponent` gesetzt, werden die Bilder jeweils sofort als einzelne Komponente pro Frame/Puffer geschickt, sobald sie verfügbar sind. Dies entspricht dem Verhalten von Clients, die MultiPart nicht unterstützen. Ist dieser Parameter auf `SynchronizedComponents` gesetzt, werden die aktivierten Komponenten auf dem *rc_visard* zeitlich synchronisiert und in einem gemeinsamen Frame/Puffer versendet – allerdings nur, falls alle für diesen Zeitpunkt verfügbar sind.

`ExposureTimeAutoMax`

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 18000 µs
- Beschreibung: Maximale Belichtungszeit im automatischen Belichtungsmodus.

`ExposureRegionOffsetX`

- Typ: Integer, Wertebereich: 0 bis maximale Bildbreite
- Voreinstellung: 0
- Beschreibung: Horizontaler Offset des Bereichs für die Belichtungszeitregelung in Pixeln.

`ExposureRegionOffsetY`

- Typ: Integer, Wertebereich: 0 bis maximale Bildhöhe
- Voreinstellung: 0
- Beschreibung: Vertikaler Offset des Bereichs für die Belichtungszeitregelung in Pixeln.

`ExposureRegionWidth`

- Typ: Integer, Wertebereich: 0 bis maximale Bildbreite
- Voreinstellung: 0
- Beschreibung: Breite des Bereichs für die Belichtungszeitregelung in Pixeln.

`ExposureRegionHeight`

- Typ: Integer, Wertebereich: 0 bis maximale Bildhöhe
- Voreinstellung: 0
- Beschreibung: Höhe des Bereichs für die Belichtungszeitregelung in Pixeln.

`RcExposureAutoAverageMax`

- Typ: Float, Wertebereich 0-1
- Voreinstellung: 0.75
- Beschreibung: Maximale Helligkeit der automatischen Belichtungszeitsteuerung als Wert zwischen 0 (dunkel) und 1 (hell).

`RcExposureAutoAverageMin`

- Typ: Float, Wertebereich 0-1
- Voreinstellung: 0.25
- Beschreibung: Minimale Helligkeit der automatischen Belichtungszeitsteuerung als Wert zwischen 0 (dunkel) und 1 (hell).

6.7.4.3 Kategorie: Scan3dControl

`FocalLengthFactor` (schreibgeschützt)

- Typ: Float
- Beschreibung: Brennweite skaliert auf eine Bildbreite von einem Pixel. Um die Brennweite für ein bestimmtes Bild in Pixeln zu ermitteln, muss dieser Wert mit der Breite des empfangenen Bilds multipliziert werden. Siehe auch den Parameter `Scan3dFocalLength`.

`Baseline` (schreibgeschützt)

- Typ: Float
- Beschreibung: Dieser Parameter ist überholt. Der Parameter `Scan3dBaseline` sollte stattdessen benutzt werden.

6.7.4.4 Kategorie: DepthControl

`DepthAcquisitionMode`

- Typ: Aufzählung, mögliche Werte: `SingleFrame`, `SingleFrameOut1` oder `Continuous`
- Voreinstellung: `Continuous`
- Beschreibung: Im Modus `SingleFrame` wird das Stereo-Matching mit jedem Aufruf von `DepthAcquisitionTrigger` durchgeführt. Der Modus `SingleFrameOut1` kann zum Kontrollieren eines externen Projektors genutzt werden. Dabei wird bei jedem Trigger `Out1` auf `ExposureAlternateActive` und nach dem Empfangen der Stereobilder auf `Low` gesetzt. Im Modus `Continuous` wird das Stereo-Matching kontinuierlich durchgeführt.

`DepthAcquisitionTrigger`

- type: Command
- Beschreibung: Dieses Kommando triggert das Stereo-Matching auf den nächsten verfügbaren Stereobildern, falls `DepthAcquisitionMode` auf `SingleFrame` oder `SingleFrameOut1` eingestellt ist.

`DepthQuality`

- Typ: Aufzählung, mögliche Werte: `Low`, `Medium`, `High` oder `Full` **(nur mit StereoPlus-Lizenz)**
- Voreinstellung: `High`
- Beschreibung: Qualität der Disparitätsbilder. Eine geringere `DepthQuality` führt zu Disparitätsbildern mit einer geringeren Auflösung (*Qualität*, Abschnitt 5.2.1.2).

`DepthDoubleShot`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: `True` zum Verbessern des Stereo-Matching-Resultats bei Szenen mit Projektor. Löcher im Tiefenbild werden gefüllt mit Tiefendaten aus dem Stereo Matching des Bildpaars ohne Projektormuster (*Double-Shot*, Abschnitt 5.2.1.2).

`DepthStaticScene`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: `True` zum Mitteln über acht aufeinanderfolgende Kamerabilder zur Verbesserung des Stereo-Matching-Resultats (*Statisch*, Abschnitt 5.2.1.2).

`DepthSmooth` (schreibgeschützt ohne StereoPlus-Lizenz)

- Typ: Boolean

- Voreinstellung: `false`
- Beschreibung: `True` um Disparitätswerte zu glätten (*Glättung*, Abschnitt 5.2.1.2).

`DepthFill`

- Typ: Integer, Wertebereich: 0–4 Pixel
- Voreinstellung: 3 Pixel
- Beschreibung: Wert in Pixeln für *Füllen* (Abschnitt 5.2.1.2).

`DepthSeg`

- Typ: Integer, Wertebereich: 0–4000 Pixel
- Voreinstellung: 200 Pixel
- Beschreibung: Wert in Pixeln für *Segmentierung* (Abschnitt 5.2.1.2).

`DepthMinConf`

- Typ: Float, Wertebereich: 0.0–1.0
- Voreinstellung: 0.0
- Beschreibung: Wert für die *Minimale Konfidenz*-Filterung (Abschnitt 5.2.1.2).

`DepthMinDepth`

- Typ: Float, Wertebereich: 0.1–100.0 m
- Voreinstellung: 0.1 m
- Beschreibung: Wert in Metern für die *Minimale Abstands*-Filterung (Abschnitt 5.2.1.2).

`DepthMaxDepth`

- Typ: Float, Wertebereich: 0.1–100.0 m
- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Abstands*-Filterung (Abschnitt 5.2.1.2).

`DepthMaxDepthErr`

- Typ: Float, Wertebereich: 0.01–100.0 m
- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Fehler*-Filterung (Abschnitt 5.2.1.2).

6.7.5 Chunk-Daten

Der *rc_visard* unterstützt Chunk-Parameter, die mit jedem Bild mitgeschickt werden. Chunk-Parameter haben alle den Präfix `Chunk`. Ihre Bedeutung entspricht den gleichlautenden Nicht-Chunk-Parametern. Sie passen jedoch immer zu dem zugehörigen Bild. Zum Beispiel hängt `Scan3dFocalLength` von `ComponentSelector` und `DepthQuality` ab, da die Bildauflösung von beiden Parametern abhängt. Der Parameter `ChunkScan3dFocalLength`, welcher zu einem Bild geliefert wird, passt hingegen zu der Auflösung dieses Bildes.

Nützliche Chunk-Parameter:

- `ChunkComponentSelector` selektiert, für welche Komponente Chunk-Daten aus dem MultiPart-Puffer gelesen werden.
- `ChunkComponentID` und `ChunkComponentIDValue` dienen der eindeutigen Zuordnung des Bildes zu seiner Komponente (z.B. Kamerabild oder Disparitätsbild), ohne dies vom Bildformat oder der Bildgröße ableiten zu müssen.
- `ChunkLineStatusAll` bietet den Status der Ein- und Ausgänge zum Zeitpunkt der Bildaufnahme. Siehe `LineStatusAll` für eine Beschreibung der Bits.

- `ChunkScan3d...` sind nützlich zur 3D-Rekonstruktion wie im Abschnitt *Umwandlung von Bild-Streams* (Abschnitt 6.7.7) beschrieben.
- `ChunkPartIndex` gibt den Index des Bild-Parts im MultiPart-Block für die ausgewählte Komponente (`ChunkComponentSelector`) zurück.
- `ChunkRcOut1Reduction` gibt den Anteil der Bildhelligkeit an, um den Bilder mit GPIO Ausgang 1 (Out1) LOW dunkler sind als Bilder mit Out1 HIGH. Ein Wert von beispielsweise 0.2 bedeutet, dass die Bilder mit GPIO Out1 LOW 20% weniger Helligkeit haben als Bilder mit GPIO Out1 HIGH. Dieser Wert ist nur verfügbar, wenn `exp_auto_mode` der Stereokamera auf `AdaptiveOut1` oder `Out1High` gesetzt ist.

Chunk-Daten werden durch das Setzen des GenICam-Parameters `ChunkModeActive` auf `True` eingeschaltet.

6.7.6 Verfügbare Bild-Streams

Der *rc_visard* stellt über die GenICam-Schnittstelle die folgenden fünf Bild-Streams zur Verfügung:

Name der Komponente	PixelFormat	Beschreibung
`Intensity`	`Mono8` (monochrome Kameras) `YCbCr411_8` (Farbkameras)	Rektifiziertes Bild der linken Kamera
`IntensityCombined`	`Mono8` (monochrome Kameras) `YCbCr411_8` (Farbkameras)	Rektifiziertes Bild der linken Kamera, gestapelt auf das rektifizierte Bild der rechten Kamera
`Disparity`	`Coord3D_C16`	Disparitätsbild in gewünschter Auflösung, d.h. `DepthQuality` in `Full`, `High`, `Medium` oder `Low`
`Confidence`	`Confidence8`	Konfidenzbild
`Error`	`Error8` (Sonderformat: 0x81080001)	Fehlerbild

Jedes Bild wird mit einem Zeitstempel und dem in der Tabelle angegebenen *PixelFormat* ausgegeben. Dieses *PixelFormat* sollte verwendet werden, um zwischen den verschiedenen Bildtypen zu unterscheiden. Bilder, die den gleichen Aufnahmezeitpunkt haben, können durch Vergleich der GenICam-Zeitstempel einander zugeordnet werden.

6.7.7 Umwandlung von Bild-Streams

Das Disparitätsbild enthält vorzeichenlose 16-Bit-Ganzzahlwerte. Diese Werte müssen mit dem im GenICam-Parameter *Scan3dCoordinateScale* angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätswerte d in Pixeln zu ermitteln. Um die 3D-Objektkoordinaten aus den Disparitätswerten berechnen zu können, werden die Brennweite, der Basisabstand und der Bildhauptpunkt benötigt. Diese Parameter werden als GenICam-Parameter *Scan3dFocalLength*, *Scan3dBaseline*, *Scan3dPrincipalPointU* und *Scan3dPrincipalPointV* zur Verfügung gestellt. Die Brennweite und der

Bildhauptpunkt hängen von der Bildauflösung der mit dem *ComponentSelector* selektierten Komponente ab. Sind diese Werte bekannt, können die Pixel-Koordinaten und die Disparitätswerte mithilfe der im Abschnitt *Berechnung von Tiefenbildern und Punktwolken* angegebenen Gleichungen in 3D-Objektkoordination im Kamera-Koordinatensystem umgerechnet werden.

Bemerkung: Das Kamera-Koordinatensystem des *rc_visard* ist in *Sensor-Koordinatensystem* (Abschnitt 3.7) definiert.

Unter der Annahme, dass es sich bei d_{ik} um den 16-Bit-Disparitätswert in der Spalte i und Zeile k eines Disparitätsbildes handelt, ist der Fließkomma-Disparitätswert in Pixeln d_{ik} gegeben durch

$$d_{ik} = d16_{ik} \cdot \mathrm{Scan3dCoordinateScale}$$

Die 3D-Rekonstruktion (in Metern) kann wie folgt mit den GenICam-Parametern durchgeführt werden:

$$\begin{aligned} P_x &= (i + 0.5 - \mathrm{Scan3dPrincipalPointU}) \frac{\mathrm{Scan3dBaseline}}{d_{ik}}, \\ P_y &= (k + 0.5 - \mathrm{Scan3dPrincipalPointV}) \frac{\mathrm{Scan3dBaseline}}{d_{ik}}, \\ P_z &= \mathrm{Scan3dFocalLength} \frac{\mathrm{Scan3dBaseline}}{d_{ik}}. \end{aligned}$$

Das Konfidenzbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Diese Werte müssen durch 255 geteilt werden, um die zwischen 0 und 1 liegenden Konfidenzwerte zu berechnen.

Das Fehlerbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Der Fehler e_{ik} muss mit dem im GenICam-Parameter *Scan3dCoordinateScale* angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätsfehlerwerte d_{eps} in Pixeln zu ermitteln. Der Beschreibung in *Konfidenz- und Fehlerbilder* zufolge lässt sich der Tiefenfehler z_{eps} (in Metern) mit den GenICam-Parametern wie folgt berechnen:

$$\begin{aligned} d_{ik} &= d16_{ik} \cdot \mathrm{Scan3dCoordinateScale}, \\ z_{eps} &= \frac{e_{ik} \cdot \mathrm{Scan3dCoordinateScale} \cdot \mathrm{Scan3dFocalLength} \cdot \mathrm{Scan3dBaseline}}{(d_{ik})^2}. \end{aligned}$$

Bemerkung: Chunk-Daten sollten nach Möglichkeit mit dem Parameter *ChunkModeActive* angeschaltet und die zum Bild zugehörigen Parameter *ChunkScan3dCoordinateScale*, *ChunkScan3dFocalLength*, *ChunkScan3dBaseline*, *ChunkScan3dPrincipalPointU* und *ChunkScan3dPrincipalPointV* genutzt werden, denn deren Werte passen zu der Auflösung des zugehörigen Bildes.

Für nähere Informationen zu Disparitäts-, Fehler- und Konfidenzbildern siehe *Stereo-Matching Modul* (Abschnitt 5.2).

6.8 gRPC Bilddatenschnittstelle

Die gRPC Bilddatenschnittstelle ist eine Alternative zur *GigE Vision / GenICam Schnittstelle* (Abschnitt 6.7) zum Streamen von Kamerabildern und synchronisierten Bilddaten (z.B. linkes Kamerabild und das dazugehörige Disparitätsbild).

gRPC ist ein System zur Interprozesskommunikation über Rechnergrenzen hinweg, welches auch das Streamen von Daten unterstützt. Es benutzt Protocol Buffers (siehe https://developers.google.com/protocol-buffers/) als Beschreibungssprache und zur Datenserialisierung. Eine Einführung und mehr Details zu gRPC sind auf der offiziellen Webseite verfügbar (https://grpc.io/).

Die Vorteile der gRPC Schnittstelle gegenüber GigE Vision sind:

- Es ist in eigenen Programmen einfacher zu benutzen als GigE Vision.

- Es gibt gRPC Unterstützung für sehr viele Programmiersprachen (siehe https://grpc.io/).
- Die Kommunikation basiert auf TCP statt auf UDP und funktioniert deshalb besser über weniger stabile Netzwerke wie z.B. WLAN.

Die Nachteile der gRPC Schnittstelle im Vergleich zu GigE Vision sind:

- Es unterstützt nicht das Ändern von Parametern. Allerdings können alle Parameter über die *REST-API-Schnittstelle* (Abschnitt 6.2) geändert werden.
- Es ist keine Standard-Bildverarbeitungsschnittstelle wie z.B. GigE Vision.

Der *rc_visard* bietet synchronisierte Bilddaten über gRPC Serverstreams auf Port 50051 an.

Die Kommunikation wird gestartet indem eine `ImageSetRequest` Nachricht an den Server geschickt wird. Die Nachricht enthält die Information über angeforderte Bilder, d.h. linkes, rechtes, Disparitäts-, Konfidenz- oder Fehlerbild, die separat an- und abgeschaltet werden können.

Nach dem Empfangen der Anfrage sendet der Server kontinuierlich `ImageSet` Nachrichten, welche alle angeforderten Bilder mit allen Parametern enthalten, die notwendig sind, um die Bilder zu interpretieren. Die Bilder in einer `ImageSet` Nachricht sind synchronisiert, d.h. sie sind alle zum selben Zeitpunkt aufgenommen. Die einzige Ausnahme von dieser Regel besteht, wenn der *out1_mode* (Abschnitt 5.5.4.1) auf `AlternateExposureActive` gesetzt ist. In diesem Fall werden die Kamera- und Disparitätsbilder um 40 ms versetzt aufgenommen, sodass GPIO Out1 auf *aus* (LOW) steht, wenn das linke und rechte Bild aufgenommen werden, und auf *an* (HIGH) für das Disparitäts-, Konfidenz- und Fehlerbild. Dies ist sinnvoll, wenn ein Musterprojektor genutzt wird, da der Projektor dann bei der Aufnahme des linken und rechten Bildes aus ist und für das Disparitätsbild an, wodurch die Kamerabilder ungestört sind, aber das Disparitätsbild deutlich dichter und genauer wird.

Das Streamen von Bildern wird beendet, sobald der Client die Verbindung schließt.

Eine `ImageEventsRequest` Nachricht kann gesendet werden, um das Streaming von `ImageEvents` zu starten. Diese Nachricht enthält das Ereignis `depth_acquisition_done`, welches signalisiert, dass die Bildaufnahme zur Tiefenberechnung abgeschlossen ist. Sie enthält außerdem den `imageset_timestamp` des zugehörigen `ImageSet`. Dieses Ereignis kann verwendet werden, um die Zykluszeit in einer Roboteranwendung zu optimieren, da es signalisiert, ab wann die Kamera oder die Szene nach dem Auslösen einer Erkennung sicher bewegt werden können.

6.8.1 gRPC Servicedefinition

```
syntax = "proto3";

message Time
{
  int32 sec = 1; ///< Seconds
  int32 nsec = 2; ///< Nanoseconds
}

message Gpios
{
  uint32 inputs  = 1; ///< bitmask of available inputs
  uint32 outputs = 2; ///< bitmask of available outputs
  uint32 values  = 3; ///< bitmask of GPIO values
}

message Image
{
  Time timestamp              = 1; ///< Acquisition timestamp of the image
  uint32 height               = 2; ///< image height (number of rows)
  uint32 width                = 3; ///< image width (number of columns)
  float focal_length          = 4; ///< focal length in pixels
  float principal_point_u     = 5; ///< horizontal position of the principal point
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  float principal_point_v     = 6; ///< vertical position of the principal point
  string encoding             = 7; ///< Encoding of pixels ["mono8", "mono16", "rgb8"]
  bool is_bigendian           = 8; ///< is data bigendian, (in our case false)
  uint32 step                 = 9; ///< full row length in bytes
  bytes data                  = 10; ///< actual matrix data, size is (step * height)
  Gpios gpios                 = 11; ///< GPIOs as of acquisition timestamp
  float exposure_time         = 12; ///< exposure time in seconds
  float gain                  = 13; ///< gain factor in decibel
  float noise                 = 14; ///< noise
  float out1_reduction        = 16; ///< Fraction of reduction (0.0 - 1.0) of exposure time for
↪images with GPIO Out1=Low in exp_auto_mode=AdaptiveOut1
  float brightness            = 17; ///< Current brightness of the image as value between 0 and 1
  float gamma                 = 18; ///< Current gamma value of the image, where 1.0 is a linear
↪mapping. Lower gamma values let dark image parts appear brighter.
  bool auto_exposure_adapting = 19; ///< True if auto exposure is currently still adapting
}

message DisparityImage
{
  Time timestamp           = 1; ///< Acquisition timestamp of the image
  float scale              = 2; ///< scale factor
  float offset             = 3; ///< offset in pixels (in our case 0)
  float invalid_data_value = 4; ///< value used to mark pixels as invalid (in our case 0)
  float baseline           = 5; ///< baseline in meters
  float delta_d            = 6; ///< Smallest allowed disparity increment. The smallest
↪achievable depth range resolution is delta_Z = (Z^2/image.focal_length*baseline)*delta_d.
  Image image              = 7; ///< disparity image
}

message Mesh
{
  Time timestamp           = 1; ///< Acquisition timestamp of disparity image from which the mesh
↪is computed
  string format            = 2; ///< currently only "ply" is supported
  bytes data               = 3; ///< actual mesh data
}

message ImageSet
{
  Time timestamp             = 1;
  Image left                 = 2;
  Image right                = 3;
  DisparityImage disparity   = 4;
  Image disparity_error      = 5;
  Image confidence           = 6;
  Mesh mesh                  = 7;
}

message MeshOptions
{
  uint32 max_points            = 1; ///< limit maximum number of points, zero means default (up
↪to 3.1MP), minimum is 1000
  enum BinningMethod {
    AVERAGE = 0;                   ///< average over all points in bin
    MIN_DEPTH = 1;                 ///< use point with minimum depth (i.e. closest to camera) in
↪bin
  }
  BinningMethod binning_method = 2; ///< method used for binning if limited by max_points
  bool watertight              = 3; ///< connect all edges and fill all holes, e.g. for collision
↪checking
  bool textured                = 4; ///< add texture information to mesh
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}

message ImageSetRequest
{
  bool left_enabled            = 1;
  bool right_enabled           = 2;
  bool disparity_enabled       = 3;
  bool disparity_error_enabled = 4;
  bool confidence_enabled      = 5;
  bool mesh_enabled            = 6;
  MeshOptions mesh_options     = 7;
  bool color                   = 8; ///< send left/right image as color (rgb8) images
}

service ImageInterface
{
  // A server-to-client streaming RPC.
  rpc StreamImageSets(ImageSetRequest) returns (stream ImageSet) {}
}

message Event
{
  Time timestamp = 1; ///< timestamp of the event
  string message = 2; ///< optional message of the event
}

message ImageEvents
{
  Time imageset_timestamp      = 1; ///< timestamp of the ImageSet that the event belongs to
  Event depth_acquisition_done = 2; ///< Depth image acquisition is done (e.g. stereo images
↪captured)
}

message ImageEventsRequest
{
  bool depth_acquisition_done_enabled = 1; ///< send event when depth acquisition is done
}

service ImageEventsInterface
{
  // A server-to-client streaming RPC.
  rpc StreamImageEvents(ImageEventsRequest) returns (stream ImageEvents) {}
}
```

6.8.1.1 Umwandlung von Bild-Streams

Das Disparitätsbild enthält vorzeichenlose 16-Bit-Ganzzahlwerte. Diese Werte müssen mit dem `scale` Wert der `DisparityImage` Nachricht multipliziert werden, um die Disparitätswerte d in Pixeln zu ermitteln. Um die 3D-Objektkoordinaten aus den Disparitätswerten berechnen zu können, werden der Basisabstand `baseline` = t aus der `DisparityImage` Nachricht, die Brennweite `focal_length` = f, und der Bildhauptpunkt `principal_point_u` = c_x und `principal_point_v` = c_y aus der `ImageData` Nachricht benötigt. Die Brennweite und der Bildhauptpunkt hängen von der Bildauflösung der Kamera ab und müssen auf die Auflösung des Disparitätsbilds skaliert werden. Sind diese Werte bekannt, können die Pixel-Koordinaten und die Disparitätswerte mithilfe der im Abschnitt *Berechnung von Tiefenbildern und Punktwolken* angegebenen Gleichungen in 3D-Objektkoordination im Kamera-Koordinatensystem umgerechnet werden.

Bemerkung: Das Kamera-Koordinatensystem des *rc_visard* ist in *Sensor-Koordinatensystem* (Abschnitt 3.7) definiert.

Unter der Annahme, dass es sich bei d_{ik} um den 16-Bit-Disparitätswert in der Spalte i und Zeile k eines Disparitätsbildes handelt, ist der Fließkomma-Disparitätswert in Pixeln d_{ik} gegeben durch

$$d_{ik} = d16_{ik} \cdot \text{scale}$$

Die 3D-Rekonstruktion (in Metern) kann wie folgt durchgeführt werden:

$$P_x = (i + 0.5 - c_x) \frac{t}{d_{ik}},$$
$$P_y = (k + 0.5 - c_y) \frac{t}{d_{ik}},$$
$$P_z = f \frac{t}{d_{ik}}.$$

Das Konfidenzbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Diese Werte müssen durch 255 geteilt werden, um die zwischen 0 und 1 liegenden Konfidenzwerte zu berechnen.

Das Fehlerbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Der Fehler e_{ik} muss mit dem im `scale` Wert der `DisparityImage` Nachricht angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätsfehlerwerte d_{eps} in Pixeln zu ermitteln. Der Beschreibung in *Konfidenz- und Fehlerbilder* zufolge lässt sich der Tiefenfehler z_{eps} (in Metern) wie folgt berechnen:

$$d_{ik} = d16_{ik} \cdot \text{scale},$$
$$z_{eps} = \frac{e_{ik} \cdot \text{scale} \cdot f \cdot t}{(d_{ik})^2}.$$

Für nähere Informationen zu Disparitäts-, Fehler- und Konfidenzbildern siehe *Stereo-Matching Modul* (Abschnitt 5.2).

6.8.2 Beispielclient

Ein einfacher C++ Client kann von https://github.com/roboception/grpc_image_client_example heruntergeladen werden.

6.9 Zeitsynchronisierung

Der *rc_visard* stellt für alle Bilder und Nachrichten Zeitstempel zur Verfügung. Um diese mit der Zeit auf dem Applikations-Rechner zu vergleichen, muss die Zeit synchronisiert werden.

Dies kann über das Network Time Protocol (NTP), welches die Standardeinstellung ist, oder über das Precision Time Protocol (PTP) erfolgen.

Bemerkung: Der *rc_visard* verfügt über keine Backup-Batterie für seine Echtzeituhr und behält daher die Zeit nicht, wenn er vom Strom getrennt wird. Die Systemzeit startet beim Anschalten bei der letzten gespeicherten Zeit (wird bei einem Reboot und alle 15 Minuten gespeichert) und wird dann automatisch über NTP gesetzt, falls ein Server gefunden wird.

Die aktuelle Systemzeit sowie der Status der Zeitsynchronisierung können über die *REST-API* (Abschnitt 6.2) abgerufen und darüber hinaus auch in der *Web GUI* (Abschnitt 6.1) auf der Seite *System* eingesehen werden.

Bemerkung: Abhängig von der Erreichbarkeit von NTP- oder PTP-Servern, kann es bis zu mehreren Minuten dauern, bis die Zeit synchronisiert ist.

6.9.1 NTP

Das Network Time Protocol (NTP) ist ein TCP/IP Protokoll um Zeit über ein Netzwerk zu synchronisieren. Im Wesentlichen fordert ein Client die aktuelle Zeit periodisch von einem Server an und nutzt diese, um seine eigene Uhr zu stellen bzw. zu korrigieren.

Standardmäßig versucht der *rc_visard* den NTP-Server des NTP-Pool-Projekts zu erreichen, wozu eine Verbindung zum Internet nötig ist.

Falls die Netzwerkkonfiguration des *rc_visard* auf *DHCP* (Abschnitt 4.4.2) (entspricht der Werkseinstellung) konfiguriert ist, werden NTP-Server auch vom DHCP-Server angefordert und verwendet.

Weiterhin können über den /**system**/**time** Endpoint der REST-API bis zu drei NTP-Server manuell gesetzt werden. Eine einfachere Möglichkeit zum Setzen der manuellen NTP-Server gibt es auf der Seite *Systemzeit* der *Web GUI* (Abschnitt 6.1).

6.9.2 PTP

Das Precision Time Protocol (PTP, auch als IEEE1588 bekannt) ist ein Protokoll, welches eine genauere und robustere Synchronisation der Uhren erlaubt als NTP.

Der *rc_visard* kann als PTP-Slave konfiguriert werden. Dies ist über die Standard *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 6.7) mit dem Parameter `GevIEEE1588` möglich.

Mindestens ein PTP-Master muss die Zeit im Netzwerk zur Verfügung stellen. Unter Linux kann ein PTP-Master beispielsweise auf dem Netzwerkport `eth0` gestartet werden mit `sudo ptpd --masteronly --foreground -i eth0`.

Während der *rc_visard* mit einem PTP-Master synchronisiert ist (Sensor ist im PTP SLAVE-Status), ist die Synchronisierung via NTP pausiert.

6.9.3 Manuelles Setzen der Zeit

Der *rc_visard* ermöglicht das manuelle Setzen des aktuellen Datums und der Uhrzeit über den /**system**/**time** Endpoint der REST-API, wenn keine Zeitsynchronisation aktiv ist (siehe *System und Logs*, Abschnitt 6.2.3.3). Eine einfachere Möglichkeit zum Setzen der Systemzeit gibt es auf der Seite *Systemzeit* der *Web GUI* (Abschnitt 6.1).

7 Wartung

Warnung: Das Gehäuse des *rc_visard* muss für Wartungsarbeiten nicht geöffnet werden. Das unbefugte Öffnen des Produkts führt zum Erlöschen der Garantie.

7.1 Reinigung der Kameralinsen

Glaslinsen sind mit einer Anti-Reflex-Beschichtung versehen, um Spiegelungen zu verringern. Bei der Reinigung der Linsen ist besonders vorsichtig vorzugehen. Mit einer weichen Linsenbürste lassen sich Staub und Schmutzpartikel entfernen. Anschließend kann die Linse mit einem Tuch in kreisenden Bewegungen abgewischt werden: Dabei ist ein Spezialreinigungstuch aus Mikrofaser zu verwenden, um Kratzer zu vermeiden, die die Leistung des Sensors beeinträchtigen können. Hartnäckiger Schmutz lässt sich mit hochreinem Isopropanol oder einer für beschichtete Linsen geeigneten Reinigungslösung (z.B. „Uvex Clear“-Produkte) entfernen.

7.2 Kamerakalibrierung

Die Kameras werden ab Werk kalibriert. Unter normalen Betriebsbedingungen bleibt die Kalibrierung für die Lebensdauer des Sensors erhalten. Wenn der *rc_visard* einer starken mechanischen Belastung ausgesetzt wird, wenn er beispielsweise fallen gelassen wird, können sich die Parameter der Kamera jedoch leicht verändern. In diesem Fall lässt sich die Kalibrierung über die Web GUI überprüfen und bei Bedarf neu durchführen (siehe *Kamerakalibrierung*, Abschnitt 5.5.3).

7.3 Backup der Einstellungen

Der *rc_visard* bietet die Möglichkeit, die aktuellen Einstellungen als Backup oder zum Übertragen auf einen anderen *rc_visard* oder *rc_cube* herunterzuladen.

Die aktuellen Einstellungen des *rc_visard* können über die *Web GUI* (Abschnitt 6.1) auf der Seite *System* im Abschnitt *rc_visard Einstellungen* heruntergeladen werden, oder über die *REST-API-Schnittstelle* (Abschnitt 6.2) des *rc_visard* mit Hilfe des Aufrufs `GET /system/backup`.

Beim Herunterladen des Backups kann der Nutzer entscheiden, welche Einstellungen das Backup enthalten soll:

- `nodes`: die Einstellungen aller Module (Parameter, bevorzugte TCP-Orientierungen und Sortierstrategien)
- `load_carriers`: die erstellten Load Carrier
- `regions_of_interest`: die erstellten 2D und 3D Regions of Interest
- `grippers`: die erstellten Greifer (ohne CAD Elemente)

Das zurückgelieferte Backup sollte als .json-Datei gespeichert werden.

Die Templates des SilhouetteMatch Moduls sind nicht im Backup enthalten, aber können manuell über die REST-API oder die Web GUI heruntergeladen werden (siehe *Template API*, Abschnitt 5.4.7.14).

Ein Backup kann auf dem *rc_visard* über die *Web GUI* (Abschnitt 6.1) auf der Seite *System* im Abschnitt *rc_visard Einstellungen* eingespielt werden, indem die Backup .json-Datei hochgeladen wird. In der *Web GUI* werden die im Backup enthaltenen Einstellungen angezeigt und können für das Einspielen ausgewählt werden. Der zugehörige Aufruf der *REST-API-Schnittstelle* (Abschnitt 6.2) ist `POST /system/backup`.

Warnung: Wenn ein Backup von Load Carriern eingespielt wird, gehen alle bestehenden Load Carrier auf dem *rc_visard* verloren und werden durch die Load Carrier im Backup ersetzt. Das gleiche trifft auf das Einspielen von Greifern und Regions of Interest zu.

Wenn ein Backup eingespielt wird, werden nur die Einstellungen gesetzt, die für den jeweiligen *rc_visard* zutreffend sind. Parameter für Module, die nicht existieren oder keine gültige Lizenz haben, werden ignoriert. Wenn ein Backup nur teilweise eingespielt werden konnte, wird der Benutzer über Warnungen darüber informiert.

7.4 Aktualisierung der Softwarelizenz

Lizenzen, die von Roboception zur Aktivierung zusätzlicher Funktionen erworben werden, können über die Seite *System* → *Firmware & Lizenz* der *Web GUI* (Abschnitt 6.1) installiert werden. Der *rc_visard* muss neu gestartet werden, um die Lizenz nutzen zu können.

7.5 Download der Logdateien

Während des Betriebs dokumentiert der *rc_visard* wichtige Informationen, Hinweise und Fehler in sogenannten Logdateien. Zeigt der *rc_visard* ein unerwartetes oder fehlerhaftes Verhalten, kann mithilfe der Logdateien nach der Fehlerursache geforscht werden. Logeinträge lassen sich über die Seite *System* → *Logs* auf der *Web GUI* (Abschnitt 6.1) ansehen und filtern. Wird der Support kontaktiert (*Kontakt*, Abschnitt 10), sind die Logdateien sehr hilfreich, um Probleme aufzuspüren. Um diese als tar.gz-Datei herunterzuladen, ist der Button *Alle Logs herunterladen* auf der Seite *System* → *Logs* der Web GUI unter *System* zu klicken.

Die Logs sind nicht nur über die Web GUI, sondern auch über die *REST-API-Schnittstelle* (Abschnitt 6.2) des *rc_visard* zugänglich. Hierfür können die Anfragen des Typs `GET /logs` und `GET /logs/{log}` verwendet werden.

7.6 Aktualisierung der Firmware

Angaben zur aktuellen Firmware-Version sind auf der Seite *System* → *Firmware & Lizenz* in der *Web GUI* (Abschnitt 6.1) angegeben. Diese Informationen lassen sich mithilfe einer `GET /system`-Anfrage über die die *REST-API-Schnittstelle* (Abschnitt 6.2) des *rc_visard* abrufen. Die Aktualisierung der Firmware kann entweder über die Web GUI oder über die REST-API vorgenommen werden.

Warnung: Ausgehend von einer Firmware-Version älter als 21.07 werden alle konfigurierten Parameter der Softwaremodule nach einem Firmware-Update auf die Werkseinstellungen zurückgesetzt. Nur beim Update ausgehend von Version 21.07 oder höher bleiben die zuletzt gespeicherten Parameter erhalten. Bevor das Update vorgenommen wird, sollten daher alle Einstellungen (über die

REST-API-Schnittstelle, Abschnitt 6.2) abgefragt und in der Anwendung oder auf dem Client-PC gesichert werden.

Folgende Einstellungen sind davon ausgeschlossen und bleiben auch nach einem Firmware-Update erhalten:

- die Netzwerkkonfiguration des *rc_visard*, samt der ggf. vergebenen festen IP-Adresse und des benutzerdefinierten Gerätenamens,
- das letzte Ergebnis der *Hand-Auge-Kalibrierung* (Abschnitt 5.5.1), was bedeutet, dass der *rc_visard* nicht neu zum Roboter kalibriert werden muss, es sei denn, die Montage wurde verändert, und
- das letzte Ergebnis der *Kamerakalibrierung* (Abschnitt 5.5.3), was bedeutet, dass die Stereokamera des *rc_visard* nicht neu kalibriert werden muss.

Schritt 1: Download der neuesten Firmware Firmware-Updates werden in Form einer Mender-Artefakt-Datei bereitgestellt, die an ihrem `.mender`-Suffix erkennbar ist.

Ist ein neues Firmware-Update für den *rc_visard* erhältlich, kann die Datei von der Roboception-Homepage (https://www.roboception.com/download) auf den lokalen Rechner heruntergeladen werden.

Warnung: Stellen Sie sicher, dass die hochzuladende Firmware-Version noch innerhalb des Software-Wartungszeitraums Ihres *rc_visard* liegt. Sie können die Firmware-Versionsbeschränkungen auf der Web GUI des *rc_visard* unter *System* → *Firmware & Lizenz* einsehen. Sollte die neueste Firmware-Version den Software-Wartungszeitraums überschreiten, muss eine neue Lizenz erworben werden, um eine neuere Firmware nutzen zu können.

Schritt 2: Hochladen der Update-Datei Soll das Update über die REST-API des *rc_visard* vorgenommen werden, kann der Benutzer auf die Anfrage `POST /system/update` zurückgreifen.

Um die Firmware über die Web GUI zu aktualisieren, muss auf der Seite *System* → *Firmware & License* die Schaltfläche „*rc_visard* Update hochladen“ betätigt werden. Nachdem die gewünschte Update-Image-Datei (Dateierweiterung `.mender`) aus dem lokalen Dateisystem ausgewählt und geöffnet wurde, startet das Update.

Je nach Netzwerkarchitektur und Konfiguration kann das Hochladen mehrere Minuten in Anspruch nehmen. Während das Update über die Web GUI läuft, zeigt ein Statusbalken an, wie weit das Update bereits vorangeschritten ist.

Bemerkung: Je nach Webbrowser kann es vorkommen, dass der angezeigte Statusbalken den Abschluss des Updates zu früh angibt. Es empfiehlt sich, zu warten, bis sich ein Benachrichtigungsfenster öffnet, das das Ende des Updatevorgangs anzeigt. Insgesamt ist mit einer Update-Dauer von mindestens fünf Minuten zu rechnen.

Warnung: Die Webbrowser-Registerkarte, die die Web GUI enthält, darf weder geschlossen noch aktualisiert werden, da der Update-Vorgang anderenfalls unterbrochen wird. Ist dies der Fall, muss der Update-Vorgang neu gestartet werden.

Schritt 3: Neustart des *rc_visard* Um ein Firmware-Update auf den *rc_visard* aufzuspielen, muss nach dem Upload der neuen Image-Datei ein Neustart vorgenommen werden.

Bemerkung: Die neue Firmware-Version wird in die inaktive Partition des *rc_visard* hochgeladen. Erst nach dem Neustart wird die inaktive Partition aktiviert und die aktive Partition deaktiviert. Kann das aktualisierte Firmware-Image nicht geladen werden, bleibt diese Partition des *rc_visard* inaktiv und es wird automatisch die zuvor installierte Firmware-Version von der aktiven Partition verwendet.

Über die REST-API lässt sich der Neustart mittels der Anfrage `PUT /system/reboot` vornehmen.

Nachdem die neue Firmware über die Web GUI hochgeladen wurde, öffnet sich ein Benachrichtigungsfenster, in dem der Benutzer aufgefordert wird, das Gerät sofort neu zu starten oder aber den Neustart zu verschieben. Soll der *rc_visard* zu einem späteren Zeitpunkt neu gestartet werden, kann dies über die Schaltfläche *Neustart* auf der Web GUI-Seite *System* vorgenommen werden.

Schritt 4: Bestätigung des Firmware-Updates Nach dem Neustart des *rc_visard* ist die Versionsnummer des derzeit aktiven Firmware-Images zu überprüfen, sodass sichergestellt ist, dass das aktualisierte Image erfolgreich geladen wurde. Dies kann entweder über die Web GUI auf der Seite *System* → *Firmware & Lizenz* oder über die REST-API mittels der Anfrage `GET /system/update` vorgenommen werden.

Kann das Firmware-Update nicht erfolgreich aufgespielt werden, ist der Roboception-Support zu kontaktieren.

7.7 Wiederherstellung der vorherigen Firmware-Version

Nach einem erfolgreichen Firmware-Update wird das vorherige Firmware-Image auf der inaktiven Partition des *rc_visard* hinterlegt und kann von dort bei Bedarf wiederhergestellt werden. Dieses Verfahren wird auch als *Rollback* bezeichnet.

Warnung: Rollbacks werden im UserSpace generell nicht unterstützt. Vor dem Durchführen eines Rollbacks sollten alle UserSpace Apps gesichert werden, um Datenverluste zu vermeiden.

Bemerkung: Es wird dringend empfohlen, die neueste Firmware-Version zu verwenden, die von Roboception zur Verfügung gestellt wurde. Auf das Rollback sollte nur dann zurückgegriffen werden, wenn es mit der aktualisierten Firmware-Version große Probleme gibt.

Die Rollback-Funktion kann lediglich über die *REST-API-Schnittstelle* (Abschnitt 6.2) des *rc_visard* aufgerufen werden – mithilfe der Anfrage `PUT /system/rollback`. Die Anfrage kann entweder mit einem HTTP-kompatiblen Client oder, wie in *Swagger UI* (Abschnitt 6.2.5) beschrieben, über einen Webbrowser ausgelöst werden. Wie beim Update-Prozess ist es auch beim Rollback nötig, das Gerät im Anschluss neu zu starten, um die wiederhergestellte Firmware-Version zu laden.

7.8 Neustart des *rc_visard*

Nach einem Firmware-Update oder einem Software-Rollback muss der *rc_visard* neu gestartet werden. Der Neustart lässt sich entweder programmgesteuert mithilfe der Anforderung `PUT /system/reboot` über die *REST-API-Schnittstelle* (Abschnitt 6.2) des *rc_visard* oder manuell auf der Seite *System* der *Web GUI* (Abschnitt 6.1) vornehmen.

Der Neustart ist abgeschlossen, wenn die LED wieder grün leuchtet.

8 Zubehör

8.1 Anschlussset

Roboception bietet ein optional erhältliches Anschlussset an, um Kunden bei der Einrichtung des *rc_visard* zu unterstützen. Bei dauerhafter Installation muss der Kunde ein geeignetes Netzteil bereitstellen. Das Anschlussset besteht aus folgenden Elementen:

- Netzwerkkabel mit gerader M12-Buchse und geradem RJ45-Stecker, Länge: 2 m, 5 m oder 10 m
- Netzteilkabel mit gerader M12-Buchse und DC-Stecker, Länge: 30 cm
- 24 V, 30 W Steckernetzteil, oder 24 V, 60 W Tischnetzteil

Für den Anschluss des *rc_visard* an ein Wohn- oder Bürogebäudenetz sind Netzteile erforderlich, die den Emissionsstandards nach EN 55011 Klasse B entsprechen. Das im Anschlussset enthaltene Netzteil E2CFS (30 W, 24 V) der EGSTON System Electronics Eggenburg GmbH (http://www.egston.com) ist entsprechend zertifiziert. Es erfüllt jedoch nicht die Anforderungen in Bezug auf Störaussendungen in Industriebereichen (EN 61000-6-2).

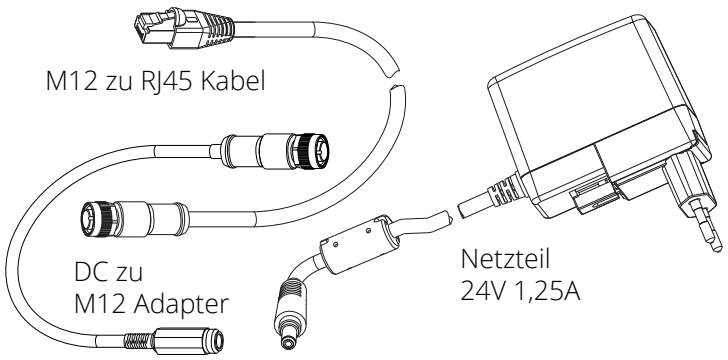


Abb. 8.1: Bestandteile des optional erhältlichen Anschlusssets

8.2 Verkabelung

Kabel sind standardmäßig nicht im Lieferumfang des *rc_visard* enthalten. Es ist Aufgabe des Kunden, geeignete Kabel zu erwerben. In den folgenden Abschnitten wird ein Überblick über die empfohlenen Artikel gegeben.

8.2.1 Ethernet-Anschluss

Der *rc_visard* besitzt eine achtpolige M12-Buchse mit A-Kodierung für den Ethernet-Anschluss. Verschiedene Kabellösungen können direkt von Drittanbietern bezogen werden.

CAT5-Kabel (1 Gbps) für die M12/RJ45-Verbindung

- Gerader M12-Stecker/Gerader RJ45-Stecker; Kabellänge: 10 m; Phoenix Contact; NBC-MS/10,0-94B/R4AC SCO; Art.-Nr.: 1407417
- Gerader M12-Stecker/Gerader RJ45-Stecker; Kabellänge: 10 m; MURR Elektronik; Art.-Nr.: 7700-48521-S4W1000
- Gewinkelter M12-Stecker/Gerader RJ45-Stecker; Kabellänge: 10 m; MURR Elektronik; Art.-Nr.: 7700-48551-S4W1000

8.2.2 Stromanschluss

Für den Stromanschluss und die GPIO-Konnektivität ist ein achtpoliger M12-Stecker mit A-Kodierung vorgesehen. Verschiedene Kabellösungen können direkt von Drittanbietern bezogen werden. Eine Auswahl an M12-Kabeln mit offenem Ende ist unten angegeben. Der Kunde muss die Strom- und GPIO-Anschlüsse gemäß der unter *Verkabelung* (Abschnitt 3.5) angegebenen Steckerbelegung vorsehen. Das Gehäuse des *rc_visard* muss geerdet werden.

Sensor-/Aktor-Kabel mit M12-Buchse und einseitig offenem Ende

- Gerade M12-Buchse/Freies Leitungsende, geschirmt; Kabellänge: 10 m; Phoenix Contact; SAC-8P-10,0-PUR/M12FS SH; Art.Nr.: 1522891
- Gewinkelte M12-Buchse/Freies Leitungsende, geschirmt; Kabellänge: 10 m; Phoenix Contact; SAC-8P-10,0-PUR/M12FR SH; Art.Nr.: 1522943

Sensor-/Aktor-Kabel mit M12-Buchse für die Feldmontage

- Phoenix Contact; SACC-M12FS-8CON-PG9-M; Art.Nr.:1513347
- TE Connectivity T4110011081-000 (Metallgehäuse)
- TE Connectivity T4110001081-000 (Kunststoffgehäuse)

8.2.3 Netzteile

Der *rc_visard* ist als EN-55011 Klasse B Gerät klassifiziert, und für kommerzielle, industrielle und geschäftliche Einsatzbereiche vorgesehen. Um den Sensor an ein Gebäudenetz anschließen zu können, wird ein Netzteil gemäß EN 55011/55022 Klasse B benötigt.

Es ist Aufgabe des Kunden, ein Netzteil zu erwerben und zu installieren, das den Anforderungen der EN 61000-6-2 für die dauerhafte Installation in einem industriellen Umfeld entspricht. Ein Beispiel, das sowohl der EN 61000-6-2 als auch der EN 55011/55022 Klasse B entspricht, ist das Hutschienen-Netzteil PULS MiniLine ML60.241 (24 VDC; 2,5 A) der PULS GmbH (http://www.pulspower.com). Die Installation muss von einem qualifizierten Elektriker vorgenommen werden.

Es darf immer nur ein *rc_visard* an ein Netzteil angeschlossen werden. Die Länge der verwendeten Kabel darf 30 Meter nicht überschreiten.

8.3 Ersatzteile

Für den *rc_visard* sind derzeit keine Ersatzteile erhältlich.

9 Fehlerbehebung

9.1 LED-Farben

Während des Boot-Vorgangs wechselt die LED mehrmals die Farbe, um die verschiedenen Boot-Phasen anzuzeigen:

Tab. 9.1: LED-Farbcodes

LED-Farbe	Boot-Vorgang
Weiß	Stromversorgung OK
Gelb	Normaler Boot-Vorgang
Violett	
Blau	
Grün	Boot-Vorgang abgeschlossen, *rc_visard* einsatzbereit

Die LED dient ferner dazu, Probleme oder Fehlerzustände zu signalisieren, um den Benutzer im Rahmen der Problembehandlung zu unterstützen.

Tab. 9.2: LED-Farbcodes

LED-Farbe	Problem oder Fehlerzustand
Aus	Der Sensor wird nicht mit Strom versorgt.
Kurzes Blinken jede Sekunde	Keine Netzwerkkonnektivität
Blaues Blinken	Systemnachrichten verfügbar, prüfen Sie die Web GUI oder /system/messages
Schnelles rotes Blinken	Temperaturwarnung (Gehäusetemperatur liegt über 60 °C)
Rot	Ein Prozess wurde beendet und kann nicht neu gestartet werden.

9.2 Probleme mit der Hardware

LED leuchtet nicht

Der *rc_visard* fährt nicht hoch.

- Vergewissern Sie sich, dass alle Kabel ordentlich angeschlossen und gesichert sind.
- Vergewissern Sie sich, dass eine geeignete Gleichstromquelle (18–30 V) mit korrekter Polarität an den in der *Spezifikation der Steckerbelegung* (Abschnitt 3.6) mit **Stromzufuhr** und **Masse** gekennzeichneten Pins angeschlossen ist. Wird der Sensor außerhalb des angegebenen Spannungsbereichs, mit Wechselstrom oder mit umgekehrter Polarität betrieben, oder ist er an ein Versorgungsnetz angeschlossen, in dem Spannungsspitzen auftreten, kann dies zu dauerhaften Hardware-Schäden führen.

LED blinkt rot, obwohl der Sensor anscheinend normal funktioniert

Dies kann auf eine erhöhte Gehäusetemperatur hinweisen. Der Sensor ist ggf. so montiert, dass die Luft die Kühlrippen nicht ungehindert umströmen kann.

- Reinigen Sie die Kühlrippen und das Gehäuse.
- Stellen Sie sicher, dass in alle Richtungen um die Kühlrippen 10 cm Platz sind, damit die konvektive Kühlung ordentlich funktioniert.
- Vergewissern Sie sich, dass die Umgebungstemperatur der Spezifikation entspricht.

Der Sensor kann die Verarbeitungsgeschwindigkeit drosseln wenn die Kühlung nicht ausreicht oder die Umgebungstemperatur außerhalb des zugelassenen Bereichs liegt.

Probleme mit der Zuverlässigkeit und/oder mechanische Schäden

Dies kann darauf hinweisen, dass die Umgebungsbedingungen (Vibrationen, Erschütterungen, Schwingungen und Temperatur) außerhalb der *entsprechenden Spezifikationen* (Abschnitt 3.4) liegen.

- Wird der *rc_visard* außerhalb der angegebenen Umgebungsbedingungen betrieben, kann dies zu Schäden am Gerät und zum Erlöschen der Garantie führen.

Stromschlag bei Berührung des Sensors

Dies deutet auf einen elektrischen Defekt im Sensor, in der Verkabelung, im Netzteil oder im angrenzenden System hin.

- Schalten Sie das System unverzüglich aus, ziehen Sie alle Kabel und lassen Sie die Einrichtung des Geräts durch einen qualifizierten Elektriker überprüfen.
- Vergewissern Sie sich, dass das Sensorgehäuse ordentlich geerdet ist. Prüfen Sie auf große Erdschleifen.

9.3 Probleme mit der Konnektivität

LED blinkt jede Sekunde

Wenn die LED jede Sekunde kurz blinkt, kann der *rc_visard* keine Netzwerkverbindung herstellen.

- Überprüfen Sie, ob das Netzwerkkabel ordentlich mit dem *rc_visard* und dem Netzwerk verbunden ist.
- Ist kein Problem erkennbar, tauschen Sie das Ethernet-Kabel aus.

Die Kamera wird vom GigE Vision-Client oder vom rcdiscover-gui-Tool nicht erkannt

- Überprüfen Sie, ob die LED an der Gerätefront des *rc_visard* jede Sekunde kurz blinkt (überprüfen Sie das Kabel, wenn dies der Fall ist).
- Vergewissern Sie sich, dass der *rc_visard* an das gleiche Subnetz angeschlossen ist (der Discovery-Mechanismus nutzt Broadcasts, die nicht über verschiedene Subnetze funktionieren).

Die Web GUI kann nicht aufgerufen werden

- Vergewissern Sie sich, dass der *rc_visard* eingeschaltet und an das gleiche Subnetz wie der Host-Computer angeschlossen ist.
- Überprüfen Sie, ob die LED an der Gerätefront des *rc_visard* jede Sekunde kurz blinkt (überprüfen Sie das Kabel, wenn dies der Fall ist).
- Überprüfen Sie, ob die `rcdiscover-gui` den Sensor erkennt. Gibt das Tool an, dass der *rc_visard* nicht erreichbar ist, ist die *Netzwerkkonfiguration* (Abschnitt 4.4) des *rc_visard* fehlerhaft.
- Wird der *rc_visard* als unerreichbar angegeben, versuchen Sie, einen Doppelklick auf den Geräteeintrag zu machen, um die Web GUI in einem Browser zu öffnen.
- Funktioniert das nicht, versuchen Sie, die vom *rc_visard* gemeldete IP-Adresse direkt als Zieladresse in den Browser einzugeben.

Zu viele Web-GUI-Instanzen gleichzeitig geöffnet

Die Web GUI verbraucht Verarbeitungsressourcen des *rc_visard*, um die zu übertragenden Bilder zu komprimieren und die regelmäßig vom Browser zusammengestellten Statistiken auszugeben. Werden gleichzeitig mehrere Instanzen der Web GUI auf einem oder mehreren Rechnern geöffnet, so kann die Leistung des *rc_visard* stark abnehmen. Die Web GUI ist für Konfigurations- und Validierungszwecke gedacht, nicht jedoch, um den *rc_visard* dauerhaft zu überwachen.

9.4 Probleme mit den Kamerabildern

Kamerabild ist zu hell

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verkürzen oder
- schalten Sie auf automatische Belichtung um.

Kamerabild ist zu dunkel

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verlängern oder
- schalten Sie auf automatische Belichtung um.

Kamerabild rauscht zu stark

Große Gain-Faktoren verursachen ein Bildrauschen mit hoher Amplitude. Wollen Sie das Bildrauschen verringern,

- verwenden Sie eine zusätzliche Lichtquelle, um die Lichtintensität der Aufnahme zu erhöhen, oder
- stellen Sie eine größere maximale Autobelichtungszeit ein.

Kamerabild ist unscharf

- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt, und erhöhen Sie bei Bedarf den Abstand zwischen dem Objekt und der Linse.
- Überprüfen Sie, ob die Kameralinsen verschmutzt sind, und reinigen Sie diese bei Bedarf.
- Trifft keiner der vorstehenden Punkte zu, kann es sein, dass ein schweres Hardware-Problem vorliegt. Bitte wenden Sie sich an den *Support* (Abschnitt 10).

Kamerabild ist verschwommen

Schnelle Bewegungen können in Kombination mit langen Belichtungszeiten zu Unschärfe führen. Um Bewegungsunschärfe zu verringern,

- verringern Sie die Bewegungsgeschwindigkeit der Kamera,
- verringern Sie die Bewegungsgeschwindigkeit von Objekten im Sichtfeld der Kamera oder
- verkürzen Sie die Belichtungszeit der Kameras.

Kamerabild ist unscharf

- Überprüfen Sie, ob die Linsen verschmutzt sind, und reinigen Sie diese bei Bedarf (siehe *Reinigung der Kameralinsen*, Abschnitt 7.1).
- Trifft keiner der vorstehenden Punkte zu, kann es sein, dass ein schweres Hardware-Problem vorliegt. Bitte wenden Sie sich an den *Support* (Abschnitt 10).

Bildwiederholrate ist zu niedrig

- Erhöhen Sie die Bildwiederholrate.
- Die maximale Bildwiederholrate der Kameras beträgt 25 Hz.

9.5 Probleme mit Tiefen-/Disparitäts-, Fehler- oder Konfidenzbildern

Die folgenden Hinweise gelten auch für Fehler- und Konfidenzbilder, da sie direkt mit den Disparitätsbildern zusammenhängen.

Disparitätsbild spärlich befüllt oder leer

- Überprüfen Sie, ob die Kamerabilder gut belichtet und scharf sind. Befolgen Sie bei Bedarf die Anweisungen in *Probleme mit den Kamerabildern* (Abschnitt 9.4).
- Überprüfen Sie, ob die Szene genügend Textur hat und installieren Sie bei Bedarf einen Musterprojektor.
- Senken Sie den *Minimalen Abstand* (Abschnitt 5.2.1).
- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 5.2.1).
- Überprüfen Sie, ob das Objekt zu nahe an der Kamera liegt. Beachten Sie die unterschiedlichen Tiefenmessbereiche der Kameravarianten.
- Senken Sie die *Minimale Konfidenz* (Abschnitt 5.2.1).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 5.2.1).
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 5.2.1). Disparitätsbilder mit einer geringeren Auflösung sind in der Regel nicht so spärlich befüllt.
- Überprüfen Sie die Kalibrierung der Kameras und führen Sie bei Bedarf eine Neukalibrierung durch (siehe *Kamerakalibrierung*, Abschnitt 5.5.3).

Bildwiederholrate der Disparitätsbilder ist zu niedrig

- Überprüfen und erhöhen Sie die Bildwiederholrate der Kamerabilder. Die Bildwiederholrate der Disparitätsbilder kann nicht größer sein als die Bildwiederholrate der Kamerabilder.
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 5.2.1).
- Erhöhen Sie den *Minimalen Abstand* (Abschnitt 5.2.1) so viel wie für die Applikation möglich.

Disparitätsbild zeigt keine nahe liegenden Objekte

- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt. Beachten Sie die unterschiedlichen Tiefenmessbereiche der Kameravarianten.
- Senken Sie den *Minimalen Abstand* (Abschnitt 5.2.1).

Disparitätsbild zeigt keine weit entfernten Objekte

- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 5.2.1).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 5.2.1).
- Senken Sie die *Minimale Konfidenz* (Abschnitt 5.2.1).

Disparitätsbild rauscht zu stark

- Erhöhen Sie den *Segmentierungswert* (Abschnitt 5.2.1).
- Erhöhen Sie den *Füllen-Wert* (Abschnitt 5.2.1).

Disparitätswerte oder resultierende Tiefenwerte sind zu ungenau

- Verringern Sie den Abstand zwischen der Kamera und der Szene. Der Tiefenmessfehler nimmt quadratisch mit dem Abstand zu den Kameras zu.
- Überprüfen Sie, ob die Szene wiederkehrende Muster enthält und entfernen Sie diese bei Bedarf. Diese könnten falsche Disparitätsmessungen verursachen.

Disparitätsbild ist zu glatt

- Senken Sie den *Füllen-Wert* (Abschnitt 5.2.1).

Disparitätsbild zeigt keine feinen Strukturen

- Senken Sie den *Segmentierungswert* (Abschnitt 5.2.1).
- Senken Sie den *Füllen-Wert* (Abschnitt 5.2.1).

9.6 Probleme mit der Zustandsschätzung

Keine Zustandsschätzungen verfügbar

- Kontrollieren Sie in der Web GUI, dass das Dynamik-Modul eingeschaltet ist (siehe *Parameter*, Abschnitt 5.3.2.1).
- Kontrollieren Sie in der Web GUI, dass die Aktualisierungsrate etwa 200 Hz beträgt.
- Überprüfen Sie die *Logs* in der Web GUI auf Fehler.

Zustandsschätzungen rauschen zu stark

- Passen Sie die Parameter für die visuelle Odometrie gemäß den Anweisungen in *Parameter* (Abschnitt 5.3.2.1) an.
- Überprüfen Sie, ob der *Kameraposen-Datenstrom* genau genug ist.

Posenschätzung weist Sprünge auf

- Ist das SLAM-Modul eingeschaltet? SLAM kann Sprünge verursachen, wenn Fehler aufgrund eines Schleifenschlusses korrigiert werden.
- Passen Sie die Parameter für die visuelle Odometrie gemäß den Anweisungen in *Parameter* (Abschnitt 5.3.2.1) an.

Posenfrequenz ist zu niedrig

- Verwenden Sie den Echtzeit-Datenstrom der Kamerapose mit einer Aktualisierungsrate im 200 Hz-Bereich. Siehe *Stereo-INS* (Abschnitt 5.3.3).

Verzögerung/Latenz der Posenschätzung ist zu groß

- Verwenden Sie den Echtzeit-Datenstrom der Kamerapose. Siehe *Stereo-INS* (Abschnitt 5.3.3).

9.7 Probleme mit GigE Vision/GenICam

Keine Bilder

- Überprüfen Sie, ob die Bildkomponenten aktiviert sind. Siehe `ComponentSelector` und `ComponentEnable` in *Wichtige Parameter der GenICam-Schnittstelle* (Abschnitt 6.7.2).

10 Kontakt

10.1 Support

Support-Anfragen können Sie uns entweder über die Seite http://www.roboception.com/support oder per E-Mail an support@roboception.de zukommen lassen.

Bei spezifischen Problemen laden Sie bitte die Logdateien herunter (siehe *Download der Logdateien*, Abschnitt 7.5). Laden Sie gegebenenfalls auch einen Snapshot der letzten Erkennung herunter: Jedes Softwaremodul ermöglicht das Herunterladen der letzten Erkennung auf seiner Seite in der *Web GUI* (Abschnitt 6.1) durch Klicken auf die blaue Kameraschaltfläche unterhalb der Bildstreams.

Das Herunterladen der letzten Erkennung kann auch mithilfe des Services `trigger_dump` des jeweiligen Softwaremoduls automatisiert werden, siehe z.B. *Services* (Abschnitt 5.4.2.7) für das LoadCarrier-Modul.

10.2 Downloads

Software-SDKs usw. können von der Roboception-Homepage heruntergeladen werden: https://roboception.com/resources/knowledge-base.

10.3 Adresse

Roboception GmbH
Kaflerstraße 2
81241 München
Deutschland

Web: http://www.roboception.com
E-Mail: info@roboception.de
Telefon: +49 89 889 50 79-0

11 Anhang

11.1 Formate für Posendaten

Eine Pose besteht aus einer Translation und einer Rotation. Die Translation definiert die Verschiebung entlang der x, y und z-Achsen. Die Rotation kann auf viele verschiedene Arten definiert werden. Der *rc_visard* benutzt Quaternionen, um Rotationen zu definieren, und Translationen werden in Metern angegeben. Dies wird als XYZ+Quaternion Format bezeichnet. Dieses Kapitel erklärt Umrechnungen zwischen verschiedenen üblichen Posenformaten und dem XYZ+Quaternion Format.

Es ist weit verbreitet, Rotationen in 3D durch drei Winkel als Drehungen um die Koordinatenachsen zu definieren. Leider existieren hierfür viele verschiedene Möglichkeiten. Übliche Konventionen sind Euler- oder Kardanwinkel (letztere werden auch als Tait-Bryan Winkel bezeichnet). In beiden Konventionen können die drei Rotationen auf die bereits gedrehten Achsen (intrinsische Rotation) oder auf die Achsen des festen Koordinatensystems (extrinsische Rotation) angewendet werden.

Wir benutzen x, y und z zur Bezeichnung der drei Koordinatenachsen. x', y' und z' bezeichnen die Achsen, die einmal rotiert wurden. x'', y'' und z'' bezeichnen die Achsen nach zwei Rotationen.

In der ursprünglichen Eulerwinkelkonvention ist die erste und dritte Drehachse immer identisch. Die Rotationsreihenfolge z-x'-z'' bedeutet z.B. eine Drehung um die z-Achse, dann eine Drehung um die gedrehte x-Achse und schließlich eine Drehung um die (zweimal) gedrehte z-Achse. In der Kardanwinkelkonvention sind alle drei Drehachsen unterschiedlich, z.B. z-y'-x''. Kardanwinkel werden häufig ebenfalls als Eulerwinkel bezeichnet.

Für jede intrinsische Rotationsreihenfolge gibt es eine äquivalente extrinsische Rotationsreihenfolge, die genau umgekehrt ist. Die intrinsische Rotationsreihenfolge z-y'-x'' ist zum Beispiel äquivalent zu der extrinsischen Rotationsreihenfolge x-y-z.

Rotationen um die x, y und z-Achse können mit Quaternionen definiert werden als

$$r_x(\alpha) = \begin{pmatrix} \sin\frac{\alpha}{2} \\ 0 \\ 0 \\ \cos\frac{\alpha}{2} \end{pmatrix}, \qquad r_y(\beta) = \begin{pmatrix} 0 \\ \sin\frac{\beta}{2} \\ 0 \\ \cos\frac{\beta}{2} \end{pmatrix}, \qquad r_z(\gamma) = \begin{pmatrix} 0 \\ 0 \\ \sin\frac{\gamma}{2} \\ \cos\frac{\gamma}{2} \end{pmatrix},$$

oder durch Rotationsmatrizen als

$$r_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix},$$
$$r_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix},$$
$$r_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Die extrinsische Rotationsreihenfolge x-y-z kann durch die Multiplikation einzelner Rotationen in umge-

kehrter Reihenfolge berechnet werden, d.h. $r_z(\gamma)r_y(\beta)r_x(\alpha)$.

Basierend auf diesen Definitionen beschreiben die folgenden Abschnitte die Umrechnung zwischen üblichen Konventionen und dem XYZ+Quaternion Format.

Bemerkung: Zu beachten sind stets die Einheiten für Positionen und Orientierungen. *rc_visard* Geräte benutzen stets Meter als Längeneinheit, während die meisten Roboterhersteller Längen in Millimeter oder Inch angeben. Winkel werden üblicherweise in Grad angegeben, können aber auch im Bogenmaß angegeben sein.

11.1.1 Rotationsmatrix und Translationsvektor

Eine Pose kann mit einer Rotationsmatrix R und einem Translationsvektor T definiert werden.

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}, \qquad T = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

Die Posentransformation für einen Punkt P ist

$$P' = RP + T.$$

11.1.1.1 Umrechnung von Rotationsmatrizen in Quaternionen

Die Umrechnung von einer Rotationsmatrix (mit $det(R) = 1$) in eine Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ kann wie folgt durchgeführt werden.

$$\begin{aligned} x &= \text{sign}(r_{21} - r_{12})\frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} - r_{11} - r_{22})} \\ y &= \text{sign}(r_{02} - r_{20})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} + r_{11} - r_{22})} \\ z &= \text{sign}(r_{10} - r_{01})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} - r_{11} + r_{22})} \\ w &= \frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} + r_{11} + r_{22})} \end{aligned}$$

Der sign Operator gibt -1 zurück, falls sein Argument negativ ist. Sonst wird 1 zurück gegeben. Er wird zur Wiederherstellung das Vorzeichens der Wurzel benutzt. Die max Funktion stellt sicher, dass das Argument der Wurzel nicht negativ ist, was in der Praxis durch Rundungsfehler passieren kann.

11.1.1.2 Umrechnung von Quaternionen in Rotationsmatrizen

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ mit $||q|| = 1$ in eine Rotationsmatrix kann wie folgt durchgeführt werden.

$$R = 2\begin{pmatrix} \frac{1}{2} - y^2 - z^2 & xy - zw & xz + yw \\ xy + zw & \frac{1}{2} - x^2 - z^2 & yz - xw \\ xz - yw & yz + xw & \frac{1}{2} - x^2 - y^2 \end{pmatrix}$$

11.1.2 ABB Posenformat

ABB Roboter beschreiben eine Pose durch Position X, Y, Z und Quaternion $Q1$, $Q2$, $Q3$, $Q4$, ähnlich wie *rc_visard* Geräte. Jedoch muss die Position in Millimetern angegeben werden und die Quaternion Reihenfolge ist wie folgt definiert:

$q = \begin{pmatrix} x & y & z & w \end{pmatrix} = \begin{pmatrix} Q2 & Q3 & Q4 & Q1 \end{pmatrix}$.

11.1.3 FANUC XYZ-WPR Format

Das Posenformat, welches von FANUC Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung WPR, welche durch drei Winkel in Grad gegeben ist. W rotiert um die x-Achse, P rotiert um die y-Achse und R rotiert um die z-Achse. Die Rotationsreihenfolge ist x-y-z und wird berechnet durch $r_z(R)r_y(P)r_x(W)$.

11.1.3.1 Umrechnung von FANUC-WPR in Quaternionen

Zur Umrechnung von WPR Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden zunächst die Winkel ins Bogenmaß umgerechnet

$$\begin{aligned} W_r &= W\frac{\pi}{180}, \\ P_r &= P\frac{\pi}{180}, \\ R_r &= R\frac{\pi}{180}, \end{aligned}$$

und damit wird die Quaternion berechnet als

$$\begin{aligned} x &= \cos(R_r/2)\cos(P_r/2)\sin(W_r/2) - \sin(R_r/2)\sin(P_r/2)\cos(W_r/2), \\ y &= \cos(R_r/2)\sin(P_r/2)\cos(W_r/2) + \sin(R_r/2)\cos(P_r/2)\sin(W_r/2), \\ z &= \sin(R_r/2)\cos(P_r/2)\cos(W_r/2) - \cos(R_r/2)\sin(P_r/2)\sin(W_r/2), \\ w &= \cos(R_r/2)\cos(P_r/2)\cos(W_r/2) + \sin(R_r/2)\sin(P_r/2)\sin(W_r/2). \end{aligned}$$

11.1.3.2 Umrechnung von Quaternionen in FANUC-WPR

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in WPR Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned} R &= \mathsf{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi} \\ P &= \mathsf{asin}(2(wy - zx))\frac{180}{\pi} \\ W &= \mathsf{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi} \end{aligned}$$

11.1.4 Franka Posenformat

Franka Roboter nutzen eine Transformationsmatrix T um eine Pose zu definieren. Eine Transformationsmatrix kombiniert eine Rotationsmatrix R und einen Translationsvektor $t = (\ x \quad y \quad z\ )^T$.

$$T = \begin{pmatrix} r_{00} & r_{01} & r_{02} & x \\ r_{10} & r_{11} & r_{12} & y \\ r_{20} & r_{21} & r_{22} & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die Posen, die Franka's „Measure Pose“ App ausgibt, bestehen aus einer Translation x, y, z in Millimetern und einer Rotation x, y, z in Grad. Die Rotationsreihenfolge ist z-y'-x'' (d.h. x-y-z) und die Rotation wird berechnet durch $r_z(z)r_y(y)r_x(x)$.

11.1.4.1 Umrechnung von Transformation in Quaternion

Die Umrechnung von einer Rotationsmatrix (mit $det(R) = 1$) in eine Quaternion $q = (\ q_x \quad q_y \quad q_z \quad q_w\ )$ kann wie folgt durchgeführt werden.

$$
\begin{aligned}
q_x &= \text{sign}(r_{21} - r_{12})\frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} - r_{11} - r_{22})} \\
q_y &= \text{sign}(r_{02} - r_{20})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} + r_{11} - r_{22})} \\
q_z &= \text{sign}(r_{10} - r_{01})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} - r_{11} + r_{22})} \\
q_w &= \frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} + r_{11} + r_{22})}
\end{aligned}
$$

Der sign Operator gibt -1 zurück, falls sein Argument negativ ist. Sonst wird 1 zurück gegeben. Er wird zur Wiederherstellung das Vorzeichens der Wurzel benutzt. Die max Funktion stellt sicher, dass das Argument der Wurzel nicht negativ ist, was in der Praxis durch Rundungsfehler passieren kann.

11.1.4.2 Umrechnung von Rotation-XYZ in Quaternion

Zur Umrechnung von der Rotationswinkel x, y, z in Grad in eine Quaternion $q = (\ q_x \quad q_y \quad q_z \quad q_w\ )$ werden zuerst alle Winkel in das Bogenmaß umgerechnet mit

$$
\begin{aligned}
X_r &= x\frac{\pi}{180}, \\
Y_r &= y\frac{\pi}{180}, \\
Z_r &= z\frac{\pi}{180},
\end{aligned}
$$

und damit die Quaternion berechnet durch

$$
\begin{aligned}
q_x &= \cos(Z_r/2)\cos(Y_r/2)\sin(X_r/2) - \sin(Z_r/2)\sin(Y_r/2)\cos(X_r/2), \\
q_y &= \cos(Z_r/2)\sin(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\cos(Y_r/2)\sin(X_r/2), \\
q_z &= \sin(Z_r/2)\cos(Y_r/2)\cos(X_r/2) - \cos(Z_r/2)\sin(Y_r/2)\sin(X_r/2), \\
q_w &= \cos(Z_r/2)\cos(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\sin(Y_r/2)\sin(X_r/2).
\end{aligned}
$$

11.1.4.3 Umrechnung von Quaternion und Translation in Transformation

Die Umrechnung von einer Quaternion $q = (\ q_x \quad q_y \quad q_z \quad q_w\ )$ und einem Translationsvektor $t = (\ x \quad y \quad z\ )^T$ in eine Transformationsmatrix T kann wie folgt durchgeführt werden.

$$
T = \begin{pmatrix}
1 - 2s(q_y^2 + q_z^2) & 2s(q_x q_y - q_z q_w) & 2s(q_x q_z + q_y q_w) & x \\
2s(q_x q_y + q_z q_w) & 1 - 2s(q_x^2 + q_z^2) & 2s(q_y q_z - q_x q_w) & y \\
2s(q_x q_z - q_y q_w) & 2s(q_y q_z + q_x q_w) & 1 - 2s(q_x^2 + q_y^2) & z \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

wobei $s = ||q||^{-2} = \frac{1}{q_x^2 + q_y^2 + q_z^2 + q_w^2}$ und $s = 1$ wenn q eine Einheitsquaternion ist.

11.1.4.4 Umrechnung von Quaternion in Rotation-XYZ

Die Umrechnung von einer Quaternion $q = (\ q_x \quad q_y \quad q_z \quad q_w\ )$ mit $||q|| = 1$ in x, y, z Winkel in Grad kann wie folgt durchgeführt werden.

$$x = \text{atan}_2(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2))\frac{180}{\pi}$$
$$y = \text{asin}(2(q_w q_y - q_z q_x))\frac{180}{\pi}$$
$$z = \text{atan}_2(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2))\frac{180}{\pi}$$

11.1.4.5 Posenrepräsentation in RaceCom Messages und Statemachines

In RaceCom Messages und in Statemachines wird eine Pose normalerweise als eindimensionales Array aus 16 Floatwerten definiert, die in spaltenweiser Anordnung eine Transformationsmatrix repräsentieren. Die Indizes der Einträge der folgenden Matrix entsprechen den Array-Indizes.

$$T = \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

11.1.5 Fruitcore HORST Posenformat

Fruitcore HORST Roboter beschreiben eine Pose durch eine Position in Metern und ein Quaternion mit $q_0 = w$, $q_1 = x$, $q_2 = y$ und $q_3 = z$ wie auch *rc_visard* Geräte. Es ist keine Konvertierung notwendig.

11.1.6 Kawasaki XYZ-OAT Format

Das Posenformat, welches von Kawasaki Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung OAT, welche durch drei Winkel in Grad angegeben wird. O rotiert um die z-Achse, A rotiert um die gedrehte y-Achse und T rotiert um die gedrehte z-Achse. Die Rotationsreihenfolge ist z-y'-z'' (d.h. z-y-z) und wird berechnet durch $r_z(O)r_y(A)r_z(T)$.

11.1.6.1 Umrechnung von Kawasaki-OAT in Quaternionen

Zur Umrechnung von OAT Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden zunächst alle Winkel in das Bogenmaß umgerechnet durch

$$O_r = O\frac{\pi}{180},$$
$$A_r = A\frac{\pi}{180},$$
$$T_r = T\frac{\pi}{180},$$

und damit wird die Quaternion berechnet durch

$$x = \cos(O_r/2)\sin(A_r/2)\sin(T_r/2) - \sin(O_r/2)\sin(A_r/2)\cos(T_r/2),$$
$$y = \cos(O_r/2)\sin(A_r/2)\cos(T_r/2) + \sin(O_r/2)\sin(A_r/2)\sin(T_r/2),$$
$$z = \sin(O_r/2)\cos(A_r/2)\cos(T_r/2) + \cos(O_r/2)\cos(A_r/2)\sin(T_r/2),$$
$$w = \cos(O_r/2)\cos(A_r/2)\cos(T_r/2) - \sin(O_r/2)\cos(A_r/2)\sin(T_r/2).$$

11.1.6.2 Umrechnung von Quaternionen in Kawasaki-OAT

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in OAT Winkel in Grad kann wie folgt durchgeführt werden.

Wenn $x = 0$ **und** $y = 0$ ist die Umrechnung

$$O = \text{atan}_2(2(z - w), 2(z + w))\frac{180}{\pi}$$
$$A = \text{acos}(w^2 + z^2)\frac{180}{\pi}$$
$$T = \text{atan}_2(2(z + w), 2(w - z))\frac{180}{\pi}$$

Wenn $z = 0$ **und** $w = 0$ ist die Umrechnung

$$O = \text{atan}_2(2(y - x), 2(x + y))\frac{180}{\pi}$$
$$A = \text{acos}(-1.0)\frac{180}{\pi}$$
$$T = \text{atan}_2(2(y + x), 2(y - x))\frac{180}{\pi}$$

In allen anderen Fällen ist die Umrechnung

$$O = \text{atan}_2(2(yz - wx), 2(xz + wy))\frac{180}{\pi}$$
$$A = \text{acos}(w^2 - x^2 - y^2 + z^2)\frac{180}{\pi}$$
$$T = \text{atan}_2(2(yz + wx), 2(wy - xz))\frac{180}{\pi}$$

11.1.7 KUKA XYZ-ABC Format

KUKA Roboter nutzen das sogenannte XYZ-ABC Format. XYZ ist die Position in Millimetern. ABC sind Winkel in Grad, wobei A um die z-Achse rotiert, B rotiert um die y-Achse und C rotiert um die x-Achse. Die Rotationsreihenfolge ist z-y'-x'' (i.e. x-y-z) und wird berechnet durch $r_z(A)r_y(B)r_x(C)$.

11.1.7.1 Umrechnung von KUKA-ABC in Quaternionen

Zur Umrechnung von ABC Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden zuerst alle Winkel in das Bogenmaß umgerechnet mit

$$A_r = A\frac{\pi}{180},$$
$$B_r = B\frac{\pi}{180},$$
$$C_r = C\frac{\pi}{180},$$

und damit die Quaternion berechnet durch

$$x = \cos(A_r/2)\cos(B_r/2)\sin(C_r/2) - \sin(A_r/2)\sin(B_r/2)\cos(C_r/2),$$
$$y = \cos(A_r/2)\sin(B_r/2)\cos(C_r/2) + \sin(A_r/2)\cos(B_r/2)\sin(C_r/2),$$
$$z = \sin(A_r/2)\cos(B_r/2)\cos(C_r/2) - \cos(A_r/2)\sin(B_r/2)\sin(C_r/2),$$
$$w = \cos(A_r/2)\cos(B_r/2)\cos(C_r/2) + \sin(A_r/2)\sin(B_r/2)\sin(C_r/2).$$

11.1.7.2 Umrechnung von Quaternionen in KUKA-ABC

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in ABC Winkel in Grad kann wie folgt durchgeführt werden.

$$A = \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi}$$
$$B = \text{asin}(2(wy - zx))\frac{180}{\pi}$$
$$C = \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}$$

11.1.8 Mitsubishi XYZ-ABC Format

Das Posenformat, welches von Mitsubishi Robotern benutzt wird, ist das gleiche wie für KUKA Roboter (siehe *KUKA XYZ-ABC Format*, Abschnitt 11.1.7), außer, dass der Winkel A um die x-Achse rotiert und C eine Rotation um die z-Achse ist. Damit wird die Rotation berechnet durch $r_z(C)r_y(B)r_x(A)$.

11.1.8.1 Umrechnung von Mitsubishi-ABC in Quaternionen

Zur Umrechnung von ABC Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden die Winkel zunächst ins Bogenmaß umgerechnet mit

$$A_r = A\frac{\pi}{180},$$
$$B_r = B\frac{\pi}{180},$$
$$C_r = C\frac{\pi}{180},$$

und damit die Quaternion berechnet durch

$$x = \cos(C_r/2)\cos(B_r/2)\sin(A_r/2) - \sin(C_r/2)\sin(B_r/2)\cos(A_r/2),$$
$$y = \cos(C_r/2)\sin(B_r/2)\cos(A_r/2) + \sin(C_r/2)\cos(B_r/2)\sin(A_r/2),$$
$$z = \sin(C_r/2)\cos(B_r/2)\cos(A_r/2) - \cos(C_r/2)\sin(B_r/2)\sin(A_r/2),$$
$$w = \cos(C_r/2)\cos(B_r/2)\cos(A_r/2) + \sin(C_r/2)\sin(B_r/2)\sin(A_r/2).$$

11.1.8.2 Umrechnung von Quaternionen in Mitsubishi-ABC

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in ABC Winkel in Grad kann wie folgt durchgeführt werden.

$$A = \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}$$
$$B = \text{asin}(2(wy - zx))\frac{180}{\pi}$$
$$C = \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi}$$

11.1.9 Universal Robots Posenformat

Das Posenformat, welches von Universal Robots verwendet wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung im Angle-Axis Format $V = (\ RX \quad RY \quad RZ\ )^T$. Der Rotationswin-

kel θ im Bogenmaß ist die Länge der Rotationsachse U.

$$V = \begin{pmatrix} RX \\ RY \\ RZ \end{pmatrix} = \begin{pmatrix} \theta u_x \\ \theta u_y \\ \theta u_z \end{pmatrix}$$

V wird als Rotationsvektor bezeichnet.

11.1.9.1 Umrechnung vom Angle-Axis Format in Quaternionen

Die Umrechnung von einem Rotationsvektor V in eine Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ kann wie folgt durchgeführt werden.

Zunächst wird der Winkel θ im Bogenmaß aus dem Rotationsvektor V gewonnen durch

$$\theta = \sqrt{RX^2 + RY^2 + RZ^2}.$$

Wenn $\theta = 0$, dann ist die Quaternion gleich $q = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$, sonst wird sie berechnet durch

$$\begin{aligned} x &= RX\frac{\sin(\theta/2)}{\theta}, \\ y &= RY\frac{\sin(\theta/2)}{\theta}, \\ z &= RZ\frac{\sin(\theta/2)}{\theta}, \\ w &= \cos(\theta/2). \end{aligned}$$

11.1.9.2 Umrechnung von Quaternionen ins Angle-Axis Format

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ mit $||q|| = 1$ in einen Rotationsvektor im Angle-Axis Format kann wie folgt durchgeführt werden.

Zunächst wird der Winkel θ im Bogenmaß aus dem Quaternion gewonnen durch

$$\theta = 2 \cdot \mathrm{acos}(w).$$

Wenn $\theta = 0$, dann ist der Rotationsvektor $V = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$, sonst wird er berechnet durch

$$\begin{aligned} RX &= \theta\frac{x}{\sqrt{1-w^2}}, \\ RY &= \theta\frac{y}{\sqrt{1-w^2}}, \\ RZ &= \theta\frac{z}{\sqrt{1-w^2}}. \end{aligned}$$

11.1.10 Yaskawa Posenformat

Das Posenformat, welches von Yaskawa Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung, welche durch drei Winkel in Grad gegeben ist. Rx rotiert um die x-Achse, Ry rotiert um die y-Achse und Rz rotiert um die z-Achse. Die Rotationsreihenfolge ist x-y-z und wird berechnet durch $r_z(Rz)r_y(Ry)r_x(Rx)$.

11.1.10.1 Umrechnung von Yaskawa Rx, Ry, Rz in Quaternionen

Zur Umrechnung von Rx, Ry, Rz Winkeln in Grad in eine Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ werden zunächst die Winkel ins Bogenmaß umgerechnet

$$\begin{aligned} X_r &= Rx\frac{\pi}{180}, \\ Y_r &= Ry\frac{\pi}{180}, \\ Z_r &= Rz\frac{\pi}{180}, \end{aligned}$$

und damit wird die Quaternion berechnet als

$$\begin{aligned} x &= \cos(Z_r/2)\cos(Y_r/2)\sin(X_r/2) - \sin(Z_r/2)\sin(Y_r/2)\cos(X_r/2), \\ y &= \cos(Z_r/2)\sin(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\cos(Y_r/2)\sin(X_r/2), \\ z &= \sin(Z_r/2)\cos(Y_r/2)\cos(X_r/2) - \cos(Z_r/2)\sin(Y_r/2)\sin(X_r/2), \\ w &= \cos(Z_r/2)\cos(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\sin(Y_r/2)\sin(X_r/2). \end{aligned}$$

11.1.10.2 Umrechnung von Quaternionen in Yaskawa Rx, Ry, Rz

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ mit $||q|| = 1$ in Rx, Ry, Rz Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned} Rx &= \mathsf{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi} \\ Ry &= \mathsf{asin}(2(wy - zx))\frac{180}{\pi} \\ Rz &= \mathsf{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi} \end{aligned}$$

HTTP Routing Table

/cad

GET /cad/gripper_elements, 296
GET /cad/gripper_elements/{id}, 296
PUT /cad/gripper_elements/{id}, 297
DELETE /cad/gripper_elements/{id}, 297

/datastreams

GET /datastreams, 321
GET /datastreams/{stream}, 322
PUT /datastreams/{stream}, 322
DELETE /datastreams/{stream}, 323

/generic_robot_interface

GET /generic_robot_interface/hec_configs, 378
GET /generic_robot_interface/hec_configs/{pipeline}, 378
GET /generic_robot_interface/jobs, 380
GET /generic_robot_interface/jobs/{job_id}, 381
PUT /generic_robot_interface/hec_configs/{pipeline}, 379
PUT /generic_robot_interface/jobs/{job_id}, 381
DELETE /generic_robot_interface/hec_configs/{pipeline}, 380
DELETE /generic_robot_interface/jobs/{job_id}, 382

/logs

GET /logs, 324
GET /logs/{log}, 325

/nodes

GET /nodes, 306
GET /nodes/{node}, 307
GET /nodes/{node}/services, 308
GET /nodes/{node}/services/{service}, 309
GET /nodes/{node}/status, 310
PUT /nodes/{node}/services/{service}, 309

/pipelines

GET /pipelines/{pipeline}/nodes, 311
GET /pipelines/{pipeline}/nodes/{node}, 312
GET /pipelines/{pipeline}/nodes/{node}/parameters, 313
GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}, 315
GET /pipelines/{pipeline}/nodes/{node}/services, 317
GET /pipelines/{pipeline}/nodes/{node}/services/{service}, 318
GET /pipelines/{pipeline}/nodes/{node}/status, 320
PUT /pipelines/{pipeline}/nodes/{node}/parameters, 314
PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}, 316
PUT /pipelines/{pipeline}/nodes/{node}/services/{service}, 319

/system

GET /system, 326
GET /system/backup, 327
GET /system/dns, 328
GET /system/license, 330
GET /system/max_power_test, 330
GET /system/network, 331
GET /system/network/settings, 332
GET /system/ntp, 333
GET /system/rollback, 334
GET /system/time, 335
GET /system/ui_lock, 336
GET /system/update, 337
POST /system/backup, 328
POST /system/license, 330
POST /system/max_power_test, 331
POST /system/ui_lock, 337
POST /system/update, 338
PUT /system/dns, 329
PUT /system/network/settings, 332
PUT /system/ntp, 333
PUT /system/reboot, 334
PUT /system/rollback, 335
PUT /system/time, 336
DELETE /system/ui_lock, 337

/templates

GET /templates/rc_boxpick, 166
GET /templates/rc_boxpick/{id}, 166
GET /templates/rc_silhouettematch, 219
GET /templates/rc_silhouettematch/{id}, 220
PUT /templates/rc_boxpick/{id}, 167
PUT /templates/rc_silhouettematch/{id}, 220
DELETE /templates/rc_boxpick/{id}, 167

```
DELETE /templates/rc_silhouettematch/{id},
        221
```

Stichwortverzeichnis

Sonderzeichen

3D Koordinaten, 50
    Disparitätsbild, 50
3D Modellierung, 50
3D-Modellierung, 54

A

Abmessungen
    Load Carrier, 265
    rc_visard, 18
Abteil
    Load Carrier, 268
AcquisitionAlternateFilter
    GenICam, 397
AcquisitionFrameRate
    GenICam, 394
AcquisitionMultiPartMode
    GenICam, 398
AdaptiveOut1
    automatische Belichtung, 35
aktive Partition, 410
Anschlussset, 412
AprilTag, **99**
    Posenschätzung, 101
    Rückgabecodes, 110
    Services, 105
    Tag-Wiedererkennung, 102
Aufnahmemodus
    Disparitätsbild, 42
automatisch
    Belichtung, 35
automatische Belichtung, 34, 35
    AdaptiveOut1, 35
    Normal, 35
    Out1High, 35

B

Backup
    Einstellungen, 408
BalanceRatio
    GenICam, 395
BalanceRatioSelector
    GenICam, 395
BalanceWhiteAuto
    GenICam, 395
Baseline
    GenICam, 399
Basisebene
    SilhouetteMatch, 182
Baumer
    IpConfigTool, 30
Belichtung
    automatisch, 34, 35
    HDR, 34
    manuell, 34
Belichtungsregion, 36
Belichtungszeit, 37
    Maximum, 35
Berechnung der Platzierungspose, 168
Beschleunigung, 55
Betriebsbedingungen, 20
Bewegungsunschärfe, 35
Bild
    Zeitstempel, 51, 401
Bildauflösung, 17
Bilder
    Download, 40
Bildmerkmale
    visuelle Odometrie, 61, 63
Bildrauschen, 35
Bildwiederholrate, 17
    GenICam, 394
    Kamera, 33
    Pose, 54, 55
    visuelle Odometrie, 61
Bin-Packing, 168
Bin-Picking, 110, 134
BoxPick, **134**
    bevorzugte TCP-Orientierung, 138
    Dimensionierung, 136
    Ebene, 280
    Füllstand, 82
    Griff, 137
    Load Carrier, 81, 265
    Objektmodelle, 134
    Parameter, 140
    RECTANGLE, 135
    Region of Interest, 273
    Rückgabecodes, 165
    services, 147
    Sortierung, 138
    Statuswerte, 147
    Template API, 165
    Template Download, 165
    Template löschen, 165
    Template Upload, 165

Textur, 135
TEXTURED_BOX, 135
Views, 135
Brennweite, 40
Brennweitenfaktor
GenICam, 399

C

CAD-Greiferelement API, 296
CAD-Greiferelement Download, 296
CAD-Greiferelement löschen, 296
CAD-Greiferelement Upload, 296
CAD-Modell, 19
Chunk-Daten
GenICam, 397
collision check, 245, 284
CollisionCheck, **245**
Rückgabecodes, 253
ComponentEnable
GenICam, 393
ComponentIDValue
GenICam, 393
ComponentSelector
GenICam, 393
Confidence
GenICam Bild-Stream, 401

D

Daten
IMU, 55
Inertialmesseinheit, 55
Datenmodell
REST-API, 338
Datenstrom
IMU, 55
Pose, 54
pose_rt, 54, 55
REST-API, 320
Sensordynamik, 54
Datentyp
REST-API, 338
Definition
Load Carrier, 265
DepthAcquisitionMode
GenICam, 399
DepthAcquisitionTrigger
GenICam, 399
DepthDoubleShot
GenICam, 399
DepthFill
GenICam, 400
DepthMaxDepth
GenICam, 400
DepthMaxDepthErr
GenICam, 400
DepthMinConf
GenICam, 400
DepthMinDepth
GenICam, 400
DepthQuality
GenICam, 399
DepthSeg
GenICam, 400
DepthSmooth
GenICam, 399
DepthStaticScene
GenICam, 399
Detektion
Load Carrier, 81
DHCP, **12**
DHCP, 29
Dimensionierung
Ebene, 136, 280
discovery GUI, 27
Disparität, 40, 50
GenICam Bild-Stream, 401
Disparitätsbereich
visuelle Odometrie, 63
Disparitätsbild, 40, 50
3D Koordinaten, 50
Aufnahmemodus, 42
double_shot, 45
Parameter, 40
Qualität, 43
smooth, 46
static_scene, 46
Timeout Belichtungsautomatik, 43
Web GUI, 40
Disparitätsfehler, 51
DNS, **12**
DOF, **12**
double_shot
Disparitätsbild, 45
GenICam, 399
Download
Bilder, 40
Einstellungen, 408
Logdateien, 409
Punktwolke, 51
Dynamik
REST-API, 320
Web GUI, 61
Dynamik-Datenstrom, 54

E

Ebene, 280
Dimensionierung, 136
Ebenenschätzung, 74
Measure, 74
Echtzeit-Pose, 54
Ecken
visuelle Odometrie, 61, 63
Eigenbewegung, 61
Einstellungen
Backup, 408
Download, 408

Upload, 408
Wiederherstellung, 408
eki, 383
Erkennung
Tag, 97
Error
GenICam Bild-Stream, 401
Ersatzteile, 413
Ethernet
Pin-Belegung, 22
ExposureAuto
GenICam, 394
ExposureRegionHeight
GenICam, 398
ExposureRegionOffsetX
GenICam, 398
ExposureRegionOffsetY
GenICam, 398
ExposureRegionWidth
GenICam, 398
ExposureTime
GenICam, 394
ExposureTimeAutoMax
GenICam, 398
externes Referenzkoordinatensystem
Hand-Auge-Kalibrierung, 222

F

Fehler, **51**
Hand-Auge-Kalibrierung, 232
Pose, 66
Feuchtigkeit, 20
Firmware
Mender, 409
Rollback, 411
Update, 409
Version, 409
FocalLengthFactor
GenICam, 399
fps, *siehe* Bildwiederholrate
Füllen, 46
GenICam, 400
Füllstand
BoxPick, 82
ItemPick, 82
LoadCarrier, 82
SilhouetteMatch, 82

G

Gain
GenICam, 394
gamma
Kamera, 34
Gehäusetemperatur
LED, 20
Generic Robot Interface, 359
GenICam, **12**
GenICam
AcquisitionAlternateFilter, 397
AcquisitionFrameRate, 394
AcquisitionMultiPartMode, 398
BalanceRatio, 395
BalanceRatioSelector, 395
BalanceWhiteAuto, 395
Baseline, 399
Bildwiederholrate, 394
Brennweitenfaktor, 399
Chunk-Daten, 397
ComponentEnable, 393
ComponentIDValue, 393
ComponentSelector, 393
DepthAcquisitionMode, 399
DepthAcquisitionTrigger, 399
DepthDoubleShot, 399
DepthFill, 400
DepthMaxDepth, 400
DepthMaxDepthErr, 400
DepthMinConf, 400
DepthMinDepth, 400
DepthQuality, 399
DepthSeg, 400
DepthSmooth, 399
DepthStaticScene, 399
double_shot, 399
ExposureAuto, 394
ExposureRegionHeight, 398
ExposureRegionOffsetX, 398
ExposureRegionOffsetY, 398
ExposureRegionWidth, 398
ExposureTime, 394
ExposureTimeAutoMax, 398
FocalLengthFactor, 399
Füllen, 400
Gain, 394
Height, 393
HeightMax, 394
LineSelector, 395
LineSource, 395
LineStatus, 395
LineStatusAll, 395
maximaler Abstand, 400
maximaler Fehler, 400
minimale Konfidenz, 400
minimaler Abstand, 400
Parametersperrung aufheben, 397
PixelFormat, 394, 401
PtpEnable, 396
Qualität, 399
RcExposureAutoAverageMax, 398
RcExposureAutoAverageMin, 398
Scan3dBaseline, 396
Scan3dCoordinateOffset, 396
Scan3dCoordinateScale, 396
Scan3dDistanceUnit, 396
Scan3dFocalLength, 396
Scan3dInvalidDataFlag, 396

Scan3dInvalidDataValue, 397
Scan3dOutputMode, 396
Scan3dPrincipalPointU, 396
Scan3dPrincipalPointV, 396
Segmentierung, 400
smooth, 399
static_scene, 399
System bereit, 397
Width, 393
WidthMax, 393
Zeitstempel, 401
GenICam Bild-Stream
Confidence, 401
Disparität, 401
Error, 401
Intensity, 401
IntensityCombined, 401
Umwandlung, 401
Geschwindigkeit
linear, 55
Winkel-, 55
GigE, **12**
GigE Vision, *siehe* GenICam
IP-Adresse, 30
GigE Vision, **12**
GPIO
Pin-Belegung, 22
GRI, 359
Griffberechnung, 110, 134
GripperDB, **284**
Rückgabecodes, 295
gRPC, 402
gRPC Bild-Stream
Umwandlung, 405

H

Hand-Auge-Kalibrierung
externes Referenzkoordinatensystem, 222
Fehler, 232
Kalibrierung, 226
Parameter, 233
Roboterkoordinatensystem, 222
Sensormontage, 223
Slot, 229
Height
GenICam, 393
HeightMax
GenICam, 394
Host-Name, 29

I

IMU, **12**
IMU
Daten, 55
Datenstrom, 55
Inertialmesseinheit, 61
inaktive Partition, 410, 411
Inertialmesseinheit
Daten, 55
IMU, 61
Innenvolumen
Load Carrier, 265
INS, **12**
Installation, 26
Intensity
GenICam Bild-Stream, 401
IntensityCombined
GenICam Bild-Stream, 401
IP, **12**
IP-Adresse, **12**
IP-Adresse, 28
GigE Vision, 30
IpConfigTool
Baumer, 30
IP 54, 20
ItemPick, **110**
bevorzugte TCP-Orientierung, 113
Füllstand, 82
Griff, 111
Load Carrier, 81, 265
Parameter, 115
Region of Interest, 273
Rückgabecodes, 133
services, 120
Sortierung, 112
Statuswerte, 119

K

Kabel, 21, 412
Kalibriermuster, 255
Kalibrierung
Hand-Auge-Kalibrierung, 226
Kamera, 254
Kamera-zu-IMU, 55
Rektifizierung, 40
Kalibrierung der Basisebene
SilhouetteMatch, 182
Kamera
Bildwiederholrate, 33
gamma, 34
Kalibrierung, 254
Parameter, 31
Posen-Datenstrom, 54
Web GUI, 31
Kamera-zu-IMU
Kalibrierung, 55
Transformation, 55
Kamerakalibrierung
Monokalibrierung, 259
Parameter, 260
Services, 261
Stereokalibrierung, 257
Keyframes, 61
visuelle Odometrie, 61, 63
Komponenten
rc_visard, 16

Konfidenz, **51**
Minimum, 47
Koordinatensysteme
Montage, 24
Sensordynamik, 55
Zustandsschätzung, 52
Korrespondenzen
visuelle Odometrie, 61
Kühlung, 20

L

LED, 26
Farben, 414
Gehäusetemperatur, 20
linear
Geschwindigkeit, 55
LineSelector
GenICam, 395
LineSource
GenICam, 395
LineStatus
GenICam, 395
LineStatusAll
GenICam, 395
Link-Local, **12**
Link-Local, 30
Load Carrier
Abmessungen, 265
Abteil, 268
BoxPick, 81, 265
Definition, 265
Detektion, 81
Innenvolumen, 265
ItemPick, 81, 265
Orientierungsprior, 265
Pose, 265
Rand, 265
SilhouetteMatch, 81, 265
Load Carrier Erkennung, 81
Load Carrier Modell, 265
LoadCarrier, **81**
Füllstand, 82
Parameter, 84
Rückgabecodes, 97
Services, 86
LoadCarrierDB, **265**
Rückgabecodes, 273
Services, 269
Logdateien
Download, 409
Logs
REST-API, 324

M

MAC-Adresse, **12**
MAC-Adresse, 29
manuelle Belichtung, 34, 37
maximaler Abstand, 45
GenICam, 400
maximaler Fehler, 47
GenICam, 400
Maximum
Belichtungszeit, 35
Tiefenfehler, 47
mDNS, **12**
Measure, **74**
Ebenenschätzung, 74
Parameter, 75
Rückgabecodes, 80
Services, 76
Mender
Firmware, 409
minimale Konfidenz, 47
GenICam, 400
minimaler Abstand, 44
GenICam, 400
Minimum
Konfidenz, 47
Monokalibrierung
Kamerakalibrierung, 259
Montage, 24

N

Netzteil, 413
Netzwerkkabel, 412
Netzwerkkonfiguration, 28
Neustart, 411
node
REST-API, 305
Normal
automatische Belichtung, 35
NTP, **12**
NTP
Synchronisierung, 407

O

Objekterkennung, 180
OPC UA, 383
Orientierungsprior
Load Carrier, 265
Out1High
automatische Belichtung, 35

P

Pack, **168**
bevorzugte TCP-Orientierung, 169
Packvolumen, 168
Platzierungspose, 168
Rückgabecodes, 180
Services, 173
Startpunkt, 169
Statuswerte, 172
Parameter
Disparitätsbild, 40
Hand-Auge-Kalibrierung, 233
Kamera, 31

Kamerakalibrierung, 260
REST-API, 306
Services, 39
visuelle Odometrie, 61
Parametersperrung aufheben
GenICam, 397
Pin-Belegung
Ethernet, 22
GPIO, 22
Stromzufuhr, 22
PixelFormat
GenICam, 394, 401
Pose
Bildwiederholrate, 54, 55
Datenstrom, 54
Fehler, 66
Load Carrier, 265
Zeitstempel, 53
pose_rt
Datenstrom, 54, 55
Posen-Datenstrom, 54, 55
Kamera, 54
Posenschätzung, *siehe* Zustandsschätzung
AprilTag, 101
QR-Code, 101
possible_jump
Sensordynamik, 55
SLAM, 55
PrimitiveDB, **280**
Rückgabecodes, 284
Services, 282
PTP, **13**
PTP
Synchronisierung, 396, 407
PtpEnable
GenICam, 396
Punktwolke, 50
Download, 51

Q

QR-Code, **98**
Posenschätzung, 101
Rückgabecodes, 110
Services, 105
Tag-Wiedererkennung, 102
Qualität
Disparitätsbild, 43
GenICam, 399
Quaternion
Rotation, 55

R

Rand
Load Carrier, 265
rc_dynamics, 355
rc_visard
Komponenten, 16
RcExposureAutoAverageMax
GenICam, 398
RcExposureAutoAverageMin
GenICam, 398
Rektifizierung, 40
REST-API, 302
Datenmodell, 338
Datenstrom, 320
Datentyp, 338
Dynamik, 320
Einstiegspunkt, 303
Logs, 324
node, 305
Parameter, 306
Services, 306
Statuswert, 305
System, 324
Version, 303
Roboterkoordinatensystem
Hand-Auge-Kalibrierung, 222
ROI, 273
RoiDB, **273**
Rückgabecodes, 280
Services, 275
Rollback
Firmware, 411
Rotation
Quaternion, 55
Rückgabecodes
AprilTag, 110
BoxPick, 165
CollisionCheck, 253
GripperDB, 295
ItemPick, 133
LoadCarrier, 97
LoadCarrierDB, 273
Measure, 80
Pack, 180
PrimitiveDB, 284
QR-Code, 110
RoiDB, 280
SilhouetteMatch, 218

S

Scan3dBaseline
GenICam, 396
Scan3dCoordinateOffset
GenICam, 396
Scan3dCoordinateScale
GenICam, 396
Scan3dDistanceUnit
GenICam, 396
Scan3dFocalLength
GenICam, 396
Scan3dInvalidDataFlag
GenICam, 396
Scan3dInvalidDataValue
GenICam, 397
Scan3dOutputMode

GenICam, 396
Scan3dPrinciplePointU
GenICam, 396
Scan3dPrinciplePointV
GenICam, 396
Schleifenschluss, 66
Schutzklasse, 20
SDK, **13**
Segmentierung, 47
GenICam, 400
Selbstkalibrierung, 254
Sensordatenfusion, 61
Sensordynamik
Datenstrom, 54
Koordinatensysteme, 55
possible_jump, 55
Services, 55
Sensormontage
Hand-Auge-Kalibrierung, 223
Seriennummer, 27, 29
Services
AprilTag, 105
Kamerakalibrierung, 261
Parameter, 39
QR-Code, 105
REST-API, 306
Sensordynamik, 55
Tagerkennung, 105
visuelle Odometrie, 64
Setzen
Zeit, 407
SGM, **13**
Silhouette, 180
SilhouetteMatch, **180**
Basisebene, 182
bevorzugte TCP-Orientierung, 185
Füllstand, 82
Greifpunkte, 183
Kalibrierung der Basisebene, 182
Kollisionsprüfung, 189
Load Carrier, 81, 265
Objekt-Template, 183
Objekterkennung, 186
Parameter, 190
Region of Interest, 183, 273
Rückgabecodes, 218
Services, 196
Sortierung, 186
Statuswerte, 196
Template API, 219
Template Download, 219
Template löschen, 219
Template Upload, 219
Simultane Lokalisierung und Kartierung, *siehe* SLAM
SLAM, **13**
SLAM, **66**
possible_jump, 55
Web GUI, 66
Slot
Hand-Auge-Kalibrierung, 229
smooth
Disparitätsbild, 46
GenICam, 399
Spezifikationen
rc_visard, 17
static_scene
Disparitätsbild, 46
GenICam, 399
Stativ, 24
Statuswert
REST-API, 305
Stereo Region of Interest, 44
Stereokalibrierung
Kamerakalibrierung, 257
Stromkabel, 412, 413
Stromversorgung, **20**
Stromzufuhr
Pin-Belegung, 22
Swagger UI, 351
Synchronisierung
NTP, 407
PTP, 396, 407
Zeit, 396, 406
System
REST-API, 324
System bereit
GenICam, 397

T

Tag-Wiedererkennung
AprilTag, 102
QR-Code, 102
Tagerkennung, **97**
Familien, 99
Posenschätzung, 101
Services, 105
Tag-Wiedererkennung, 102
TCP, **13**
Temperaturbereich, 20
Tiefenbild, 40, **50**, 50
Web GUI, 40
Tiefenfehler
Maximum, 47
Tiefenmessung, 74
Timeout Belichtungsautomatik
Disparitätsbild, 43
Transformation
Kamera-zu-IMU, 55
Translation, 55

U

UDP, **13**
Umwandlung
GenICam Bild-Stream, 401
gRPC Bild-Stream, 405

Update
    Firmware, 409
Upload
    Einstellungen, 408
URI, **13**
URL, **13**

V

Version
    Firmware, 409
    REST-API, 303
Verstärkungsfaktor, 35, 37
visuelle Odometrie, **61**
    Bildmerkmale, 61, 63
    Bildwiederholrate, 61
    Disparitätsbereich, 63
    Ecken, 61, 63
    Keyframes, 61, 63
    Korrespondenzen, 61
    Parameter, 61
    Services, 64
    Web GUI, 61
VO, *siehe* visuelle Odometrie

W

Web GUI, **299**
    Backup, 408
    Disparitätsbild, 40
    Dynamik, 61
    Kamera, 31
    Logs, 409
    SLAM, 66
    Tiefenbild, 40
    Update, 409
    visuelle Odometrie, 61
Weißabgleich, 37
Width
    GenICam, 393
WidthMax
    GenICam, 393
Wiederherstellung
    Einstellungen, 408
Winkel-
    Geschwindigkeit, 55

X

XYZ+Quaternion, **13**
XYZABC, **13**

Z

Zeit
    Setzen, 407
    Synchronisierung, 396, 406
Zeitstempel
    Bild, 51, 401
    GenICam, 401
    Pose, 53
Zurücksetzen, 27
Zustandsschätzung, 53
    Koordinatensysteme, 52

roboception

rc_visard 3D Stereosensor

MONTAGE- UND BETRIEBSANLEITUNG

Roboception GmbH

Kaflerstraße 2
81241 München
Deutschland

info@roboception.de
www.roboception.com

Tutorials: https://tutorials.roboception.com
GitHub: https://github.com/roboception
Dokumentation: https://doc.rc-visard.com
https://doc.rc-viscore.com
https://doc.rc-cube.com
https://doc.rc-randomdot.com
Shop: https://roboception.com/shop

Für Kundensupport kontaktieren Sie

+49 89 889 50 790
(09:00-17:00 CET)

support@roboception.de

