

# roboception

Roboception GmbH | April 2022

## 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** 22.04.5 29.04.2022

Gültig für *rc\_visard* Firmware 22.04.x

### HERSTELLER

#### **Roboception GmbH**

Kaflerstraße 2

81241 München

Deutschland

**KUNDENSUPPORT:** [support@roboception.de](mailto: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

<b>1</b>	<b>Einführung</b>	<b>5</b>
1.1	Überblick	5
1.2	Garantie	7
1.3	Schnittstellen, Zulassungen und Normen	8
1.3.1	Schnittstellen	8
1.3.2	Zulassungen	8
1.3.3	Normen	8
1.4	Informationen zur Entsorgung	9
1.5	Glossar	11
<b>2</b>	<b>Sicherheit</b>	<b>13</b>
2.1	Allgemeine Sicherheitshinweise	13
2.2	Bestimmungsgemäße Verwendung	14
<b>3</b>	<b>Hardware-Spezifikation</b>	<b>15</b>
3.1	Lieferumfang	15
3.2	Technische Spezifikation	16
3.3	Umwelt- und Betriebsbedingungen	19
3.4	Spezifikationen für die Stromversorgung	19
3.5	Verkabelung	20
3.6	Mechanische Schnittstelle	23
3.7	Koordinatensysteme	23
<b>4</b>	<b>Installation</b>	<b>25</b>
4.1	Softwarelizenz	25
4.2	Einschalten	25
4.3	Aufspüren von <i>rc_visard</i> -Geräten	26
4.3.1	Zurücksetzen der Konfiguration	26
4.4	Netzwerkkonfiguration	27
4.4.1	Host-Name	28
4.4.2	Automatische Konfiguration (werkseitige Voreinstellung)	28
4.4.3	Manuelle Konfiguration	29
<b>5</b>	<b>Messprinzipien</b>	<b>30</b>
5.1	Stereovision	30
5.2	Sensordynamik	31
<b>6</b>	<b>Softwaremodule</b>	<b>33</b>
6.1	3D-Kamera-Module	33
6.1.1	Kamera	33
6.1.2	Stereo-Matching	42
6.2	Navigationsmodule	53
6.2.1	Sensordynamik	54
6.2.2	Visuelle Odometrie	62
6.2.3	Stereo-INS	67
6.2.4	SLAM	68

6.3	Detektionsmodule	75
6.3.1	LoadCarrier	75
6.3.2	TagDetect	88
6.3.3	ItemPick und BoxPick	101
6.3.4	SilhouetteMatch	126
6.4	Konfigurationsmodule	159
6.4.1	Hand-Auge-Kalibrierung	159
6.4.2	CollisionCheck	181
6.4.3	Kamerakalibrierung	189
6.4.4	IOControl und Projektor-Kontrolle	196
6.5	Datenbankmodule	200
6.5.1	LoadCarrierDB	200
6.5.2	RoiDB	208
6.5.3	GripperDB	215
<b>7</b>	<b>Schnittstellen</b>	<b>223</b>
7.1	Web GUI	223
7.1.1	Zugriff auf die Web GUI	223
7.1.2	Kennenlernen der Web GUI	224
7.1.3	Herunterladen von Kamerabildern	225
7.1.4	Herunterladen von Tiefenbildern und Punktwolken	225
7.2	GigE Vision 2.0/GenICam-Schnittstelle	226
7.2.1	GigE Vision Ports	226
7.2.2	Wichtige Parameter der GenICam-Schnittstelle	226
7.2.3	Wichtige Standardparameter der GenICam-Schnittstelle	226
7.2.4	Besondere Parameter der GenICam-Schnittstelle des <i>rc_visard</i>	231
7.2.5	Chunk-Daten	234
7.2.6	Verfügbare Bild-Streams	234
7.2.7	Umwandlung von Bild-Streams	235
7.3	REST-API-Schnittstelle	236
7.3.1	Allgemeine Struktur der Programmierschnittstelle (API)	236
7.3.2	Migration von API Version 1	238
7.3.3	Verfügbare Ressourcen und Anfragen	239
7.3.4	Datentyp-Definitionen	267
7.3.5	Swagger UI	277
7.4	Die <i>rc_dynamics</i> -Schnittstelle	281
7.4.1	Starten/Stoppen der Dynamik-Zustandsschätzungen	281
7.4.2	Konfiguration von Datenströmen	282
7.4.3	Datenstromprotokoll	283
7.4.4	Die <i>rc_dynamics</i> -Programmierschnittstelle	284
7.5	KUKA Ethernet KRL Schnittstelle	285
7.5.1	Konfiguration der Ethernet-Verbindung	285
7.5.2	Allgemeine XML-Struktur	286
7.5.3	Services	287
7.5.4	Parameter	290
7.5.5	Migration zu Firmware version 22.01	292
7.5.6	Beispielanwendungen	292
7.6	Zeitsynchronisierung	292
7.6.1	NTP	293
7.6.2	PTP	293
<b>8</b>	<b>Wartung</b>	<b>294</b>
8.1	Reinigung der Kameralinsen	294
8.2	Kamerakalibrierung	294
8.3	Backup der Einstellungen	294
8.4	Aktualisierung der Firmware	295
8.5	Wiederherstellung der vorherigen Firmware-Version	296
8.6	Neustart des <i>rc_visard</i>	297

8.7	Aktualisierung der Softwarelizenz . . . . .	297
8.8	Download der Logdateien . . . . .	297
<b>9</b>	<b>Zubehör</b>	<b>298</b>
9.1	Anschlussset . . . . .	298
9.2	Verkabelung . . . . .	298
9.2.1	Ethernet-Anschluss . . . . .	298
9.2.2	Stromanschluss . . . . .	299
9.2.3	Netzteile . . . . .	299
9.3	Ersatzteile . . . . .	299
<b>10</b>	<b>Fehlerbehebung</b>	<b>300</b>
10.1	LED-Farben . . . . .	300
10.2	Probleme mit der Hardware . . . . .	300
10.3	Probleme mit der Konnektivität . . . . .	301
10.4	Probleme mit den Kamerabildern . . . . .	302
10.5	Probleme mit Tiefen-/Disparitäts-, Fehler- oder Konfidenzbildern . . . . .	303
10.6	Probleme mit der Zustandsschätzung . . . . .	304
10.7	Probleme mit GigE Vision/GenICam . . . . .	304
10.8	Probleme mit EKI (KUKA Ethernet KRL Interface) . . . . .	304
<b>11</b>	<b>Kontakt</b>	<b>305</b>
11.1	Support . . . . .	305
11.2	Downloads . . . . .	305
11.3	Adresse . . . . .	305
<b>12</b>	<b>Anhang</b>	<b>306</b>
12.1	Formate für Posendaten . . . . .	306
12.1.1	Rotationsmatrix und Translationsvektor . . . . .	307
12.1.2	ABB Posenformat . . . . .	307
12.1.3	FANUC XYZ-WPR Format . . . . .	308
12.1.4	Kawasaki XYZ-OAT Format . . . . .	308
12.1.5	KUKA XYZ-ABC Format . . . . .	309
12.1.6	Mitsubishi XYZ-ABC Format . . . . .	310
12.1.7	Universal Robots Posenformat . . . . .	310
12.1.8	Fruitcore HORST Posenformat . . . . .	311
	<b>HTTP Routing Table</b>	<b>312</b>
	<b>Stichwortverzeichnis</b>	<b>313</b>

# 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

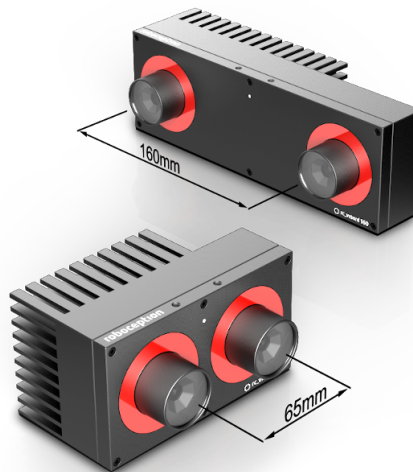
Der 3D-Sensor *rc\_visard* ist eine selbstregistrierende Stereokamera, die über eine integrierte Recheneinheit verfügt und der Schutzklasse IP 54 angehört.

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 GenICam-Schnittstelle 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>i</i>CAM

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



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:



EG-Konformitätserklärung



NRTL-Zertifizierung durch den TÜV Süd



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.

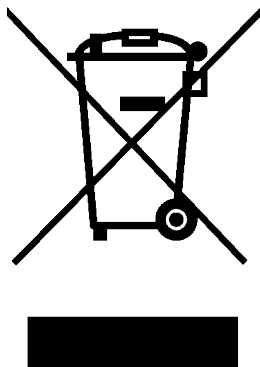
### 1.3.3 Normen

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](#) (Section 11) 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 inertielle 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 inertieller 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 6.2.1.2) über die *rc\_dynamics-Schnittstelle* (Abschnitt 7.4). 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 weist.
- 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 12.1.1).
- XYZABC** Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *KUKA XYZ-ABC Format* (Abschnitt 12.1.5).

## 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 Medizinprodukterichtlinie.

### 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 für die Datenerfassung (z.B. Kamerabilder, Disparitätsbilder und Eigenbewegung) in stationären oder mobilen Robotik-Anwendungen bestimmt. Der *rc\_visard* kann dabei auf einem Roboter, einer automatischen Maschine, einer mobilen Plattform oder einer stationären Vorrichtung montiert sein. Er eignet sich zudem für die Datenerfassung in anderen Anwendungen.

**Warnung:** Der *rc\_visard* ist **NICHT** für sicherheitskritische Anwendungen bestimmt.

Der vom *rc\_visard* verwendete Schnittstellenstandard GigE Vision® unterstützt weder Authentifizierung noch Verschlüsselung. Alle von diesem und an dieses Gerät gesandten Daten werden ohne Authentifizierung und Verschlüsselung übermittelt und könnten daher von einem Dritten abgefangen oder manipuliert werden. Es liegt in der Verantwortung des Bedieners, den *rc\_visard* nur an ein gesichertes internes Netzwerk anzuschließen.

**Warnung:** Der *rc\_visard* muss an gesicherte interne Netzwerke angeschlossen werden.

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 Automationsicherheit 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 7.1) oder über die Roboception-Homepage <http://www.roboception.com/documentation> aufrufen.

**Bemerkung:** Folgende Elemente sind, sofern nicht anders angegeben, NICHT im Lieferumfang enthalten:

- Kupplungen, Adapter, Halterungen,
- Netzteil, Kabel und Sicherungen,
- Netzkabel.

In Abschnitt *Zubehör* (Abschnitt 9) ist angegeben, welche Kabelanbieter empfohlen werden.

Für den *rc\_visard* ist ein Anschlussset verfügbar. Dieses Set umfasst das M12/RJ45-Netzkabel, ein 24-V-Netzteil und einen DC/M12-Adapter. Für nähere Informationen siehe *Zubehör* (Abschnitt 9).

**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.



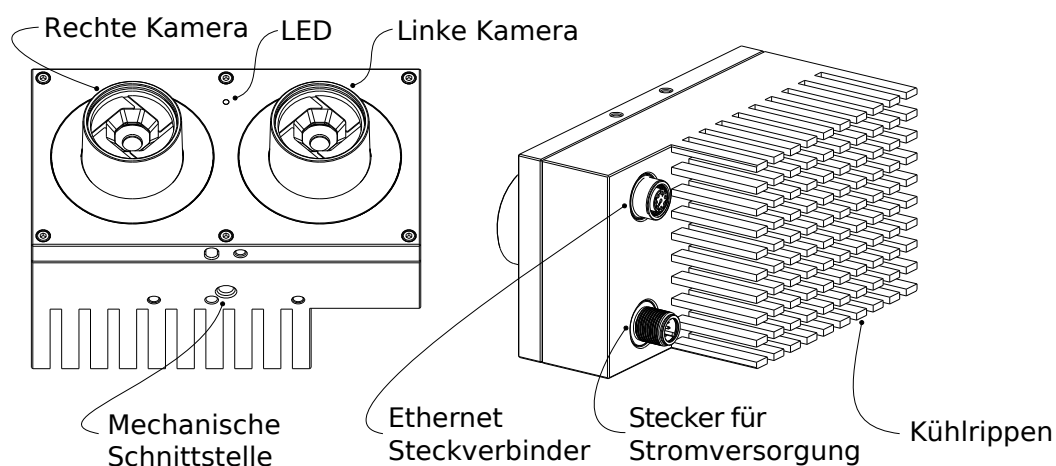


Abb. 3.1: Beschreibung der Bauteile

## 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.

	<i>rc_visard 65 / rc_visard 160</i>
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 (Auflösung)	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

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

Der *rc\_visard* kann für zusätzliche Funktionalitäten mit On-Board-Softwaremodulen, wie z.B. SLAM, ausgestattet werden. Diese Softwaremodule können bei Roboception bestellt werden und benötigen ein Lizenz-Update.

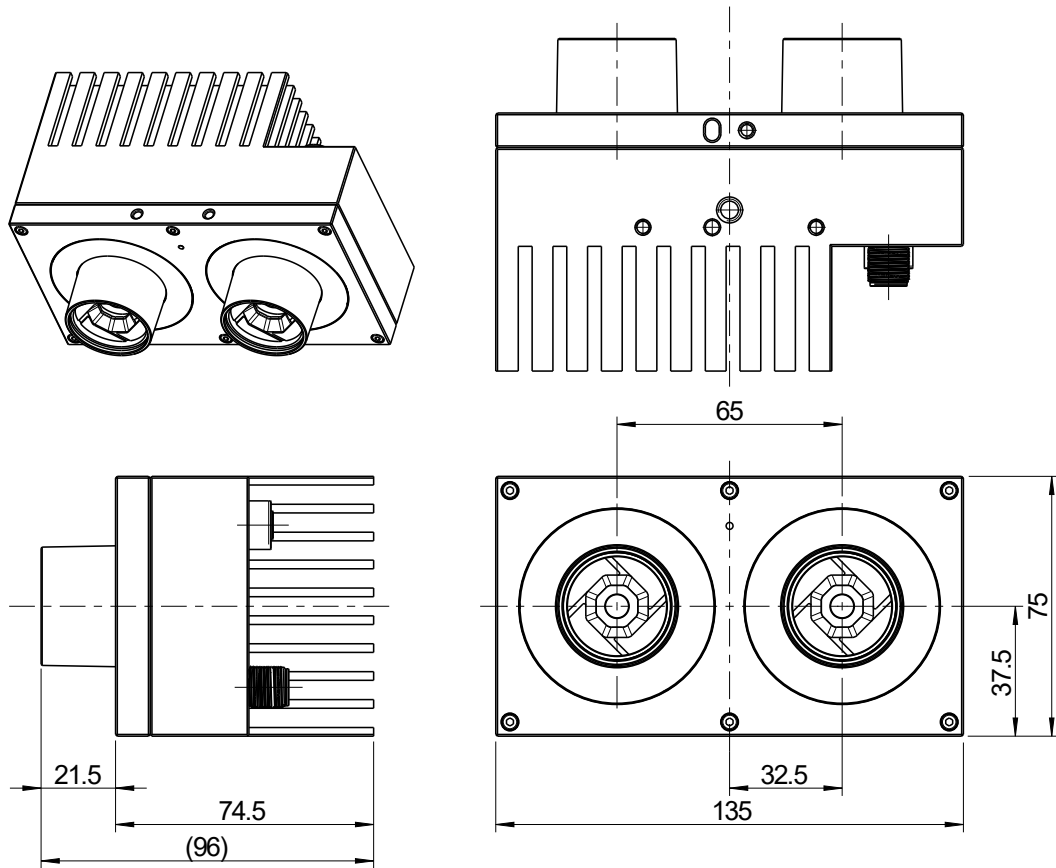


Abb. 3.2: Abmessungen des *rc\_visard 65*

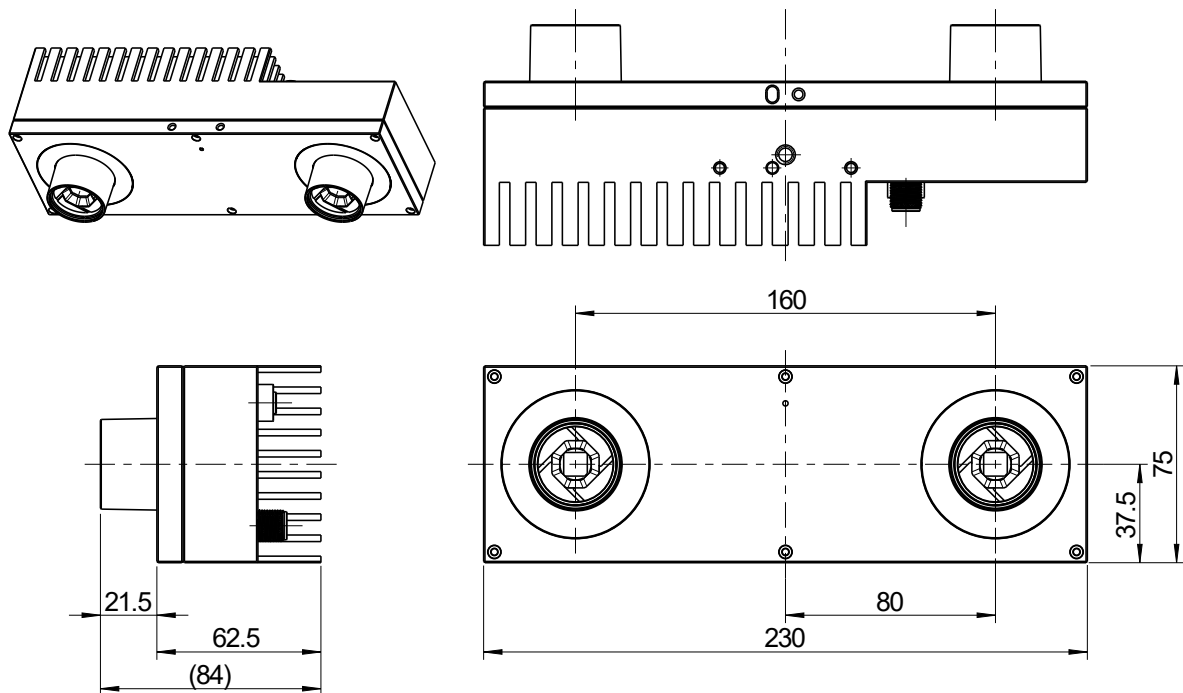


Abb. 3.3: Abmessungen des *rc\_visard 160*

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:  $2.76 \frac{\text{g}}{\text{cm}^3}$ ), 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

	<i>rc_visard</i> 65 / <i>rc_visard</i> 160
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	<ul style="list-style-type: none"> <li>• Von korrosiven Flüssigkeiten oder Gasen fernhalten.</li> <li>• Von explosiven Flüssigkeiten oder Gasen fernhalten.</li> <li>• Von starken elektromagnetischen Störungen fernhalten.</li> </ul>

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

	<i>Minimum</i>	<i>Bemessungswert</i>	<i>Maximum</i>
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 <i>Normen</i> (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 9) 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 wird am Beispiel des *rc\_visard* 65 in *Abb. 3.4* dargestellt.

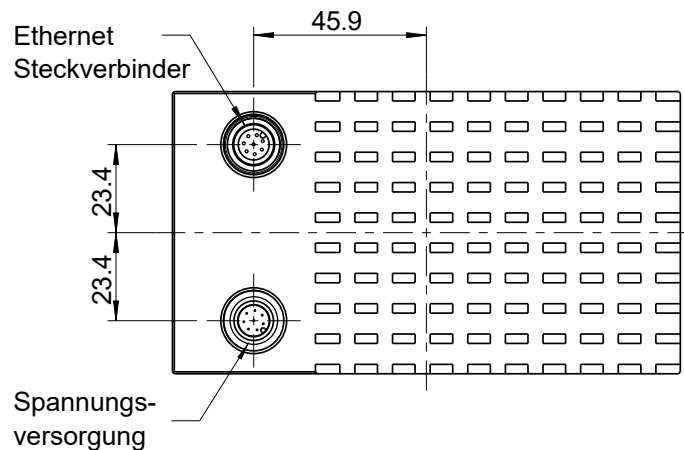


Abb. 3.4: Lage der elektrischen Anschlüsse des *rc\_visard* 65 für die Ethernetverbindung (oben) und die Stromversorgung (unten)

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

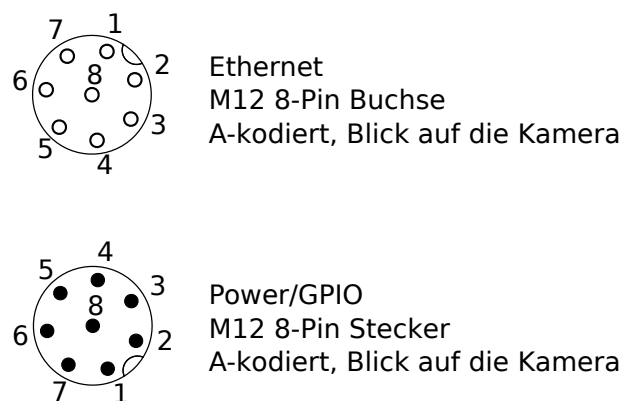


Abb. 3.5: Steckerbelegung für den Strom- und Ethernetanschluss

Die Steckerbelegung für den Ethernetanschluss ist in [Abb. 3.6](#) angegeben.

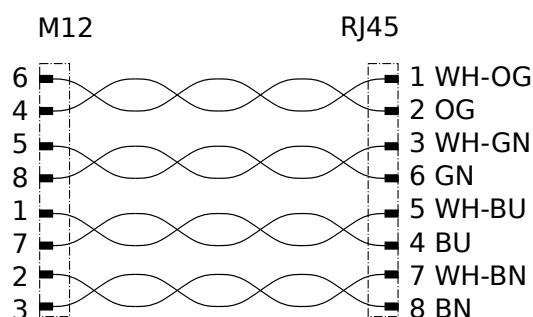


Abb. 3.6: Steckerbelegung für die M12/Ethernet-Verkabelung

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 optionale IOControl-Modul kontrolliert werden (*IOControl und Projektor-Kontrolle*, Abschnitt 6.4.4). 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-Schalterschema und die zugehörigen Spezifikationen sind in [Abb. 3.7](#) angegeben. Die maximale Spannung für *GPIO Eingang* und *GPIO Vcc* beträgt 30 V.

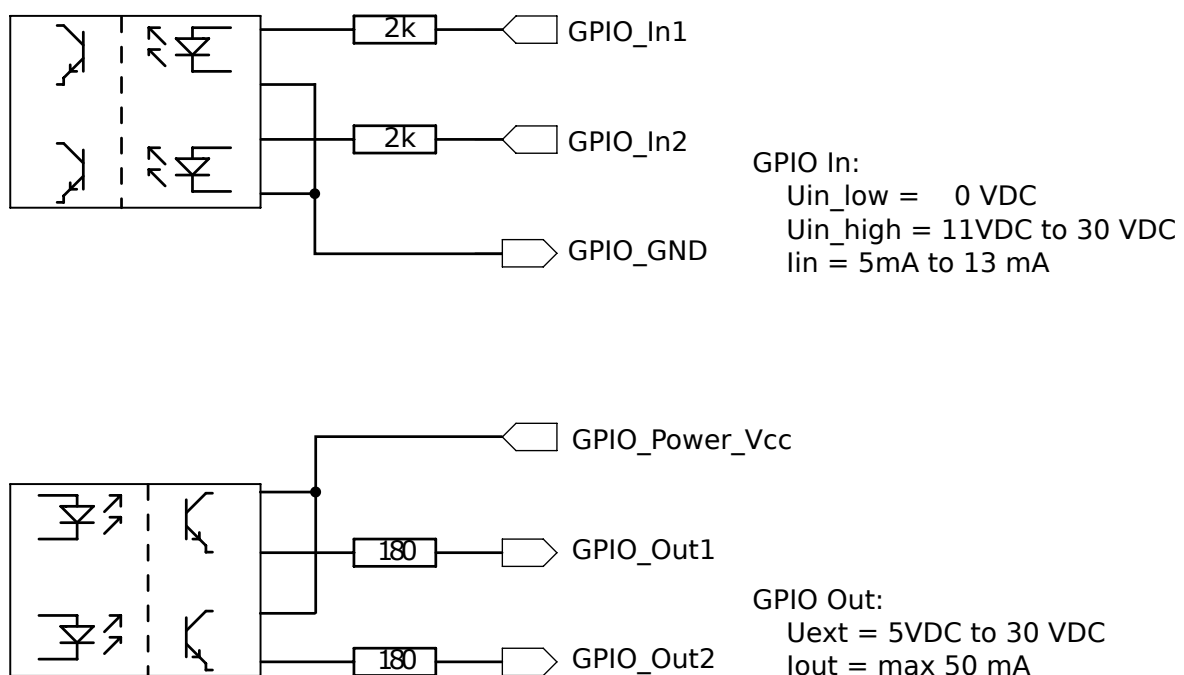


Abb. 3.7: GPIO-Schalterschema und zugehörige Spezifikationen: Keine Signale über 30 V anschließen!

**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.

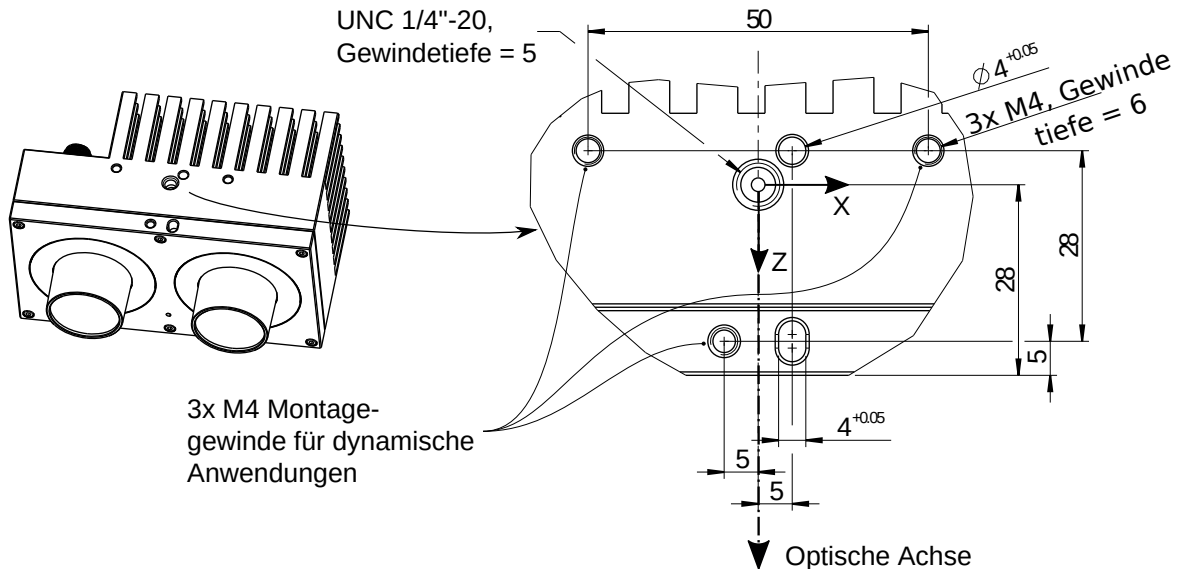


Abb. 3.8: Montagepunkt für den Anschluss des *rc\_visard* an Roboter oder andere Vorrichtungen

Zur Fehlerbehebung sowie zu Konfigurationszwecken kann der Sensor über die am Koordinatensprung 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* 65 wird auf dem nächsten Bild gezeigt.

**Bemerkung:** Der korrekte Versatz zwischen dem Sensor-/Kamerakoordinatensystem und einem Roboterkoordinatensystem kann über die [Hand-Auge Kalibrierung](#) (Abschnitt 6.4.1) bestimmt werden.

Das Montagepunkt-Koordinatensystem für beide *rc\_visard*-Geräte sitzt an der Unterseite, zentriert auf dem Gewinde, wobei die Ausrichtung der des Sensor-Koordinatensystems entspricht. [Abb. 3.9](#) zeigt den ungefähren Versatz.



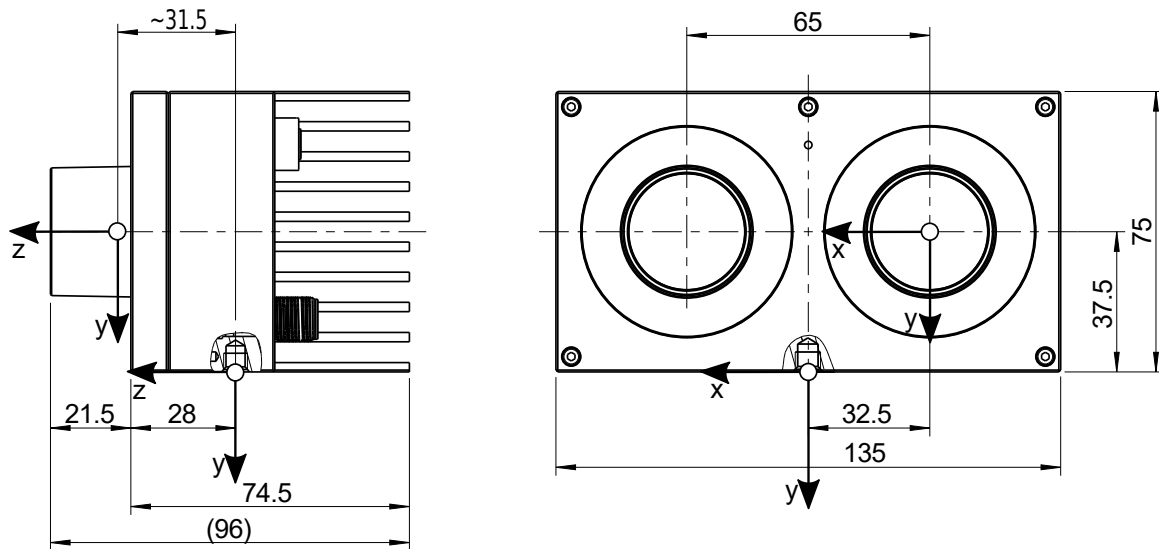


Abb. 3.9: Ungefähre Position des Sensor-/Kamerakoordinatensystems (in der linken Kameralinse) und des Montagepunkt-Koordinatensystems (am Stativgewinde) für den *rc\_visard 65*

Die ungefähre Position des Sensor-/Kamerakoordinatensystems und des Montagepunkt-Koordinatensystems für den *rc\_visard 160* ist in [Abb. 3.10](#) angegeben.

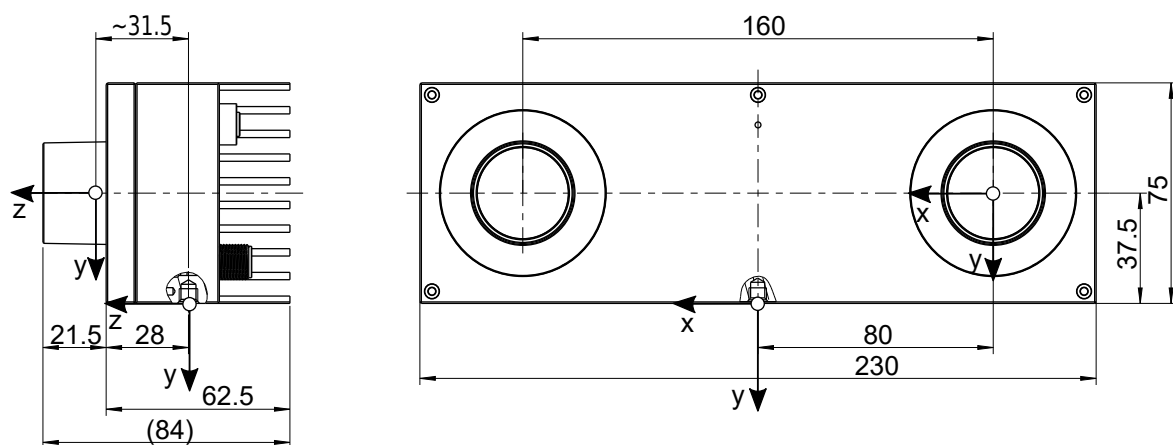


Abb. 3.10: Ungefähre Position des Sensor-/Kamerakoordinatensystems (in der linken Kameralinse) und des Montagepunkt-Koordinatensystems (am Stativgewinde) für den *rc\_visard 160*

## 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 8.7) 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 7.1).

### 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 Netzkabel 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 10.1).

## 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 von der Roboception-Homepage heruntergeladen werden: <http://www.roboception.com/download>. Dieses Tool besteht für Windows 7 und Windows 10 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.

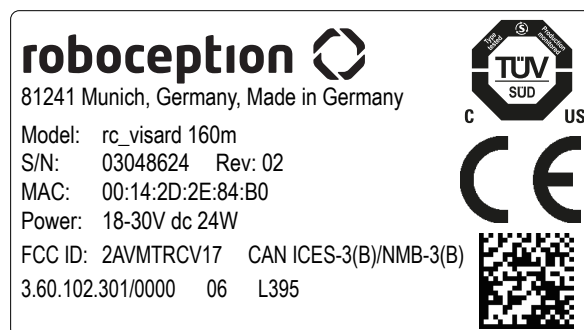


Abb. 4.1: Typenschild des *rc\_visard* mit Modellart, Seriennummer und MAC-Adresse

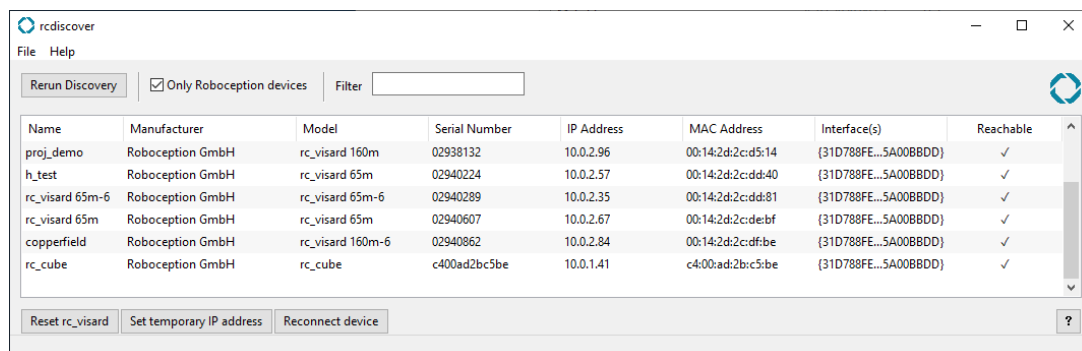


Abb. 4.2: *rcdiscover-gui*-Tool zum Aufspüren angeschlossener GigE Vision®-Geräte

Wurde das Gerät erfolgreich gefunden, öffnet sich nach einem Doppelklick auf den Geräteeintrag die *Web GUI* (Abschnitt 7.1) 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.

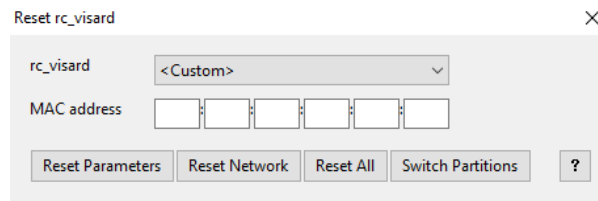


Abb. 4.3: Reset-Dialog des rcdiscover-gui-Tools

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 7.1) 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 8.5).

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

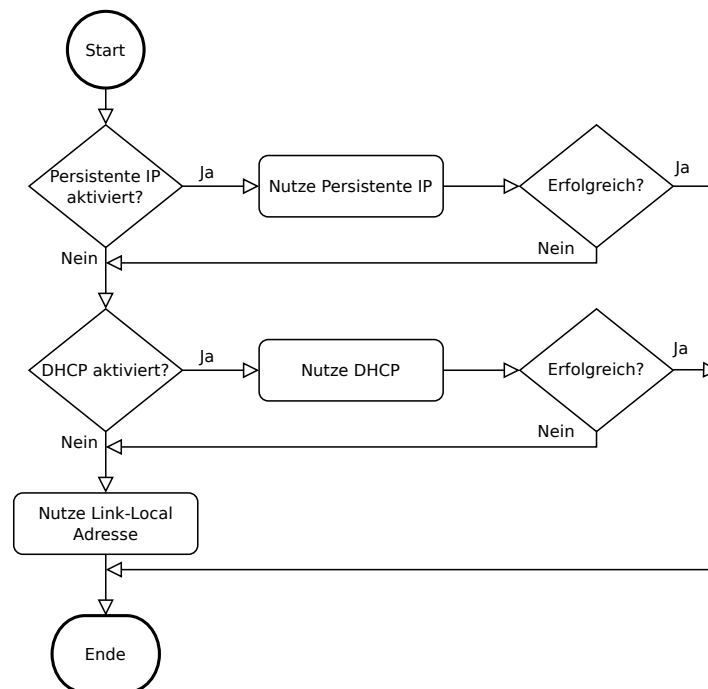


Abb. 4.4: Flussdiagramm für die Auswahl der IP-Konfigurationsmethoden des *rc\_visard*

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 7.1)
- 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 (werkseitige Voreinstellung)

Für die Zuweisung von IP-Adressen wird bevorzugt auf *DHCP* zugegriffen. Ist *DHCP* (werkseitige 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 Netzkabels 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 andernfalls möglicherweise nicht auf den *rc\_visard* zugegriffen werden kann. Dieses Problem lässt sich vermeiden, indem die unter *Automatische Konfiguration (werkseitige 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 Messprinzipien

Der *rc\_visard* ist eine selbstregistrierende 3D-Kamera. Er erstellt rektifizierte Bilder sowie Disparitäts-, Konfidenz- und Fehlerbilder, mit denen sich die Tiefenwerte der Aufnahme berechnen lassen. Zusätzlich werden intern gemessene Beschleunigungs- und Drehraten mit Bewegungsschätzungen aus den Kamerabildern kombiniert, um Echtzeit-Schätzungen der aktuellen Pose (Position und Orientierung), Geschwindigkeit und Beschleunigung des Sensors anbieten zu können.

Im Folgenden sind die zugrunde liegenden Messprinzipien genauer dargestellt.

### 5.1 Stereovision

Bei der *Stereovision* werden 3D-Informationen gewonnen, indem zwei aus verschiedenen Blickwinkeln aufgenommene Bilder miteinander verglichen werden. Das zugrunde liegende Prinzip ist darin begründet, dass Objektpunkte je nach Abstand vom Kamerapaar an unterschiedlichen Stellen in beiden Kameras erscheinen. Während sehr weit entfernte Objektpunkte in beiden Kamerabildern etwa an der gleichen Position erscheinen, liegen sehr nahe Objektpunkte an unterschiedlichen Stellen im linken und rechten Kamerabild. Dieser Versatz der Objektpunkte in beiden Kamerabildern wird auch „Disparität“ genannt. Je größer die Disparität, desto näher ist das Objekt der Kamera. Das Prinzip der Stereovision wird in [Abb. 5.1](#) genauer dargestellt.

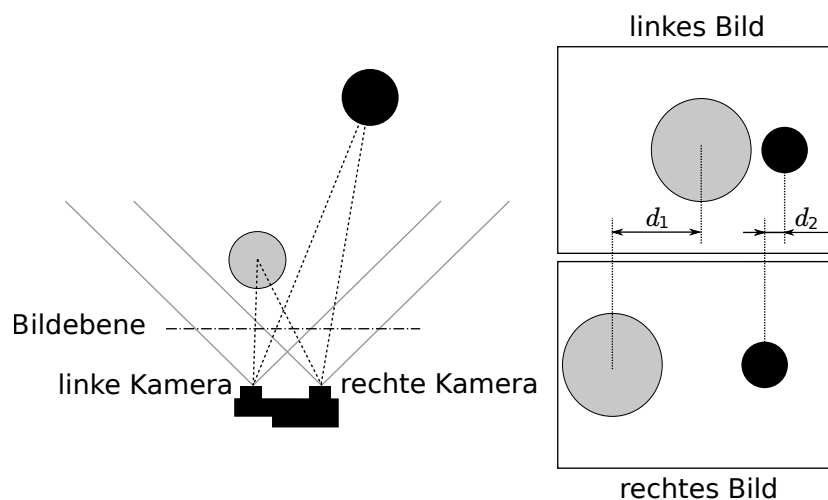


Abb. 5.1: Schematische Darstellung des Prinzips der Stereovision: Die Disparität  $d_2$  des weiter entfernten (schwarzen) Objekts ist kleiner als die Disparität  $d_1$  des nahe liegenden (grauen) Objekts.

Stereovision beruht auf passiver Wahrnehmung. Dies bedeutet, dass keine Licht- oder sonstigen Signale zur Distanzmessung ausgesandt werden, sondern nur das von der Umgebung ausgehende oder reflektierte Licht genutzt wird. Dadurch können die Roboception-*rc\_visard*-Sensoren sowohl im Innen- als auch im Außenbereich eingesetzt werden. Zudem können problemlos mehrere Sensoren störungsfrei zusammen auf engem Raum betrieben werden.

Um die 3D-Informationen berechnen zu können, muss der Stereo-Matching-Algorithmus die zusammengehörenden Objektpunkte im linken und rechten Kamerabild finden. Hierfür bedient er sich der Bildtextur, d.h. der durch Muster oder Oberflächenstrukturen der Objekte verursachten Schwankungen in der Bildintensität. Das Stereo-Matching-Verfahren kann bei Oberflächen ohne jede Textur, wie z.B. bei glatten, weißen Wänden, keine Werte liefern. Das Stereo-Matching-Verfahren, das der *rc\_visard* verwendet, ist *SGM (Semi-Global Matching)*, welches selbst bei feineren Strukturen den bestmöglichen Kompromiss aus Laufzeit und Genauigkeit bietet.

Für die Berechnung der 3D-Informationen werden folgende Softwaremodule benötigt:

- *Kamera*: Dieses Modul dient dazu, synchronisierte Bildpaare aufzunehmen und diese in Bilder umzuwandeln, die weitestgehend den Aufnahmen einer idealen Kamera entsprechen (Rektifizierung).
- *Stereo-Matching*: Dieses Modul errechnet mithilfe des Stereo-Matching-Verfahrens *SGM* die Disparitäten der rektifizierten Stereo-Bildpaare (Abschnitt 6.1.2).

Für das Stereo-Matching-Verfahren müssen die Positionen der linken und rechten Kamera sowie ihre Ausrichtung zueinander genau bekannt sein. Dies wird durch Kalibrierung erreicht. Die Kameras des *rc\_visard* werden bereits im Werk vorkalibriert. Hat sich der *rc\_visard* jedoch, beispielsweise während des Transports, dekalibriert, muss die Stereokamera neu kalibriert werden.

- *Kamerakalibrierung*: Mit diesem Modul kann der Benutzer die Stereokamera des *rc\_visard* neu kalibrieren (Abschnitt 6.4.3).

## 5.2 Sensordynamik

Neben 3D-Informationen zu einer Szene kann der *rc\_visard* auch Echtzeit-Schätzungen seiner *Eigenbewegung* oder seines *dynamischen Zustands* bereitstellen. Hierfür misst er seine aktuelle Pose, d.h. seine Position und Orientierung in Bezug auf ein Referenzkoordinatensystem, sowie seine Geschwindigkeit und Beschleunigung. Für diese Messungen werden die Messungen aus der SVO (stereobasierte visuelle Odometrie) mit den Werten des integrierten inertialen Messsystems (*IMU*) kombiniert. Diese Kombination wird auch als visuelles Trägheitsnavigationssystem (*VINS*) bezeichnet.

Die visuelle Odometrie verfolgt die Bewegung charakteristischer Punkte in Kamerabildern, um auf dieser Grundlage die Kamerabewegung abzuschätzen. Objektpunkte werden je nach Blickrichtung der Kamera auf verschiedene Pixel projiziert. Die 3D-Koordinaten jedes dieser Objektpunkte lassen sich über Stereo-Matching der Punktprojektionen im linken und rechten Kamerabild errechnen. So werden für zwei unterschiedliche Betrachtungspositionen A und B zwei zugehörige 3D-Punktwolken errechnet. Nimmt man eine statische Umgebung an, entspricht die Bewegung, die die eine Punktwolke in die andere Punktwolke transformiert, der Kamerabewegung. Das Prinzip wird in [Abb. 5.2](#) für einen vereinfachten 2D-Anwendungsfall dargestellt.



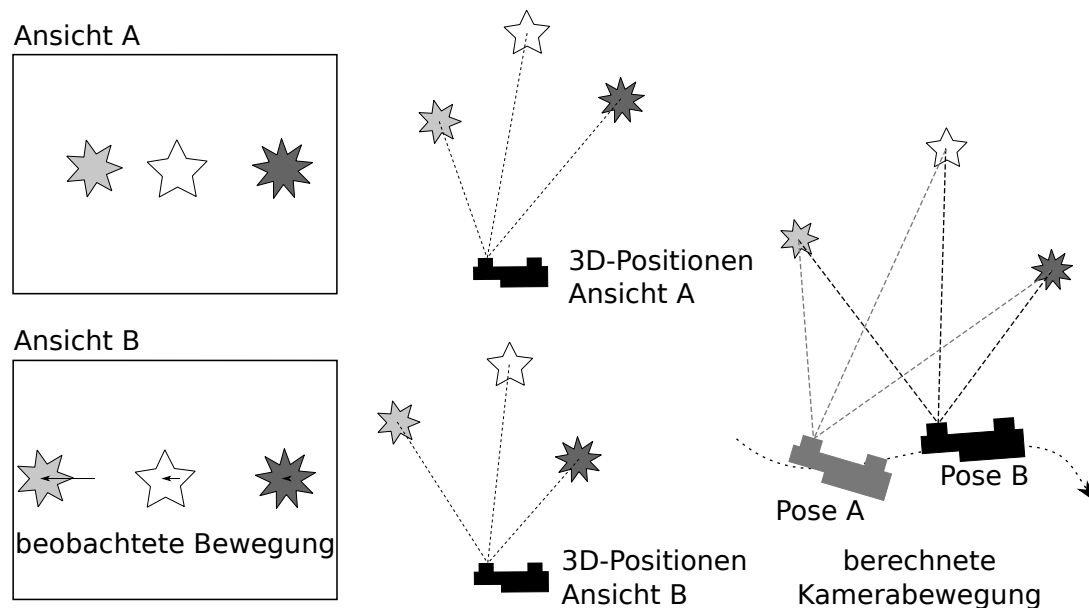


Abb. 5.2: Vereinfachte schematische Darstellung der stereobasierten visuellen Odometrie für 2D-Bewegungen: Die Kamerabewegung wird auf Grundlage der beobachteten Bewegung charakteristischer Bildpunkte berechnet.

Da die visuelle Odometrie auf eine gute Qualität der Bilddaten angewiesen ist, verschlechtern sich die Bewegungsschätzungen, wenn Bilder verschwommen oder schlecht beleuchtet sind. Zudem ist die Frequenz der visuellen Odometrie zu gering für Regelungsanwendungen. Aus diesem Grund verfügt der *rc\_visard* über eine integrierte inertielle Messeinheit (IMU), die Beschleunigungen und Winkelgeschwindigkeiten misst, die auftreten wenn sich der *rc\_visard* bewegt. Das System misst zudem die Erdbeschleunigung, was die globale Orientierung entlang der vertikalen Achse ermöglicht. Außerdem werden für IMU-Messungen hohe Messraten von 200 Hz genutzt. Lineargeschwindigkeit, Position und Orientierung des *rc\_visard* lassen sich durch Aufintegrieren der IMU-Messungen errechnen. Die Integrationsergebnisse führen jedoch im Laufe der Zeit zu einem Abdriften. Um eine akkurate, robuste und hochfrequente Schätzung der aktuellen Position, Orientierung, Geschwindigkeit und Beschleunigung des *rc\_visard* bereitzustellen, die in einem Regelkreis verwendet werden kann, kombiniert der *rc\_visard* die akkuraten, niederfrequenten und manchmal unzuverlässigen Odometriemessungen mit den robusten hochfrequenten IMU-Messungen.

Für die Berechnung der Zustandsschätzungen sind neben dem Stereokamera-Modul und dem Kalibriermodul auch folgende *rc\_visard*-Softwaremodule erforderlich:

- **Sensordynamik:** Mit diesem Modul lassen sich die für die einzelnen Submodule benötigten Schätzungen starten und stoppen und verwalten (Abschnitt 6.2.1).
  - **Visuelle Odometrie:** Dieses Modul errechnet eine Bewegungsschätzung anhand von Kamerabildern (Abschnitt 6.2.2).
  - **Stereo-INS:** Dieses Modul kombiniert die von der visuellen Odometrie bereitgestellten Bewegungsschätzungen mit den Messungen der integrierten IMU, um so hochfrequente Echtzeit-Lageschätzungen bereitstellen zu können (Abschnitt 6.2.3).
  - **SLAM:** Dieses Modul, das optional für den *rc\_visard* erhältlich ist, erstellt eine interne Karte der Umgebung, auf deren Grundlage Posenfehler korrigiert werden (Abschnitt 6.2.4).

## 6 Softwaremodule

Der *rc\_visard* wird mit einer Reihe von On-Board-Softwaremodulen mit verschiedenen Funktionalitäten ausgeliefert. Jedes Softwaremodul bietet über seinen zugehörigen *Node* eine *REST-API-Schnittstelle* (Abschnitt 7.3) als Programmierschnittstelle an.

Die Softwaremodule des *rc\_visard* können unterteilt werden in

- **3D-Kamera-Module (Abschnitt 6.1)** welche Bildpaare aufnehmen und 3D Tiefeninformationen bereitstellen, und auch über die *GigE Vision/GenICam-Schnittstelle* des *rc\_visard* konfigurierbar sind.
- **Navigationsmodule (Abschnitt 6.2)** welche Schätzungen der momentanen Pose, Geschwindigkeit und Beschleunigung des *rc\_visard* bereitstellen,
- **Detektionsmodule (Abschnitt 6.3)** welche eine Vielzahl verschiedener Detektionsfunktionen, wie Greifpunktberechnungen und Objekterkennung anbieten.
- **Konfigurationsmodule (Abschnitt 6.4)** welche es dem Nutzer ermöglichen, Kalibrierungen durchzuführen und den *rc\_visard* für spezielle Anwendungen zu konfigurieren.
- **Datenbankmodule (Abschnitt 6.5)** welche dem Nutzer die Konfiguration globaler Daten ermöglichen, die in allen anderen Modulen verfügbar sind, wie Load Carrier, Regions of Interest und Greifer.

### 6.1 3D-Kamera-Module

Die 3D-Kamera-Software des *rc\_visard* enthält die folgenden Module:

- **Kamera (rc\_camera, Abschnitt 6.1.1)** 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 (rc\_stereomatching, Abschnitt 6.1.2)** nutzt die rektifizierten Stereo-Bildpaare, um 3D-Tiefeninformationen, z.B. für Disparitäts-, Fehler- und Konfidenzbilder, zu berechnen. Diese werden auch als *GenICam-Bild-Streams* bereitgestellt.

Die Module für die *Kamera* und das *Stereo-Matching*, welche die Bildpaare und die 3D-Tiefeninformationen bereitstellen, sind auch über die *GigE Vision/GenICam-Schnittstelle* des *rc\_visard* konfigurierbar.

#### 6.1.1 Kamera

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.

### 6.1.1.1 Rektifizierung

Um die Bildverarbeitung zu vereinfachen rektifiziert das Modul alle Kamerabilder basierend auf der Kamerakalibrierung. Dies bedeutet, dass die Verzerrung entfernt und der Bildhauptpunkt genau in die Mitte des Bildes gelegt wird.

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.

Im Fall einer Stereokamera richtet die Rektifizierung die Bilder so aus, dass Objektpunkte in beiden Bildern immer in die gleiche Bildzeile projiziert werden. Die optischen Achsen der Kameras werden dadurch exakt parallel ausgerichtet.

### 6.1.1.2 Anzeigen und Herunterladen von Bildern

Der *rc\_visard* bietet über die GenICam-Schnittstelle zeitgestempelte rektifizierte Kamerabilder (siehe [Verfügbare Bild-Streams](#), Abschnitt 7.2.6). Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 7.1) 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 7.1.3) beschrieben wird.

### 6.1.1.3 Parameter

Das Kamera-Modul wird als *rc\_camera* bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Kamera* dargestellt. Der Benutzer kann die Kamera-Parameter entweder dort oder direkt über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.3) oder GigE Vision (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 7.2) ändern.

**Bemerkung:** Wird der *rc\_visard* über GigE Vision genutzt, können die Kamera-Parameter nicht über die Web GUI oder REST-API geändert werden.

## Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.1: Laufzeitparameter des rc\_camera-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
exp_auto	bool	false	true	true	Umschalten zwischen automatischer und manueller Belichtung
exp_auto_average_max	float64	0.0	1.0	0.75	Maximaler Belichtungsmittelwert, wenn exp_auto auf true gesetzt ist
exp_auto_average_min	float64	0.0	1.0	0.25	Maximaler Belichtungsmittelwert, wenn exp_auto auf true gesetzt ist
exp_auto_mode	string	-	-	Normal	Modus für automatische Belichtung: [Normal, Out1High, AdaptiveOut1]
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, wenn exp_auto auf true gesetzt ist
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, wenn exp_auto auf false gesetzt ist
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	Manuelle Verstärkung in Dezibel, wenn exp_auto auf false gesetzt ist
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)

## Beschreibung der Laufzeitparameter

The screenshot displays the Roboception web GUI for the 'rc\_visard > Kamera' interface. At the top, there are two live video feeds labeled 'Linkes Live-Bild' and 'Rechtes Live-Bild', both showing a bin filled with white rectangular blocks. Below the feeds, technical specifications are listed: Auflösung (px) 1280 x 960, FPS (Hz) 24.6, Belichtung (ms) 5.76, Verstärkung (dB) 0.0, and Helligkeit 0.25. The main settings area is titled 'IOControl Einstellungen' and includes a dropdown for 'Out1 / Projektor' set to 'ExposureAlternateActive'. Under 'Kameraeinstellungen', there is a 'Zurücksetzen' button and a 'Bildwiederholrate (Hz)' slider set to 25. The 'Belichtung' section is set to 'Auto' and includes sliders for 'Maximale Belichtungszeit (s)' (0.018), 'Modus Belichtungsautomatik' (Normal, Out1High, AdaptiveOut1), 'Max. Helligkeit' (0.75), and 'Min. Helligkeit' (0.25). A 'Bereich zur Regelung' section allows for selecting a region in the image and adjusting 'Offset X (Pixel)', 'Offset Y (Pixel)', 'Breite (Pixel)', and 'Höhe (Pixel)', all currently set to 0. The 'Manuell' options for 'Belichtungszeit (s)' (0.005) and 'Verstärkungsfaktor (dB)' (0) are also visible. The 'Weißabgleich' section is set to 'Auto' and includes sliders for 'Blau | Grün' (2.4) and 'Rot | Grün' (1.2).

Abb. 6.1: Seite *Kamera* in der Web GUI**fps (Bildwiederholrate)**

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 Abbildung [Abb. 6.2](#) 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. 6.2: Die interne Bildaufnahme geschieht immer mit 25 Hz. Der fps Parameter bestimmt, wie viele dieser Kamerabilder per GigE Vision versendet werden.

**exp\_auto (Belichtungszeit Auto oder Manuell)**

Dieser Wert lässt sich für den automatischen Belichtungsmodus auf *true* und für den manuellen Belichtungsmodus auf *false* setzen. Im manuellen Belichtungsmodus wird die gewählte Belichtungszeit konstant gehalten und die Verstärkung bleibt bei 0,0 dB, auch wenn die Bilder über- oder unterbelichtet sind. Im automatischen Belichtungsmodus werden die Belichtungszeit und der Verstärkungsfaktor automatisch angepasst, sodass das Bild korrekt belichtet wird. Wenn die Automatik abgeschaltet wird, werden *exp\_value* und *gain\_value* auf die letzten von der Automatik ermittelten Werte für Belichtungszeit und Verstärkung gesetzt.

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=
↔<value>
```

**API Version 1 (veraltet)**

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_auto=<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 des *rc\_visard* oder *rc\_viscore* angeschlossen ist. Dieser Ausgang kann durch das optionale IOControl-Modul (*IOControl und Projektor-Kontrolle*, Abschnitt [6.4.4](#)) 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](#), 6.1.2.5 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 6.1.2.5), 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)

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>
```

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 7.2).

### 6.1.1.4 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.2: Statuswerte des rc\_camera-Moduls

Name	Beschreibung
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 <i>Out1 Reduktion (%)</i> angezeigt.
baseline	Basisabstand $t$ der Stereokamera in Metern
brightness	Aktuelle Helligkeit als Wert zwischen 0 und 1
color	0 für monochrome Kameras, 1 für Farbkameras
exp	Aktuelle Belichtungszeit in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als <i>Belichtung (ms)</i> 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 <i>FPS (Hz)</i> angezeigt.
gain	Aktueller Verstärkungsfaktor in Dezibel. Dieser Wert wird unter der Bildvorschau in der Web GUI als <i>Verstärkung (dB)</i> angezeigt.
height	Höhe des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von <i>Auflösung (px)</i> angezeigt.
test	0 for Live-Bilder und 1 für Test-Bilder
width	Breite des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von <i>Auflösung (px)</i> angezeigt.
temp_left	Temperatur des linken Kamerasensors in Grad Celsius
temp_right	Temperatur des rechten Kamerasensors in Grad Celsius
time	Verarbeitungszeit für die Bilderfassung in Sekunden

### 6.1.1.5 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"
    }
  }
}
```

## 6.1.2 Stereo-Matching

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 8.7) benötigt. Diese Lizenz ist auf jedem *rc\_visard* vorhanden, der nach dem 31.01.2019 gekauft wurde.

### 6.1.2.1 Berechnung von Disparitätsbildern

Nach der Rektifizierung haben das linke und das rechte Kamerabild die Eigenschaft, dass ein Objektpunkt in beiden Bildern auf die gleiche Pixelreihe projiziert wird. Die Pixelspalte des Objektpunkts ist im rechten Bild maximal so groß wie die Pixelspalte des Objektpunkts im linken Bild. Der Begriff Disparität bezeichnet den Unterschied zwischen den Pixelspalten im rechten und linken Bild und gibt indirekt die Tiefe des Objektpunkts, d.h. dessen Abstand zur Kamera an. Das Disparitätsbild speichert die Disparitätswerte aller Pixel des linken Kamerabilds.

Je größer die Disparität, desto näher liegt der Objektpunkt. Beträgt die Disparität 0, bedeutet dies, dass die Projektionen des Objektpunkts in der gleichen Bildspalte liegen und der Objektpunkt sich in unendlicher Distanz befindet. Häufig gibt es Pixel, für welche die Disparität nicht bestimmt werden kann. Dies ist der Fall bei Verdeckungen auf der linken Seite von Objekten, da diese Bereiche von der rechten Kamera nicht eingesehen werden können. Zudem lässt sich die Disparität auch bei texturlosen Bereichen nicht bestimmen. Pixel, für welche die Disparität nicht bestimmt werden kann, werden mit dem besonderen Disparitätswert 0 als ungültig markiert. 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 den kleinstmöglichen Disparitätswert über 0 gesetzt.

Um Disparitätswerte zu berechnen, muss der Stereo-Matching-Algorithmus die zugehörigen Objektpunkte im linken und rechten Kamerabild finden. Diese Punkte stellen jeweils den gleichen Objektpunkt in der Szene dar. Für das Stereo-Matching nutzt der *rc\_visard* *SGM* (*Semi-Global Matching*). Dieser Algorithmus zeichnet sich durch eine kurze Laufzeit aus und bietet, insbesondere an Objekträndern, bei feinen Strukturen und in schwach texturierten Bereichen, eine hohe Genauigkeit.

Unabhängig vom eingesetzten Verfahren ist es beim Stereo-Matching wichtig, dass das Bild über eine gewisse Textur verfügt, durch Muster oder Oberflächenstrukturen. Bei einer gänzlich untexturierten Szene, wie einer weißen Wand ohne jede Struktur, können Disparitätswerte entweder nicht berechnet werden, oder aber die Ergebnisse sind fehlerhaft oder von geringer Konfidenz (siehe [Konfidenz- und Fehlerbilder](#), Abschnitt 6.1.2.3). Bei der Textur in der Szene sollte es sich nicht um ein künstliches, regelmäßig wiederkehrendes Muster handeln, da diese Strukturen zu Mehrdeutigkeiten und damit zu falschen Disparitätsmessungen führen können.

Für schwach texturierte Objekte oder in untexturierten Umgebungen lässt sich mithilfe eines externen Musterprojektors eine statische künstliche Struktur auf die Szene projizieren. Dieses projizierte Muster sollte zufällig sein und keine wiederkehrenden Strukturen enthalten. Der *rc\_visard* bietet das *IOControl*-Modul als optionales Softwaremodul (siehe [IOControl und Projektor-Kontrolle](#), Abschnitt 6.4.4), das einen Musterprojektor ansteuern kann.

### 6.1.2.2 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:

$$\begin{aligned} P_x &= \frac{p_x \cdot t}{d} \\ P_y &= \frac{p_y \cdot t}{d} \\ P_z &= \frac{f \cdot t}{d}, \end{aligned} \quad (6.1)$$

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 7.2.4).

**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 Gleichungen (6.1) 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.

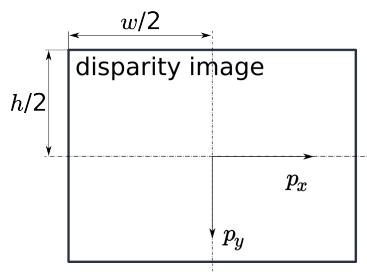


Abb. 6.3: Bildkoordinatensystem: Der Ursprung des Bildkoordinatensystems befindet sich in der Bildmitte –  $w$  ist die Bildbreite und  $h$  die Bildhöhe.

Die gleichen Formeln, aber mit den entsprechenden GenICam-Parametern, sind in *Umwandlung von Bild-Streams* (Abschnitt 7.2.7) 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.

### 6.1.2.3 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.

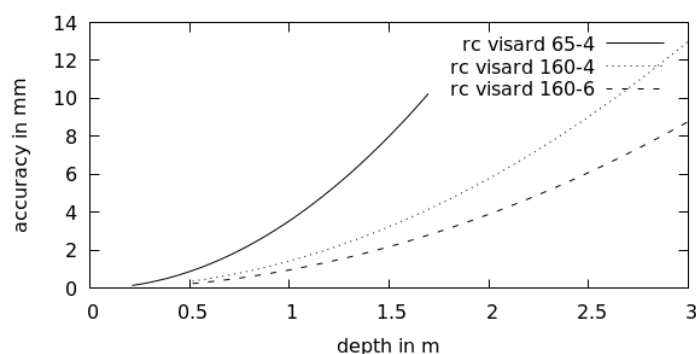
Der Disparitätsfehler  $d_{eps}$  (in Pixeln) lässt sich mithilfe der Brennweite  $f$  (in Pixeln), des Basisabstands  $t$  (in Metern) und des Disparitätswerts  $d$  (in Pixeln) desselben Pixels im Disparitätsbild in einen Tiefenfehler  $z_{eps}$  (in Metern) umrechnen:

$$z_{eps} = \frac{d_{eps} \cdot f \cdot t}{d^2}. \quad (6.2)$$

Durch Kombination der Gleichungen (6.1) und (6.2) 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 Tiefengenaugigkeit wie folgt grafisch darstellen:



#### 6.1.2.4 Anzeigen und Herunterladen von Tiefenbildern und Punktwolken

Der *rc\_visard* stellt über die GenICam-Schnittstelle zeitgestempelte Disparitäts-, Fehler- und Konfidenzbilder zur Verfügung (siehe [Verfügbare Bild-Streams](#), Abschnitt 7.2.6). Live-Streams in geringerer Qualität werden auf der *Tiefenbild* Seite in der *Web GUI* (Abschnitt 7.1) 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 7.1.4) beschrieben wird.

#### 6.1.2.5 Parameter

Das Stereo-Matching-Modul wird in der REST-API als *rc\_stereomatching* bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Tiefenbild* dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API ([REST-API-Schnittstelle](#), Abschnitt 7.3) oder über GigE Vision ([GigE Vision 2.0/GenICam-Schnittstelle](#), Abschnitt 7.2) ändern.

#### Übersicht über die Parameter

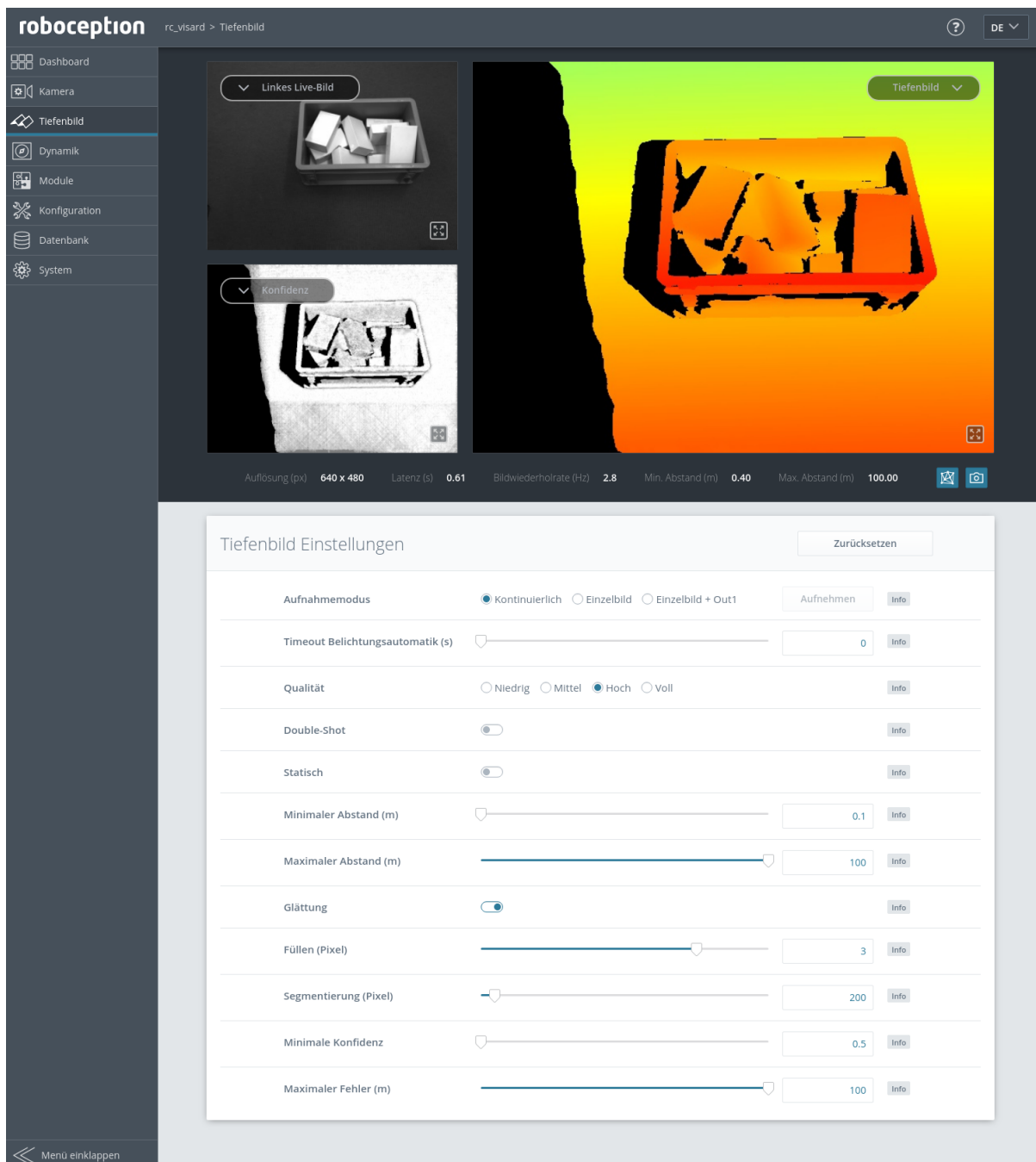
Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.3: 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
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.
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

### 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. 6.4: Seite *Tiefenbild* der Web GUI

### acquisition\_mode (Aufnahmemodus)

Der Aufnahmemodus kann auf Continuous (*Kontinuierlich*), SingleFrame (*Einzelbild*) oder SingleFrameOut1 (*Einzelbild + Out1*) eingestellt werden. *Kontinuierlich* 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 6.4.4). In diesem Modus wird out1\_mode des IOControl-Moduls automa-

tisch 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 Projekteinstellung `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. 6.4: Auflösung des Tiefenbilds in Abhängigkeit von der gewählten Qualität

Qualität	<b>Voll</b> (Full)	<b>Hoch</b> (High)	<b>Mittel</b> (Medium)	<b>Niedrig</b> (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>
```

### 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>
```

**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>
```

### 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*, Abschnitt 6.1.2.3).

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>
```

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 7.2).

### 6.1.2.6 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.5: 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 <i>Bildwiederholrate (Hz)</i> angezeigt.
latency	Zeit zwischen Bildaufnahme und Weitergabe des Disparitätsbildes in Sekunden
width	Aktuelle Breite des Disparitäts-, Fehler- und Konfidenzbildes in Pixeln
height	Aktuelle Höhe des Disparitäts-, Fehler- und Konfidenzbildes in Pixeln
mindepth	Tatsächlicher minimaler Arbeitsabstand in Metern
maxdepth	Tatsächlicher maximaler Arbeitsabstand in Metern
time_matching	Zeit in Sekunden für die Durchführung des Stereo-Matchings mittels <i>SGM</i> 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

### 6.1.2.7 Services

Das Stereo-Matching-Modul bietet folgende Services.

#### 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. 6.6: 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 Triggerruf 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"
    }
  }
}
```

### 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"
    }
  }
}
```

## 6.2 Navigationsmodule

Die Navigationsmodule des *rc\_visard* umfassen:

- **Sensordynamik (rc\_dynamics, Abschnitt 6.2.1)** 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 6.2.2)** schätzt die Bewegung des *rc\_visard* anhand der Bewegungen charakteristischer Merkmale in den Bildern der linken Kamera.

- **Stereo-INS** (`rc_stereo_ins`, [Abschnitt 6.2.3](#)) 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 6.2.4](#)) übernimmt die simultane Lokalisierung und Kartenerstellung, um akkumulierte Posendaten zu korrigieren. Die Trajektorie des `rc_visard` lässt sich über die [REST-API-Schnittstelle](#) ([Abschnitt 7.3](#)) abfragen.

### 6.2.1 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 6.2.2](#)).
- **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 6.2.3](#)).
- **SLAM** (`rc_slam`) übernimmt die simultane Lokalisierung und Kartenerstellung (SLAM), um akkumulierte Posendaten zu korrigieren ([Abschnitt 6.2.4](#)).

**Bemerkung:** Die gleichzeitige Nutzung des Dynamik-Moduls und des [Stereo-Matching](#) ([Abschnitt 6.1.2](#)) kann zu verringerter Genauigkeit der Bewegungsschätzung führen. Der [Abschnitt Visuelle Odometrie](#) ([6.2.2](#)) erklärt, wie dies vermieden werden kann.

#### 6.2.1.1 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.

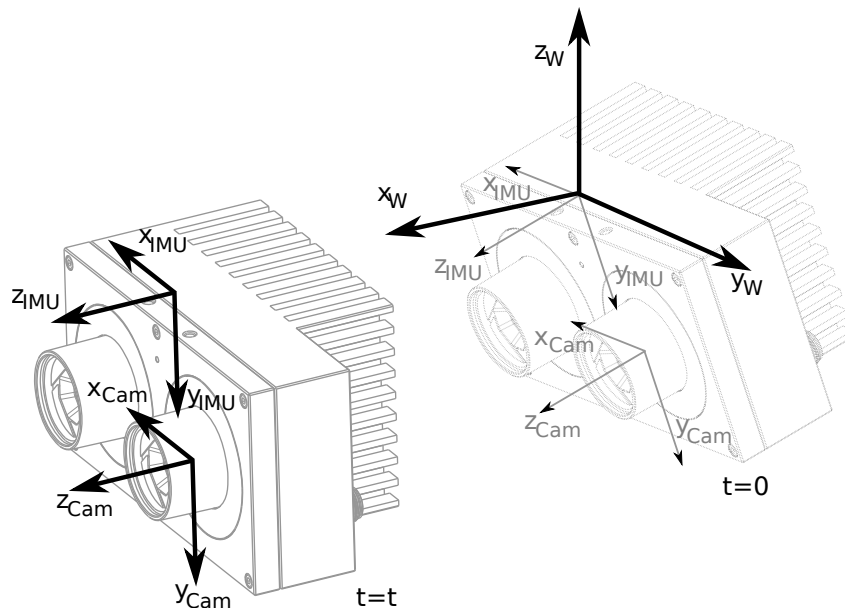


Abb. 6.5: Koordinatensysteme für die Zustandsschätzung: Das IMU-Koordinatensystem liegt im Gehäuse des *rc\_visard*, das *Kamera-Koordinatensystem* (Abschnitt 3.7) im Fokuspunkt der linken Kamera.

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 7).

**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.

### 6.2.1.2 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 7.4):

Name	Frequenz	Quelle	Beschreibung
<i>pose</i>	25 Hz	Best Effort	genaueste Schätzung der Pose des Kamerakoordinatensystems, aber leicht zeitverzögert
<i>pose_ins</i>	25 Hz	<i>Stereo-INS</i>	genaueste Schätzung der Pose des Kamerakoordinatensystems, aber leicht zeitverzögert
<i>pose_rt</i>	200 Hz	Best Effort	Pose des Kamerakoordinatensystems
<i>pose_rt_ins</i>	200 Hz	<i>Stereo-INS</i>	Pose des Kamerakoordinatensystems
<i>dynamics</i>	200 Hz	Best Effort	Pose, Geschwindigkeit und Beschleunigung im IMU-Koordinatensystem
<i>dynamics_ins</i>	200 Hz	<i>Stereo-INS</i>	Pose, Geschwindigkeit und Beschleunigung im IMU-Koordinatensystem
<i>imu</i>	200 Hz	<i>Stereo-INS</i>	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 6.2.1.1). 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 6.2.1.1). 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* und *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  $\mathbf{q} = (q_x, q_y, q_z, q_w)^T$  als Einheitsquaternion,
- Lineargeschwindigkeit  $\mathbf{v} = (v_x, v_y, v_z)^T$  in  $\frac{m}{s}$ ,
- Winkelgeschwindigkeit  $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^T$  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 6.2.1.1). 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 6.2.4) 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.

#### 6.2.1.3 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.7: Die Statuswerte des `rc_dynamics` Moduls

Name	Beschreibung
<code>state</code>	Der aktuelle Zustand des Sensordynamik-Moduls

#### 6.2.1.4 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. 6.8](#) angegeben.

Tab. 6.8: Mögliche Zustände des Sensordynamik-Moduls

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

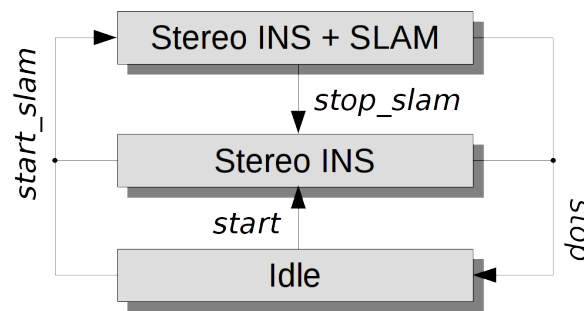


Abb. 6.6: Vereinfachtes Zustandsdiagramm

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. 6.8](#) 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"
  }
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}  
}
```

### 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 Trajektorien-schätzung 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 Trajektorien-schätzung 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 7.4.3).

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"
  }
}
```

### 6.2.2 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 erhö-

hen, 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 von *Stereo-Matching* 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 der *Kamera* 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 6.2.3).

### 6.2.2.1 Parameter

Das Odometrie-Modul heißt *rc\_stereovisodo* und wird in der *Web GUI* (Abschnitt 7.1) auf der Seite *Dynamik* dargestellt. Der Benutzer kann die Parameter der visuellen Odometrie entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.3) ändern.

#### Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

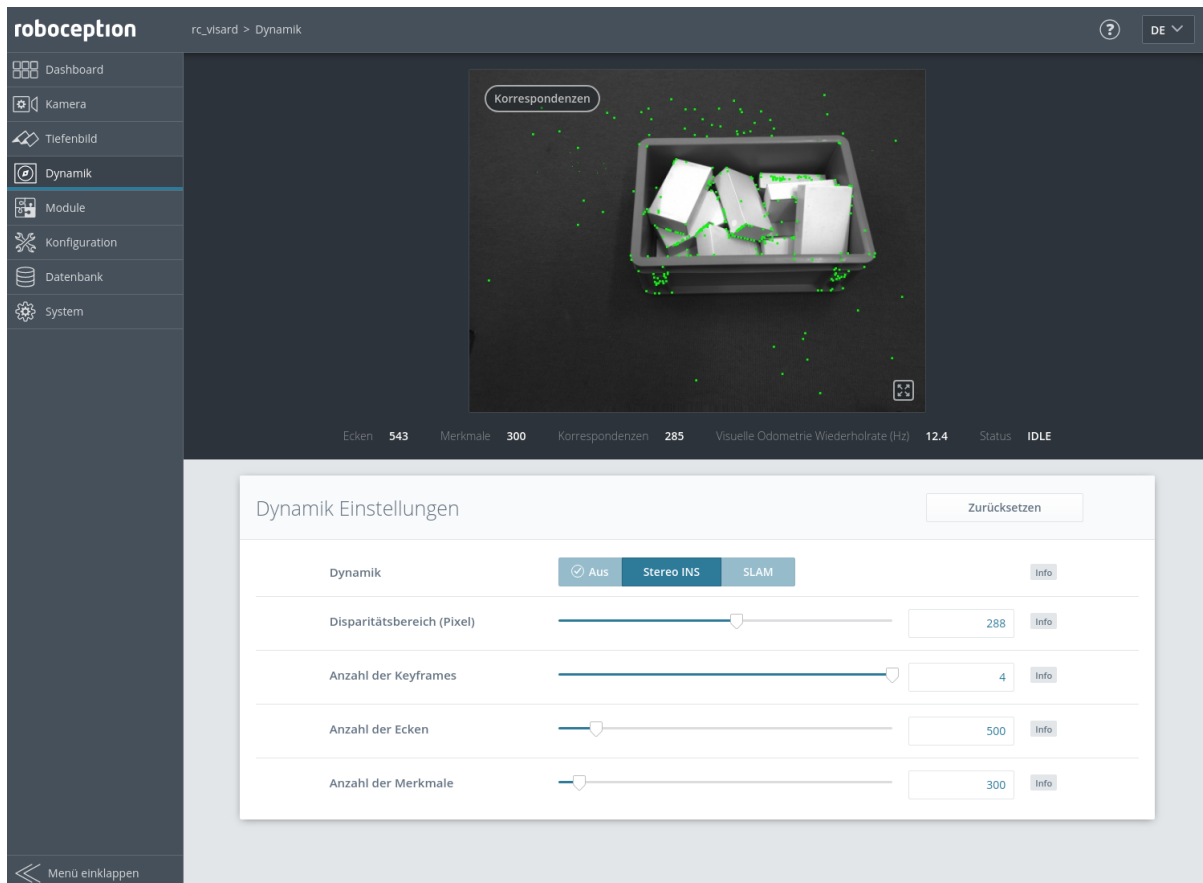
Tab. 6.9: Laufzeitparameter des *rc\_stereovisodo*-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
<i>disprange</i>	int32	32	512	256	Disparitätsbereich in Pixeln
<i>ncorner</i>	int32	50	4000	500	Anzahl der Eckpunkte
<i>nfeature</i>	int32	50	4000	300	Anzahl der Bildmerkmale
<i>nkey</i>	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. 6.7: *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.

**6.2.2.2 Statuswerte**

Dieses Modul meldet folgende Statuswerte:

Tab. 6.10: Statuswerte des rc\_stereovisodo-Moduls

Name	Beschreibung
corner	Anzahl der erkannten Eckpunkte. Dieser Wert wird unter der Bildvorschau in der Web GUI als <i>Ecken</i> angezeigt.
correspondences	Anzahl der Übereinstimmungen. Dieser Wert wird unter der Bildvorschau in der Web GUI als <i>Korrespondenzen</i> angezeigt.
feature	Anzahl der Bildmerkmale. Dieser Wert wird unter der Bildvorschau in der Web GUI als <i>Merkmale</i> angezeigt.
fps	Frequenz der visuellen Odometrie in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als <i>Rate visuelle Odometrie (Hz)</i> 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

**6.2.2.3 Services**

Das Modul bietet keine eigenen Funktionen zum Starten bzw. Stoppen, da es über das *Dynamik-Modul* (Abschnitt 6.2.1) 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"
    }
  }
}
```

## 6.2.3 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.

### 6.2.3.1 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.

### 6.2.3.2 Parameter

Das Stereo-INS-Modul heißt `rc_stereo_ins`.

Dieses Modul besitzt keine Laufzeitparameter.

### 6.2.3.3 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.11: Statuswerte des rc\_stereo\_ins-Moduls

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

## 6.2.4 SLAM

Das SLAM-Modul ist ein optionales Modul, welches intern auf dem *rc\_visard* läuft, und eine separate SLAM-Lizenz (Abschnitt 8.7) 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 Posenfänger 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*, Abschnitt 6.1.2.2). Zu diesem Zweck kann die gesamte Trajektorie des *rc\_visard* beim SLAM-Modul abgefragt werden.

### 6.2.4.1 Verwendung

Das SLAM-Modul kann jederzeit entweder über das *rc\_dynamics* Interface (siehe Dokumentation der zugehörigen *Services*, Abschnitt 6.2.1.4) 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 6.2.4.5) 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.

#### 6.2.4.2 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 6.2.1.4).

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](#) 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.

#### 6.2.4.3 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 7.3) setzen.

#### Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.12: Laufzeitparameter des `rc_slam`-Moduls

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

#### 6.2.4.4 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.13: 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 Trajektorien-schätzung

Der Parameter state kann folgende Werte annehmen:

Tab. 6.14: 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.

#### 6.2.4.5 Services

**Bemerkung:** Die Aktivierung und Deaktivierung des SLAM-Moduls wird über das Service-Interface von rc\_dynamics gesteuert (siehe [Services](#), Abschnitt 6.2.1.4).

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.

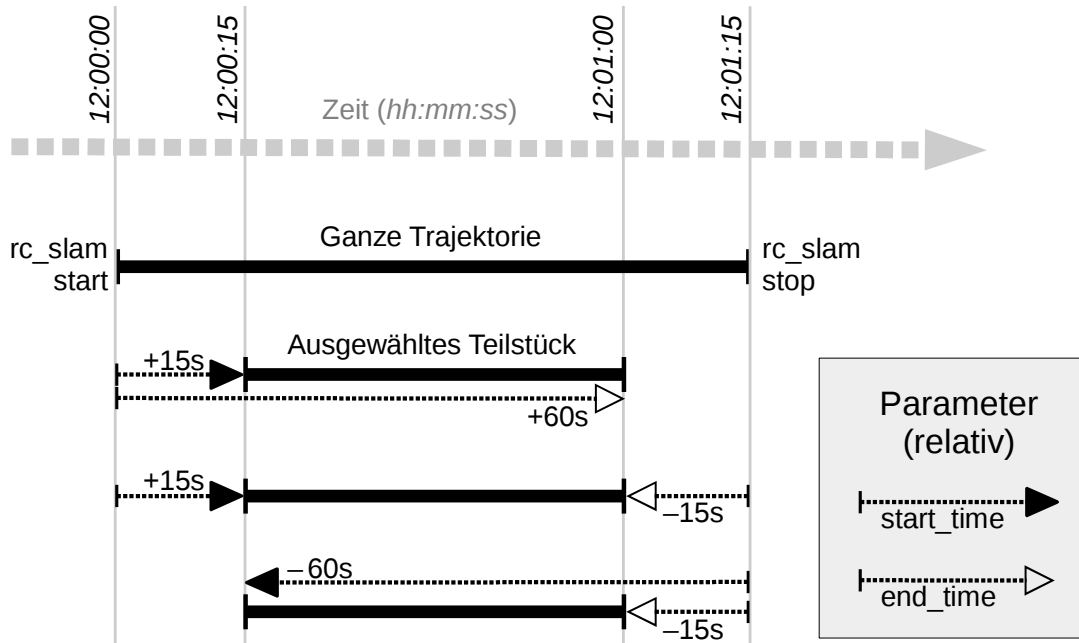


Abb. 6.8: Beispielhafte Kombinationen positiver und negativer relativer Zeitangaben. Alle gezeigten Kombinationen resultieren im gleichen Teilstück.

**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",
    "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"
    }
  }
}
```

### 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 6.2.1.4) 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"
  }
}
```

## 6.3 Detektionsmodule

Der *rc\_visard* bietet Softwaremodule für unterschiedliche Detektionsanwendungen:

- **LoadCarrier** (*rc\_load\_carrier*, **Abschnitt 6.3.1**) ermöglicht die Erkennung von Load Carriern (Behältern) und ihres Füllstands.
- **TagDetect** (*rc\_april\_tag\_detect* und *rc\_qr\_code\_detect*, **Abschnitt 6.3.2**) ermöglicht die Erkennung von AprilTags und QR-Codes sowie die Schätzung von deren Pose.
- **ItemPick und BoxPick** (*rc\_itempick* und *rc\_boxpick*, **Abschnitt 6.3.3**) bietet eine Standardlösung für robotische Pick-and-Place-Anwendungen für Vakuum-Greifsysteme.
- **SilhouetteMatch** (*rc\_silhouettematch*, **Abschnitt 6.3.4**) bietet eine Objekterkennungslösung für Objekte, die sich auf einer Ebene befinden.

Diese Module sind optional und können durch Kauf einer separaten *Lizenz* (**Abschnitt 8.7**) aktiviert werden.

### 6.3.1 LoadCarrier

#### 6.3.1.1 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 6.5.1**) 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 und BoxPick* (**Abschnitt 6.3.3**) oder *SilhouetteMatch* (**Abschnitt 6.3.4**) vorhanden ist. Andernfalls benötigt es eine gesonderte LoadCarrier *Lizenz* (**Abschnitt 8.7**), welche erworben werden muss.

#### 6.3.1.2 Erkennung von Load Carriern

Der Algorithmus zur Erkennung von Load Carriern basiert auf der Erkennung des rechteckigen Load Carrier Rands. Standardmäßig wird nach einem Load Carrier gesucht, dessen durch den Rand definierte Ebene senkrecht zur gemessenen Gravitationsrichtung ausgerichtet ist. Um einen geeigneten Load Carrier zu erkennen, muss seine 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 6.5.2) genutzt wird, um das Volumen für die Load Carrier Erkennung einzuschränken, muss nur der Rand des Load Carriers vollständig in der Region of Interest enthalten sein.

Die Erkennung liefert die Pose des Load Carrier Koordinatensystems (siehe [Load Carrier Definition](#), Abschnitt 6.5.1.2) im gewünschten Referenzkoordinatensystem zurück.

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

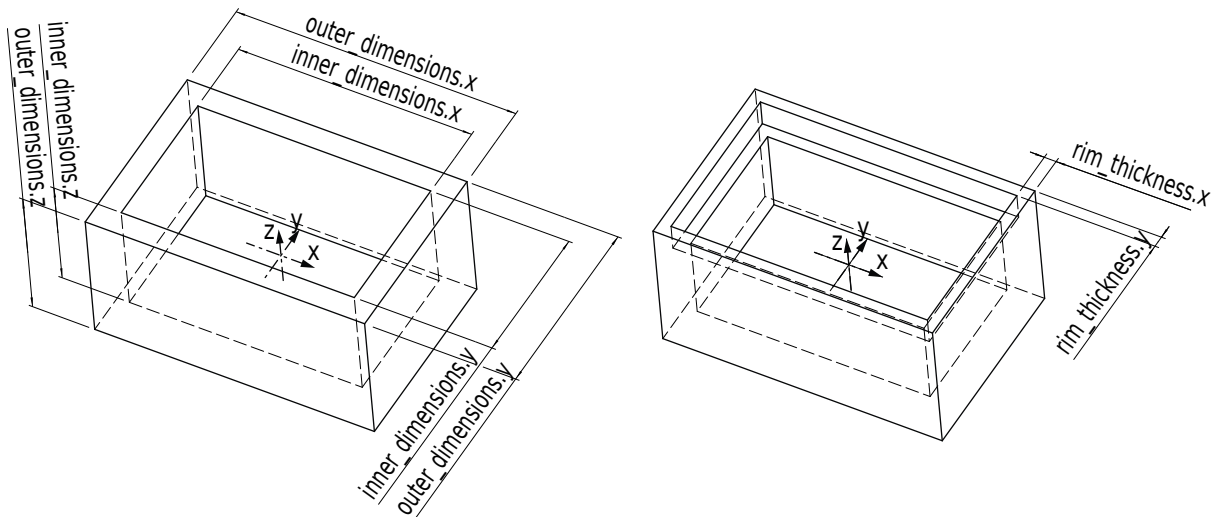


Abb. 6.9: Illustration verschiedener Load Carrier Modelle und deren Koordinatensysteme

### 6.3.1.3 Füllstandserkennung

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

Dazu wird der Load Carrier in eine konfigurierbare Anzahl von Zellen unterteilt, welche in einem 2D-Raster angeordnet sind. Die maximale Anzahl der Zellen beträgt 10x10. 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 6.3.3.4). 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. 6.10: 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.

#### 6.3.1.4 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.

#### Stereokamera und Stereo-Matching

Folgende Daten werden vom LoadCarrier Modul verarbeitet:

- Die rektifizierten Bilder des *Kamera*-Moduls (*rc\_camera*, Abschnitt 6.1.1);
- Die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching*-Moduls (*rc\_stereomatching*, Abschnitt 6.1.2).

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

#### Schätzung der Gravitationsrichtung

Jedes Mal, wenn eine Load Carrier oder Füllstandserkennung durchgeführt wird, wird die Gravitationsrichtung basierend auf den IMU-Daten des *rc\_visard* geschätzt.

**Bemerkung:** Die Richtung des Gravitationsvektors wird durch Messungen der linearen Beschleunigung der IMU bestimmt. Für eine korrekte Schätzung des Gravitationsvektors muss der *rc\_visard* stillstehen.

#### 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 6.4.4) 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 6.1.2.5), 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 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren (siehe *Stereokamera-Parameter*, Abschnitt 6.1.1.3).

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 6.3.1.6) 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 die Load Carrier Funktionalität alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). 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.

#### 6.3.1.5 Parameter

Das LoadCarrier Modul wird in der REST-API als `rc_load_carrier` bezeichnet in der *Web GUI* (Abschnitt 7.1) unter *Module* → *LoadCarrier* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.3) ändern.

#### Übersicht über die Parameter

Dieses Modul bietet folgende Laufzeitparameter:

Tab. 6.15: Laufzeitparameter des `rc_load_carrier` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
<code>crop_distance</code>	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
<code>model_tolerance</code>	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 7.3) eines anderen Moduls verwendet werden, sind sie durch den Präfix load\_carrier\_ gekennzeichnet.

#### 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. 6.42](#)).

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>
```

#### 6.3.1.6 Services

Die angebotenen Services des LoadCarrier Moduls können mithilfe der [REST-API-Schnittstelle](#) (Abschnitt 7.3) oder der rc\_visard [Web GUI](#) (Abschnitt 7.1) 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 6.3.1.2) 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 6.3.1.4).

load\_carrier\_ids: IDs der zu erkennenden Load Carrier.

Möglicherweise benötigte Serviceargumente:

robot\_pose: siehe *Hand-Auge Kalibrierung* (Abschnitt 6.3.1.4).

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.

**Warnung:** Es kann nur eine Art von Region of Interest 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": [
      {
        "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"
    }
  }
}
```

### detect\_filling\_level

löst eine Load Carrier Füllstandserkennung aus, wie in [Füllstandserkennung](#) (Abschnitt 6.3.1.3) 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 6.3.1.4).

load\_carrier\_ids: IDs der zu erkennenden Load Carrier.

Möglicherweise benötigte Serviceargumente:

robot\_pose: siehe [Hand-Auge Kalibrierung](#) (Abschnitt 6.3.1.4).

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.

**Warnung:** Es kann nur eine Art von Region of Interest 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",
              "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"
        },
        "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"
          }
        }
      }
    ]
  }
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
"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 6.5.1.5) 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 6.5.1.5) 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 6.5.1.5) 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 6.5.1.5) 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 6.5.1.5) 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 6.5.1.5) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) 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 6.5.2.4) in `rc_roi_db` beschrieben.

### 6.3.1.7 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. 6.16: 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
-302	Es wurde mehr als ein Load Carrier an den <code>detect_load_carriers</code> oder <code>detect_filling_level</code> Service übergeben, aber nur einer wird unterstützt.
10	Die maximal speicherbare Anzahl an Load Carriern wurde erreicht.
11	Mit dem Aufruf von <code>set_load_carrier</code> wurde ein bereits existierendes Objekt mit derselben <code>id</code> überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
102	Der erkannte Load Carrier ist leer.
300	Ein gültiges <code>robot_pose</code> -Argument wurde angegeben, ist aber nicht erforderlich.

## 6.3.2 TagDetect

### 6.3.2.1 Einführung

Die TagDetect-Module sind optionale Module, die intern auf dem `rc_visard` laufen, und benötigen gesonderte [Lizenzen](#) (Abschnitt 8.7), 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-Barcodes und Marker zu erkennen. Derzeit gibt es TagDetect-Module für *QR-Codes* und *AprilTags*. Neben der Erkennung berechnen die Module die Position und Orientierung jedes Markers im 3D-Kamerakoordinatensystem, um diesen beispielsweise mit einem Roboter zu manipulieren oder die Pose der Kamera in Bezug auf den Marker zu berechnen.

Die Markererkennung besteht aus drei Schritten:

1. Markererkennung auf dem 2D-Bildpaar (siehe [Markererkennung](#), Abschnitt 6.3.2.2).
2. Schätzung der Pose jedes Markers (siehe [Posenschätzung](#), Abschnitt 6.3.2.3).
3. Wiedererkennung von bisher gesehenen Markern (siehe [Marker-Wiedererkennung](#), Abschnitt 6.3.2.4).

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

## QR-Code



Abb. 6.11: Beispiel eines QR-Codes

QR-Codes sind zweidimensionale Barcodes, 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 Marker mit dem TagDetect-Modul nicht garantiert. Gleiches gilt für QR-Codes, welche Zeichen außerhalb des ASCII-Zeichensatzes beinhalten.

## AprilTag

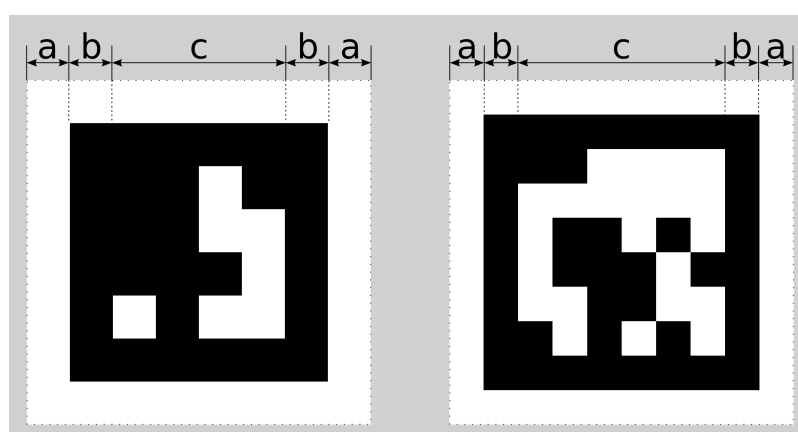


Abb. 6.12: Ein 16h5 Marker (links) und ein 36h11 Marker (rechts). AprilTags bestehen aus einem obligatorischen weißen (a) und schwarzen (b) Rahmen und einer variablen Menge an Datenmodulen (c).

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. 6.12](#) veranschaulicht den Aufbau von AprilTags. Sie sind von einem obligatorischen weißen und schwarzen Rahmen

umschlossen, welcher jeweils ein Modul breit ist. Innen enthalten sie eine variable Menge an Datenmodulen. Anders als QR-Codes speichern sie keine benutzerdefinierten Informationen, sondern werden durch eine vordefinierte *Familie* und *ID* identifiziert. Die Tags in [Abb. 6.12](#) sind zum Beispiel aus Familie 16h5 bzw. 35h11 und besitzen ID 0 bzw. 11. Alle unterstützten Familien werden in [Tab. 6.17](#) aufgelistet.

Tab. 6.17: AprilTag-Familien

Familie	Anzahl IDs	Empfohlen
16h5	30	-
25h7	242	-
25h9	35	o
36h10	2320	o
36h11	587	+

Die Zahl vor dem „h“ jeder Familie bezeichnet die Anzahl der Datenmodule, welche im Marker enthalten sind: Während ein 16h5 Marker 16 (4x4) Datenmodule enthält ((c) in [Abb. 6.12](#)), besteht ein 36h11 Marker aus 36 (6x6) Datenmodulen. Die Zahl hinter dem „h“ bezeichnet den Hamming-Abstand zwischen zwei Markern der Familie. Je höher, desto höher ist die Robustheit, aber desto weniger IDs stehen bei gleicher Anzahl an Datenmodulen zur Verfügung (siehe [Tab. 6.17](#)).

Der Vorteil von Familien mit weniger Datenmodulen (bspw. 16h5 im Vergleich zu 36h11) ist die niedrigere Auflösung der Marker. Jedes Modul ist somit größer, weshalb der Marker 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 Markererkennung signifikant reduziert wird, da es zu einer höheren Falsch-Positiv-Rate kommt. Dies bedeutet, dass Marker verwechselt werden oder nicht existierende Marker in zufälliger Bildtextur oder im Bildrauschen erkannt werden.

Aus diesen Gründen empfehlen wir die Verwendung der 36h11-Familie und raten ausdrücklich von den Familien 16h5 und 25h7 ab. Letztgenannte Familien 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 AprilTag-Projektwebseite (<https://april.eecs.umich.edu/software/apriltag.html>) heruntergeladen werden. Jede Familie besteht aus mehreren PNGs, welche jeweils einen AprilTag enthalten, und einem PDF, welches jeden AprilTag auf einer eigenen Seite enthält. Jedes Pixel im PNG entspricht dabei einem Modul des AprilTags. Beim Drucken der Marker sollte darauf geachtet werden, den weißen Rand um den AprilTag mit einzuschließen – dieser ist sowohl in den PNGs also auch in den PDFs enthalten (siehe (a) in [Abb. 6.12](#)). Die Marker 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 Marker 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 jeden Apriltag nach außen hin begrenzende, durchgängige weiß-zu-schwarz-Kante bei einer präziseren Posenschätzung.

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

### 6.3.2.2 Markererkennung

Der erste Schritt der Markererkennung ist die Detektion der Marker auf dem Stereo-Bildpaar. Dieser Schritt benötigt die meiste Zeit und seine Präzision ist entscheidend für die Präzision der finalen Markerpose. Um die Dauer dieses Schritts zu kontrollieren, kann der Parameter *quality* vom Benutzer

konfiguriert werden. Er hat ein Herunterskalieren des Stereo-Bildpaares vor der Markererkennung zur Folge. *Hoch* (High) ergibt die höchste maximale Erkennungssdistanz und Präzision, aber auch die längste Dauer der Erkennung. *Niedrig* (Low) führt zur kleinsten maximalen Erkennungssdistanz 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* (Abschnitt 6.1.2) hat.

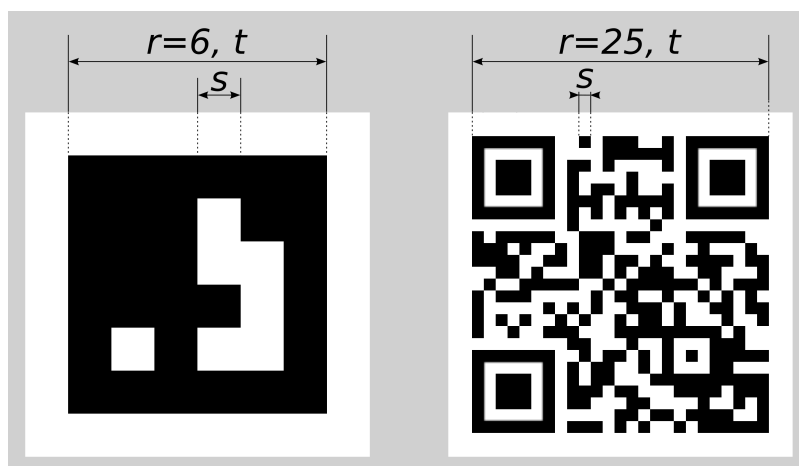


Abb. 6.13: Visualisierung der Modulgröße  $s$ , der Größe eines Markers in Modulen  $r$  und der Größe eines Markers in Metern  $t$  für QR-Codes (links) und AprilTags (rechts)

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 6.1.1.1) in Pixeln und  $s$  die Größe jedes Moduls in Metern bezeichnet.  $s$  kann leicht mit letztgenannter Formel berechnet werden, in welcher  $t$  der Markergröße in Metern und  $r$  der Breite des Markers in Modulen entspricht (bei AprilTags ohne den weißen Rahmen). [Abb. 6.13](#) 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 Markers 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. 6.18: Beispiele zur maximalen Erkennungsdistanz für AprilTags mit einer Breite von  $t = 4$  cm

AprilTag-Familie	Markerbreite	Maximale Distanz
36h11 (empfohlen)	8 Module	1.1 m
16h5	6 Module	1.4 m

Tab. 6.19: Beispiele zur maximalen Erkennungsdistanz für QR-Codes mit einer Breite von  $t = 8$  cm

Markerbreite	Maximale Distanz
29 Module	0.49 m
21 Module	0.70 m

### 6.3.2.3 Posenschätzung

Für jeden erkannten Marker wird dessen Pose im Kamerakoordinatensystem geschätzt. Eine Bedingung dafür ist, dass der Marker vollständig im linken und rechten Bild zu sehen ist. Das Koordinatensystem ist wie unten gezeigt am Marker ausgerichtet.

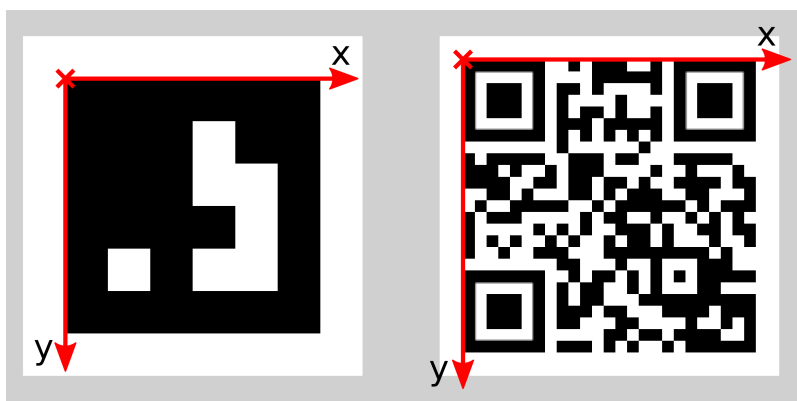


Abb. 6.14: Koordinatensysteme für AprilTags (links) bzw. QR-Codes (rechts)

Die z-Achse zeigt „in“ den Marker. 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 Markers geschätzt unter der Annahme, dass der Marker quadratisch ist. Bei QR-Codes bezieht sich die Größe auf den gesamten Marker, bei AprilTags dagegen nur auf den schwarzen Rand und nicht auf den äußeren weißen.

Der Benutzer kann auch die ungefähre Größe ( $\pm 10\%$ ) eines Markers angeben. Alle Marker, 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 Marker mit derselben ID im linken und rechten Bild sichtbar und diese Marker parallel zu den Bildzeilen ausgerichtet sind.

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

**Warnung:** Wir empfehlen, die ungefähre Größe der Marker anzugeben. Ansonsten, falls mehrere Marker mit derselben ID im linken oder rechten Bild sichtbar sind, kann es zu einer fehlerhaften Posenschätzung kommen, wenn die Marker 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 Marker.

Unten stehende Tabellen enthalten grobe Angaben zur Präzision der geschätzten Posen von AprilTags und QR-Codes. 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 Markers ist. Die Größe eines Markers hat keinen signifikanten Einfluss auf die Präzision in lateraler und z-Richtung. Im Allgemeinen verbessert ein größerer Marker allerdings die Präzision. Im Bezug auf die Präzision der Rotation, im speziellen um die x- und y-Achsen, übertreffen große Marker kleinere deutlich.

Tab. 6.20: Ungefähre Präzision der Pose von AprilTags

Distanz	<code>rc_visard 65</code> - lateral	<code>rc_visard 65</code> - z	<code>rc_visard 160</code> - lateral	<code>rc_visard 160</code> - z
0.3 m	0.4 mm	0.9 mm	0.4 mm	0.8 mm
1.0 m	0.7 mm	3.3 mm	0.7 mm	3.3 mm

Tab. 6.21: Ungefähre Präzision der Pose von QR-Codes

Distanz	<code>rc_visard 65</code> - lateral	<code>rc_visard 65</code> - z	<code>rc_visard 160</code> - lateral	<code>rc_visard 160</code> - z
0.3 m	0.6 mm	2.0 mm	0.6 mm	1.3 mm
1.0 m	2.6 mm	15 mm	2.6 mm	7.9 mm

### 6.3.2.4 Marker-Wiedererkennung

Jeder Marker 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 Marker mit derselben ID in einer Szene vorkommen können.

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

Die Marker-Wiedererkennung vergleicht die Positionen der Ecken der Marker in einem statischen Koordinatensystem, um identische Marker wiederzufinden. Marker 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 6.2.1) angeschaltet sein. Falls dies nicht der Fall ist, wird der Sensor als statisch angenommen. Die Marker-Wiedererkennung funktioniert in diesem Fall nicht über Bewegungen des Sensors hinweg.

Über den `max_corner_distance`-Parameter kann der Benutzer festlegen, wie weit ein Marker sich zwischen zwei Erkennungsläufen bewegen darf, um als identisch zu gelten. Der Parameter definiert die maximale Distanz zwischen den Ecken zweier Marker, was in [Abb. 6.15](#) dargestellt ist. Die euklidischen Abstände der vier zusammengehörenden Markerecken in 3D werden berechnet. Falls keiner dieser Abstände den Grenzwert überschreitet, gilt der Marker als wiedererkannt.

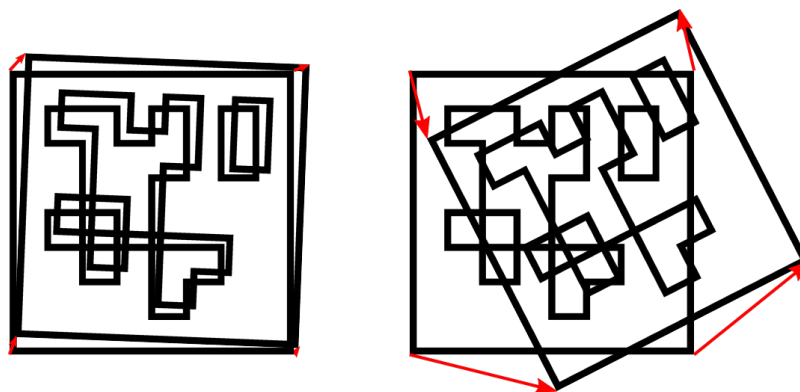


Abb. 6.15: Vereinfachte Darstellung der Marker-Wiedererkennung. Die euklidischen Abstände zwischen zusammengehörenden Markerecken in 3D werden berechnet (rote Pfeile).

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

### 6.3.2.5 Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das TagDetect-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die [Services](#) (Abschnitt 6.3.2.8) 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 6.4.1). 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.

### 6.3.2.6 Parameter

Es stehen zwei getrennte Module für die Markererkennung 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 7.3) 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. 6.22: 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 Markererkennungsläufen, nach denen ein vorher gesehener Marker während der Marker-Wiedererkennung verworfen wird
max_corner_distance	float64	0.001	0.01	0.005	Maximale Distanz zusammengehöriger Ecken zweier Marker während der Marker-Wiedererkennung
quality	string	-	-	High	Qualität der Markererkennung: [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>
```

### 6.3.2.7 Statuswerte

Die TagDetect-Module melden folgende Statuswerte:

Tab. 6.23: 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_timestamp_processed	Zeitstempel des letzten verarbeiteten Bilddatensatzes
state	Der aktuelle Zustand des Moduls
tag_detection_time	Berechnungszeit für die Markererkennung beim letzten Aufruf in Sekunden

Der Parameter state kann folgende Werte annehmen:



Tab. 6.24: Mögliche Zustände der TagDetect-Module

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

### 6.3.2.8 Services

Die TagDetect-Module implementieren einen Zustandsautomaten, welcher zum Starten und Stoppen genutzt werden kann. Die eigentliche Markererkennung 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 7.3) oder der `rc_visard Web GUI` (Abschnitt 7.1) ausprobiert und getestet werden.

#### `detect`

löst eine Markererkennung 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 Markererkennung 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 Marker-Wiedererkennung nicht. Es wird daher ausdrücklich empfohlen, das Modul manuell zu starten, bevor `detect` aufgerufen wird.

Marker können vom `detect`-Ergebnis aus mehreren Gründen ausgeschlossen werden, z.B. falls ein Marker nur in einem der Kamerabilder sichtbar war, oder falls die Posenschätzung fehlschlug. Diese herausgefilterten Marker werden im Log aufgelistet, auf welches wie in [Download der Logdateien](#) (Abschnitt 8.8) beschrieben zugegriffen werden kann.

Auf den Web GUI-Seiten der TagDetect-Module wird eine Visualisierung der letzten Markererkennung bereitgestellt. Diese Visualisierung wird allerdings erst angezeigt, sobald die Markererkennung mindestens einmal ausgeführt wurde. In der Web GUI kann die Markererkennung 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 (siehe [Zeitsynchronisierung](#), Abschnitt 7.6). 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 Marker-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 Marker zurückgegeben. Dieses Feature sollte nur während der Entwicklung einer Applikation oder zur Fehlerbehebung benutzt werden. Wann immer möglich sollten die konkreten Marker-IDs aufgelistet werden, zum einen zur Vermeidung von Fehldetektionen, zum anderen auch um die Markererkennung zu beschleunigen, da nicht benötigte Marker aussortiert werden können.

Bei AprilTags kann der Benutzer nicht nur einzelne Marker, sondern auch eine gesamte Familie spezifizieren, indem die ID auf „<family>“ gesetzt wird, bspw. „36h11“. Dadurch werden alle Marker dieser Familie erkannt. Es ist auch möglich, mehrere Familien oder eine Kombination aus Familien und einzelnen Markern 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 Markers angegeben werden. Wie in *Posenschätzung* (Abschnitt 6.3.2.3) erklärt, verhilft dies Mehrdeutigkeiten aufzulösen, die in bestimmten Situationen auftreten können.

Das Feld pose\_frame gibt an, ob die Posen im Kamera- oder im externen Koordinatensystem zurückgegeben werden (siehe *Hand-Auge-Kalibrierung*, Abschnitt 6.3.2.5). Der Standardwert ist camera.

Die Definition der Response 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",
        "size": "float64"
      }
    ]
  }
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}
}
```

**Response**

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

tags enthält alle erkannten Marker.

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

instance\_id ist der zufällige, eindeutige Identifikator eines Markers, welcher von der Marker-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 geschätzte Markergröße gesetzt. Bei AprilTags ist hier der weiße Rahmen nicht enthalten.

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}  
}
```

## start

startet das Modul durch einen Übergang von IDLE nach RUNNING.

### Details

Wenn das Modul läuft, empfängt es die Bilder der Stereokamera und ist bereit, Marker 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 Marker-Wiedererkennung-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"
    }
  }
}
```

### 6.3.2.9 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
-101	Ein interner Fehler trat während der Markererkennung 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 <code>message</code> .
201	Das Modul war nicht im Zustand <code>RUNNING</code> .

## 6.3.3 ItemPick und BoxPick

### 6.3.3.1 Einführung

Die `ItemPick` und `BoxPick` Module liefern eine gebrauchsfertige, modellfreie Perzeptionslösung, um robotische Pick-and-Place-Anwendungen für Vakuum-Greifsysteme zu realisieren. Dazu analysieren die

Module die sichtbare 3D-Szene, extrahieren mittels Clustering-Verfahren ebene Greifflächen und berechnen daraus mögliche 3D-Greifposen für die Positionierung des Sauggreifers.

Darüber hinaus bieten beide Module:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc\_visard Web GUI* (Abschnitt 7.1)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *RoiDB*, Abschnitt 6.5.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 6.3.1), 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
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) 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

**Bemerkung:** In diesem Kapitel werden die Begriffe Cluster und Oberfläche synonym verwendet und bezeichnen eine Menge von Punkten (oder Pixeln) mit ähnlichen geometrischen Eigenschaften.

Die ItemPick und BoxPick Module sind optional erhältliche Module, welche intern auf dem *rc\_visard* laufen und gesonderte ItemPick- bzw. BoxPick-*Lizenzen* (Abschnitt 8.7) benötigen.

### 6.3.3.2 Erkennung von Rechtecken (BoxPick)

Das BoxPick-Modul unterstützt die Erkennung von mehreren Objektmodellen (*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 von Boxen läuft in mehreren Schritten ab. Zuerst wird die Punktwolke in möglichst ebene Oberflächen (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 „Oberflächen“ 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.

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 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. Jedes erkannte Rechteck beinhaltet eine *uuid* (Universally Unique Identifier) und den Zeitstempel *timestamp* des ältesten Bildes, das für die Erkennung benutzt wurde.

### 6.3.3.3 Berechnung der Greifpunkte

Die ItemPick- und BoxPick-Module bieten 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.

Bei BoxPick werden die Greifpunkte auf den erkannten Rechtecken berechnet (siehe [Erkennung von Rechtecken \(BoxPick\)](#), Abschnitt 6.3.3.2).

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 6.5.1.3).
- 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 6.3.3.4) gegeben.

Ein vom ItemPick- oder 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.

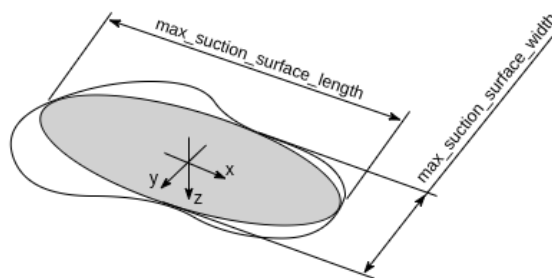


Abb. 6.16: Veranschaulichung eines berechneten Greifpunktes mit Greifpose und der zugehörigen Ellipse, welche die Greiffläche bestmöglich beschreibt.

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.

Im BoxPick-Modul entspricht der Greifpunkt dem Zentrum des detektierten Rechtecks, wobei die Achslängen der Greiffläche durch Länge und Breite des Rechtecks gegeben sind. Falls mehr als 15% der Rechtecksfläche ungültige Datenpunkte enthält oder durch andere Objekte verdeckt ist, wird dem Rechteck kein Greifpunkt zugeordnet.

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. Folgende Sortierstrategien sind verfügbar und können über die *Web GUI* (Abschnitt 7.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `gravity`: höchste 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.

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`.

#### 6.3.3.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem `rc_visard` laufenden Module liefern Daten für das `ItemPick`- und `BoxPick`-Modul oder haben Einfluss auf die Datenverarbeitung.

**Bemerkung:** Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten der `ItemPick`- und `Boxpick`-Module haben.

#### Stereokamera und Stereo-Matching

Folgende Daten werden vom `ItemPick`- und `BoxPick`-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera*-Moduls (`rc_camera`, Abschnitt 6.1.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching*-Moduls (`rc_stereomatching`, Abschnitt 6.1.2)

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

#### Schätzung der Gravitationsrichtung

Jedes Mal, wenn eine Erkennung oder Greifpunktberechnung innerhalb eines `Load Carriers` durchgeführt wird, schätzen die Module die Gravitationsrichtung wie in *Schätzung der Gravitationsrichtung* (Abschnitt 6.3.1.4) beschrieben.

#### 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 6.4.4) 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 6.1.2.5), 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 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren (siehe *Stereokamera-Parameter*, Abschnitt 6.1.1.3).

## Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, können die ItemPick- und BoxPick-Module automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.3.7) 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 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 6.4.1). 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.

## CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung der ItemPick und BoxPick Module aktiviert werden, indem die ID des benutzten Greifers und optional ein Greif-Offset an den `compute_grasps` Service übergeben werden. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 6.5.3.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2) gegeben.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in den Visualisierungen auf der *BoxPick*- und *ItemPick*-Seite der Web GUI kollidierende Greifpunkte als schwarze Ellipsen dargestellt.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 6.4.2.3) beschrieben.

### 6.3.3.5 Parameter

Die ItemPick- und BoxPick-Module werden in der REST-API als `rc_itempick` und `rc_boxpick` bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Module* → *ItemPick* und *Module* → *BoxPick* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.3) ändern.

## Übersicht über die Parameter

Diese Softwaremodule bieten folgende Laufzeitparameter:

Tab. 6.25: Anwendungsspezifische Laufzeitparameter der `rc_itempick` und `rc_boxpick` Module

Name	Typ	Min.	Max.	Default	Beschreibung
<code>max_grasps</code>	<code>int32</code>	1	20	5	Maximale Anzahl von bereitgestellten Greifpunkten

Tab. 6.26: Laufzeitparameter der `rc_itempick` und `rc_boxpick` Module für das Clustering-Verfahren

Name	Typ	Min.	Max.	Default	Beschreibung
<code>cluster_max_dimension</code>	float64	0.05	0.8	0.3	<b>Nur für <code>rc_itempick</code>.</b> Maximal erlaubter Durchmesser eines Clusters in Metern. Cluster mit einem größeren Durchmesser werden nicht für die Greifpunktberechnung berücksichtigt.
<code>cluster_max_curvature</code>	float64	0.005	0.5	0.11	Maximal erlaubte Krümmung für Greifflächen
<code>clustering_patch_size</code>	int32	3	10	4	<b>Nur für <code>rc_itempick</code>.</b> Pixelgröße der Patches für die Unterteilung des Tiefenbildes im ersten Clustering-Schritt
<code>clustering_max_surface_rmse</code>	float64	0.0005	0.01	0.004	Maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern
<code>clustering_discontinuity_factor</code>	float64	0.1	5.0	1.0	Erlaubte Unebenheit von Greifflächen

Tab. 6.27: Laufzeitparameter des `rc_boxpick` Moduls für die Rechteckerkennung

Name	Typ	Min.	Max.	Default	Beschreibung
<code>mode</code>	string	-	-	Unconstrained	Modus der Rechteckerkennung: [Unconstrained, Packed-GridLayout]
<code>manual_line_sensitivity</code>	bool	false	true	false	gibt an, ob die benutzerdefinierte Liniempfindlichkeit oder die automatische genutzt werden soll
<code>line_sensitivity</code>	float64	0.1	1.0	0.1	Empfindlichkeit des Liniendetektors
<code>prefer_splits</code>	bool	false	true	false	Gibt an, ob Rechtecke in kleinere Rechtecke gesplittet werden sollen, falls möglich

### Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf den *ItemPick*- bzw. *BoxPick*-Seiten 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|rc_boxpick>/parameters/  
↔parameters?max_grasps=<value>
```

**API Version 1 (veraltet)**

```
PUT http://<host>/api/v1/nodes/<rc_itempick|rc_boxpick>/parameters?max_grasps=<value>
```

**cluster\_max\_dimension (Nur für ItemPick, 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|rc_boxpick>/parameters/  
↔parameters?cluster_max_curvature=<value>
```

**API Version 1 (veraltet)**

```
PUT http://<host>/api/v1/nodes/<rc_itempick|rc_boxpick>/parameters?cluster_max_  
↔curvature=<value>
```

**clustering\_patch\_size (Nur für ItemPick, 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|rc_boxpick>/parameters/  
↔parameters?clustering_discontinuity_factor=<value>
```

**API Version 1 (veraltet)**

```
PUT http://<host>/api/v1/nodes/<rc_itempick|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_itempick|rc_boxpick>/parameters/  
↔parameters?clustering_max_surface_rmse=<value>
```

**API Version 1 (veraltet)**

```
PUT http://<host>/api/v1/nodes/<rc_itempick|rc_boxpick>/parameters?clustering_max_  
↔surface_rmse=<value>
```

**mode (Nur für BoxPick, Modus)**

legt den Modus der Rechteckerkennung fest. Mögliche Werte sind Unconstrained (*Unbeschränkt*) und PackedGridLayout (*Dichtes Gitterlayout*). Im Modus PackedGridLayout werden Rechtecke eines Clusters in einem dichten Gittermuster erkannt. Dieser Modus ist für viele Depalettieszenarien geeignet. Im Modus Unconstrained (Standardwert) werden Rechtecke unabhängig von ihren relativen Positionen zueinander erkannt. [Abb. 6.17](#) stellt verschiedene Szenarien für die beiden Modi dar.

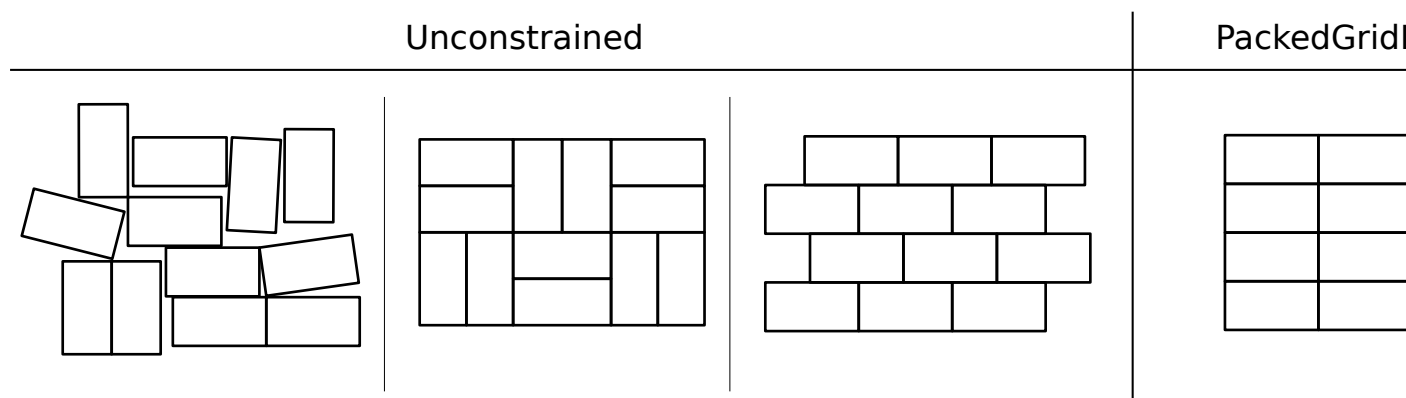


Abb. 6.17: Darstellung geeigneter Szenarien für die verschiedenen BoxPick Modi

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 (Nur für BoxPick, 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 „Oberflächenvisualisierung 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 (Nur für BoxPick, 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 „Oberflächen“ 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 (Nur für BoxPick, 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>
```

### 6.3.3.6 Statuswerte

Statuswerte der *rc\_itempick* und *rc\_boxpick* Module:

Tab. 6.28: Statuswerte der *rc\_itempick* und *rc\_boxpick* Module

Name	Beschreibung
<code>data_acquisition_time</code>	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste. Normalerweise sollte dieser Wert zwischen 0.5 und 0.6 Sekunden bei Tiefenbildern der Auflösung High liegen.
<code>grasp_computation_time</code>	Laufzeit für die Greifpunktberechnung beim letzten Aufruf in Sekunden
<code>last_timestamp_processed</code>	Zeitstempel des letzten verarbeiteten Bilddatensatzes
<code>load_carrier_detection_time</code>	Laufzeit für die letzte Load Carrier Erkennung in Sekunden
<code>state</code>	Aktueller Zustand des ItemPick- bzw. BoxPick-Moduls

Folgende state-Werte werden gemeldet.

Tab. 6.29: Mögliche Werte für den Zustand der ItemPick und Box-Pick Module

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.

### 6.3.3.7 Services

Die angebotenen Services von `rc_itempick` bzw. `rc_boxpick` können mithilfe der [REST-API-Schnittstelle](#) (Abschnitt 7.3) oder der [rc\\_visard Web GUI](#) (Abschnitt 7.1) ausprobiert und getestet werden.

Das ItemPick- bzw. BoxPick-Modul stellt folgende Services zur Verfügung.

#### detect\_items (nur BoxPick)

löst die Erkennung von Rechtecken aus, wie in [Erkennung von Rechtecken \(BoxPick\)](#) (Abschnitt 6.3.3.2) 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 6.3.3.4).

`item_models`: Liste von Rechtecken mit minimaler und maximaler Größe, wobei die minimale Größe kleiner als die maximale Größe sein muss. Der Typ muss immer `RECTANGLE` sein. Die Abmessungen sollten relativ genau angegeben werden, um Fehldetektionen zu verhindern, jedoch eine gewisse Toleranz beinhalten, um Messunsicherheiten und mögliche Produktionsabweichungen zu berücksichtigen.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe [Hand-Auge-Kalibrierung](#) (Abschnitt 6.3.3.4).

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 6.5.1.3).

`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.

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

```
{
  "args": {
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
```

(Fortsetzung auf der nächsten Seite)



(Fortsetzung der vorherigen Seite)

```

        "x": "float64",
        "y": "float64"
    }
  },
  "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.

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",
  "response": {
    "items": [
      {

```

(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",
"rectangle": {
  "x": "float64",
  "y": "float64"
},
"timestamp": {
  "nsec": "int32",
  "sec": "int32"
},
"type": "string",
"uuid": "string"
}
],
"load_carriers": [
  {
    "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",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "y": "float64"
      },
      "type": "string"
    }
  ],
  "return_code": {
    "message": "string",
    "value": "int16"
  },
  "timestamp": {
    "nsec": "int32",
    "sec": "int32"
  }
}
}
```

### compute\_grasps (für ItemPick)

löst die Erkennung von Greifpunkten für einen Sauggreifer aus, wie in *Berechnung der Greifpunkte* (Abschnitt 6.3.3.3) 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 6.3.3.4).

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 6.3.3.4).

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 6.5.1.3).

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.

item\_models: Liste von unbekanntem Objekten mit minimaler und maximaler Größe, wobei die minimale Größe kleiner als die maximale Größe sein muss. Nur ein Objekt item\_model vom Typ UNKNOWN wird aktuell unterstützt.

collision\_detection: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2)

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

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  },
  "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": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "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.

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",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

        "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 (für BoxPick)**

löst die Erkennung von Rechtecken und Berechnung von Greifposen für diese Rechtecke aus, wie in *Berechnung der Greifpunkte* (Abschnitt 6.3.3.3) 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 6.3.3.4).

`item_models`: Liste von Rechtecken mit minimaler und maximaler Größe, wobei die minimale Größe kleiner als die maximale Größe sein muss. Der Typ muss immer `RECTANGLE` sein. Die Abmessungen sollten relativ genau angegeben werden, um Fehldetektionen zu verhindern, jedoch eine gewisse Toleranz beinhalten, um Messunsicherheiten und mögliche Produktionsabweichungen zu berücksichtigen.

`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 6.3.3.4).

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 6.5.1.3).

`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 6.4.2.2)

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

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

(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"
      }
    }
  },
  "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).

items: Liste von erkannten Rechtecken.

grasps: sortierte Liste von Sauggreifpunkten.

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",

```

(Fortsetzung auf der nächsten Seite)



(Fortsetzung der vorherigen Seite)

```
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "pose_frame": "string",
    "quality": "float64",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    },
    "type": "string",
    "uuid": "string"
  }
],
"items": [
  {
    "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"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    },
    "type": "string",
    "uuid": "string"
  }
],
"load_carriers": [
  {
    "id": "string",
    "inner_dimensions": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "outer_dimensions": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  }
]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    },
    "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\_sorting\_strategies

speichert die gewählte Strategie zur Sortierung der Greifpunkte, die vom `compute_grasps` Service zurückgeliefert werden (siehe *Berechnung der Greifpunkte*, Abschnitt 6.3.3.3).

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_itempick|rc_boxpick>/services/set_
↔sorting_strategies
```

#### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|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.

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

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "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 vom `compute-grasps` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Berechnung der Greifpunkte*, Abschnitt 6.3.3.3).

### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_itempick|rc_boxpick>/services/get_
↔sorting_strategies
```

#### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|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",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "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|rc_boxpick>/services/reset_
↔defaults
```

### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|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": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}  
}
```

## 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|rc_boxpick>/services/reset_  
↔defaults
```

### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|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 dieses Moduls wieder her und wendet sie an („factory reset“). Dies betrifft nicht die gewählte Sortierstrategie.

### Details

Dieser Service kann wie folgt aufgerufen werden.

### API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/<rc_itempick|rc_boxpick>/services/reset_  
↔defaults
```

### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|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"
    }
  }
}
```

### set\_region\_of\_interest (veraltet)

speichert eine 3D Region of Interest auf dem *rc\_visard*.

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen [set\\_region\\_of\\_interest](#) (Abschnitt 6.5.2.4) in *rc\_roi\_db*.

#### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|rc_boxpick>/services/set_region_of_interest
```

### get\_regions\_of\_interest (veraltet)

gibt die mit *region\_of\_interest\_ids* spezifizierten, gespeicherten 3D Regions of Interest zurück.

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen [get\\_regions\\_of\\_interest](#) (Abschnitt 6.5.2.4) in *rc\_roi\_db*.

#### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_itempick|rc_boxpick>/services/get_regions_of_
↪interest
```

### delete\_regions\_of\_interest (veraltet)

löscht die mit *region\_of\_interest\_ids* spezifizierten, gespeicherten 3D Regions of Interest.

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen [delete\\_regions\\_of\\_interest](#) (Abschnitt 6.5.2.4) in *rc\_roi\_db*.

**API Version 1 (veraltet)**

```
PUT http://<host>/api/v1/nodes/<rc_itempick|rc_boxpick>/services/delete_regions_of_
↪interest
```

**6.3.3.8 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. 6.30: Rückgabecodes der Services des ItemPick- bzw. 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.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern oder ROIs überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
-301	Für die Anfrage zur Greifpunktberechnung <code>compute_grasps</code> wurden mehrere Objektmodelle ( <code>item_models</code> ) vom Typ UNKNOWN übergeben.
-302	Mehr als ein Load Carrier wurde für die Anfrage <code>detect_load_carriers</code> oder <code>detect_filling_level</code> angegeben. Momentan wird nur ein Load Carrier gleichzeitig unterstützt.
10	Die maximal speicherbare Anzahl an Load Carriern oder ROIs wurde erreicht.
11	Mit dem Aufruf von <code>set_load_carrier</code> oder <code>set_region_of_interest</code> wurde ein bereits existierendes Objekt mit derselben <code>id</code> 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 mit dem Load Carrier.
200	Das Modul ist im Zustand IDLE.
300	Ein gültiges <code>robot_pose</code> -Argument wurde angegeben, ist aber nicht erforderlich.
400	Der Serviceanfrage <code>compute_grasps</code> wurden keine Objektmodelle ( <code>item_models</code> ) als Argumente mitgegeben.

**6.3.4 SilhouetteMatch****6.3.4.1 Einführung**

Das SilhouetteMatch-Modul ist ein optionales Modul, welches intern auf dem `rc_visard` läuft, und benötigt eine eigene *Lizenz* (Abschnitt 8.7), welche erworben werden muss.

Das Modul erkennt Objekte, indem eine vordefinierte Silhouette („Template“) mit Kanten im Bild verglichen wird.

Für jedes Objekt, das mit dem SilhouetteMatch-Modul erkannt werden soll, wird ein Template benötigt. Roboception bietet hierfür einen Template-Generierungsservice auf ihrer [Website \(https://roboception.com/de/template-request-de/\)](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 7.1)
- eine *REST-API-Schnittstelle* (Abschnitt 7.3) und eine *KUKA Ethernet KRL Schnittstelle* (Abschnitt 7.5)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche des Kamerabilds auszuwählen (siehe *Setzen einer Region of Interest*, Abschnitt 6.3.4.3)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 6.3.1), 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
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern
- Auswahl einer Strategie zum Sortieren der erkannten Objekte und zurückgelieferten Greifpunkte

### Taugliche Objekte

Das SilhouetteMatch-Modul ist für Objekte ausgelegt, die prägnante Kanten auf einer Ebene besitzen, welche parallel zu der Basisebene ist, auf der die Objekte liegen. Das trifft beispielsweise auf flache, nicht-transparente 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.

Das SilhouetteMatch-Modul funktioniert am besten für Objekte, die auf einer texturlosen Basisebene liegen. Die Farbe der Basisebene sollte so gewählt werden, dass im Intensitätsbild ein klarer Kontrast zwischen den Objekten und der Basisebene sichtbar ist.

### 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.
- Alle sichtbaren Objekte befinden sich auf einer gemeinsamen Basisebene, welche kalibriert werden muss.
- Die Verkippung der Basisebene zur Blickrichtung der Kamera darf 10 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.

#### 6.3.4.2 Kalibrierung der Basisebene

Bevor Objekte erkannt werden können, muss die Basisebene kalibriert werden. 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 6.3.2) 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 Kalibrieremethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 7.3) 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 7.3) 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 7.3) und nicht über die `rc_visard` Web GUI verfügbar.

#### 6.3.4.3 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 6.5.2.2) beschrieben wird.

#### 6.3.4.4 Setzen von Greifpunkten

Um das `SilhouetteMatch`-Modul direkt in einer Roboteranwendung zu nutzen, können für jedes Template 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. 6.18](#)).

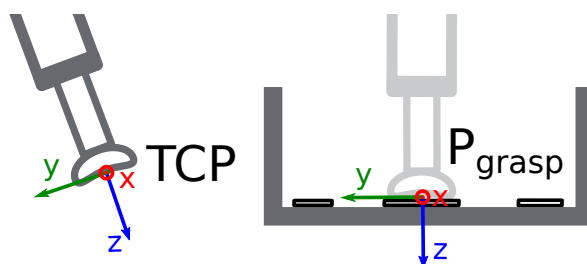


Abb. 6.18: Definition von Greifpunkten bezogen auf den Roboter-TCP

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 7.3), oder über die interaktive Visualisierung in der Web GUI definiert werden. Der `rc_visard` kann bis zu 50 Greifpunkte pro Template speichern.

Wird ein Greifpunkt auf einem symmetrischen Objekt definiert, werden alle Greifpunkte, die zu diesem symmetrisch sind, automatisch im `detect_object` 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.

### 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 er Greifpunkt wird 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 hineingerichtet 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 aktiv ist (das ist der Standardwert), kann der Greifpunkt durch Klicken auf *Verschieben* und anschließendes Klicken auf den Greifpunkt über die Oberfläche des Modells oder des Templates zur gewünschten Position bewegt werden. Die Orientierung des Greifpunkts um die Oberflächennormale kann ebenfalls interaktiv verändert werden, in dem auf *Rotieren* geklickt wird und anschließend der Greifpunkt mit der Maus rotiert wird. Wenn *An Oberfläche anheften* nicht aktiv ist, kann der Greifpunkt mit der Maus frei in allen drei Raumrichtungen verschoben und rotiert werden.

Wenn das Template Symmetrien hat, können die Greifpunkte, die symmetrisch zum definierten Greifpunkt sind, durch Klick auf *symmetrische Greifpunkte anzeigen* angezeigt werden.

### Setzen von Greifpunkten über die REST-API

Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 7.3) mithilfe des `set_grasp` oder `set_all_grasps` Services gesetzt werden (siehe *Services*, Abschnitt 6.3.4.11). Im *SilhouetteMatch*-Modul besteht ein Greifpunkt 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.

#### 6.3.4.5 Setzen der bevorzugten TCP-Orientierung

Das *SilhouetteMatch*-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des Greifers oder 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 der Sensor am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden, damit die bevorzugte Orientierung zur Filterung und optional zur Sortierung der Greifpunkte auf den erkannten

Objekten genutzt werden kann. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera als die bevorzugte TCP-Orientierung genutzt.

#### 6.3.4.6 Setzen der Sortierstrategie

Die vom `detect_object` 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 7.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `preferred_orientation`: Objekte und Greifpunkte mit der geringsten Rotationsänderung zwischen ihrer Orientierung und der bevorzugten 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.

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.

#### 6.3.4.7 Objekterkennung

Objekte können 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.

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 6.3.4.8)

Optional können auch folgende Serviceargumente an das `SilhouetteMatch`-Modul übergeben werden:

- ein Versatz, 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.
- 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 6.3.4.8) gegeben.

Im *Ausprobieren*-Abschnitt der Seite *SilhouetteMatch* der *Web GUI* kann die Objektdetektion ausprobiert werden. Das Ergebnis wird, wie in [Abb. 6.19](#) dargestellt, visualisiert.

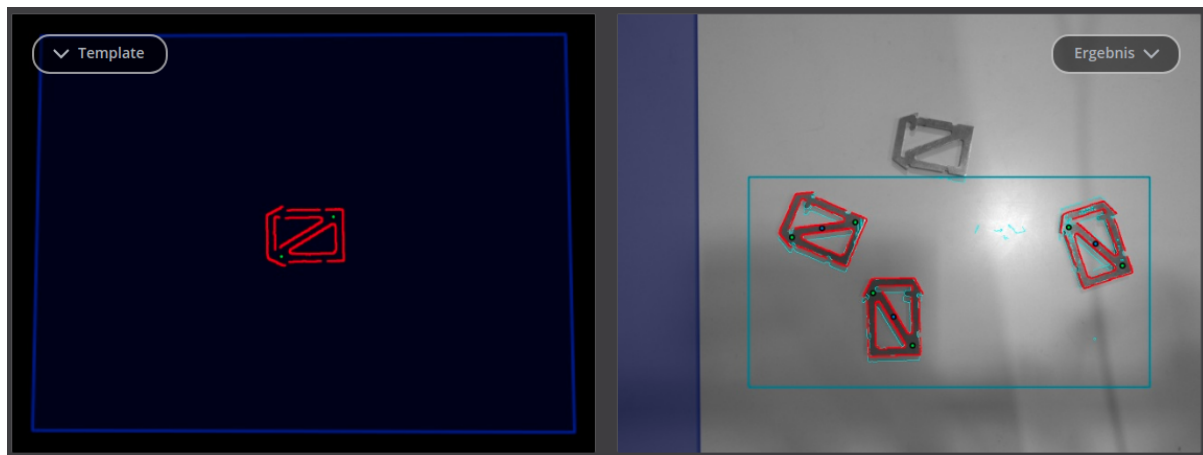


Abb. 6.19: Ergebnisbild des SilhouetteMatch-Moduls, wie über die Web GUI dargestellt

Das linke Bild zeigt die kalibrierte Basisebene in blau und das zu erkennende Template in rot mit den Greifpunkten (siehe [Setzen von Greifpunkten](#), Abschnitt 6.3.4.4) in grün. Das Template wird passend zu Abstand und Verkippung der Basisebene verformt dargestellt.

Das rechte Bild zeigt das Detektionsergebnis. Die blauschattierte 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. Die gewählte Region of Interest wird als petrolfarbenes Rechteck dargestellt. Erkannte Kanten im Bild werden in hellem Blau und erkannte Objekte (instances) in rot visualisiert. Blaue Punkte markieren jeweils den Ursprung der detektierten Objekte, wie im Template festgelegt. Erreichbare Greifpunkte sind als grüne Punkte dargestellt. Nicht erreichbare Greifpunkte werden als rote Punkte visualisiert (nicht im Bild dargestellt).

Die Posen der Objektursprünge werden im gewählten Koordinatensystem zurückgegeben. Die Orientierung der erkannten Objekte ist mit der Normalen der Basisebene ausgerichtet. 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 6.3.4.6). 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.

#### 6.3.4.8 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.

### Stereokamera und Stereo-Matching

Das SilhouetteMatch-Modul verarbeitet intern die rektifizierten Bilder des *Kamera*-Moduls (*rc\_camera*, Abschnitt 6.1.1). Es sollte deshalb auf eine passende Belichtungszeit geachtet werden, um optimale Ergebnisse zu erhalten.

Für die Kalibrierung der Basisebene mit der Stereo-Methode wird das Disparitätsbild des *Stereo-Matching*-Moduls (*rc\_stereomatching*, Abschnitt 6.1.2) verarbeitet. Abgesehen davon 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 6.1.2.5) für *Stereo-Matching* zu aktivieren.

### Schätzung der Gravitationsrichtung

Jedes Mal, wenn eine Objekterkennung innerhalb eines Load Carriers durchgeführt wird, schätzt das Modul die Gravitationsrichtung wie in *Schätzung der Gravitationsrichtung* (Abschnitt 6.3.1.4) beschrieben.

### IOControl und Projektor-Kontrolle

Wenn der *rc\_visard* in Verbindung mit einem externen Musterprojektor und dem Modul *IOControl und Projektor-Kontrolle* (*rc\_iocontrol*, Abschnitt 6.4.4) betrieben wird, sollte der Projektor für die stereobasierte Kalibrierung der Basisebene benutzt werden.

Das projizierte Muster darf während der Objektdetektion nicht im linken oder rechten Kamerabild sichtbar sein, da es den Detektionsvorgang behindert. Der Projektor sollte deshalb entweder ausgeschaltet sein oder im Modus *ExposureAlternateActive* betrieben werden.

### 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 6.3.4.11) 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 6.4.1). 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` und der bevorzugten TCP-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 bevorzugte TCP-Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn `camera` als `pose_frame` ausgewählt ist und die bevorzugte TCP-Orientierung in `external` definiert ist, 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` oder `detect_objects` 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.

## 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 6.5.3.2) und Details über die Kollisionsprüfung werden in [Integrierte Kollisionsprüfung in anderen Modulen](#) (Abschnitt 6.4.2.2) gegeben. 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 die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in der Ergebnis-Visualisierung oben auf der `SilhouetteMatch`-Seite der Web GUI kollidierende Greifpunkte als rote Punkte dargestellt. Die Objekte, die bei der Kollisionsprüfung betrachtet werden, werden auch mit roten Kanten visualisiert.

Die Laufzeitparameter des `CollisionCheck`-Moduls beeinflussen die Kollisionserkennung wie in [CollisionCheck-Parameter](#) (Abschnitt 6.4.2.3) beschrieben.

### 6.3.4.9 Parameter

Das `SilhouetteMatch`-Modul wird in der REST-API als `rc_silhouettematch` bezeichnet und in der [Web GUI](#) (Abschnitt 7.1) unter `Module` → `SilhouetteMatch` dargestellt. Der Benutzer kann die Parameter entweder dort oder über die [REST-API-Schnittstelle](#) (Abschnitt 7.3) ändern.

## Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.31: Laufzeitparameter des rc\_silhouettematch-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
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
edge_sensitivity	float64	0.1	1.0	0.6	Empfindlichkeit der Kantenerkennung
match_max_distance	float64	0.1	10.0	2.5	Der maximale tolerierte Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln
match_percentile	float64	0.7	1.0	0.85	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
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 in den Kamerabildern 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.

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>
```

### check\_collisions\_with\_base\_plane (Kollisionsprüfung mit Basisebene)

Wenn dieser Parameter 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 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)

Wenn dieser Parameter 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>
```

### 6.3.4.10 Statuswerte

Dieses Modul meldet folgende Statuswerte.

Tab. 6.32: Statuswerte des rc\_silhouettematch-Moduls

Name	Beschreibung
data_acquisition_time	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
load_carrier_detection_time	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden
detect_service_time	Berechnungszeit für die Objekterkennung, einschließlich der Zeit für Datenaufnahme und Load Carrier Erkennung
last_timestamp_processed	Zeitstempel des letzten verarbeiteten Bilddatensatzes

### 6.3.4.11 Services

Die angebotenen Services des rc\_silhouettematch-Moduls können mithilfe der [REST-API-Schnittstelle](#) (Abschnitt 7.3) oder der [rc\\_visard Web GUI](#) (Abschnitt 7.1) ausprobiert und getestet werden.

Das SilhouetteMatch-Modul bietet folgende Services.

#### detect\_object

führt eine Objekterkennung durch, wie in [Objekterkennung](#) (Abschnitt 6.3.4.7) 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 6.3.4.8).

Potentiell obligatorische Serviceargumente:

- robot\_pose: siehe [Hand-Auge-Kalibrierung](#) (Abschnitt 6.3.4.8).

Optionale Serviceargumente:

- 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 6.4.2.2)

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_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.

`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": [
      {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
"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",
"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"
      }
    },
    "pose_frame": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    },
    "uuid": "string"
  }
],
"load_carriers": [
  {
    "id": "string",
    "inner_dimensions": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "outer_dimensions": {
      "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "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"
}
}
```

### calibrate\_base\_plane

führt die Kalibrierung der Basisebene durch, wie in *Kalibrierung der Basisebene* (Abschnitt 6.3.4.2) 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 6.3.4.8).

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 6.3.4.8).

`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"
    }
  }
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    },
    "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"
    }
  }
}

```

**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 6.3.4.8).

Potentiell obligatorische Serviceargumente:



robot\_pose: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.4.8).

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",
        "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` Service zurückgelieferten Greifpunkte verwendet wird (siehe [Setzen der bevorzugten TCP-Orientierung](#), Abschnitt 6.3.4.5).

### 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",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "value": "int16"
  }
}
```

### get\_preferred\_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional zur Sortierung der vom `detect_object` Service zurückgelieferten Greifpunkte verwendet wird (siehe [Setzen der bevorzugten TCP-Orientierung](#), Abschnitt 6.3.4.5).

#### 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` Service zurückgeliefert werden (siehe [Objekterkennung](#), Abschnitt 6.3.4.7).

#### 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.

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

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "preferred_orientation": {
      "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` Service zurückgelieferten Objekte und Greifpunkte verwendet wird (siehe [Objekterkennung](#), Abschnitt 6.3.4.7).

### 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"
    },
    "preferred_orientation": {
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

**reset\_defaults**

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“). Dies betrifft nicht die konfigurierten Templates, die Kalibrierung der Basisebene, die bevorzugte TCP-Orientierung und die gewählte Sortierstrategie.

**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": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

### set\_region\_of\_interest\_2d (veraltet)

speichert eine 2D Region of Interest auf dem *rc\_visard*.

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen [set\\_region\\_of\\_interest\\_2d](#) (Abschnitt 6.5.2.4) in *rc\_roi\_db*.

#### API Version 1 (veraltet)

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

### get\_regions\_of\_interest\_2d (veraltet)

gibt die mit *region\_of\_interest\_2d\_ids* spezifizierten, gespeicherten 2D Regions of Interest zurück.

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen [get\\_regions\\_of\\_interest\\_2d](#) (Abschnitt 6.5.2.4) in *rc\_roi\_db*.

#### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_regions_of_interest_2d
```

### delete\_regions\_of\_interest\_2d (veraltet)

löscht die mit *region\_of\_interest\_2d\_ids* spezifizierten, gespeicherten 2D Regions of Interest.

#### Details

Dieser Service kann wie folgt aufgerufen werden.

#### API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen [delete\\_regions\\_of\\_interest\\_2d](#) (Abschnitt 6.5.2.4) in *rc\_roi\_db*.

#### API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_regions_of_interest_
↔2d
```

### 6.3.4.12 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 6.3.4.4) beschrieben.

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

```
{
  "args": {
    "grasp": {
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "template_id": "string"
  }
}
```

#### Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_grasp",
  "response": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

"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 6.3.4.4) beschrieben.

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

```

{
  "args": {
    "grasps": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "template_id": "string"
      }
    ],
    "template_id": "string"
  }
}

```

#### Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```

{
  "name": "set_all_grasps",
  "response": {

```

(Fortsetzung auf der nächsten Seite)



(Fortsetzung der vorherigen Seite)

```

"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": [
      {
        "id": "string",
        "pose": {
          "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"
        },
        "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"
    ]
  }
}

```

#### Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```

{
  "name": "delete_grasps",
  "response": {
    "return_code": {
      "message": "string",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    "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 der *Request*-Argumente mit jeweiligen Datentypen ist:

```

{
  "args": {
    "grasp": {
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "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 6.3.4.4) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```

{
  "name": "get_symmetric_grasps",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
"response": {
  "grasps": [
    {
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "template_id": "string"
    }
  ],
  "return_code": {
    "message": "string",
    "value": "int16"
  }
}
```

#### 6.3.4.13 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. 6.33: 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 Verkipfung 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.
151	Das Objekt-Template hat eine kontinuierliche Symmetrie.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

#### 6.3.4.14 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"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}  
]
```

**Antwort-Header**

- **Content-Type** – application/json

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabewert: Array der Templates*)
- **404 Not Found** – Modul nicht gefunden

**Referenzierte Datenmodelle**

- *Template* (Abschnitt 7.3.4)

**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/octet-stream

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabewert: Template*)
- **404 Not Found** – Modul oder Template wurden nicht gefunden.

**Referenzierte Datenmodelle**

- *Template* (Abschnitt 7.3.4)

**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 (*obligatorisch*)

**Anfrage-Header**

- **Accept** – multipart/form-data application/json

**Antwort-Header**

- **Content-Type** – application/json

**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 7.3.4)

**DELETE** /templates/rc\_silhouettematch/{id}  
entfernt ein rc\_silhouettematch-Template.

**Musteranfrage**

```
DELETE /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: application/json
```

**Parameter**

- **id** (*string*) – ID des Templates (*obligatorisch*)

**Anfrage-Header**

- **Accept** – application/json

**Antwort-Header**

- **Content-Type** – application/json

**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.

## 6.4 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 6.4.1**) ermöglicht dem Benutzer, die Kamera entweder über die Web GUI oder die REST-API zu einem Roboter zu kalibrieren.
- **CollisionCheck** (`rc_collision_check`, **Abschnitt 6.4.2**) bietet eine einfache Möglichkeit zu prüfen, ob ein Greifer in Kollision ist.
- **Kamerakalibrierung** (`rc_stereocalib`, **Abschnitt 6.4.3**) ermöglicht die Überprüfung und Durchführung der Kamerakalibrierung über die *Web GUI* (Abschnitt 7.1).
- **IOControl und Projektor-Kontrolle** (`rc_iocontrol`, **Abschnitt 6.4.4**) bietet die Kontrolle über die Ein- und Ausgänge des Sensors mit speziellen Betriebsarten zur Kontrolle eines externen Musterprojektors.

### 6.4.1 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 Kalibrieroutine, 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 *Kalibrieroutine* (Abschnitt 6.4.1.3) ist ein benutzerfreundliches mehrstufiges Verfahren, für das mit einem Kalibriermuster gearbeitet wird. Entsprechende Kalibriermuster können von Roboception bezogen werden.

#### 6.4.1.1 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 7.3) des `rc_visard` enthalten. Der REST-API-Name dieses Moduls lautet `rc_hand_eye_calibration` und seine Services werden in *Services* (Abschnitt 6.4.1.5) 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 7.1) einen geführten Prozess vor, mit dem sich die Kalibrieroutine **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.

### 6.4.1.2 Kameramontage

Wie in [Abb. 6.20](#) und [Abb. 6.22](#) 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 *Kalibrieroutine* (Abschnitt 6.4.1.3) 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 Kalibrierusters.

**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. 6.20](#)).

**Warnung:** Es ist äußerst wichtig, dass sich das Kalibriermuster in Schritt 2 der *Kalibrieroutine* (Abschnitt 6.4.1.3) 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 *Kalibrieroutine*, Abschnitt 6.4.1.3) ist eine Pose  $\mathbf{T}_{\text{camera}}^{\text{robot}}$ , die die (zuvor unbekannt) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{robot}} = \mathbf{R}_{\text{camera}}^{\text{robot}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}_{\text{camera}}^{\text{robot}}, \quad (6.3)$$

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  $3 \times 3$  Drehmatrix und den  $3 \times 1$  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 12.1).

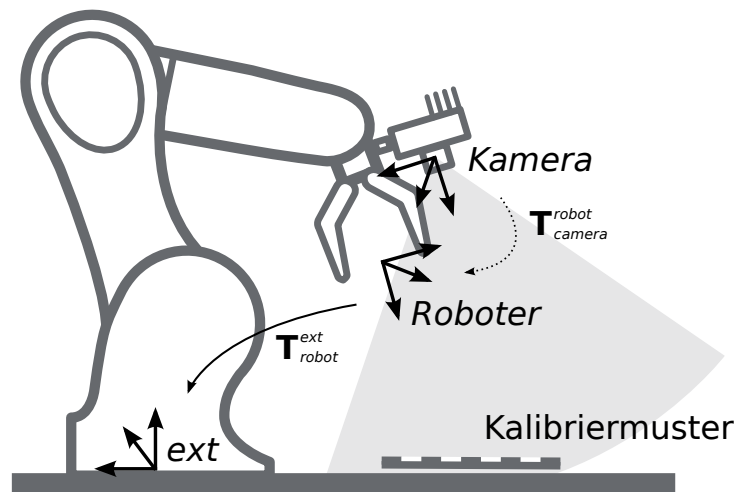


Abb. 6.20: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer robotergeführten Kamera: Sie wird mit einer festen relativen Position zu einem benutzerdefinierten *Roboter*-Koordinatensystem (z.B. Flansch oder Werkzeugmittelpunkt) montiert. Es ist wichtig, dass die Pose  $T_{robot}^{ext}$  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  $T_{camera}^{robot}$ , 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. 6.21](#) 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.

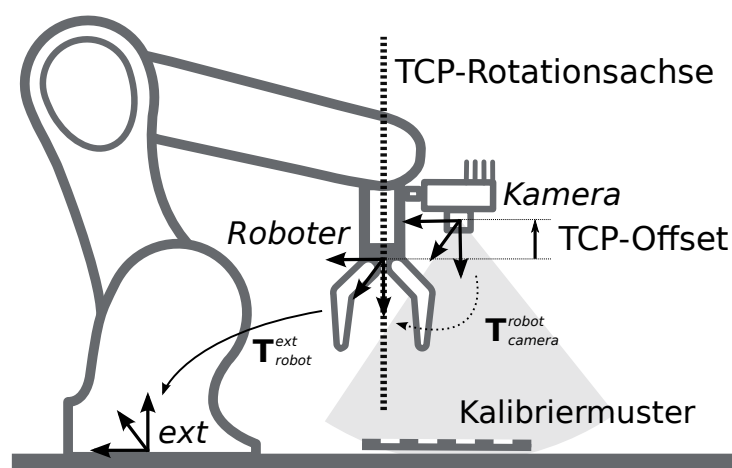


Abb. 6.21: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

**Kalibrierung einer statisch montierten Kamera** In Anwendungsfällen, bei denen die Kamera statisch verglichen zum Roboter montiert wird, muss das Kalibrieremuster, wie im Beispiel in [Abb.](#)

6.22 und Abb. 6.23 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 Kalibrierusters zu diesem Koordinatensystem weder bekannt sein muss, noch für die Kalibrierroutine relevant ist (siehe in Abb. 6.23).

**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 6.4.1.3) vom Benutzer definierte *Roboter*-Koordinatensystem nicht verändert.

In diesem Anwendungsfall ist das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 6.4.1.3) die Pose  $\mathbf{T}_{\text{camera}}^{\text{ext}}$ , die die (zuvor unbekannt) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{ext}} = \mathbf{R}_{\text{camera}}^{\text{ext}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}_{\text{camera}}^{\text{ext}}, \quad (6.4)$$

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  $\mathbf{R}_{\text{camera}}^{\text{ext}}$  sowie  $\mathbf{t}_{\text{camera}}^{\text{ext}}$  die  $3 \times 3$  Rotationsmatrix und  $3 \times 1$  Translationsvektor der Pose  $\mathbf{T}_{\text{camera}}^{\text{ext}}$  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 12.1).

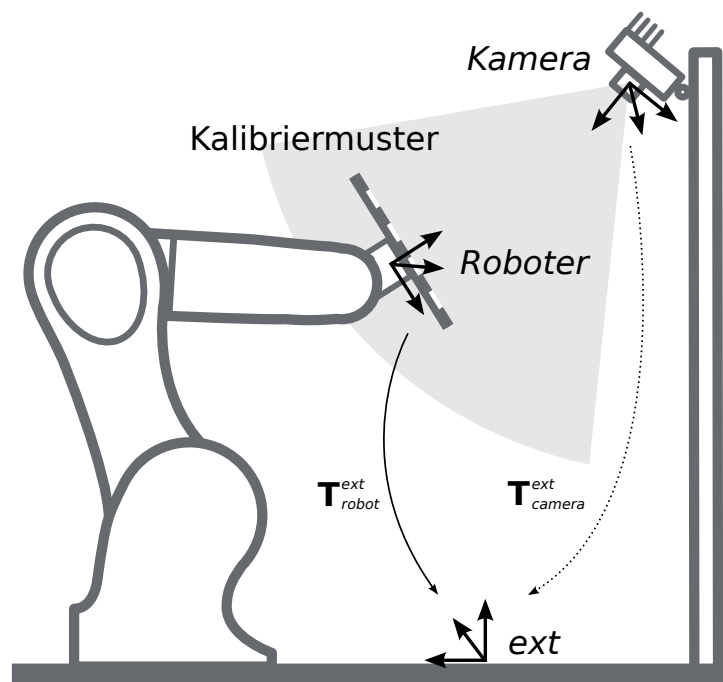


Abb. 6.22: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer statisch montierten Kamera: Sie wird mit einer festen Position relativ zu einem benutzerdefinierten externen Referenzkoordinatensystem *ext* (z.B. Weltkoordinatensystem oder Roboter-Montagepunkt) montiert. Es ist wichtig, dass die Pose  $\mathbf{T}_{\text{robot}}^{\text{ext}}$  des benutzerdefinierten *Roboter*-Koordinatensystems in Bezug auf dieses Koordinatensystem während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibrierprozesses ist die gewünschte Kalibriertransformation  $\mathbf{T}_{\text{camera}}^{\text{ext}}$ , d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten externen Koordinatensystem *ext*.

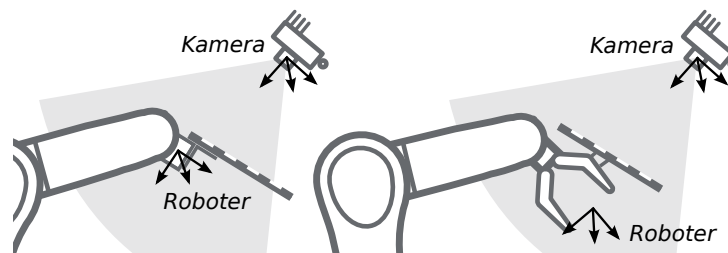


Abb. 6.23: Alternative Montageoptionen für die Befestigung des Kalibrieramusters am Roboter

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 Kalibrieramusters entlang der TCP-Rotationsachse angegeben werden. Das Kalibrieramuster muss so angebracht werden, dass die TCP-Rotationsachse orthogonal zum Kalibrieramuster verläuft. [Abb. 6.24](#) zeigt die Situation.

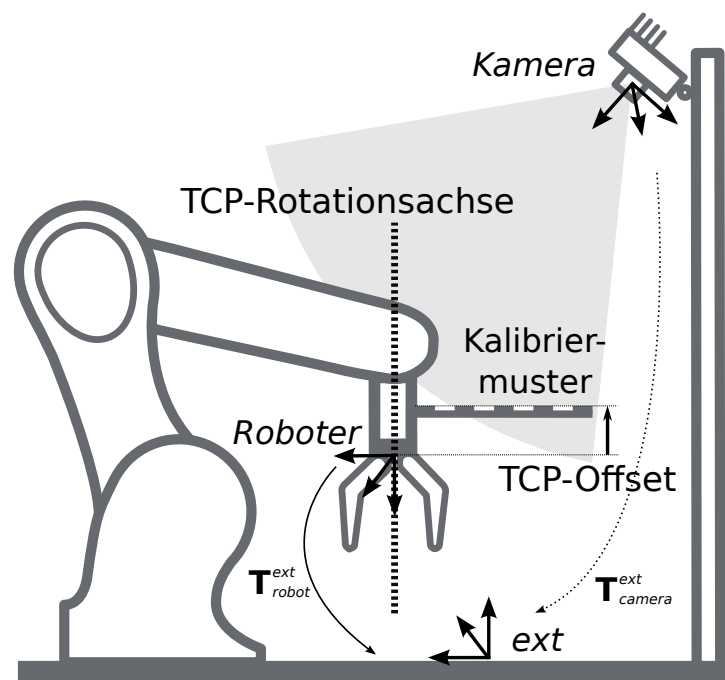


Abb. 6.24: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zur sichtbaren Oberfläche des Kalibrieramusters entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

### 6.4.1.3 Kalibrierroutine

Die Hand-Auge-Kalibrierung kann manuell über die [Web GUI](#) (Abschnitt 7.1) oder programmgesteuert über die [REST-API-Schnittstelle](#) (Abschnitt 7.3) 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. 6.25).

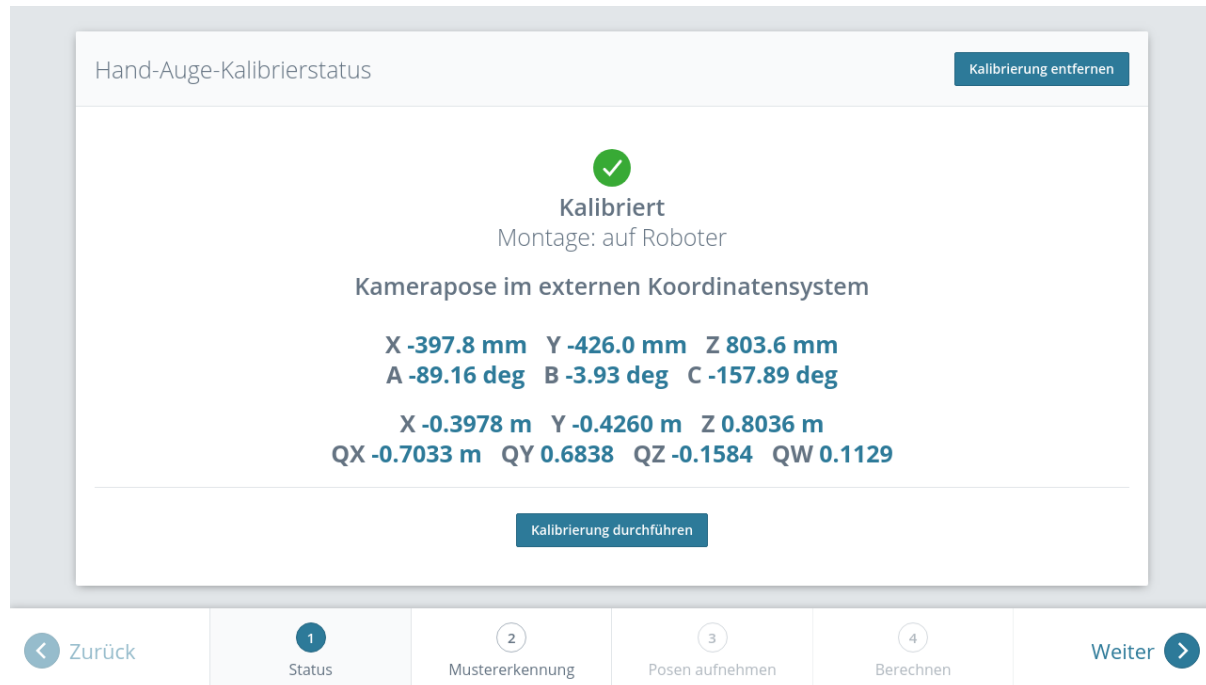


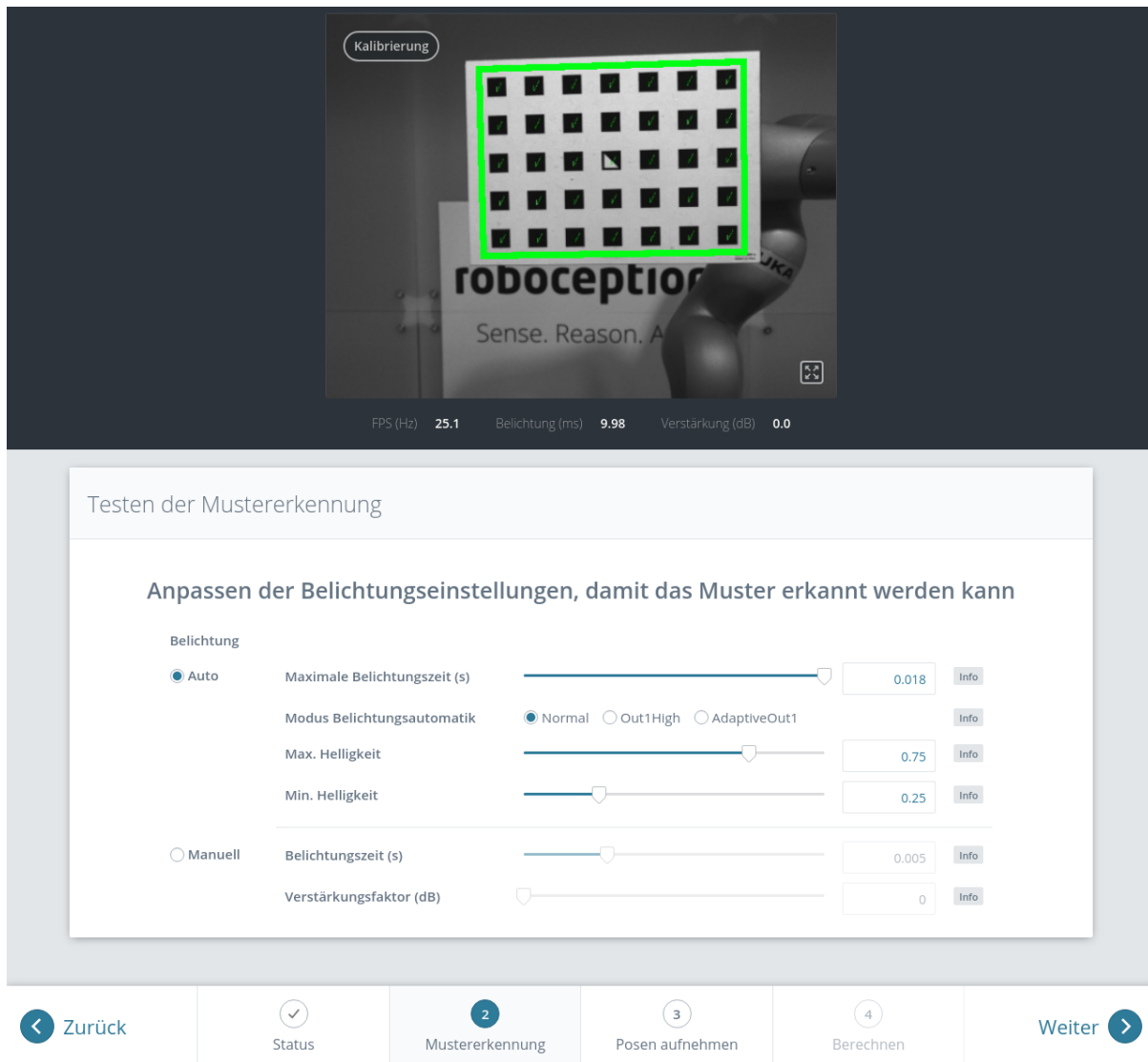
Abb. 6.25: Aktueller Status der Hand-Auge-Kalibrierung falls eine Hand-Auge-Kalibrierung gespeichert ist

Um den Status der Hand-Auge-Kalibrierung programmgesteuert abzufragen bietet die REST-API den Service `get_calibration` (siehe [Services](#), Abschnitt 6.4.1.5). Eine vorhandene Hand-Auge-Kalibrierung kann über *Kalibrierung entfernen* oder den REST-API Service `remove_calibration` (siehe [Services](#), Abschnitt 6.4.1.5) 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. Die erfolgreiche Erkennung des Kalibriermusters wird durch grüne Häkchen auf jedem Quadrat und einen dicken grünen Rahmen um das Kalibriermuster visualisiert, wie in Abb. 6.26 dargestellt ist.



The screenshot displays the 'Kalibrierung' (Calibration) interface. At the top, a video feed shows a 5x5 grid of markers on a robot arm, with a green box highlighting the grid. Below the video, the following parameters are shown: FPS (Hz) 25.1, Belichtung (ms) 9.98, and Verstärkung (dB) 0.0.

The main configuration panel is titled 'Testen der Mustererkennung' (Testing Pattern Recognition) and contains the following settings:

- Anpassen der Belichtungseinstellungen, damit das Muster erkannt werden kann** (Adjust lighting settings so the pattern can be recognized)
- Belichtung** (Lighting):
  - Auto
    - Maximale Belichtungszeit (s): 0.018 (Info)
    - Modus Belichtungsautomatik:  Normal,  Out1High,  AdaptiveOut1 (Info)
    - Max. Helligkeit: 0.75 (Info)
    - Min. Helligkeit: 0.25 (Info)
  - Manuell
    - Belichtungszeit (s): 0.005 (Info)
    - Verstärkungsfaktor (dB): 0 (Info)

At the bottom, a navigation bar shows the following steps: Zurück, Status, **2 Mustererkennung**, 3 Posen aufnehmen, 4 Berechnen, and Weiter.

Abb. 6.26: Testen der Mustererkennung

### Schritt 3: Posen aufnehmen

In diesem Schritt werden Bilder des Kalibrieremusters an verschiedenen Roboterposen aufgenommen. Dabei ist sicherzustellen, dass das Kalibrieremuster bei allen Posen im linken Kamerabild vollständig sichtbar ist. Zudem müssen die Roboterpositionen sorgsam ausgewählt werden, damit das Kalibrieremuster aus unterschiedlichen Perspektiven aufgenommen wird. [Abb. 6.27](#) zeigt eine schematische Darstellung der empfohlenen acht Ansichten.

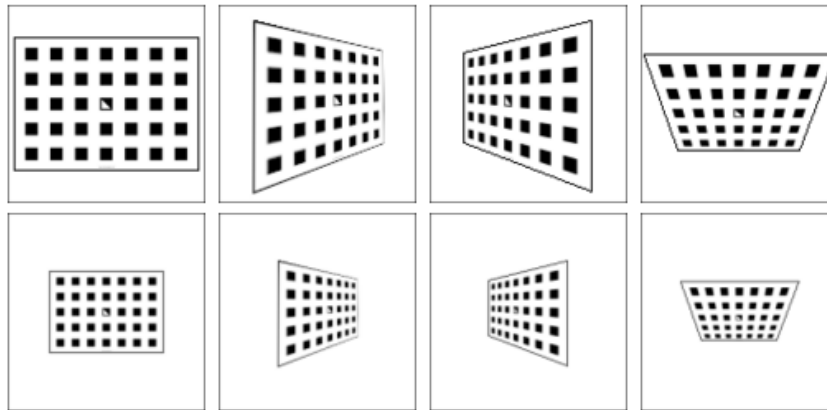


Abb. 6.27: Empfohlene Ansichten des Kalibrierusters während des Kalibriervorgangs. Im Fall von 4DOF-Robotern müssen andere Ansichten gewählt werden, welche so unterschiedlich wie möglich sein sollten.

**Warnung:** Die Kalibrierqualität, d.h. die Genauigkeit des berechneten Kalibrierergebnisses, hängt von den Ansichten des Kalibrierusters 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  $T_{\text{robot}}^{\text{ext}}$  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 zwei verschiedenen Formaten für die Kalibrierposen wählen: *XYZABC* oder *XYZ+Quaternion* (siehe [Formate für Posendaten](#), Abschnitt 12.1). 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 Kalibrierusters angezeigt. Der Roboter sollte für jeden Slot so bewegt werden, dass die empfohlene Ansicht erreicht wird.

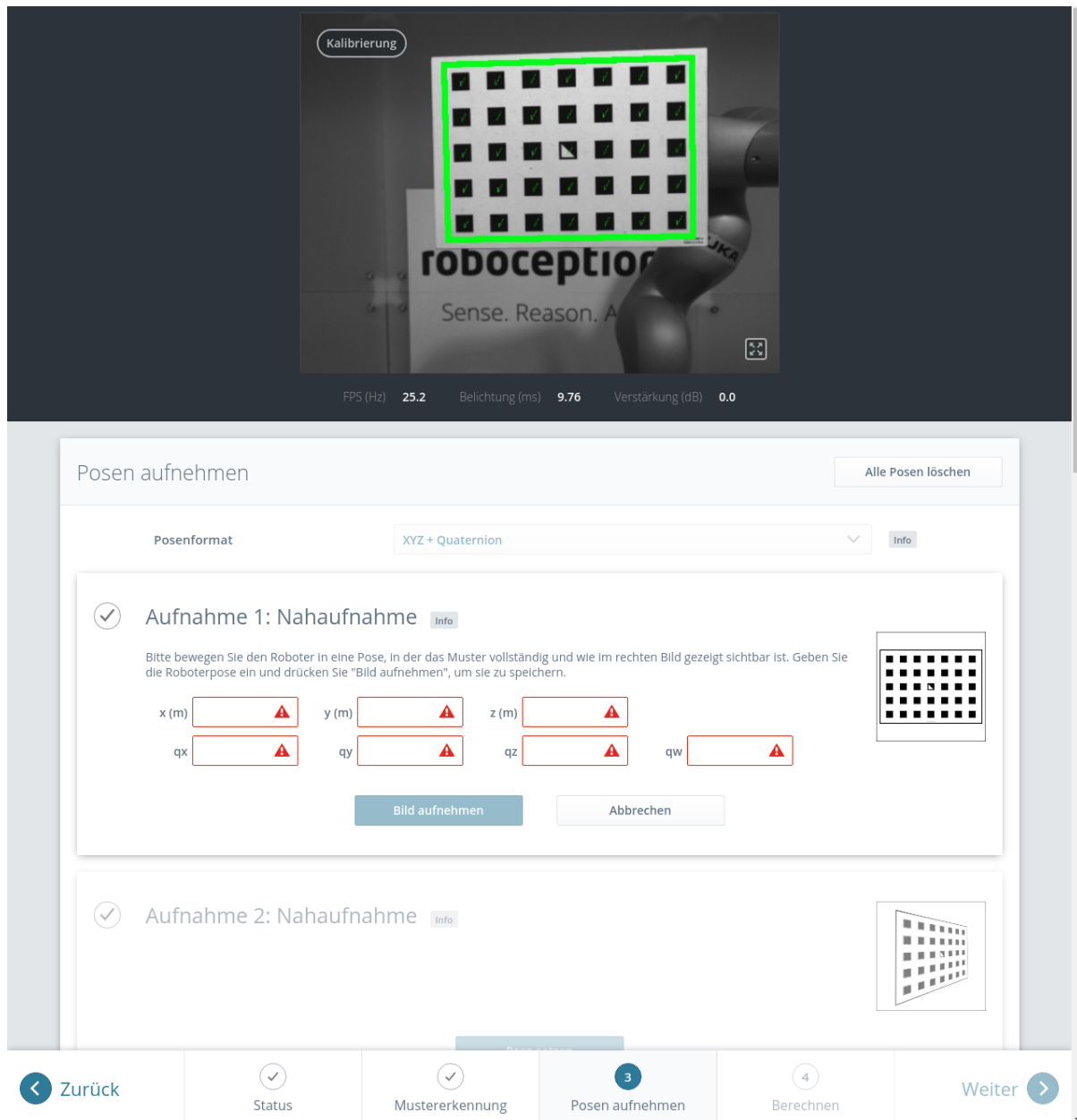


Abb. 6.28: Setzen der ersten Kalibrierpose für die Hand-Auge-Kalibrierung bei einer statisch montierten Kamera

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 6.4.1.5).

**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 6.4.1.5).

Wenn mindestens vier Posen gesetzt wurden, gelangt man über die Schaltfläche *Weiter* zur Berechnung des Kalibrierergebnisses.

Vorausgesetzt, die in [Abb. 6.27](#) dargestellten Empfehlungen zur Aufnahme des Kalibrierusters aus verschiedenen Blickwinkeln jeweils aus der Nähe und aus der Ferne wurden eingehalten, sind die folgenden Kamerabilder mit den jeweiligen Roboterposen im Softwaremodul zur Hand-Auge-Kalibrierung gespeichert:

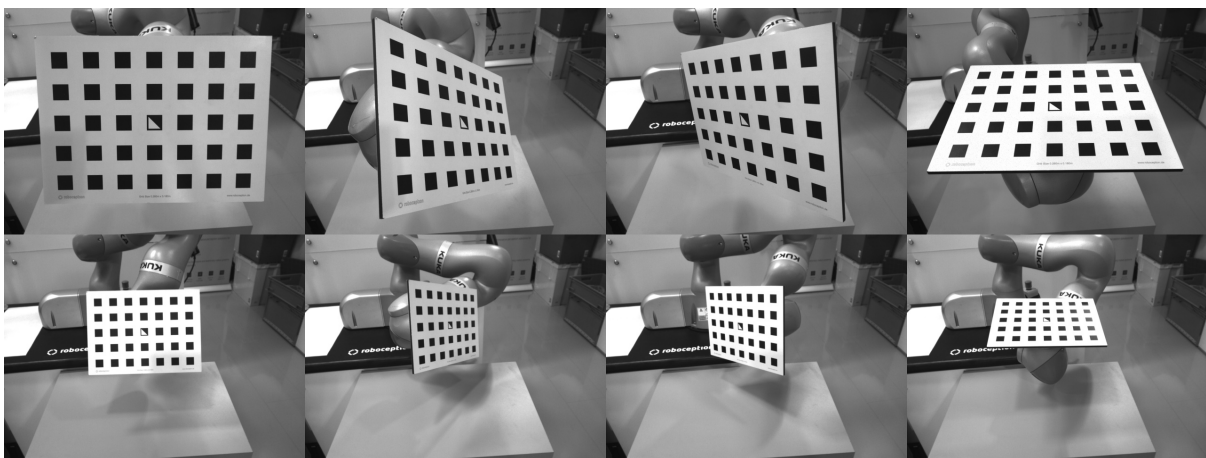


Abb. 6.29: Beispiel-Kamerabilder für die Hand-Auge-Kalibrierung

**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 Kalibrierusters 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 Kalibrierusters (für statische Kameras) angegeben werden. Für die REST-API sind die entsprechenden *Parameter* (Abschnitt 6.4.1.4) aufgelistet.

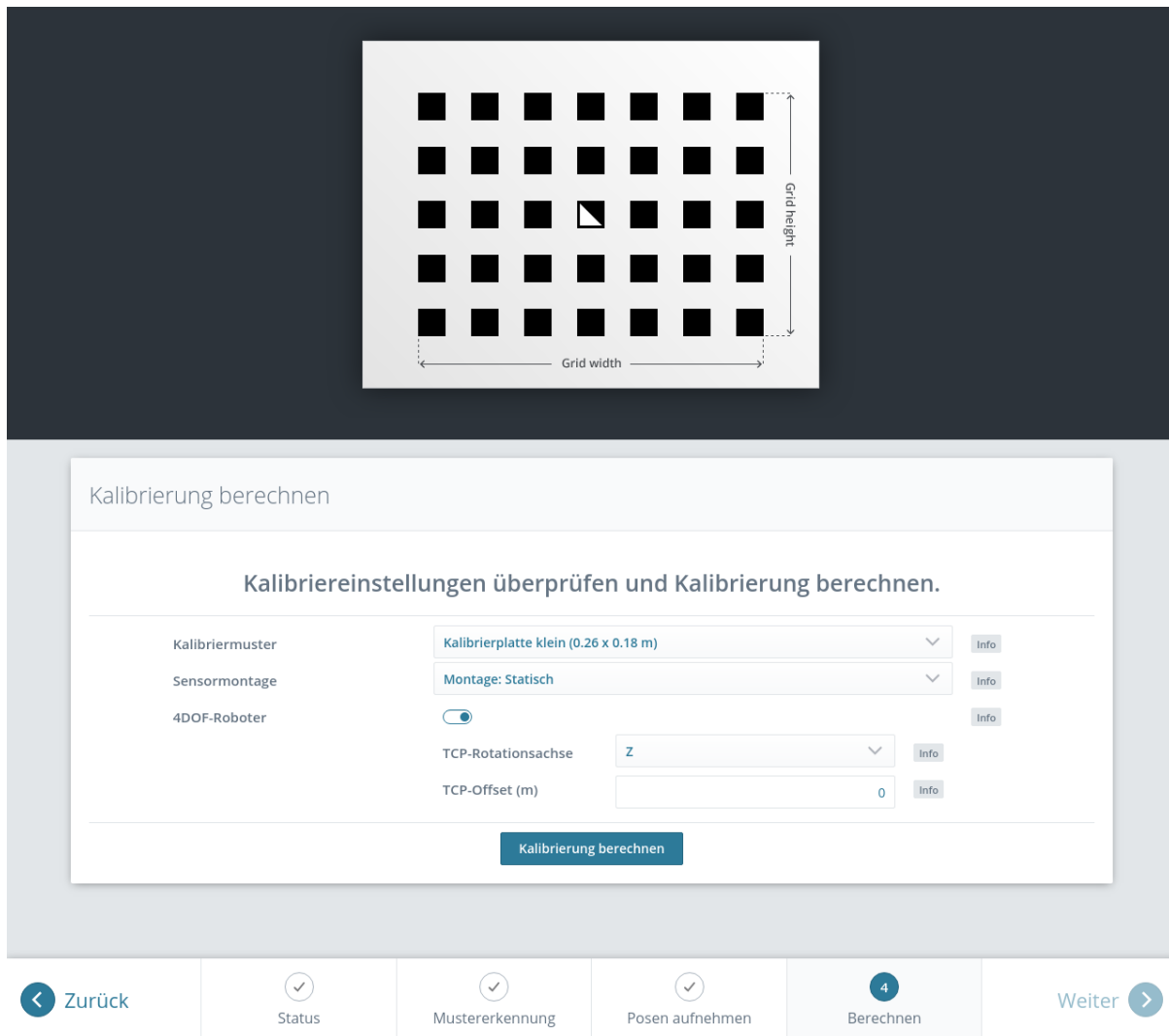


Abb. 6.30: Setzen der Parameter und Berechnen der Hand-Auge-Kalibrierung in der Web GUI des *rc\_visard*

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 6.4.1.5).

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 6.4.1.2).

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 6.4.1.5) wird das Kalibrierergebnis gespeichert.

### 6.4.1.4 Parameter

Das Modul zur Hand-Auge-Kalibrierung wird in der REST-API als `rc_hand_eye_calibration` bezeichnet und in der [Web GUI](#) (Abschnitt 7.1) unter *Konfiguration* → *Hand-Auge Kalibrierung* dargestellt. Der Benutzer kann die Kalibrierparameter entweder dort oder über die [REST-API-Schnittstelle](#) (Abschnitt 7.3) ändern.

#### Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.34: Laufzeitparameter des `rc_hand_eye_calibration`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
<code>grid_height</code>	float64	0.0	10.0	0.0	Höhe des Kalibrierusters in Metern
<code>grid_width</code>	float64	0.0	10.0	0.0	Breite des Kalibrierusters in Metern
<code>robot_mounted</code>	bool	false	true	true	Angabe, ob der <code>rc_visard</code> auf einem Roboter montiert ist
<code>tcp_offset</code>	float64	-10.0	10.0	0.0	Offset vom TCP entlang <code>tcp_rotation_axis</code>
<code>tcp_rotation_axis</code>	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 Kalibrierusters 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 Kalibrierusters 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 Kalibrierusters (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>
```

### 6.4.1.5 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. 6.35: 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",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    "status": "int32",
    "success": "bool",
    "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. 6.36: 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 Kalibrierpatterns 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. 6.37: 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

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_pose",
  "response": {
    "message": "string",
    "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. 6.38: Rückgabewerte des get\_poses-Services

status	success	Beschreibung
0	true	gespeicherte Posen werden zurückgeliefert
1	true	keine Kalibrierposen verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_poses",
  "response": {
    "message": "string",
    "poses": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "slot": "uint32"
      }
    ],
    "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"
    ]
  }
}
```

**Response**

Tab. 6.39: 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. 6.40: 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": {
        "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",
    "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. 6.41: 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"
      },

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "robot_mounted": "bool"
  }
}

```

**Response**

Tab. 6.42: 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 Kalibrieremusters 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": {

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

"return_code": {
  "message": "string",
  "value": "int16"
}
}
}

```

## 6.4.2 CollisionCheck

### 6.4.2.1 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* und *BoxPick* (Abschnitt 6.3.3) oder *SilhouetteMatch* (Abschnitt 6.3.4) vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 8.7).

Das Modul ermöglicht die Kollisionsprüfung zwischen dem Greifer und dem Load Carrier, oder anderen detektierten Objekten (nur in Kombination mit *SilhouetteMatch* (Abschnitt 6.3.4)). Es ist in die Module *ItemPick* und *BoxPick* (Abschnitt 6.3.3) und *SilhouetteMatch* (Abschnitt 6.3.4) integriert, kann aber auch als eigenständiges Modul genutzt werden. Die Greifermodelle für die Kollisionsprüfung müssen über das *GripperDB* (Abschnitt 6.5.3) Modul definiert werden.

**Warnung:** Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch, anderen Objekten oder dem Objekt im Greifer. Nur in Kombination mit *SilhouetteMatch* (Abschnitt 6.3.4), 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. 6.43: Spezifikationen des CollisionCheck-Moduls

Kollisionsprüfung mit	detektiertem Load Carrier, detektierte Objekte (nur <i>SilhouetteMatch</i> (Abschnitt 6.3.4)), Basisebene (nur <i>SilhouetteMatch</i> , Abschnitt 6.3.4)
Kollisionsprüfung verfügbar in	<i>ItemPick</i> und <i>BoxPick</i> (Abschnitt 6.3.3), <i>SilhouetteMatch</i> (Abschnitt 6.3.4)

### 6.4.2.2 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 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 Carrier 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. 6.31). 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.

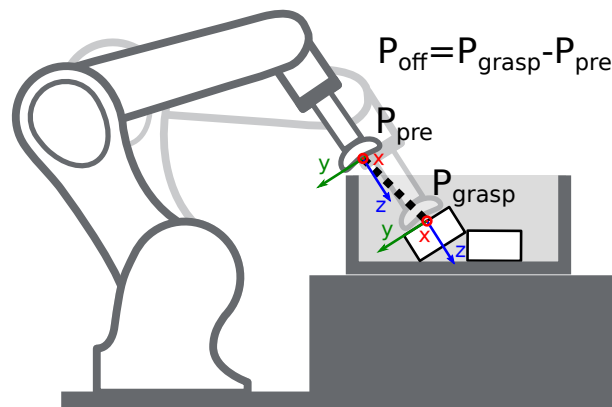


Abb. 6.31: Darstellung des Greif-Offsets für die Kollisionsprüfung. Im dargestellten Fall sind sowohl die Vorgreifposition als auch die Greifposition kollisionsfrei, aber die Trajektorie zwischen diesen Punkten hätte eine Kollision mit dem Load Carrier. Deswegen wird dieser Greifpunkt als Kollision erkannt.

### Integrierte Kollisionsprüfung in anderen Modulen

Die Kollisionsprüfung ist in die Services der folgenden Softwaremodule integriert:

- *ItemPick* und *BoxPick* (Abschnitt 6.3.3): `compute_grasps` (siehe *compute\_grasps für ItemPick*, Abschnitt 6.3.3.7 und *compute\_grasps für BoxPick*, Abschnitt 6.3.3.7)
- *SilhouetteMatch* (Abschnitt 6.3.4): `detect_object` (siehe *detect\_object*, Abschnitt 6.3.4.11)

Jedem dieser Services kann ein `collision_detection`-Argument übergeben werden, das aus der ID des Greifers (`grripper_id`) und optional aus dem Greif-Offset (`pre_grasp_offset`, siehe *Stand-Alone Kollisionsprüfung*, Abschnitt 6.4.2.2) besteht. 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. Dazu muss dem jeweiligen Service auch die ID des zu erkennenden Load Carriers übergeben werden.

Nur in *SilhouetteMatch* (Abschnitt 6.3.4), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im *SilhouetteMatch* 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.

In *SilhouetteMatch*, Abschnitt 6.3.4 werden Kollisionen zwischen dem Greifer und der Basisebene geprüft, wenn der Parameter `check_collisions_with_base_plane` in *SilhouetteMatch* aktiviert ist.

**Warnung:** Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch, anderen Objekten oder dem Objekt im Greifer. Nur in Kombination mit *SilhouetteMatch* (Abschnitt 6.3.4), 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.

#### 6.4.2.3 Parameter

Das `CollisionCheck`-Modul wird in der REST-API als `rc_collision_check` bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Konfiguration* → *CollisionCheck* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.3) ändern.

## Übersicht der Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.44: 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.

## 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.

**Warnung:** Der Sicherheitsabstand wird nicht für die Kollisionsprüfung zwischen 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 6.5.3.2) 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.

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>
```

**6.4.2.4 Statuswerte**

Statuswerte des rc\_collision\_check-Moduls:

Tab. 6.45: 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

**6.4.2.5 Services**

Die angebotenen Services von rc\_collision\_check können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.3) oder der rc\_visard *Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Das CollisionCheck-Modul stellt folgende Services zur Verfügung.

**check\_collisions**

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 6.3.1.2) 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 Vorgeifposition. Wird ein Greif-Offset angegeben, dann wird die Kollisionsprüfung auf der gesamten linearen Trajektorie von der Vorgeifposition bis zur Greifposition durchgeführt.

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

```
{
  "args": {
    "grasps": [
      {
        "pose": {
          "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"
          }
        }
      }
    ]
  }
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    },
    "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",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "pose_frame": "string",
    "uuid": "string"
  }
],
"return_code": {
  "message": "string",
  "value": "int16"
}
}
```

### 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"
    }
  }
}
```

### 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 6.5.3.3) 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 6.5.3.3) 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 6.5.3.3) 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 6.5.3.3) 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 6.5.3.3) 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 6.5.3.3) in *rc\_gripper\_db* beschrieben.

#### 6.4.2.6 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. 6.46: 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.

### 6.4.3 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 6.1.1.1).

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.

#### 6.4.3.1 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 8.8):

*„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.

#### 6.4.3.2 Kalibriervorgang

Die Kamerakalibrierung kann über die *Web GUI* (Abschnitt 7.1) 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 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. 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.

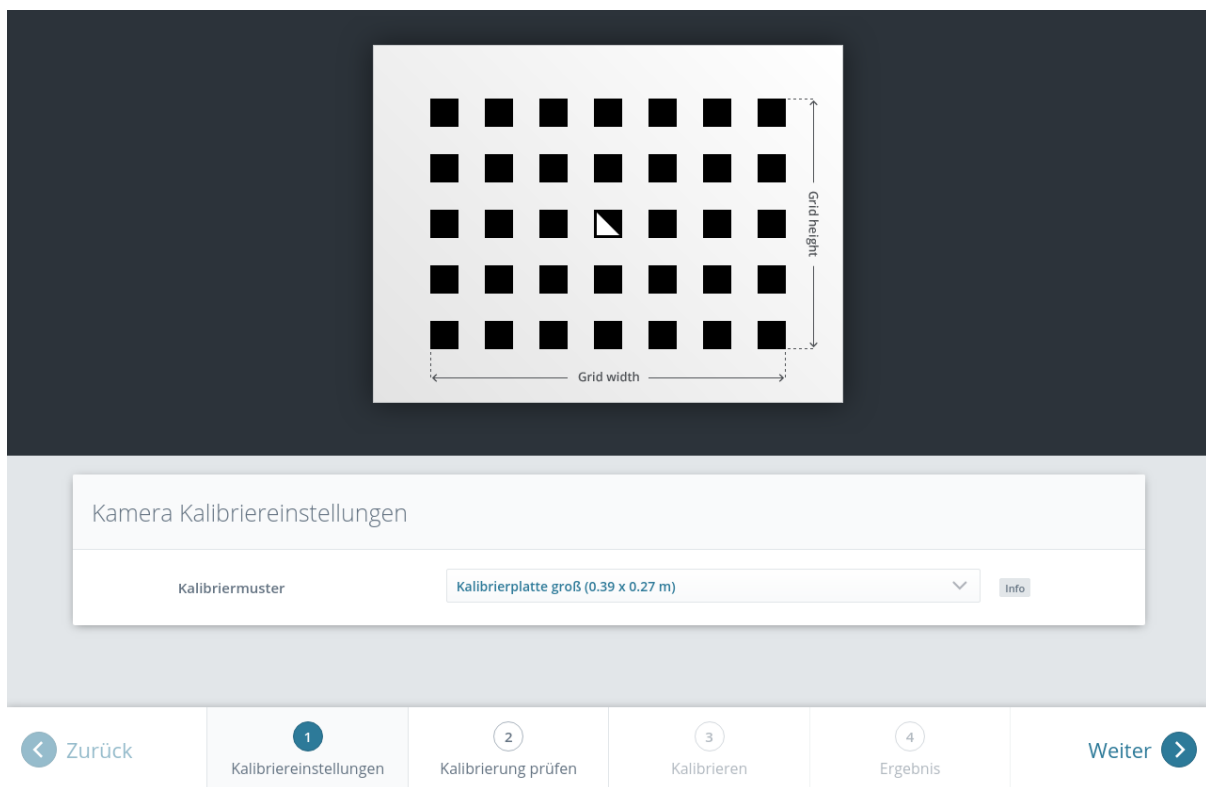


Abb. 6.32: Kalibriereinstellungen

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. 6.33: 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 Mus-



ter 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. 6.35](#) 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 Kalibrieremusters zu vereinfachen.

Als erstes muss das Kalibrieremuster 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 Kalibrieremuster gerichtet sein. Wenn das Kalibrieremuster nicht senkrecht zur Sichtachse der Kameras ausgerichtet ist, erscheinen kleine grüne Pfeile auf dem Kamerabild, die auf die erwarteten Positionen der Ecken des Kalibrieremusters zeigen (siehe [Abb. 6.34](#)).

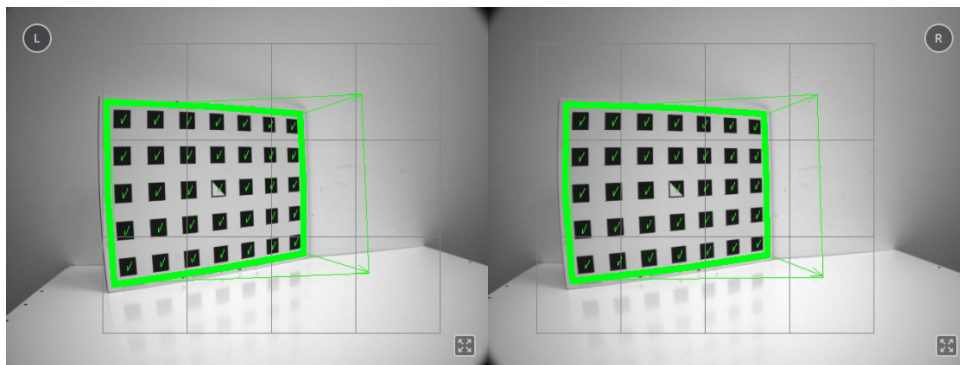


Abb. 6.34: Pfeile weisen darauf hin, wenn das Muster während der Stereokalibrierung nicht senkrecht zur Blickrichtung der Kamera gehalten wird.

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 Kalibrieremuster abgedeckt werden. Dies wird durch eine grüne Füllung der erfassten Zellen dargestellt (siehe [Abb. 6.35](#)).

Beim *rc\_visard* können alle Zellen mit einer einzigen Aufnahme erfasst werden, wenn das Kalibrieremuster nah genug gehalten wird.

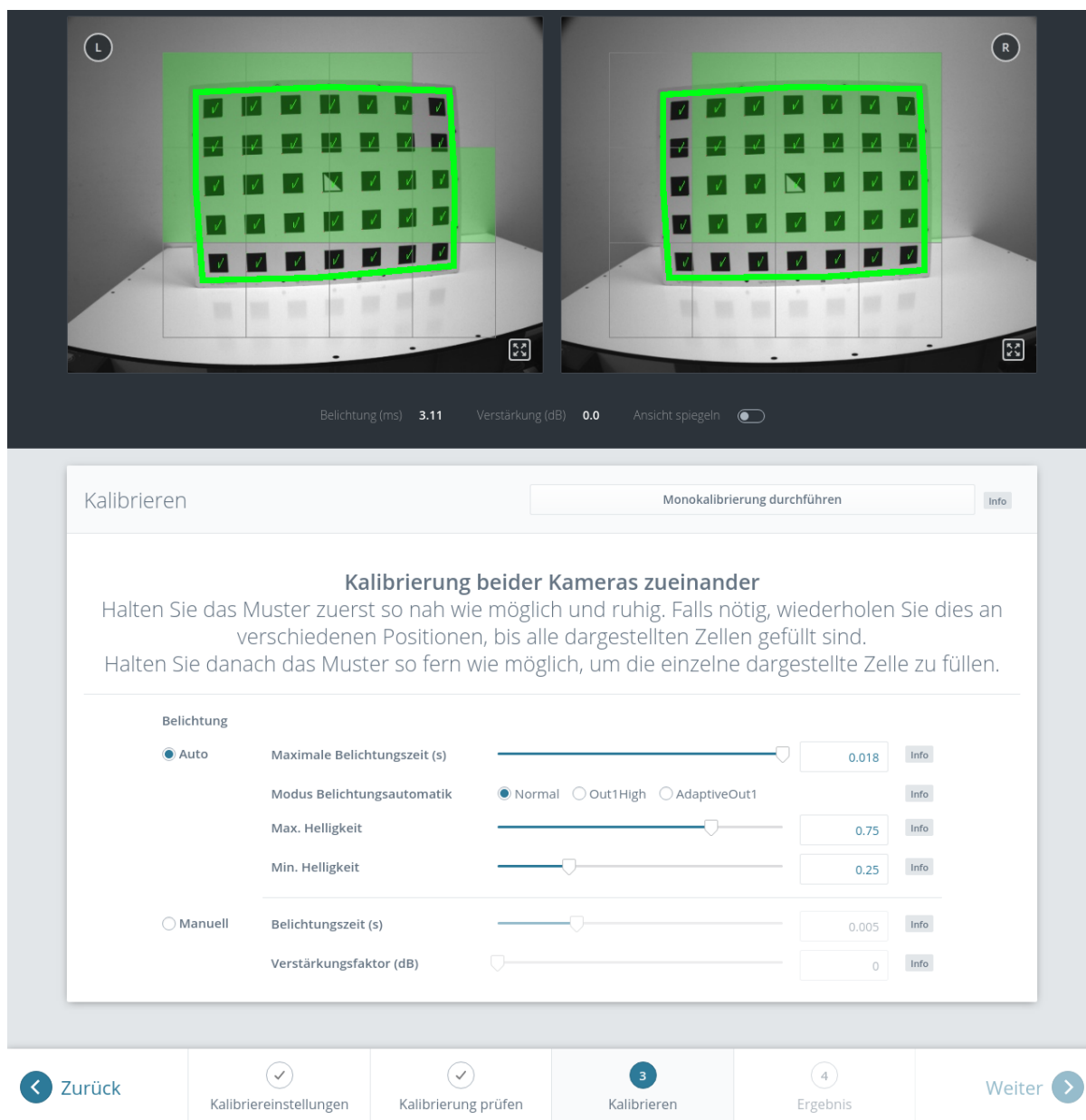


Abb. 6.35: Stereokalibrierung: Das Muster sollte so nah wie möglich gehalten werden, um alle dargestellten Zellen zu füllen.

**Bemerkung:** Falls alle Häkchen auf dem Kalibrieremuster 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 Kalibrieremuster 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 Kalibrieremuster erfolgreich detektiert wurde, wird die Zelle grün und das Kalibrierergebnis kann berechnet werden (siehe Abb. 6.36).

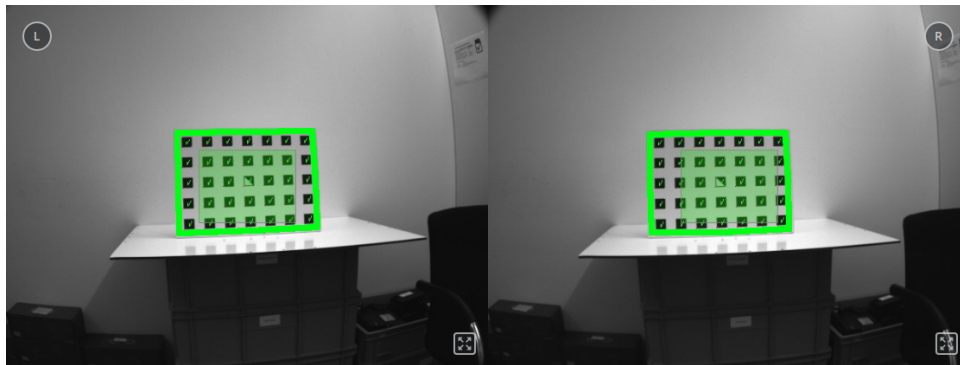


Abb. 6.36: Positionierung des Musters so entfernt wie möglich während der Stereokalibrierung

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 Kalibrieremuster 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 Kalibrierengenauigkeit 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. 6.37](#)).

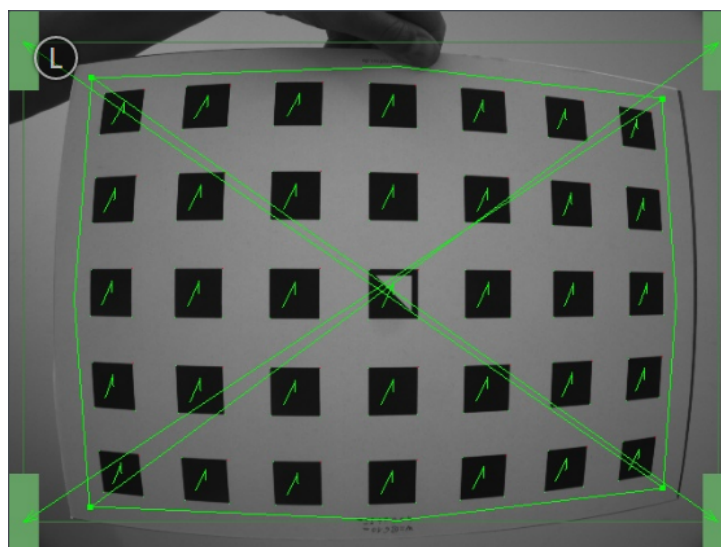


Abb. 6.37: Wird das Kalibrieremuster falsch herum gehalten, kreuzen sich die grünen Linien.

**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. 6.38](#) angegebenen Stellungen zu halten.

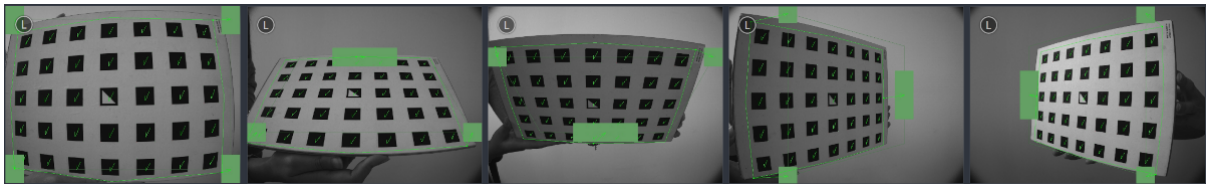


Abb. 6.38: Musterposen für die Monokalibrierung

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.

#### 6.4.3.3 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.

#### 6.4.3.4 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.

### 6.4.4 IOControl und Projektor-Kontrolle

Das IOControl Modul ist ein optionales Modul, welches intern auf dem *rc\_visard* läuft und eine gesonderte IOControl *Lizenz* (Abschnitt 8.7) benötigt, die erworben werden muss. Diese Lizenz ist auf jedem *rc\_visard*, der nach dem 01.07.2020 gekauft wurde, vorhanden.

Das IOControl-Modul ermöglicht das Lesen der digitalen Eingänge und die Kontrolle der digitalen Ausgänge (GPIOs) des *rc\_visard*. 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 des *rc\_visard* 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.

#### 6.4.4.1 Parameter

Das IOControl-Modul wird in der REST-API als *rc\_iocontrol* bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Konfiguration* → *IOControl* dargestellt. Der Benutzer kann die Parameter über die Web GUI, die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.3), oder die GenICam-Schnittstelle mit den DigitalIOControl-Parametern *LineSelector* und *LineSource* ändern (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 7.2).

#### Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.47: Laufzeitparameter des *rc\_iocontrol*-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
<i>out1_mode</i>	string	-	-	Low	Out1 mode: [Low, High, ExposureActive, ExposureAlternateActive]
<i>out2_mode</i>	string	-	-	Low	Out2 mode: [Low, High, ExposureActive, ExposureAlternateActive]

#### Beschreibung der Laufzeitparameter

##### *out1\_mode* und *out2\_mode* (*Ausgang 1 (Out1)* 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>
```

**Bemerkung:** Die Parameter können nur verändert werden, wenn eine IOControl-Lizenz auf dem *rc\_visard* verfügbar ist. Sonst gelten die Voreinstellungen für die Parameter, d.h. *out1\_mode* = Low und *out2\_mode* = Low.

Abb. 6.39 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.

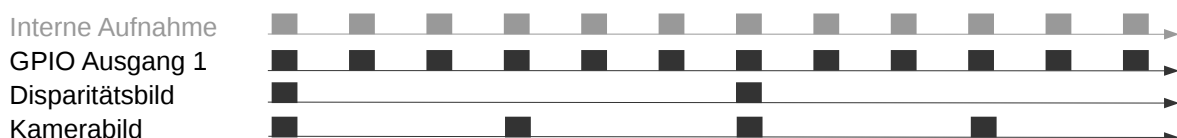


Abb. 6.39: Beispiel für die Nutzung der Betriebsart ExposureActive für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes Bildes *an* (HIGH). Disparitätsbilder werden für 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 des *rc\_visard* angeschlossen ist. In diesem Fall nutzt das *Stereo-Matching-Modul* (Abschnitt 6.1.2) 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 (siehe *FPS*, Abschnitt 6.1.1.3). Alle Module, die Intensitätsbilder benutzen, wie z.B. *TagDetect* (Abschnitt 6.3.2) und *ItemPick* (Abschnitt 6.3.3), benutzen die Intensitätsbilder, bei denen der GPIO-Ausgang 1 *aus* (LOW) ist, d.h. der Projektor ist aus. Abb. 6.40 zeigt ein Beispiel.

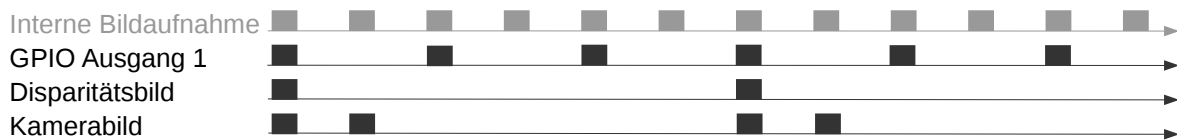


Abb. 6.40: Beispiel für die Nutzung der Betriebsart `ExposureAlternateActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes zweiten Bildes *an* (HIGH). Disparitätsbilder 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 7.2).

#### 6.4.4.2 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) des `rc_visard` 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 <code>IOControl</code> ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_io_values",
  "response": {
    "input_mask": "uint32",
    "output_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": {
```

(Fortsetzung auf der nächsten Seite)



(Fortsetzung der vorherigen Seite)

```

    "message": "string",
    "value": "int16"
  }
}
}

```

## 6.5 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 7.3) sind die Datenbankmodule nur in API Version 2 verfügbar.

Die Datenbankmodule sind:

- **LoadCarrierDB** (*rc\_load\_carrier\_db*, Abschnitt 6.5.1) ermöglicht das Erstellen, Abfragen und Löschen von Load Carriern.
- **RoiDB** (*rc\_roi\_db*, Abschnitt 6.5.2) ermöglicht das Erstellen, Abfragen und Löschen von 2D und 3d Regions of Interest.
- **GripperDB** (*rc\_gripper\_db*, Abschnitt 6.5.3) ermöglicht das Erstellen, Abfragen und Löschen von Greifern für die Kollisionsprüfung.

### 6.5.1 LoadCarrierDB

#### 6.5.1.1 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. 6.48: Spezifikationen des LoadCarrierDB Moduls

Unterstützte Load Carrier Typen	Standard 4-seitig, mit durchgängigem oder abgestuftem Rand
Min. Load Carrier Abmessungen	0.1 m x 0.1 m x 0.05 m
Max. Load Carrier Abmessungen	2 m x 2 m x 2 m
Max. Anzahl von Load Carriern	50
Load Carrier verfügbar in	<i>ItemPick</i> und <i>BoxPick</i> (Abschnitt 6.3.3) und <i>SilhouetteMatch</i> (Abschnitt 6.3.4)
Mögliche Posen-Arten	keine Pose, Orientierungsprior, exakte Pose
Mögliche Referenzkoordinatensysteme	camera, external

#### 6.5.1.2 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 2.0 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. 6.41](#)).

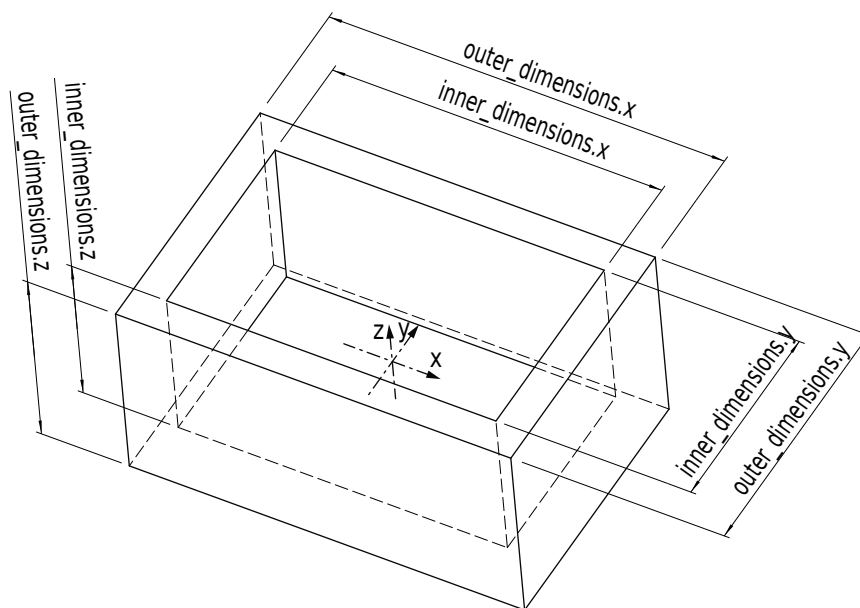


Abb. 6.41: Load Carrier mit Koordinatensystem und inneren und äußeren Abmessungen

**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 [6.3.1.5](#)). [Abb. 6.42](#) zeigt das Innenvolumen eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

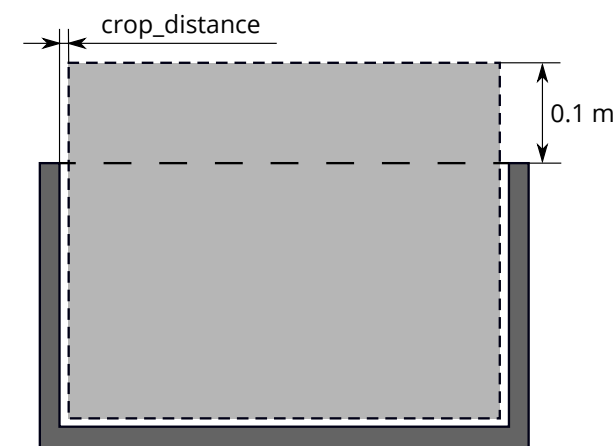


Abb. 6.42: Darstellung des Innenvolumens eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

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 6.4.2). Beispiele für Load Carrier, bei denen die Angabe der Randstärke für die Erkennung notwendig ist, werden in [Abb. 6.43](#) gezeigt.

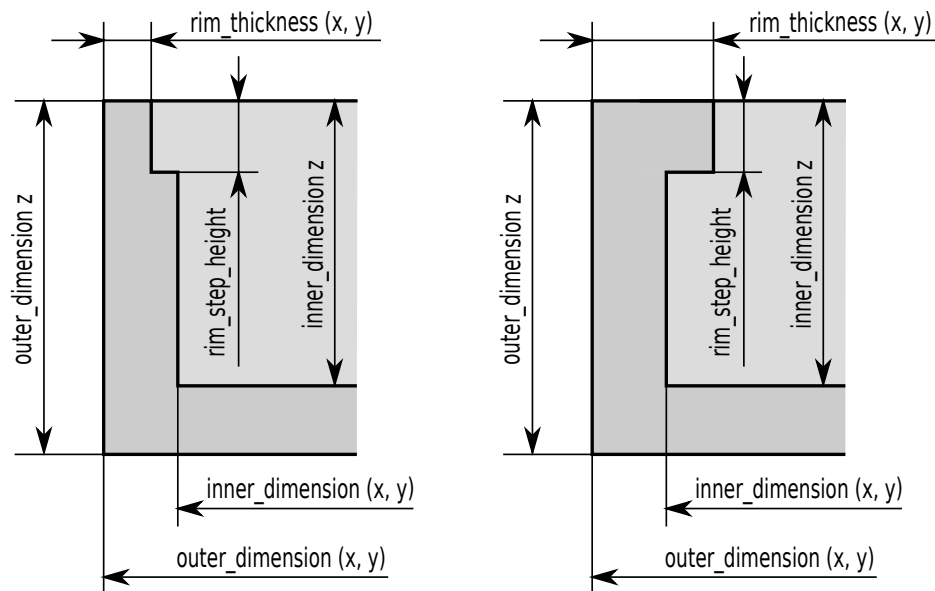


Abb. 6.43: Beispiele für Load Carrier, bei denen die Angabe der Randstärke für die Erkennung notwendig ist

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 7.3) konfiguriert werden.

**Bemerkung:** Die konfigurierten Load Carrier sind persistent auf dem `rc_visard` gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

### 6.5.1.3 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 6.3.3.7). 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. 6.44). Das Load Carrier Abteil ist nicht Teil der Load Carrier Definition im LoadCarrierDB Modul, sondern muss für jeden Detektionsaufruf separat definiert werden.

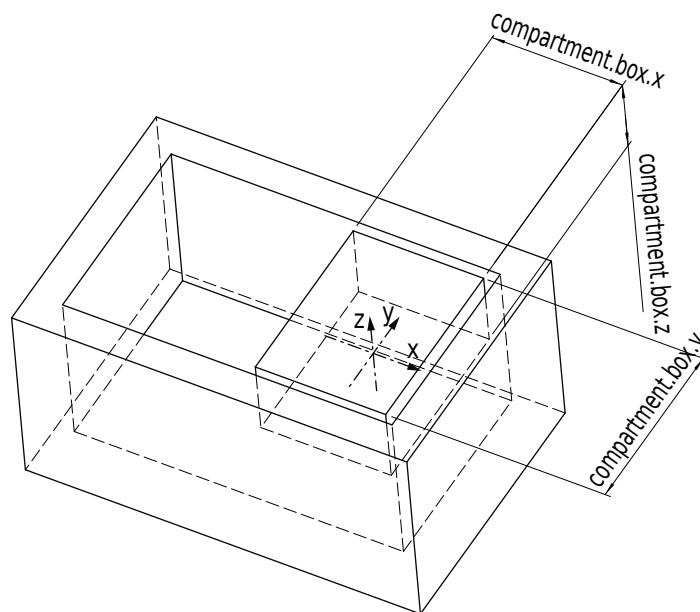


Abb. 6.44: Beispiel für ein Abteil innerhalb eines Load Carriers. Das gezeigte Koordinatensystem das Koordinatensystem des Load Carrier Abteils.

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.

### 6.5.1.4 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 6.4.1).

**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.

### 6.5.1.5 Services

Das LoadCarrierDB Modul wird in der REST-API als `rc_load_carrier_db` bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Datenbank* → *Load Carrier* dargestellt. Die angebotenen Services des LoadCarrierDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.3) 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 6.5.1.2) beschrieben.

Das Feld `type` ist optional und akzeptiert derzeit nur `STANDARD`.

Das Feld `rim_ledge` ist optional und akzeptiert derzeit nur `0`.

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

```
{
  "args": {
    "load_carrier": {
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

(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",
    "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": {
      "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

Das Feld `type` ist immer `STANDARD` und `rim_ledge` ist immer 0.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_load_carriers",
  "response": {
    "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",
        "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 löschen- den 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"
    }
  }
}
```

### 6.5.1.6 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. 6.49: 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 <code>set_load_carrier</code> wurde ein bereits existierendes Objekt mit derselben <code>id</code> überschrieben.



## 6.5.2 RoiDB

### 6.5.2.1 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* und *BoxPick* (Abschnitt 6.3.3) Modulen verwendet werden. 2D ROIs werden vom *SilhouetteMatch* (Abschnitt 6.3.4) Modul und dem *LoadCarrier* (Abschnitt 6.3.1) Modul unterstützt.

Tab. 6.50: Spezifikationen des ROI DB 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: <i>SilhouetteMatch</i> (Abschnitt 6.3.4), <i>LoadCarrier</i> (Abschnitt 6.3.1), 3D: <i>ItemPick</i> und <i>BoxPick</i> (Abschnitt 6.3.3)
Unterstützte Referenzkoordinatensysteme	camera, external

### 6.5.2.2 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 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 6.4.1) 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 7.3) als auch über die *rc\_visard Web GUI* (Abschnitt 7.1) 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 7.3) des RoiDB Moduls vorgenommen werden, oder über die *rc\_visard Web GUI* (Abschnitt 7.1) 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.

### 6.5.2.3 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 6.4.1).

**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.

### 6.5.2.4 Services

Das RoiDB Modul wird in der REST-API als *rc\_roi\_db* bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Datenbank* → *Regions of Interest* dargestellt. Die angebotenen Services des RoiDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.3) 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 6.5.2.2) 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"
    }
  }
}
```

### 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 6.5.2.2) 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"
    }
  }
}

```

### 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",
      "value": "int16"
    }
  }
}
```

### 6.5.2.5 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. 6.51: 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 <code>set_region_of_interest</code> oder <code>set_region_of_interest_2d</code> wurde ein bereits existierendes Objekt mit derselben <code>id</code> überschrieben.

## 6.5.3 GripperDB

### 6.5.3.1 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 und BoxPick* (Abschnitt 6.3.3) oder *SilhouetteMatch* (Abschnitt 6.3.4) vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 8.7).

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 6.3.4)) genutzt werden können. Die angelegten Greifer sind in allen Modulen auf dem *rc\_visard* verfügbar, die eine Kollisionsprüfung anbieten.

Tab. 6.52: Spezifikationen des GripperDB Moduls

Max. Anzahl Greifer	50
Mögliche Greiferelement-Geometrien	Box, Zylinder
Max. Anzahl Elemente pro Greifer	15
Kollisionsprüfung verfügbar in	<i>ItemPick und BoxPick</i> (Abschnitt 6.3.3), <i>SilhouetteMatch</i> (Abschnitt 6.3.4)

### 6.5.3.2 Erstellen eines Greifers

Der Greifer ist eine Kollisionsgeometrie, die zur Prüfung auf Kollisionen zwischen dem geplanten Griff und dem Load Carrier 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`.

Weiterhin müssen für jeden Greifer der Flanschradius und der Tool Center Point (TCP) definiert werden.

Die Konfiguration des Greifers wird in der Regel während des Setups der Zielanwendung durchgeführt. Das kann über die *REST-API-Schnittstelle* (Abschnitt 7.3) oder die *rc\_visard Web GUI* (Abschnitt 7.1) 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 6.4.2.3) 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. 6.45). Der Test basiert auf der Greifergeometrie und dem Flanschradius.

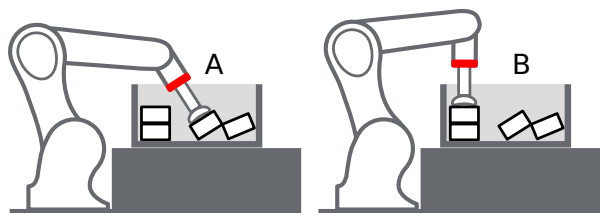


Abb. 6.45: Fall A: Der Griff wird nur als Kollision erkannt, wenn `check_flange` auf `true` gesetzt ist, denn der Flansch (rot) befindet sich im Load Carrier. Fall B: Der Griff ist in jedem Fall kollisionsfrei.



### Erstellen eines Greifers über die REST-API oder die Web GUI

Bei der Greifererstellung über die *REST-API-Schnittstelle* (Abschnitt 7.3) oder die *Web GUI* (Abschnitt 7.1) 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 jedes Elements 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. 6.46).

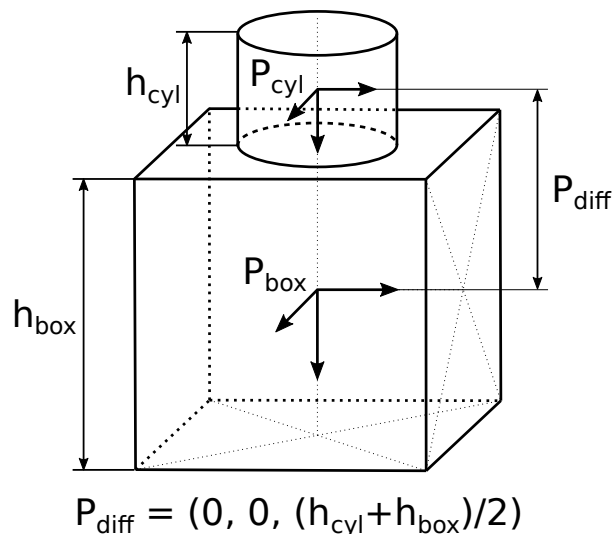


Abb. 6.46: Bezugskordinatensysteme für das Erstellen von Greifern über die REST-API und die Web GUI

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. Das automatische Anheften ist nur möglich, wenn das Kind-Element nicht rotiert ist in Bezug auf seinen Parent.

Das Bezugskordinatensystem 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` wird die TCP-Position im Flanschkoordinatensystem berechnet und als `tcp_pose_flange` 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.

### 6.5.3.3 Services

Das GripperDB Modul wird in der REST-API als `rc_gripper_db` bezeichnet und in der [Web GUI](#) (Abschnitt 7.1) unter *Datenbank* → *Greifer* dargestellt. Die angebotenen Services des GripperDB Moduls können mithilfe der [REST-API-Schnittstelle](#) (Abschnitt 7.3) 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‘ haben, wobei die Dimensionen im Feld `cylinder` bzw. `box` angegeben werden. Die Pose jedes Elements muss im *Parent*-Koordinatensystem angegeben werden (siehe [Erstellen eines Greifers](#), Abschnitt 6.5.3.2). 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.

`flange_radius`: Flanschradius der benutzt wird, falls der Parameter `check_flange` aktiviert ist.

`id`: Eindeutiger Name des Greifers.

`tcp_parent_id`: ID des Elements, auf dem der TCP definiert ist.

`tcp_pose_parent`: 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"
        },
        "cylinder": {
          "height": "float64",
          "radius": "float64"
        },
        "id": "string",
        "parent_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

        "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"
    }
}
}
}
}
}

```

**Response**

gripper: Gibt den Greifer mit dem zusätzlichen Feld `tcp_pose_flange` 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"
          },
          "cylinder": {
            "height": "float64",
            "radius": "float64"
          },
          "id": "string",
          "parent_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "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",
    "y": "float64",
    "z": "float64"
  },
  "position": {
    "x": "float64",
    "y": "float64",
    "z": "float64"
  }
},
"type": "string"
},
"return_code": {
  "message": "string",
  "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

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_grippers",
  "response": {
    "grippers": [
      {
        "elements": [
          {
            "box": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "cylinder": {
              "height": "float64",
              "radius": "float64"
            },
            "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",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

        "y": "float64",
        "z": "float64"
    }
},
"tcp_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"
    }
  }
}

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

    }
  }
}

```

### 6.5.3.4 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. 6.53: 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.

## 7 Schnittstellen

Es stehen die folgenden Schnittstellen zur Konfiguration und Datenübertragung des *rc\_visard* zur Verfügung:

- *Web GUI* (Abschnitt 7.1)  
Leicht zu bedienendes grafisches Interface zum Konfigurieren und Kalibrieren des *rc\_visard*, zum Anzeigen von Livebildern, Aufrufen von Services, Visualisieren von Ergebnissen, usw.
- *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 7.2)  
Konfiguration bild- und kamerabezogener Einstellungen.
- *REST-API-Schnittstelle* (Abschnitt 7.3)  
Programmierschnittstelle zur Konfiguration des *rc\_visard*, zur Abfrage von Statusinformationen, zum Anfordern von Datenströmen, usw.
- *rc\_dynamics-Schnittstelle* (Abschnitt 7.4)  
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.
- *Ethernet KRL Schnittstelle (EKI)* (Abschnitt 7.5)  
API zum Konfigurieren des *rc\_visard* und Ausführen von Service-Anfrage von KUKA KSS Robotern aus.
- *Zeitsynchronisation* (Abschnitt 7.6)  
Zeitsynchronisation zwischen dem *rc\_visard* und dem Anwendungs-PC.

### 7.1 Web GUI

Die Web GUI des *rc\_visard* dient dazu, das Gerät zu testen, zu kalibrieren und zu konfigurieren.

#### 7.1.1 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`.

## 7.1.2 Kennenlernen der Web GUI

Die Dashboard-Seite der Web GUI enthält die wichtigsten Informationen über das Gerät und die Softwaremodule.

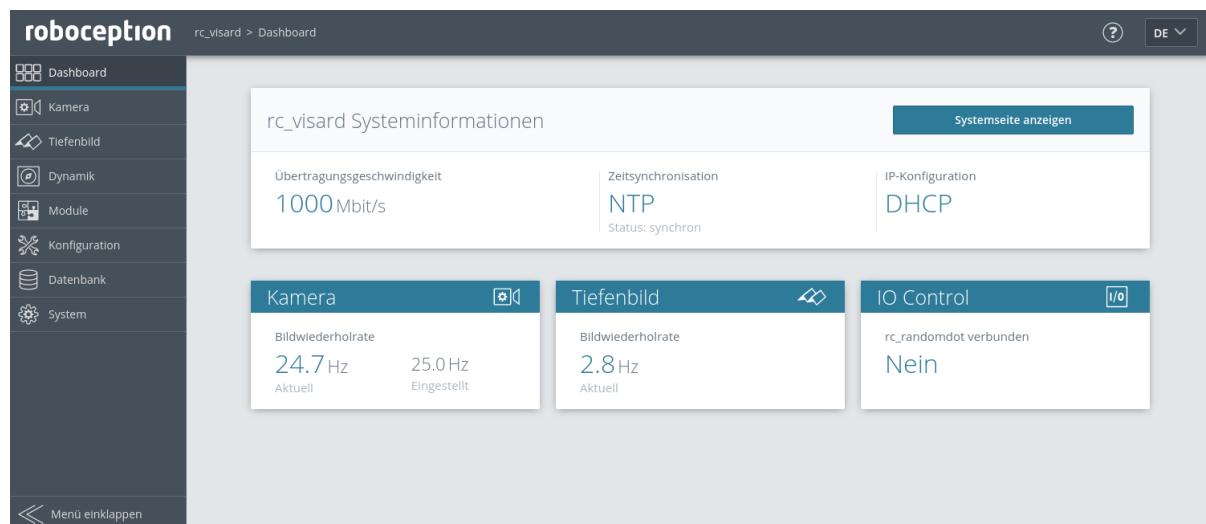


Abb. 7.1: Dashboard-Seite der Web GUI des *rc\_visard*

Über das Menü kann auf die einzelnen Seiten der Web GUI des *rc\_visard* zugegriffen werden:

**Kamera** zeigt einen Live-Stream der rektifizierten Kamerabilder. Die Bildwiederholrate lässt sich reduzieren, um Bandbreite zu sparen, wenn über einen GigE Vision®-Client gestreamt wird. Außerdem lässt sich die Belichtung manuell oder automatisch einstellen. Für nähere Informationen siehe [Parameter](#) (Abschnitt 6.1.1.3).

**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 [Parameter](#) (Abschnitt 6.1.2.5).

**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 6.2.2.1).

**Module** ermöglicht den Zugriff auf die Detektionsmodule des *rc\_visard* (siehe [Detektionsmodule](#), Abschnitt 6.3).

**Konfiguration** ermöglicht den Zugriff auf die Konfigurationsmodule des *rc\_visard* (siehe [Konfigurationsmodule](#), Abschnitt 6.4).

**Datenbank** ermöglicht den Zugriff auf die Datenbankmodule des *rc\_visard* (siehe [Datenbankmodule](#), Abschnitt 6.5).

**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.

### 7.1.3 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.

### 7.1.4 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.

## 7.2 GigE Vision 2.0/GenICam-Schnittstelle

Gigabit Ethernet for Machine Vision (oder kurz „GigE Vision®“) ist ein industrieller Standard für Kameratechnologien, 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* (PT-Pv2).

**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>).

### 7.2.1 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.

### 7.2.2 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 der Module *Kamera* (Abschnitt 6.1.1) und *Stereo-Matching* (Abschnitt 6.1.2) beziehen.

### 7.2.3 Wichtige Standardparameter der GenICam-Schnittstelle

#### 7.2.3.1 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 7.2.6).

**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 7.2.6).

**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.

**7.2.3.2 Kategorie: AcquisitionControl****AcquisitionFrameRate**

- Typ: Float, Wertebereich: 1–25 Hz
- Voreinstellung: 25 Hz
- Beschreibung: Bildwiederholrate der Kamera (*FPS*, Abschnitt 6.1.1.3).

**ExposureAuto**

- Typ: Aufzählung, mögliche Werte: Continuous, Out1High, AdaptiveOut1 oder Off
- Voreinstellung: Continuous
- Beschreibung: Lässt sich für die manuelle Belichtung auf Off bzw. für die *automatische Belichtung* (Abschnitt 6.1.1.3) auf Continuous, Out1High oder AdaptiveOut1 setzen. Der Wert Continuous entspricht dem Wert Normal für exp\_auto\_mode (*Modus Belichtungsautomatik*, Abschnitt 6.1.1.3) und Out1High und AdaptiveOut1 den gleichnamigen Modi.

**ExposureTime**

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 5000 µs
- Beschreibung: Belichtungszeit der Kameras für den manuellen Belichtungsmodus, ausgedrückt in Mikrosekunden (*Belichtungszeit*, Abschnitt 6.1.1.3).

**7.2.3.3 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 (*Verstärkungsfaktor*, Abschnitt 6.1.1.3).

**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 (*Weißabgleich*, Abschnitt 6.1.1.3).

**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 (*wb\_ratio*, Abschnitt 6.1.1.3).

### 7.2.3.4 Kategorie: DigitalIOControl

**Bemerkung:** Falls die IOControl-Lizenz nicht verfügbar ist, dann werden die Ausgänge entsprechend den Voreinstellungen gesetzt und können nicht geändert werden. Die Eingänge liefern immer den logischen Wert für falsch, unabhängig davon welche Signale an den physikalischen Eingängen anliegen.

#### 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. 7.1: Bedeutung der Bits des LineStatusAll Parameters.

Bit	4	3	2	1
GPIO	Eingang 2	Eingang 1	Ausgang 2	Ausgang 1

#### LineSource (schreibgeschützt, falls das IOControl-Modul nicht lizenziert ist)

- 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 6.4.4.1). Siehe auch den Parameter AcquisitionAlternateFilter zum Filtern von Bildern im Betriebszustand ExposureAlternateActive.

### 7.2.3.5 Kategorie: TransportLayerControl / PtpControl

#### PtpEnable

- Typ: Boolean
- Voreinstellung: false
- Beschreibung: Schaltet die PTP-Synchronisierung ein und aus.

### 7.2.3.6 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.

### 7.2.3.7 Kategorie: ChunkDataControl

#### ChunkModeActive

- Typ: Boolean
- Voreinstellung: false
- Beschreibung: Schaltet Chunk-Daten an, die mit jedem Bild mitgeliefert werden.

## 7.2.4 Besondere Parameter der GenICam-Schnittstelle des *rc\_visard*

### 7.2.4.1 Kategorie: AcquisitionControl

#### AcquisitionAlternateFilter (schreibgeschützt, falls das IOControl-Modul nicht lizenziert ist)

- 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 6.4.4.1).

#### 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 (*Maximale Belichtungszeit*, Abschnitt 6.1.1.3).

#### ExposureRegionOffsetX

- Typ: Integer, Wertebereich: 0 bis maximale Bildbreite
- Voreinstellung: 0
- Beschreibung: Horizontaler Offset des *Bereichs für die Belichtungszeitregelung* (Abschnitt 6.1.1.3) in Pixeln.

#### ExposureRegionOffsetY

- Typ: Integer, Wertebereich: 0 bis maximale Bildhöhe
- Voreinstellung: 0
- Beschreibung: Vertikaler Offset des *Bereichs für die Belichtungszeitregelung* (Abschnitt 6.1.1.3) in Pixeln.



**ExposureRegionWidth**

- Typ: Integer, Wertebereich: 0 bis maximale Bildbreite
- Voreinstellung: 0
- Beschreibung: Breite des *Bereichs für die Belichtungszeitregelung* (Abschnitt 6.1.1.3) in Pixeln.

**ExposureRegionHeight**

- Typ: Integer, Wertebereich: 0 bis maximale Bildhöhe
- Voreinstellung: 0
- Beschreibung: Höhe des *Bereichs für die Belichtungszeitregelung* (Abschnitt 6.1.1.3) in Pixeln.

**RcExposureAutoAverageMax**

- Typ: Float, Wertebereich 0-1
- Voreinstellung: 0.75
- Beschreibung: Maximale Helligkeit der *automatischen Belichtungszeitsteuerung* (Abschnitt 6.1.1.3) als Wert zwischen 0 (dunkel) und 1 (hell).

**RcExposureAutoAverageMin**

- Typ: Float, Wertebereich 0-1
- Voreinstellung: 0.25
- Beschreibung: Minimale Helligkeit der *automatischen Belichtungszeitsteuerung* (Abschnitt 6.1.1.3) als Wert zwischen 0 (dunkel) und 1 (hell).

**7.2.4.2 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.

**7.2.4.3 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. Dies ist jedoch nur möglich, wenn die IOControl-Lizenz verfügbar ist. 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 6.1.2.5).

#### 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 6.1.2.5).

#### DepthStaticScene

- Typ: Boolean
- Voreinstellung: false
- Beschreibung: True zum Mitteln über acht aufeinanderfolgende Kamerabilder zur Verbesserung des Stereo-Matching-Resultats (*Statisch*, Abschnitt 6.1.2.5).

#### DepthSmooth (schreibgeschützt ohne StereoPlus-Lizenz)

- Typ: Boolean
- Voreinstellung: false
- Beschreibung: True um Disparitätswerte zu glätten (*Glättung*, Abschnitt 6.1.2.5).

#### DepthFill

- Typ: Integer, Wertebereich: 0–4 Pixel
- Voreinstellung: 3 Pixel
- Beschreibung: Wert in Pixeln für *Füllen* (Abschnitt 6.1.2.5).

#### DepthSeg

- Typ: Integer, Wertebereich: 0–4000 Pixel
- Voreinstellung: 200 Pixel
- Beschreibung: Wert in Pixeln für *Segmentierung* (Abschnitt 6.1.2.5).

#### DepthMinConf

- Typ: Float, Wertebereich: 0.0–1.0
- Voreinstellung: 0.0
- Beschreibung: Wert für die *Minimale Konfidenz*-Filterung (Abschnitt 6.1.2.5).

#### DepthMinDepth

- Typ: Float, Wertebereich: 0.1–100.0 m
- Voreinstellung: 0.1 m
- Beschreibung: Wert in Metern für die *Minimale Abstands*-Filterung (Abschnitt 6.1.2.5).

#### DepthMaxDepth

- Typ: Float, Wertebereich: 0.1–100.0 m

- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Abstands*-Filterung (Abschnitt 6.1.2.5).

**DepthMaxDepthErr**

- Typ: Float, Wertebereich: 0.01–100.0 m
- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Fehler*-Filterung (Abschnitt 6.1.2.5).

## 7.2.5 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 7.2.7) 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 (*auto exposure mode*, Abschnitt 6.1.1.3).

Chunk-Daten werden durch das Setzen des GenICam-Parameters `ChunkModeActive` auf `True` eingeschaltet.

## 7.2.6 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.

### 7.2.7 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](#) (Abschnitt 6.1.2.2) angegebenen Gleichungen in 3D-Objektkoordinaten 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

$$d_{ik} = d16_{ik} \cdot \text{Scan3dCoordinateScale}$$

Die 3D-Rekonstruktion (in Metern) kann wie folgt mit den GenICam-Parametern durchgeführt werden:

$$P_x = (i + 0.5 - \text{Scan3dPrincipalPointU}) \frac{\text{Scan3dBaseline}}{d_{ik}},$$

$$P_y = (k + 0.5 - \text{Scan3dPrincipalPointV}) \frac{\text{Scan3dBaseline}}{d_{ik}},$$

$$P_z = \text{Scan3dFocalLength} \frac{\text{Scan3dBaseline}}{d_{ik}}.$$

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 GenICam-Parameter *Scan3dCoordinateScale* angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätsfehlerwerte  $d_{eps}$  in Pixeln zu ermitteln. Der Beschreibung in *Konfidenz- und Fehlerbilder* (Abschnitt 6.1.2.3) zufolge lässt sich der Tiefenfehler  $z_{eps}$  (in Metern) mit den GenICam-Parametern wie folgt berechnen:

$$d_{ik} = d16_{ik} \cdot \text{Scan3dCoordinateScale},$$

$$z_{eps} = \frac{e_{ik} \cdot \text{Scan3dCoordinateScale} \cdot \text{Scan3dFocalLength} \cdot \text{Scan3dBaseline}}{(d_{ik})^2}.$$

**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* (Abschnitt 6.1.2).

## 7.3 REST-API-Schnittstelle

Neben der *GenICam-Schnittstelle* (Abschnitt 7.2) bietet der *rc\_visard* 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 7.1) 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.

### 7.3.1 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 *Netzwerkconfiguration*, 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 7.3.5) erkunden und testen.

Für die eigentlichen HTTP-Anfragen wird dem Einstiegspunkt der Programmierschnittstelle die **aktuelle Version der Schnittstelle als Postfix angehängen**, 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 7.3.3) 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 Zielparame-ter 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 7.3.3) des `rc_visard` und einzelnen Parameter und Services jedes *Softwaremoduls* (Abschnitt 6).

### 7.3.2 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`.

### 7.3.3 Verfügbare Ressourcen und Anfragen

Die für die REST-API verfügbaren Ressourcen lassen sich in folgende Teilbereiche gliedern:

- **/nodes** Zugriff auf die globalen *Datenbankmodule* (Abschnitt 6.5) 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** Zugriff auf den Status und die Konfiguration der Kamerapipelines. Es gibt immer nur eine Pipeline mit der Nummer 0.
- **/pipelines/0/nodes** Zugriff auf die 3D-Kamera-, Navigations-, Detektions- und Konfigurations-*Softwaremodule* (Abschnitt 6) des *rc\_visard* mit ihren jeweiligen Laufzeitzuständen, Parametern und verfügbaren Services.
- **/templates** Zugriff auf die im *rc\_visard* hinterlegten Objekttemplates.
- **/datastreams** Zugriff auf die und Verwaltung der Datenströme der *rc\_dynamics Schnittstelle* (Abschnitt 7.4) 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.

#### 7.3.3.1 Module, Parameter und Services

Die *Softwaremodule* (Abschnitt 6) 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 6) 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 *Software-modul* (Abschnitt 6) dokumentiert.

**Bemerkung:** Die meisten Parameter, die die Module über die REST-API anbieten, lassen sich auch über die benutzerfreundliche *Web GUI* (Abschnitt 7.1) 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 7.2) 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:** Einige 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 6.4.1.5) 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 7.3.4). 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 *Software-modul* (Abschnitt 6) dokumentiert.

Die folgende Liste enthält alle REST-API-Anfragen zum Status der globalen Datenbankmodule und ihrer Parameter und Services:

#### GET /nodes

Abfrage 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

### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: NodeInfo-Array*)

### Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.3.4)

**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

**Statuscodes**

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: NodeInfo*)
- 404 Not Found – Modul nicht gefunden

**Referenzierte Datenmodelle**

- *NodeInfo* (Abschnitt 7.3.4)

**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

**Statuscodes**

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: Service-Array*)
- 404 Not Found – Modul nicht gefunden

#### Referenzierte Datenmodelle

- *Service* (Abschnitt 7.3.4)

#### 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

#### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: Service*)
- 404 Not Found – Modul oder Service nicht gefunden

#### Referenzierte Datenmodelle

- *Service* (Abschnitt 7.3.4)

#### 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

{
  "args": {}
}
```

#### Musteranfrage

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
"args": {},  
"description": "string",  
"name": "string",  
"response": {}  
}
```

**Parameter**

- **node** (*string*) – Modulname (*obligatorisch*)
- **service** (*string*) – Name des Service (*obligatorisch*)

**JSON-Objekt zur Anfrage**

- **service args** (*Service*) – Beispielargumente (*obligatorisch*)

**Anfrage-Header**

- **Accept** – application/json

**Antwort-Headers**

- **Content-Type** – application/json

**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 7.3.4)

**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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: NodeStatus*)
- **404 Not Found** – Modul nicht gefunden

### Referenzierte Datenmodelle

- [NodeStatus](#) (Abschnitt 7.3.4)

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"
    ],
    "status": "idle"
  },
]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
{
  "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

**Statuscodes**

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: NodeInfo-Array*)

**Referenzierte Datenmodelle**

- *NodeInfo* (Abschnitt 7.3.4)

**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"
  ],
  "services": [
    "reset_defaults"
  ],
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
"status": "running"
}
```

#### Parameter

- **pipeline** (*string*) – Name der Pipeline (0) (*obligatorisch*)
- **node** (*string*) – Modulname (*obligatorisch*)

#### Antwort-Headers

- **Content-Type** – application/json

#### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: NodeInfo*)
- 404 Not Found – Modul nicht gefunden

#### Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.3.4)

**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",
    "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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: NodeInfo-Array*)
- **404 Not Found** – Modul nicht gefunden

**Referenzierte Datenmodelle**

- *Parameter* (Abschnitt 7.3.4)

**PUT /pipelines/{pipeline}/nodes/{node}/parameters**

Aktualisierung mehrerer Parameter.

**Musteranfrage**

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters HTTP/1.1
Accept: application/json

[
  {
    "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,
    "name": "exp_auto",
    "type": "bool",
    "value": false
  },
  {
    "default": 0.005,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
"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

### Antwort-Headers

- **Content-Type** – application/json

### 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 7.3.4)
- *ParameterNameValue* (Abschnitt 7.3.4)

**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",
  "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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: Parameter*)
- **404 Not Found** – Modul oder Parameter nicht gefunden

**Referenzierte Datenmodelle**

- *Parameter* (Abschnitt 7.3.4)

**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

{
  "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

**Antwort-Headers**

- **Content-Type** – application/json

### 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

- *Parameter* (Abschnitt 7.3.4)
- *ParameterValue* (Abschnitt 7.3.4)

### 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"
    }
  }
]
```

### Parameter

- **pipeline** (*string*) – Name der Pipeline (0) (*obligatorisch*)
- **node** (*string*) – Modulname (*obligatorisch*)

**Antwort-Headers**

- **Content-Type** – application/json

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: Service-Array*)
- **404 Not Found** – Modul nicht gefunden

**Referenzierte Datenmodelle**

- *Service* (Abschnitt 7.3.4)

**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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: Service*)

- 404 Not Found – Modul oder Service nicht gefunden

#### Referenzierte Datenmodelle

- [Service](#) (Abschnitt 7.3.4)

#### 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

{
  "args": {}
}
```

#### 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** (*Service*) – Beispielargumente (*obligatorisch*)

#### Anfrage-Header

- **Accept** – application/json

#### Antwort-Header

- **Content-Type** – application/json

#### 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 7.3.4)

**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

**Statuscodes**

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: NodeStatus*)
- 404 Not Found – Modul nicht gefunden

**Referenzierte Datenmodelle**

- *NodeStatus* (Abschnitt 7.3.4)

**7.3.3.2 Datenströme**

Über die folgenden Ressourcen und Anfragen ist es möglich, auf die Datenströme der *rc\_dynamics-Schnittstelle* (Abschnitt 7.4) 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

#### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: Datenstrom-Array*)

#### Referenzierte Datenmodelle



- *Stream* (Abschnitt 7.3.4)

**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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: Datenstrom*)
- **404 Not Found** – Datenstrom nicht gefunden

**Referenzierte Datenmodelle**

- *Stream* (Abschnitt 7.3.4)

**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)",
  "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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: Datenstrom*)
- **404 Not Found** – Datenstrom nicht gefunden

**Referenzierte Datenmodelle**

- *Stream* (Abschnitt 7.3.4)

**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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: Datenstrom*)
- **404 Not Found** – Datenstrom nicht gefunden

**Referenzierte Datenmodelle**

- *Stream* (Abschnitt 7.3.4)

### 7.3.3.3 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, Uhrzeit-synchronisierungsstatus 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

#### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: LogInfo-Array*)

#### Referenzierte Datenmodelle

- *LogInfo* (Abschnitt 7.3.4)

#### 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 V0",
      "timestamp": 1503060034.913
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Playback mode: false",
      "timestamp": 1503060035.132
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Ready",

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "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 7.3.4)

#### 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

{
  "firmware": {
    "active_image": {
      "image_version": "rc_visard_v1.1.0"
    },
    "fallback_booted": true,
    "inactive_image": {
      "image_version": "rc_visard_v1.0.0"
    },
    "next_boot_image": "active_image"
  },
  "hostname": "rc-visard-02873515",
  "link_speed": 1000,
  "mac": "00:14:2D:2B:D8:AB",
  "ntp_status": {
    "accuracy": "48 ms",
    "synchronized": true
  },
  "ptp_status": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

"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

**Statuscodes**

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: SysInfo*)

**Referenzierte Datenmodelle**

- *SysInfo* (Abschnitt 7.3.4)

**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

**Statuscodes**

- 200 OK – Erfolgreiche Verarbeitung

**POST /system/backup**

Backup einspielen.

**Musteranfrage**

```

POST /api/v2/system/backup HTTP/1.1
Accept: application/json

{}

```

**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

#### Antwort-Headers

- **Content-Type** – application/json

#### Statuscodes

- **200 OK** – Erfolgreiche Verarbeitung

### 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

#### Statuscodes

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: LicenseInfo*)

#### Referenzierte Datenmodelle

- *LicenseInfo* (Abschnitt 7.3.4)

**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/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

**Statuscodes**

- **200 OK** – Erfolgreiche Verarbeitung (*Rückgabe: NetworkInfo*)

**Referenzierte Datenmodelle**

- *NetworkInfo* (Abschnitt 7.3.4)

**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-Header

- Content-Type – application/json

### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: NetworkSettings*)

### Referenzierte Datenmodelle

- [NetworkSettings](#) (Abschnitt 7.3.4)

**PUT** /system/network/settings

Setzen der aktuellen Netzwerkeinstellungen.

### Musteranfrage

```
PUT /api/v2/system/network/settings HTTP/1.1
Accept: application/json

{}
```

### 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

### Antwort-Header

- Content-Type – application/json

### 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 7.3.4)

#### 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": "rc_visard_v1.1.0"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "rc_visard_v1.0.0"
  },
  "next_boot_image": "active_image"
}
```

#### Antwort-Headers

- [Content-Type](#) – application/json

#### Statuscodes

- [200 OK](#) – Erfolgreiche Verarbeitung (*Rückgabe: FirmwareInfo*)

#### Referenzierte Datenmodelle

- [FirmwareInfo](#) (Abschnitt 7.3.4)

#### 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/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": "rc_visard_v1.1.0"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "rc_visard_v1.0.0"
  },
  "next_boot_image": "active_image"
}
```

##### Antwort-Headers

- Content-Type – application/json

##### Statuscodes

- 200 OK – Erfolgreiche Verarbeitung (*Rückgabe: FirmwareInfo*)

##### Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 7.3.4)

#### 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

### 7.3.4 Datentyp-Definitionen

Die REST-API definiert folgende Datenmodelle, die verwendet werden, um auf die *verfügbaren Ressourcen* (Abschnitt 7.3.3) zuzugreifen oder diese zu ändern, entweder als benötigte Attribute/Parameter oder als Rückgabewerte.

**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`

**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](#)

**NetworkInfo:** Aktuelle Netzwerk Konfiguration.

Ein Objekt des Typs NetworkInfo besitzt folgende Eigenschaften:

- **current\_method** (string) - Methode mit der die aktuellen Einstellungen gesetzt wurden (eine von 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`

**NtpStatus:** Status der NTP-Zeitsynchronisierung.

Ein Objekt des Typs NtpStatus besitzt folgende Eigenschaften:

- **accuracy** (string): vom Network Time Protocol (NTP) gemeldete Genauigkeit der Zeitsynchronisierung.
- **synchronized** (boolean): synchronisiert mit dem NTP-Server.

#### Musterobjekt

```
{
  "accuracy": "string",
  "synchronized": false
}
```

NtpStatus-Objekte sind in *SysInfo* enthalten.

**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:

- **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*
- **ntp\_status** (*NtpStatus*): siehe Beschreibung von *NtpStatus*.
- **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.
- **uptime** (float): Betriebszeit in Sekunden.

#### Musterobjekt

```
{
  "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_status": {
    "accuracy": "string",
    "synchronized": false
  },
  "ptp_status": {
    "master_ip": "string",
    "offset": 0,
    "offset_dev": 0,
    "offset_mean": 0,
    "state": "string"
  },
  "ready": false,
  "serial": "string",
  "time": 0,
  "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\_silhouettematch*
- *GET /templates/rc\_silhouettematch/{id}*
- *PUT /templates/rc\_silhouettematch/{id}*

### 7.3.5 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 7.3.3). 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*.

The screenshot displays the Swagger UI for the `rc_visard` REST API, categorized into several resource groups:

- global nodes** (Nodes that are global to all pipelines):
  - GET `/nodes`
  - GET `/nodes/{node}`
  - GET `/nodes/{node}/status`
  - GET `/nodes/{node}/services`
  - GET `/nodes/{node}/services/{service}`
  - PUT `/nodes/{node}/services/{service}`
- pipeline nodes** (Nodes that are specific to a pipeline):
  - GET `/pipelines/{pipeline}/nodes`
  - GET `/pipelines/{pipeline}/nodes/{node}`
  - GET `/pipelines/{pipeline}/nodes/{node}/status`
  - 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}`
  - GET `/pipelines/{pipeline}/nodes/{node}/services`
  - GET `/pipelines/{pipeline}/nodes/{node}/services/{service}`
  - PUT `/pipelines/{pipeline}/nodes/{node}/services/{service}`
- pipelines** (Status of active pipelines):
  - GET `/pipelines`
  - GET `/pipelines/{pipeline}`
- templates** (Template management for specific nodes):
  - GET `/templates/rc_silhouettematch`
  - GET `/templates/rc_silhouettematch/{id}`
  - PUT `/templates/rc_silhouettematch/{id}`
  - DELETE `/templates/rc_silhouettematch/{id}`
- datastreams** (Management of `rc_dynamics` data streams):
  - GET `/datastreams`
  - GET `/datastreams/{stream}`
  - PUT `/datastreams/{stream}`
  - DELETE `/datastreams/{stream}`
- system** (Query system status, configure network and handle license as well as updates):
  - GET `/system`
  - GET `/system/license`
  - POST `/system/license`
  - PUT `/system/reboot`

Abb. 7.2: 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 Schaltfläche *Try it out!* betätigt, der erforder-

derliche Parameter (pipeline-Nummer und node-Name) ausgefüllt 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.

The screenshot displays the Swagger UI for the endpoint `GET /pipelines/{pipeline}/nodes/{node}/status`. The parameters are set to `pipeline: 0` and `node: rc_stereomatching`. The response is a 200 status with a JSON body containing status, timestamp, and values for time\_matching, time\_postprocessing, latency, fps, width, height, mindepth, maxdepth, and reduced\_depth\_range.

**Parameters:**

- pipeline** (required, string, path): name of the pipeline, value: 0
- node** (required, string, path): name of the node, value: rc\_stereomatching

**Response (200):**

```

{
  "status": "running",
  "timestamp": 1642562700.7127633,
  "values": {
    "time_matching": "0.021",
    "time_postprocessing": "0.037",
    "latency": "0.065",
    "fps": "9.1",
    "width": "640",
    "height": "480",
    "mindepth": "0.4",
    "maxdepth": "100",
    "reduced_depth_range": "0"
  }
}

```

**Response headers:**

```

access-control-allow-headers: Origin,X-Requested-With,Content-Type,Accept,Authorization
access-control-allow-methods: GET,PUT,POST,DELETE
access-control-allow-origin: *
access-control-expose-headers: Location
cache-control: no-store
connection: keep-alive
content-length: 258
content-type: application/json
date: Wed, 19 Jan 2022 09:59:21 GMT
server: nginx/1.18.0 (Ubuntu)

```

**Responses:**

- 200:** successful operation
- 404:** node not found

Abb. 7.3: 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 benö-



tigten 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 Antwort enthält eine vollständige Beschreibung des angebotenen Services, einschließlich aller erforderlichen Argumente mit ihren Namen und Typen in einer Zeichenfolge im JSON-Format.

The screenshot shows a REST client interface for a GET request to the endpoint `/pipelines/{pipeline}/nodes/{node}/services/{service}`. The parameters section is filled with the following values:

- pipeline** (required, string, path): 0
- node** (required, string, path): rc\_hand\_eye\_calibration
- service** (required, string, path): set\_pose

The response section shows a 200 status code with the following JSON body:

```
{
  "response": {
    "status": "int32",
    "message": "string",
    "success": "bool"
  },
  "args": {
    "slot": "uint32",
    "pose": {
      "position": {
        "y": "float64",
        "x": "float64",
        "z": "float64"
      },
      "orientation": {
        "y": "float64",
        "x": "float64",
        "z": "float64",
        "w": "float64"
      }
    }
  },
  "name": "set_pose",
  "description": "Save a pose (grid or gripper) for later calibration."
}
```

The response headers are also visible:

```
access-control-allow-headers: Origin,X-Requested-With,Content-Type,Accept,Authorization
access-control-allow-methods: GET,PUT,POST,DELETE
access-control-allow-origin: *
access-control-expose-headers: Location
cache-control: no-store
connection: keep-alive
content-length: 346
content-type: application/json
date: Wed, 19 Jan 2022 09:03:26 GMT
server: nginx/1.14.0 (Ubuntu)
```

Abb. 7.4: 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:

PUT /pipelines/{pipeline}/nodes/{node}/services/{service}

Call a service of a node. The required args and resulting response depend on the specific node and service.

Parameters Cancel

Name	Description
pipeline * required string (path)	name of the pipeline 0
node * required string (path)	name of the node rc_hand_eye_calibration
service * required string (path)	name of the service set_pose
service args * required (body)	example args Edit Value   Model <pre>{   "args": {     "slot": 0,     "pose": {       "position": {         "x": 1.02,         "y": -0.35,         "z": 0.201       },       "orientation": {         "w": 0.0,         "x": "float64",         "y": "float64",         "z": "float64",         "w": "float64"       }     }   } }</pre>

Cancel

Parameter content type  
application/json

Execute

Abb. 7.5: Ausfüllen der Argumente des set\_pose-Services

## 7.4 Die rc\_dynamics-Schnittstelle

Die rc\_dynamics-Schnittstelle bietet über Echtzeit-Datenströme kontinuierlichen Zugang zu verschiedenen *Dynamik-Zustandsschätzungen* (Abschnitt 6.2.1.2). 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 7.4.3) unterstützt alle gängigen Betriebssysteme und Programmiersprachen.

### 7.4.1 Starten/Stoppen der Dynamik-Zustandsschätzungen

Die Dynamik-Zustandsschätzungen des rc\_visard sind nur verfügbar, wenn die zugehörige Komponente, d.h. das *Dynamik-Modul* (Abschnitt 6.2.1), 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.

## 7.4.2 Konfiguration von Datenströmen

Verfügbare Datenströme, d.h. Dynamik-Zustandsschätzungen, lassen sich über die [REST-API](#) (Abschnitt 7.3.3.2) 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 6.2.1.2).

Tab. 7.2: Datenströme, die über die rc\_dynamics-Schnittstelle verfügbar sind

Name	Protokoll	ProtoBuf	Beschreibung
dynamics	UDP	<i>Dynamics</i>	Dynamik des <i>rc_visard</i> (Pose, Geschwindigkeit, Beschleunigung), in Echtzeit (IMU-Frequenz) bereitgestellt vom INS- oder SLAM-Modul (Best-Effort-Prinzip)
dynamics_ins	UDP	<i>Dynamics</i>	Dynamik des <i>rc_visard</i> (Pose, Geschwindigkeit, Beschleunigung), in Echtzeit (IMU-Frequenz) bereitgestellt vom Stereo-INS-Modul
imu	UDP	<i>Imu</i>	Rohdaten der inertialen Messeinheit (IMU), in Echtzeit (IMU-Frequenz) bereitgestellt
pose	UDP	<i>Frame</i>	Pose der linken Kamera, mit maximaler Kamerafrequenz bereitgestellt vom INS- oder SLAM-Modul (Best-Effort-Prinzip)
pose_ins	UDP	<i>Frame</i>	Pose der linken Kamera, mit maximaler Kamerafrequenz bereitgestellt vom INS-Modul
pose_rt	UDP	<i>Frame</i>	Pose der linken Kamera, in Echtzeit (IMU-Frequenz) bereitgestellt vom INS- oder SLAM-Modul (Best-Effort-Prinzip)
pose_rt_ins	UDP	<i>Frame</i>	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 7.4.3) 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.

### 7.4.3 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 6.2.1.2) 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 6.2.1.2) 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/s²
  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 6.2.1.2) 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/s2 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;
}
```

#### 7.4.4 Die rc\_dynamics-Programmierschnittstelle

Das Open-Source-`rc_dynamics_api`-Paket bietet einen einfachen, benutzerfreundlichen C++-Wrapper, mit dem `rc_dynamics`-Datenströme angefragt und geparkt werden können. Siehe <http://www.roboception.com/download>.

## 7.5 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 EKI-Bridge-Lizenz (Abschnitt 8.7).

**Bemerkung:** Das KUKA.EthernetKRL add-on Software-Paket Version 2.2 oder neuer 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.

### 7.5.1 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 7.3). 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 6) 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.

## 7.5.2 Allgemeine XML-Struktur

Für die Datenanfrage nutzt die EKI-Bridge `<req>` als Wurzelement (kurz für „Request“).

Das Wurzelement 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 Wurzelement (kurz für „Response“). Das Wurzelement 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.

### 7.5.2.1 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. 7.3: Rückgabecodes der EKI-Bridge

Code	Beschreibung
0	Erfolgreich
-1	Parsing-Fehler in der Konvertierung von XML zu JSON
-2	Interner Fehler
-9	Fehlende oder ungültige Lizenz für das EKI-Bridge-Modul
-11	Verbindungsfehler von der REST-API

**Bemerkung:** Die EKI-Bridge liefert auch Rückgabecodes spezifisch zu den individuellen Softwaremodulen zurück. Diese sind im jeweiligen *Softwaremodul* (Abschnitt 6) 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.

### 7.5.3 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 6) 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>
```

### 7.5.3.1 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>"/>
    <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 6.5.1) 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>
  <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")

```

### 7.5.3.2 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 6.3.2) wie folgt aus.

```

<RECEIVE>
  <XML>

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

<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
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.

## 7.5.4 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="B00L"/>
</XML>

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

</SEND>
<RECEIVE>
  <XML>
    <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>
</req>

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

</node>
<parameters>
  <quality value="Low"></quality>
</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>

```

## 7.5.5 Migration zu Firmware version 22.01

Von Firmware Version 22.01 an bildet die EKI Bridge die *rc\_visard REST-API v2* (Section 7.3) ab.

Dies macht folgende Änderungen nötig:

- **Das Konfigurieren von Load Carriern, Greifern und Regions of Interest ist jetzt nur noch in den globalen Da**
  - 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.

## 7.5.6 Beispielanwendungen

Ausführlichere Beispielanwendungen können unter [https://github.com/roboception/eki\\_examples](https://github.com/roboception/eki_examples) abgerufen werden.

## 7.6 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 im Jahr 2000 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 7.3) abgerufen und darüber hinaus auch in der [Web GUI](#) (Abschnitt 7.1) 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.

### 7.6.1 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.

### 7.6.2 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 7.2) 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.

## 8 Wartung

**Warnung:** Das Gehäuse des *rc\_visard* muss für Wartungsarbeiten nicht geöffnet werden. Das unbefugte Öffnen des Produkts führt zum Erlöschen der Garantie.

### 8.1 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.

### 8.2 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 6.4.3).

### 8.3 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 7.1) auf der Seite *System* im Abschnitt *rc\_visard Einstellungen* heruntergeladen werden, oder über die *REST-API-Schnittstelle* (Abschnitt 7.3) 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

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 6.3.4.14).

Ein Backup kann auf dem *rc\_visard* über die [Web GUI](#) (Abschnitt 7.1) 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 7.3) 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.

## 8.4 Aktualisierung der Firmware

Angaben zur aktuellen Firmware-Version sind auf der Seite *System* → *Firmware & Lizenz* in der [Web GUI](#) (Abschnitt 7.1) angegeben. Diese Informationen lassen sich mithilfe einer `GET /system`-Anfrage über die [REST-API-Schnittstelle](#) (Abschnitt 7.3) 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 7.3) 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 6.4.1), 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 6.4.3), 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.

**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 & Lizenz* 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.

## 8.5 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.

**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 7.3) 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 7.3.5) 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.

## 8.6 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 7.3) des *rc\_visard* oder manuell auf der Seite *System* der *Web GUI* (Abschnitt 7.1) vornehmen.

Der Neustart ist abgeschlossen, wenn die LED wieder grün leuchtet.

## 8.7 Aktualisierung der Softwarelizenz

Lizenzen, die von Roboception zur Aktivierung zusätzlicher Funktionen erworben werden, können über die Seite *System* → *Firmware & Lizenz* der *Web GUI* (Abschnitt 7.1) installiert werden. Der *rc\_visard* muss neu gestartet werden, um die Lizenz nutzen zu können.

## 8.8 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 7.1) ansehen und filtern. Wird der Support kontaktiert (*Kontakt*, Abschnitt 11), 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 7.3) des *rc\_visard* zugänglich. Hierfür können die Anfragen des Typs `GET /logs` und `GET /logs/{log}` verwendet werden.

## 9 Zubehör

### 9.1 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:

- Netzkabel 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).

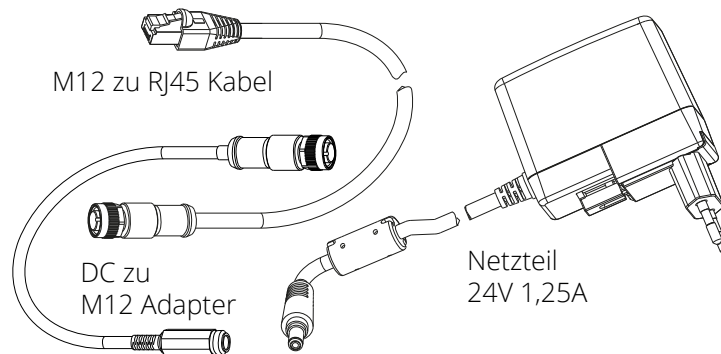


Abb. 9.1: Bestandteile des optional erhältlichem Anschlusssets

### 9.2 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.

#### 9.2.1 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

### 9.2.2 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)

### 9.2.3 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 Hutschiene-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.

## 9.3 Ersatzteile

Für den *rc\_visard* sind derzeit keine Ersatzteile erhältlich.

# 10 Fehlerbehebung

## 10.1 LED-Farben

Während des Boot-Vorgangs wechselt die LED mehrmals die Farbe, um die verschiedenen Boot-Phasen anzuzeigen:

Tab. 10.1: LED-Farbcodes

LED-Farbe	Boot-Vorgang
Weiß	Stromversorgung OK
Gelb	Normaler Boot-Vorgang
Violett	
Blau	
Grün	Boot-Vorgang abgeschlossen, <i>rc_visard</i> einsatzbereit

Die LED dient ferner dazu, Probleme oder Fehlerzustände zu signalisieren, um den Benutzer im Rahmen der Problembehandlung zu unterstützen.

Tab. 10.2: LED-Farbcodes

LED-Farbe	Problem oder Fehlerzustand
Aus	Der Sensor wird nicht mit Strom versorgt.
Kurzes rotes Blinken alle fünf Sekunden	Keine Netzwerkkonnektivität
Rot (obwohl der Sensor anscheinend normal funktioniert)	Temperaturwarnung (Gehäusetemperatur liegt über 60 °C)
Rot (obwohl die Gehäusetemperatur unter 60 °C liegt)	Ein Prozess wurde beendet und kann nicht neu gestartet werden.

## 10.2 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 leuchtet 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.

## **10.3 Probleme mit der Konnektivität**

### **LED blinkt alle 5 Sekunden rot**

Wenn die LED alle fünf Sekunden kurz rot blinkt, kann der *rc\_visard* keine Netzwerkverbindung herstellen.

- Überprüfen Sie, ob das Netzkabel 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* alle fünf Sekunden 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* alle fünf Sekunden 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.

## 10.4 Probleme mit den Kamerabildern

**Kamerabild ist zu hell**

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verkürzen (siehe *Parameter*, Abschnitt 6.1.1.3) oder
- schalten Sie auf automatische Belichtung um (siehe *Parameter*, Abschnitt 6.1.1.3).

**Kamerabild ist zu dunkel**

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verlängern (siehe *Parameter*, Abschnitt 6.1.1.3) oder
- schalten Sie auf automatische Belichtung um (siehe *Parameter*, Abschnitt 6.1.1.3).

**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 (siehe *Parameter*, Abschnitt 6.1.1.3).

**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 11).

**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 (siehe *Parameter*, Abschnitt 6.1.1.3).

**Kamerabild ist verzerrt**

- Überprüfen Sie, ob die Linsen verschmutzt sind, und reinigen Sie diese bei Bedarf (siehe *Reinigung der Kameralinsen*, Abschnitt 8.1).
- Trifft keiner der vorstehenden Punkte zu, kann es sein, dass ein schweres Hardware-Problem vorliegt. Bitte wenden Sie sich an den *Support* (Abschnitt 11).

**Bildwiederholrate ist zu niedrig**

- Erhöhen Sie die Bildwiederholrate gemäß den Anweisungen in *Parameter* (Abschnitt 6.1.1.3).
- Die maximale Bildwiederholrate der Kameras beträgt 25 Hz.

## 10.5 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 10.4).
- Überprüfen Sie, ob die Szene genügend Textur hat (siehe *Stereo-Matching*, Abschnitt 6.1.2) und installieren Sie bei Bedarf einen Musterprojektor.
- Senken Sie den *Minimalen Abstand* (Abschnitt 6.1.2.5).
- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 6.1.2.5).
- Überprüfen Sie, ob das Objekt zu nahe an der Kamera liegt. Beachten Sie die unterschiedlichen Tiefenmessbereiche der Kameravarianten.
- Senken Sie die *Minimale Konfidenz* (Abschnitt 6.1.2.5).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 6.1.2.5).
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 6.1.2.5). 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 6.4.3).

### Bildwiederholrate der Disparitätsbilder ist zu niedrig

- Überprüfen und erhöhen Sie die Bildwiederholrate der Kamerabilder (siehe *Parameter*, Abschnitt 6.1.1.3). Die Bildwiederholrate der Disparitätsbilder kann nicht größer sein als die Bildwiederholrate der Kamerabilder.
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 6.1.2.5).
- Erhöhen Sie den *Minimalen Abstand* (Abschnitt 6.1.2.5) 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 6.1.2.5).

### Disparitätsbild zeigt keine weit entfernten Objekte

- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 6.1.2.5).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 6.1.2.5).
- Senken Sie die *Minimale Konfidenz* (Abschnitt 6.1.2.5).

### Disparitätsbild rauscht zu stark

- Erhöhen Sie den *Segmentierungs-Wert* (Abschnitt 6.1.2.5).
- Erhöhen Sie den *Füllen-Wert* (Abschnitt 6.1.2.5).

### 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 6.1.2.5).

#### Disparitätsbild zeigt keine feinen Strukturen

- Senken Sie den *Segmentierungs-Wert* (Abschnitt 6.1.2.5).
- Senken Sie den *Füllen-Wert* (Abschnitt 6.1.2.5).

## 10.6 Probleme mit der Zustandsschätzung

#### Keine Zustandsschätzungen verfügbar

- Kontrollieren Sie in der Web GUI, dass das Dynamik-Modul eingeschaltet ist (siehe *Parameter*, Abschnitt 6.2.2.1).
- 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 6.2.2.1) 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 6.2.2.1) an.

#### Posenfrequenz ist zu niedrig

- Verwenden Sie den Echtzeit-Datenstrom der Kamerapose mit einer Aktualisierungsrate im 200 Hz-Bereich. Siehe *Stereo-INS* (Abschnitt 6.2.3).

#### Verzögerung/Latenz der Posenschätzung ist zu groß

- Verwenden Sie den Echtzeit-Datenstrom der Kamerapose. Siehe *Stereo-INS* (Abschnitt 6.2.3).

## 10.7 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 7.2.2).

## 10.8 Probleme mit EKI (KUKA Ethernet KRL Interface)

#### SmartPad Fehlermeldung: Limit of element memory reached

Dieser Fehler kann auftreten, wenn die Anzahl der Matches das Speicherlimit überschreitet.

- Erhöhen Sie den Wert `BUFFERING` und setzen Sie `BUFFSIZE` in den EKI Konfigurationsdateien. Passen Sie diese Einstellungen an Ihre spezielle KRC an.
- Verringern Sie den Parameter ‚Maximale Matches‘ im Detektionsmodul.

# 11 Kontakt

## 11.1 Support

Support-Anfragen können Sie uns entweder über die Seite <http://www.roboception.com/support> oder per E-Mail an [support@roboception.de](mailto:support@roboception.de) zukommen lassen.

## 11.2 Downloads

Software-SDKs usw. können von der Roboception-Homepage heruntergeladen werden: <http://www.roboception.com/download>.

## 11.3 Adresse

Roboception GmbH  
Kaflerstraße 2  
81241 München  
Deutschland

Web: <http://www.roboception.com>  
E-Mail: [info@roboception.de](mailto:info@roboception.de)  
Telefon: +49 89 889 50 79-0

# 12 Anhang

## 12.1 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

$$r_x(\alpha) = \begin{pmatrix} \sin \frac{\alpha}{2} \\ 0 \\ 0 \\ \cos \frac{\alpha}{2} \end{pmatrix}, \quad r_y(\beta) = \begin{pmatrix} 0 \\ \sin \frac{\beta}{2} \\ 0 \\ \cos \frac{\beta}{2} \end{pmatrix}, \quad r_z(\gamma) = \begin{pmatrix} 0 \\ 0 \\ \sin \frac{\gamma}{2} \\ \cos \frac{\gamma}{2} \end{pmatrix},$$

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 um-

gekehrter 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.

### 12.1.1 Rotationsmatrix und Translationsvektor

Eine Pose kann mit einer Rotationsmatrix  $R$  und einem Translationsvektor  $T$  definiert werden.

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, \quad T = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

Die Posentransformation für einen Punkt  $P$  ist

$$P' = RP + T.$$

#### 12.1.1.1 Umrechnung von Rotationsmatrizen in Quaternionen

Die Umrechnung von einer Rotationsmatrix (mit  $\det(R) = 1$ ) in eine Quaternion  $q = (x \ y \ z \ w)^T$  kann wie folgt durchgeführt werden.

$$\begin{aligned} x &= \text{sign}(r_{21} - r_{12}) \frac{1}{2} \sqrt{\max(0, 1 + r_{00} - r_{11} - r_{22})} \\ y &= \text{sign}(r_{02} - r_{20}) \frac{1}{2} \sqrt{\max(0, 1 - r_{00} + r_{11} - r_{22})} \\ z &= \text{sign}(r_{10} - r_{01}) \frac{1}{2} \sqrt{\max(0, 1 - r_{00} - r_{11} + r_{22})} \\ w &= \frac{1}{2} \sqrt{\max(0, 1 + r_{00} + r_{11} + r_{22})} \end{aligned}$$

Der sign Operator gibt -1 zurück, falls sein Argument negativ ist. Sonst wird 1 zurück gegeben. Er wird zur Wiederherstellung des 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.

#### 12.1.1.2 Umrechnung von Quaternionen in Rotationsmatrizen

Die Umrechnung von einer Quaternion  $q = (x \ y \ z \ w)^T$  mit  $\|q\| = 1$  in eine Rotationsmatrix kann wie folgt durchgeführt werden.

$$R = 2 \begin{pmatrix} \frac{1}{2} - y^2 - z^2 & xy - zw & xz + yw \\ xy + zw & \frac{1}{2} - x^2 - z^2 & yz - xw \\ xz - yw & yz + xw & \frac{1}{2} - x^2 - y^2 \end{pmatrix}$$

### 12.1.2 ABB Posenformat

ABB Roboter beschreiben eine Pose durch Position und Quaternion so wie *rc\_visard* Geräte. Es ist keine Konvertierung der Orientierung notwendig.

### 12.1.3 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  $r_z(R)r_y(P)r_x(W)$ .

#### 12.1.3.1 Umrechnung von FANUC-WPR in Quaternionen

Zur Umrechnung von  $WPR$  Winkeln in Grad in eine Quaternion  $q = (x \ y \ z \ w)^T$  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

$$\begin{aligned}x &= \cos(R_r/2) \cos(P_r/2) \sin(W_r/2) - \sin(R_r/2) \sin(P_r/2) \cos(W_r/2), \\y &= \cos(R_r/2) \sin(P_r/2) \cos(W_r/2) + \sin(R_r/2) \cos(P_r/2) \sin(W_r/2), \\z &= \sin(R_r/2) \cos(P_r/2) \cos(W_r/2) - \cos(R_r/2) \sin(P_r/2) \sin(W_r/2), \\w &= \cos(R_r/2) \cos(P_r/2) \cos(W_r/2) + \sin(R_r/2) \sin(P_r/2) \sin(W_r/2).\end{aligned}$$

#### 12.1.3.2 Umrechnung von Quaternionen in FANUC-WPR

Die Umrechnung von einer Quaternion  $q = (x \ y \ z \ w)^T$  mit  $\|q\| = 1$  in  $WPR$  Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned}R &= \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2)) \frac{180}{\pi} \\P &= \text{asin}(2(wy - zx)) \frac{180}{\pi} \\W &= \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2)) \frac{180}{\pi}\end{aligned}$$

### 12.1.4 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  $r_z(O)r_y(A)r_z(T)$ .

#### 12.1.4.1 Umrechnung von Kawasaki-OAT in Quaternionen

Zur Umrechnung von  $OAT$  Winkeln in Grad in eine Quaternion  $q = (x \ y \ z \ w)^T$  werden zunächst alle Winkel in das Bogenmaß umgerechnet durch

$$\begin{aligned}O_r &= O \frac{\pi}{180}, \\A_r &= A \frac{\pi}{180}, \\T_r &= T \frac{\pi}{180},\end{aligned}$$

und damit wird die Quaternion berechnet durch

$$\begin{aligned}x &= \cos(O_r/2) \sin(A_r/2) \sin(T_r/2) - \sin(O_r/2) \sin(A_r/2) \cos(T_r/2), \\y &= \cos(O_r/2) \sin(A_r/2) \cos(T_r/2) + \sin(O_r/2) \sin(A_r/2) \sin(T_r/2), \\z &= \sin(O_r/2) \cos(A_r/2) \cos(T_r/2) + \cos(O_r/2) \cos(A_r/2) \sin(T_r/2), \\w &= \cos(O_r/2) \cos(A_r/2) \cos(T_r/2) - \sin(O_r/2) \cos(A_r/2) \sin(T_r/2).\end{aligned}$$

#### 12.1.4.2 Umrechnung von Quaternionen in Kawasaki-OAT

Die Umrechnung von einer Quaternion  $q = (x \ y \ z \ w)^T$  mit  $\|q\| = 1$  in OAT Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned}O &= \operatorname{atan}_2(2(yz - wx), 2(xz + wy)) \frac{180}{\pi} \\A &= \operatorname{acos}(w^2 - x^2 - y^2 + z^2) \frac{180}{\pi} \\T &= \operatorname{atan}_2(2(yz + wx), -2(xz - wy)) \frac{180}{\pi}\end{aligned}$$

#### 12.1.5 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  $r_z(A)r_y(B)r_x(C)$ .

##### 12.1.5.1 Umrechnung von KUKA-ABC in Quaternionen

Zur Umrechnung von  $ABC$  Winkeln in Grad in eine Quaternion  $q = (x \ y \ z \ w)^T$  werden zuerst alle Winkel in das Bogenmaß umgerechnet mit

$$\begin{aligned}A_r &= A \frac{\pi}{180}, \\B_r &= B \frac{\pi}{180}, \\C_r &= C \frac{\pi}{180},\end{aligned}$$

und damit die Quaternion berechnet durch

$$\begin{aligned}x &= \cos(A_r/2) \cos(B_r/2) \sin(C_r/2) - \sin(A_r/2) \sin(B_r/2) \cos(C_r/2), \\y &= \cos(A_r/2) \sin(B_r/2) \cos(C_r/2) + \sin(A_r/2) \cos(B_r/2) \sin(C_r/2), \\z &= \sin(A_r/2) \cos(B_r/2) \cos(C_r/2) - \cos(A_r/2) \sin(B_r/2) \sin(C_r/2), \\w &= \cos(A_r/2) \cos(B_r/2) \cos(C_r/2) + \sin(A_r/2) \sin(B_r/2) \sin(C_r/2).\end{aligned}$$

##### 12.1.5.2 Umrechnung von Quaternionen in KUKA-ABC

Die Umrechnung von einer Quaternion  $q = (x \ y \ z \ w)^T$  mit  $\|q\| = 1$  in  $ABC$  Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned}A &= \operatorname{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2)) \frac{180}{\pi} \\B &= \operatorname{asin}(2(wy - zx)) \frac{180}{\pi} \\C &= \operatorname{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2)) \frac{180}{\pi}\end{aligned}$$

### 12.1.6 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 12.1.5), 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  $r_z(C)r_y(B)r_x(A)$ .

#### 12.1.6.1 Umrechnung von Mitsubishi-ABC in Quaternionen

Zur Umrechnung von  $ABC$  Winkeln in Grad in eine Quaternion  $q = (x \ y \ z \ w)^T$  werden die Winkel zunächst ins Bogenmaß umgerechnet mit

$$\begin{aligned} A_r &= A \frac{\pi}{180}, \\ B_r &= B \frac{\pi}{180}, \\ C_r &= C \frac{\pi}{180}, \end{aligned}$$

und damit die Quaternion berechnet durch

$$\begin{aligned} x &= \cos(C_r/2) \cos(B_r/2) \sin(A_r/2) - \sin(C_r/2) \sin(B_r/2) \cos(A_r/2), \\ y &= \cos(C_r/2) \sin(B_r/2) \cos(A_r/2) + \sin(C_r/2) \cos(B_r/2) \sin(A_r/2), \\ z &= \sin(C_r/2) \cos(B_r/2) \cos(A_r/2) - \cos(C_r/2) \sin(B_r/2) \sin(A_r/2), \\ w &= \cos(C_r/2) \cos(B_r/2) \cos(A_r/2) + \sin(C_r/2) \sin(B_r/2) \sin(A_r/2). \end{aligned}$$

#### 12.1.6.2 Umrechnung von Quaternionen in Mitsubishi-ABC

Die Umrechnung von einer Quaternion  $q = (x \ y \ z \ w)^T$  mit  $\|q\| = 1$  in  $ABC$  Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned} A &= \operatorname{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2)) \frac{180}{\pi} \\ B &= \operatorname{asin}(2(wy - zx)) \frac{180}{\pi} \\ C &= \operatorname{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2)) \frac{180}{\pi} \end{aligned}$$

### 12.1.7 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  $V = (RX \ RY \ RZ)^T$ . Der Rotationswinkel  $\theta$  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.

#### 12.1.7.1 Umrechnung vom Angle-Axis Format in Quaternionen

Die Umrechnung von einem Rotationsvektor  $V$  in eine Quaternion  $q = (x \ y \ z \ w)^T$  kann wie folgt durchgeführt werden.

Zunächst wird der Winkel  $\theta$  im Bogenmaß aus dem Rotationsvektor  $V$  gewonnen durch

$$\theta = \sqrt{RX^2 + RY^2 + RZ^2}.$$

Wenn  $\theta = 0$ , dann ist die Quaternion gleich  $q = ( 0 \ 0 \ 0 \ 1 )^T$ , sonst wird sie berechnet durch

$$\begin{aligned}x &= RX \frac{\sin(\theta/2)}{\theta}, \\y &= RY \frac{\sin(\theta/2)}{\theta}, \\z &= RZ \frac{\sin(\theta/2)}{\theta}, \\w &= \cos(\theta/2).\end{aligned}$$

### 12.1.7.2 Umrechnung von Quaternionen ins Angle-Axis Format

Die Umrechnung von einer Quaternion  $q = ( x \ y \ z \ w )^T$  mit  $\|q\| = 1$  in einen Rotationsvektor im Angle-Axis Format kann wie folgt durchgeführt werden.

Zunächst wird der Winkel  $\theta$  im Bogenmaß aus dem Quaternion gewonnen durch

$$\theta = 2 \cdot \text{acos}(w).$$

Wenn  $\theta = 0$ , dann ist der Rotationsvektor  $V = ( 0 \ 0 \ 0 )^T$ , sonst wird er berechnet durch

$$\begin{aligned}RX &= \theta \frac{x}{\sqrt{1-w^2}}, \\RY &= \theta \frac{y}{\sqrt{1-w^2}}, \\RZ &= \theta \frac{z}{\sqrt{1-w^2}}.\end{aligned}$$

### 12.1.8 Fruitcore HORST Posenformat

Fruitcore HORST Roboter beschreiben eine Pose durch Position in Metern und Quaternion so wie *rc\_visard* Geräte. Es ist keine Konvertierung notwendig.



# HTTP Routing Table

## /datastreams

GET /datastreams, 255  
GET /datastreams/{stream}, 256  
PUT /datastreams/{stream}, 256  
DELETE /datastreams/{stream}, 257

## /logs

GET /logs, 258  
GET /logs/{log}, 258

## /nodes

GET /nodes, 240  
GET /nodes/{node}, 241  
GET /nodes/{node}/services, 242  
GET /nodes/{node}/services/{service}, 243  
GET /nodes/{node}/status, 244  
PUT /nodes/{node}/services/{service}, 243

## /pipelines

GET /pipelines/{pipeline}/nodes, 245  
GET /pipelines/{pipeline}/nodes/{node}, 246  
GET /pipelines/{pipeline}/nodes/{node}/parameters,  
247  
GET /pipelines/{pipeline}/nodes/{node}/parameters/{param},  
249  
GET /pipelines/{pipeline}/nodes/{node}/services,  
251  
GET /pipelines/{pipeline}/nodes/{node}/services/{service},  
252  
GET /pipelines/{pipeline}/nodes/{node}/status,  
253  
PUT /pipelines/{pipeline}/nodes/{node}/parameters,  
248  
PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param},  
250  
PUT /pipelines/{pipeline}/nodes/{node}/services/{service},  
253

## /system

GET /system, 260  
GET /system/backup, 261  
GET /system/license, 262  
GET /system/network, 263  
GET /system/network/settings, 263  
GET /system/rollback, 265  
GET /system/update, 266  
POST /system/backup, 261  
POST /system/license, 262

POST /system/update, 266  
PUT /system/network/settings, 264  
PUT /system/reboot, 265  
PUT /system/rollback, 265

## /templates

GET /templates/rc\_silhouettematch, 156  
GET /templates/rc\_silhouettematch/{id}, 157  
PUT /templates/rc\_silhouettematch/{id}, 157  
DELETE /templates/rc\_silhouettematch/{id},  
158

# Stichwortverzeichnis

## Sonderzeichen

3D-Koordinaten, [43](#)

Disparitätsbild, [43](#)

3D-Modellierung, [43](#), [56](#)

## A

Abmessungen

Load Carrier, [200](#)

rc\_visard, [17](#)

Abteil

Load Carrier, [203](#)

AcquisitionAlternateFilter

GenICam, [231](#)

AcquisitionFrameRate

GenICam, [227](#)

AcquisitionMultiPartMode

GenICam, [231](#)

AdaptiveOut1

automatische Belichtung, [37](#)

aktive Partition, [296](#)

Anschlussset, [298](#)

AprilTag, [89](#)

Marker-Wiedererkennung, [93](#)

Posenschätzung, [92](#)

Rückgabecodes, [101](#)

Services, [96](#)

Aufnahmemodus

Disparitätsbild, [46](#)

automatisch

Belichtung, [37](#)

automatische Belichtung, [37](#), [38](#)

AdaptiveOut1, [37](#)

Normal, [37](#)

Out1High, [37](#)

## B

Backup

Einstellungen, [294](#)

BalanceRatio

GenICam, [228](#)

BalanceRatioSelector

GenICam, [228](#)

BalanceWhiteAuto

GenICam, [228](#)

Baseline

GenICam, [232](#)

Basisabstand, [34](#)

Basisebene

SilhouetteMatch, [128](#)

Baumer

IpConfigTool, [29](#)

Belichtung

automatisch, [37](#)

manuell, [37](#)

Belichtungsregion, [39](#)

Belichtungszeit, [34](#), [39](#)

Maximum, [38](#)

Beschleunigung, [56](#), [57](#)

Sensordynamik, [31](#)

Betriebsbedingungen, [19](#)

Bewegungsunschärfe, [38](#)

Bild

Zeitstempel, [44](#), [235](#)

Bildauflösung, [16](#)

Bilder

Download, [34](#)

Bildmerkmale

visuelle Odometrie, [62](#), [63](#), [65](#)

Bildrauschen, [38](#)

Bildwiederholrate, [16](#)

GenICam, [227](#)

Kamera, [36](#)

Pose, [56](#), [57](#)

visuelle Odometrie, [63](#)

Bin-Picking, [101](#)

BoxPick, [101](#)

Füllstand, [76](#)

Greifpunktsortierung, [103](#)

Griff, [103](#)

Load Carrier, [75](#), [200](#)

Objektmodell, [102](#)

Parameter, [105](#)

Rückgabecodes, [126](#)

Region of Interest, [208](#)

services, [111](#)

Statuswerte, [110](#)

Brennweite, [34](#)

Brennweitenfaktor

GenICam, [232](#)

## C

CAD-Modell, [18](#)

Chunk-Daten

GenICam, [231](#)

collision check, [181](#), [215](#)

CollisionCheck, [181](#)

- Rückgabecodes, 188
- ComponentEnable
  - GenICam, 227
- ComponentIDValue
  - GenICam, 227
- ComponentSelector
  - GenICam, 226
- Confidence
  - GenICam Bild-Stream, 234
- D**
- Daten
  - IMU, 57
  - Inertialmesseinheit, 57
- Datenmodell
  - REST-API, 267
- Datenstrom
  - IMU, 57
  - Pose, 56
  - pose\_rt, 56, 57
  - REST-API, 254
  - Sensordynamik, 56
- Datentyp
  - REST-API, 267
- Definition
  - Load Carrier, 200
- DepthAcquisitionMode
  - GenICam, 232
- DepthAcquisitionTrigger
  - GenICam, 232
- DepthDoubleShot
  - GenICam, 233
- DepthFill
  - GenICam, 233
- DepthMaxDepth
  - GenICam, 233
- DepthMaxDepthErr
  - GenICam, 234
- DepthMinConf
  - GenICam, 233
- DepthMinDepth
  - GenICam, 233
- DepthQuality
  - GenICam, 233
- DepthSeg
  - GenICam, 233
- DepthSmooth
  - GenICam, 233
- DepthStaticScene
  - GenICam, 233
- Detektion
  - Load Carrier, 75
- DHCP, 11
- DHCP, 28
- discovery GUI, 26
- Disparität, 30, 34, 42
  - GenICam Bild-Stream, 234
- Disparitätsbereich
  - visuelle Odometrie, 64
- Disparitätsbild, 30, 42
  - 3D-Koordinaten, 43
  - Aufnahmemodus, 46
  - double\_shot, 48
  - Parameter, 44
  - Qualität, 47
  - smooth, 49
  - static\_scene, 48
  - Timeout Belichtungsautomatik, 47
  - Web GUI, 44
- Disparitätsfehler, 43
- DNS, 11
- DOF, 11
- double\_shot
  - Disparitätsbild, 48
  - GenICam, 233
- Download
  - Bilder, 34
  - Einstellungen, 294
  - Logdateien, 297
  - Punktwolke, 44
- Dynamik
  - REST-API, 254
  - Web GUI, 63
- Dynamik-Datenstrom, 56
- dynamischer Zustand, 31
- E**
- Echtzeit-Pose, 56
- Ecken
  - visuelle Odometrie, 62, 65
- Eigenbewegung, 31, 62
- Einstellungen
  - Backup, 294
  - Download, 294
  - Upload, 294
  - Wiederherstellung, 294
- eki, 285
- Erkennung
  - Marker, 88
- Error
  - GenICam Bild-Stream, 234
- Ersatzteile, 299
- Ethernet
  - Pin-Belegung, 21
- ExposureAuto
  - GenICam, 227
- ExposureRegionHeight
  - GenICam, 232
- ExposureRegionOffsetX
  - GenICam, 231
- ExposureRegionOffsetY
  - GenICam, 231
- ExposureRegionWidth
  - GenICam, 231
- ExposureTime
  - GenICam, 228

ExposureTimeAutoMax  
 GenICam, 231  
 externes Referenzkoordinatensystem  
 Hand-Auge-Kalibrierung, 159

## F

Füllen, 50  
 GenICam, 233  
 Füllstand  
 BoxPick, 76  
 ItemPick, 76  
 LoadCarrier, 76  
 SilhouetteMatch, 76  
 Fehler, **43**  
 Hand-Auge-Kalibrierung, 169  
 Pose, 68  
 Feuchtigkeit, 19  
 Firmware  
 Mender, 295  
 Rollback, 296  
 Update, 295  
 Version, 295  
 FocalLengthFactor  
 GenICam, 232  
 fps, *siehe* Bildwiederholrate

## G

Gain  
 GenICam, 228  
 Gehäusetemperatur  
 LED, 19  
 GenICam, **11**  
 GenICam  
 AcquisitionAlternateFilter, 231  
 AcquisitionFrameRate, 227  
 AcquisitionMultiPartMode, 231  
 BalanceRatio, 228  
 BalanceRatioSelector, 228  
 BalanceWhiteAuto, 228  
 Baseline, 232  
 Bildwiederholrate, 227  
 Brennweitenfaktor, 232  
 Chunk-Daten, 231  
 ComponentEnable, 227  
 ComponentIDValue, 227  
 ComponentSelector, 226  
 DepthAcquisitionMode, 232  
 DepthAcquisitionTrigger, 232  
 DepthDoubleShot, 233  
 DepthFill, 233  
 DepthMaxDepth, 233  
 DepthMaxDepthErr, 234  
 DepthMinConf, 233  
 DepthMinDepth, 233  
 DepthQuality, 233  
 DepthSeg, 233  
 DepthSmooth, 233  
 DepthStaticScene, 233

double\_shot, 233  
 ExposureAuto, 227  
 ExposureRegionHeight, 232  
 ExposureRegionOffsetX, 231  
 ExposureRegionOffsetY, 231  
 ExposureRegionWidth, 231  
 ExposureTime, 228  
 ExposureTimeAutoMax, 231  
 Füllen, 233  
 FocalLengthFactor, 232  
 Gain, 228  
 Height, 227  
 HeightMax, 227  
 LineSelector, 229  
 LineSource, 229  
 LineStatus, 229  
 LineStatusAll, 229  
 maximaler Abstand, 233  
 maximaler Fehler, 234  
 minimale Konfidenz, 233  
 minimaler Abstand, 233  
 PixelFormat, 227, 234  
 PtpEnable, 229  
 Qualität, 233  
 RcExposureAutoAverageMax, 232  
 RcExposureAutoAverageMin, 232  
 Scan3dBaseline, 230  
 Scan3dCoordinateOffset, 230  
 Scan3dCoordinateScale, 230  
 Scan3dDistanceUnit, 229  
 Scan3dFocalLength, 230  
 Scan3dInvalidDataFlag, 230  
 Scan3dInvalidDataValue, 230  
 Scan3dOutputMode, 229  
 Scan3dPrinciplePointU, 230  
 Scan3dPrinciplePointV, 230  
 Segmentierung, 233  
 smooth, 233  
 static\_scene, 233  
 Width, 227  
 WidthMax, 227  
 Zeitstempel, 235  
 GenICam Bild-Stream  
 Confidence, 234  
 Disparität, 234  
 Error, 234  
 Intensity, 234  
 IntensityCombined, 234  
 Umwandlung, 235  
 Geschwindigkeit  
 linear, 56  
 Sensordynamik, 31  
 Winkel-, 56, 57  
 GigE, **11**  
 GigE Vision, *siehe* GenICam  
 IP-Adresse, 29  
 GigE Vision, **11**  
 GM", 30

## GPIO

- Pin-Belegung, 21

- Griffberechnung, 101

- GripperDB, **215**

- Rückgabecodes, 222

## H

- Hand-Auge-Kalibrierung

- externes Referenzkoordinatensystem, 159

- Fehler, 169

- Kalibrierung, 163

- Parameter, 170

- Roboterkoordinatensystem, 159

- Sensormontage, 160

- Slot, 166

- Height

- GenICam, 227

- HeightMax

- GenICam, 227

- Host-Name, 28

## I

- IMU, **11**

- IMU, 31

- Daten, 57

- Datenstrom, 57

- Inertialmesseinheit, 63

- inaktive Partition, 296

- Inertialmesseinheit

- Daten, 57

- IMU, 63

- Innenvolumen

- Load Carrier, 200

- INS, **11**

- INS, 31

- Installation, 25

- Intensity

- GenICam Bild-Stream, 234

- IntensityCombined

- GenICam Bild-Stream, 234

- IP, **11**

- IP-Adresse, **11**

- IP-Adresse, 27

- GigE Vision, 29

- IP 54, 19

- IpConfigTool

- Baumer, 29

- ItemPick, **101**

- Füllstand, 76

- Greifpunktsortierung, 103

- Griff, 103

- Load Carrier, 75, 200

- Parameter, 105

- Rückgabecodes, 126

- Region of Interest, 208

- services, 111

- Statuswerte, 110

## K

- Kühlung, 19

- Kabel, 20, 298

- Kalibriermuster, 190

- Kalibrierung

- Hand-Auge-Kalibrierung, 163

- Kamera, 189

- Kamera-zu-IMU, 56

- Rektifizierung, 34

- Kalibrierung der Basisebene

- SilhouetteMatch, 128

- Kamera

- Bildwiederholrate, 36

- Kalibrierung, 189

- Parameter, 34, 36

- Posen-Datenstrom, 56

- Web GUI, 34

- Kamera-zu-IMU

- Kalibrierung, 56

- Transformation, 56

- Kamerakalibrierung

- Monokalibrierung, 194

- Parameter, 195

- Services, 196

- Stereokalibrierung, 192

- Kameramodell, 34

- Keyframes, 63

- visuelle Odometrie, 62, 65

- Komponenten

- rc\_visard, 15

- Konfidenz, **43**

- Minimum, 50

- Koordinatensysteme

- Montage, 23

- Sensordynamik, 56

- Zustandsschätzung, 54

- Korrespondenzen

- visuelle Odometrie, 63

## L

- LED, 25

- Farben, 300

- Gehäusetemperatur, 19

- linear

- Geschwindigkeit, 56

- LineSelector

- GenICam, 229

- LineSource

- GenICam, 229

- LineStatus

- GenICam, 229

- LineStatusAll

- GenICam, 229

- Link-Local, **11**

- Link-Local, 29

- Load Carrier

- Abmessungen, 200

- Abteil, 203

- BoxPick, 75, 200
- Definition, 200
- Detektion, 75
- Innenvolumen, 200
- ItemPick, 75, 200
- Orientierungsprior, 200
- Pose, 200
- Rand, 200
- SilhouetteMatch, 75, 200
- Load Carrier Erkennung, 75
- Load Carrier Modell, 200
- LoadCarrier, **75**
  - Füllstand, 76
  - Parameter, 78
  - Rückgabecodes, 88
  - Services, 79
- LoadCarrierDB, **200**
  - Rückgabecodes, 207
  - Services, 204
- Logdateien
  - Download, 297
- Logs
  - REST-API, 258
- M**
- MAC-Adresse, **11**
- MAC-Adresse, 28
- manuelle Belichtung, 37, 39
- Marker-Wiedererkennung
  - AprilTag, 93
  - QR-Code, 93
- Markererkennung, **88**
  - Familien, 89
  - Marker-Wiedererkennung, 93
  - Posenschätzung, 92
  - Services, 96
- maximaler Abstand, 49
  - GenICam, 233
- maximaler Fehler, 51
  - GenICam, 234
- Maximum
  - Belichtungszeit, 38
  - Tiefenfehler, 51
- mDNS, **11**
- Mender
  - Firmware, 295
- minimale Konfidenz, 50
  - GenICam, 233
- minimaler Abstand, 49
  - GenICam, 233
- Minimum
  - Konfidenz, 50
- Monokalibrierung
  - Kamerakalibrierung, 194
- Montage, 23
- N**
- Netzteil, 299
- Netzwerkkabel, 298
- Netzwerkkonfiguration, 27
- Neustart, 297
- node
  - REST-API, 239
- Normal
  - automatische Belichtung, 37
- NTP, **11**
- NTP
  - Synchronisierung, 293
- O**
- Objekterkennung, 126
- Orientierungsprior
  - Load Carrier, 200
- OutHigh
  - automatische Belichtung, 37
- P**
- Parameter
  - Disparitätsbild, 44
  - Hand-Auge-Kalibrierung, 170
  - Kamera, 34, 36
  - Kamerakalibrierung, 195
  - REST-API, 239
  - Services, 41
  - visuelle Odometrie, 63
- Pin-Belegung
  - Ethernet, 21
  - GPIO, 21
  - Stromzufuhr, 21
- PixelFormat
  - GenICam, 227, 234
- Pose
  - Bildwiederholrate, 56, 57
  - Datenstrom, 56
  - Fehler, 68
  - Load Carrier, 200
  - Sensordynamik, 31
  - Zeitstempel, 55
- pose\_rt
  - Datenstrom, 56, 57
- Posen-Datenstrom, 56, 57
  - Kamera, 56
- Posenschätzung, *siehe* Zustandsschätzung
  - AprilTag, 92
  - QR-Code, 92
- possible\_jump
  - Sensordynamik, 57
  - SLAM, 57
- PTP, **12**
- PTP
  - Synchronisierung, 229, 293
- PtpEnable
  - GenICam, 229
- Punktwolke, 43
  - Download, 44

## Q

- QR-Code, **89**
  - Marker-Wiedererkennung, **93**
  - Posenschätzung, **92**
  - Rückgabecodes, **101**
  - Services, **96**
- Qualität
  - Disparitätsbild, **47**
  - GenICam, **233**
- Quaternion
  - Rotation, **56**

## R

- Rückgabecodes
  - AprilTag, **101**
  - BoxPick, **126**
  - CollisionCheck, **188**
  - GripperDB, **222**
  - ItemPick, **126**
  - LoadCarrier, **88**
  - LoadCarrierDB, **207**
  - QR-Code, **101**
  - RoiDB, **214**
  - SilhouetteMatch, **155**
- Rand
  - Load Carrier, **200**
- rc\_dynamics, **281**
- rc\_visard
  - Komponenten, **15**
- RcExposureAutoAverageMax
  - GenICam, **232**
- RcExposureAutoAverageMin
  - GenICam, **232**
- Rektifizierung, **34**
- REST-API, **236**
  - Datenmodell, **267**
  - Datenstrom, **254**
  - Datentyp, **267**
  - Dynamik, **254**
  - Einstiegspunkt, **236**
  - Logs, **258**
  - node, **239**
  - Parameter, **239**
  - Services, **240**
  - Statuswert, **239**
  - System, **258**
  - Version, **236**
- Roboterkoordinatensystem
  - Hand-Auge-Kalibrierung, **159**
- ROI, **208**
- RoiDB, **208**
  - Rückgabecodes, **214**
  - Services, **209**
- Rollback
  - Firmware, **296**
- Rotation
  - Quaternion, **56**

## S

- Scan3dBaseline
  - GenICam, **230**
- Scan3dCoordinateOffset
  - GenICam, **230**
- Scan3dCoordinateScale
  - GenICam, **230**
- Scan3dDistanceUnit
  - GenICam, **229**
- Scan3dFocalLength
  - GenICam, **230**
- Scan3dInvalidDataFlag
  - GenICam, **230**
- Scan3dInvalidDataValue
  - GenICam, **230**
- Scan3dOutputMode
  - GenICam, **229**
- Scan3dPrinciplePointU
  - GenICam, **230**
- Scan3dPrinciplePointV
  - GenICam, **230**
- Schleifenschluss, **68**
- Schutzklasse, **19**
- SDK, **12**
- Segmentierung, **50**
  - GenICam, **233**
- Selbstkalibrierung, **189**
- Semi-Global Matching, *siehe* SGM
- Sensordatenfusion, **63**
- Sensordynamik
  - Beschleunigung, **31**
  - Datenstrom, **56**
  - Geschwindigkeit, **31**
  - Koordinatensysteme, **56**
  - Pose, **31**
  - possible\_jump, **57**
  - Services, **57**
- Sensormontage
  - Hand-Auge-Kalibrierung, **160**
- Seriennummer, **26, 28**
- Services
  - AprilTag, **96**
  - Kamerakalibrierung, **196**
  - Markererkennung, **96**
  - Parameter, **41**
  - QR-Code, **96**
  - REST-API, **240**
  - Sensordynamik, **57**
  - visuelle Odometrie, **66**
- SGM, **12**
- SGM, **42**
- Silhouette, **126**
- SilhouetteMatch, **126**
  - Basisebene, **128**
  - bevorzugte TCP-Orientierung, **130**
  - Füllstand, **76**
  - Greifpunkte, **129**
  - Kalibrierung der Basisebene, **128**

- Kollisionsprüfung, 134
  - Load Carrier, 75, 200
  - Objekt-Template, 129
  - Objekterkennung, 131
  - Parameter, 134
  - Rückgabecodes, 155
  - Region of Interest, 129, 208
  - Services, 138
  - Sortierung, 131
  - Statuswerte, 138
  - Template API, 156
  - Template Download, 156
  - Template löschen, 156
  - Template Upload, 156
  - Simultane Lokalisierung und Kartierung, *siehe* SLAM
  - SLAM, **12**
  - SLAM, **68**
    - possible\_jump, 57
    - Web GUI, 68
  - Slot
    - Hand-Auge-Kalibrierung, 166
  - smooth
    - Disparitätsbild, 49
    - GenICam, 233
  - Spezifikationen
    - rc\_visard, 16
  - static\_scene
    - Disparitätsbild, 48
    - GenICam, 233
  - Stativ, 23
  - Statuswert
    - REST-API, 239
  - Stereo-Matching, 30
  - Stereokalibrierung
    - Kamerakalibrierung, 192
  - Stereokamera, 34
  - Stromkabel, 298, 299
  - Stromversorgung, **19**
  - Stromzufuhr
    - Pin-Belegung, 21
  - Swagger UI, 277
  - Synchronisierung
    - NTP, 293
    - PTP, 229, 293
    - Zeit, 229, 292
  - System
    - REST-API, 258
- T**
- TCP, **12**
  - Temperaturbereich, 19
  - Textur, 42
  - Tiefenbild, 42, **43**, 43
    - Web GUI, 44
  - Tiefenfehler
    - Maximum, 51
  - Timeout Belichtungsautomatik
    - Disparitätsbild, 47
  - Transformation
    - Kamera-zu-IMU, 56
  - Translation, 56
- U**
- UDP, **12**
  - Umwandlung
    - GenICam Bild-Stream, 235
  - Update
    - Firmware, 295
  - Upload
    - Einstellungen, 294
  - URI, **12**
  - URL, **12**
- V**
- Version
    - Firmware, 295
    - REST-API, 236
  - Verstärkungsfaktor, 34, 38, 40
  - Visuelle Odometrie, 31
  - visuelle Odometrie, **62**
    - Bildmerkmale, 62, 63, 65
    - Bildwiederholrate, 63
    - Disparitätsbereich, 64
    - Ecken, 62, 65
    - Keyframes, 62, 65
    - Korrespondenzen, 63
    - Parameter, 63
    - Services, 66
    - Web GUI, 63
  - V0, *siehe* visuelle Odometrie
- W**
- Web GUI, **223**
    - Backup, 294
    - Disparitätsbild, 44
    - Dynamik, 63
    - Kamera, 34
    - Logs, 297
    - SLAM, 68
    - Tiefenbild, 44
    - Update, 295
    - visuelle Odometrie, 63
  - Weißabgleich, 40
  - Width
    - GenICam, 227
  - WidthMax
    - GenICam, 227
  - Wiederherstellung
    - Einstellungen, 294
  - Winkel-
    - Geschwindigkeit, 56, 57
- X**
- XYZ+Quaternion, **12**
  - XYZABC, **12**



## Z

### Zeit

Synchronisierung, [229](#), [292](#)

### Zeitstempel

Bild, [44](#), [235](#)

GenICam, [235](#)

Pose, [55](#)

### Zurücksetzen, [26](#)

### Zustandsschätzung, [55](#)

Koordinatensysteme, [54](#)

# 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-random.com>

**Shop:**

<https://roboception.com/shop>

### Für Kundensupport kontaktieren Sie

+49 89 889 50 790  
(09:00-17:00 CET)

support@roboception.de

