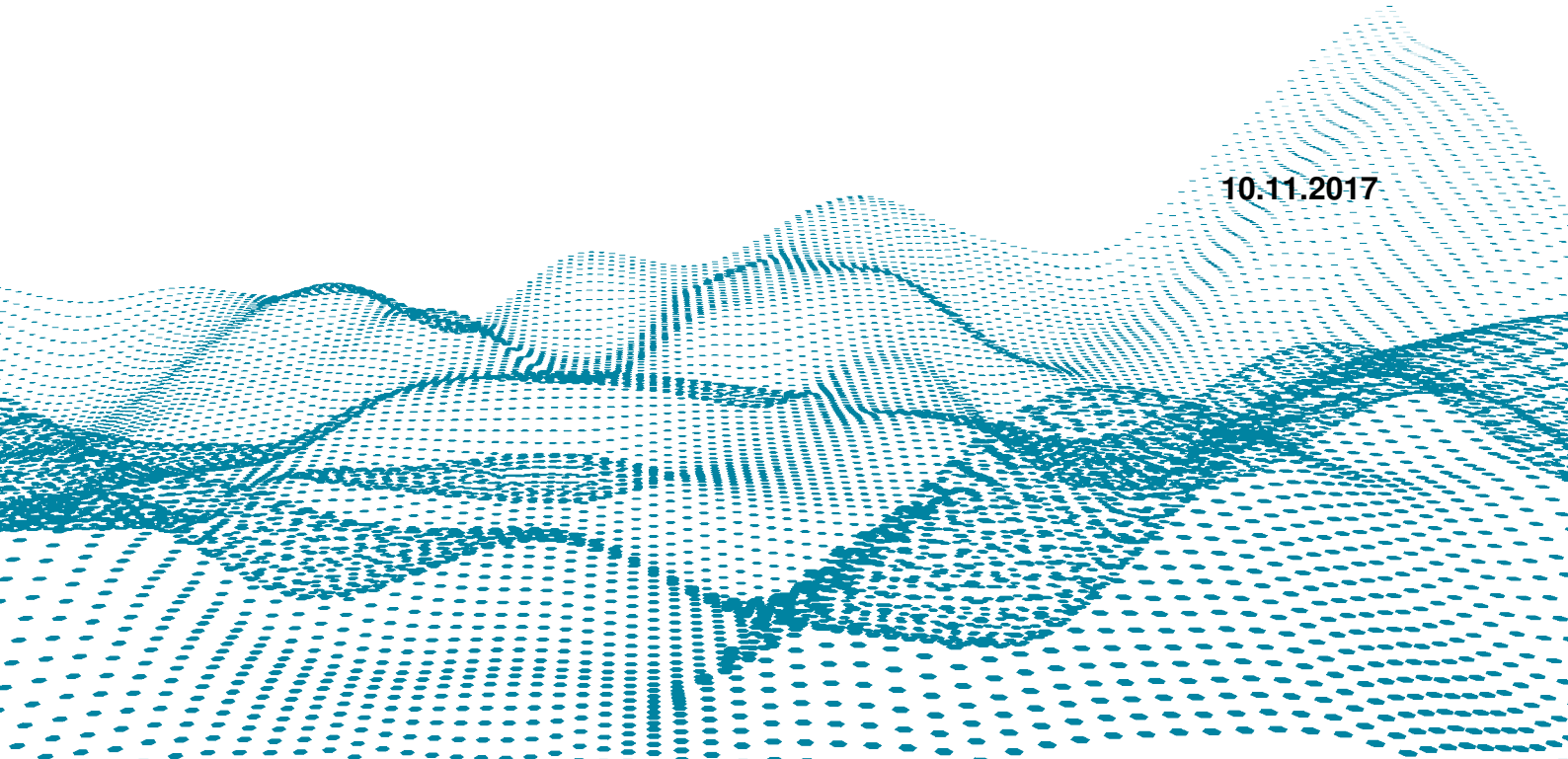
rc_visard

Release 1.0.5

Roboception GmbH

10.11.2017



Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Überblick . . . 2
- 1.2 Garantie . . . 3
- 1.3 Schnittstellen, Zulassungen und Normen . . . 4
- 1.4 Glossar . . . 5

2 Sicherheit **7**
- 2.1 Allgemeine Sicherheitshinweise . . . 7
- 2.2 Bestimmungsgemäße Verwendung . . . 8

3 Hardware-Spezifikation **9**
- 3.1 Lieferumfang . . . 9
- 3.2 Technische Spezifikation . . . 10
- 3.3 Umwelt- und Betriebsbedingungen . . . 12
- 3.4 Spezifikationen für die Stromversorgung . . . 12
- 3.5 Verkabelung . . . 13
- 3.6 Mechanische Schnittstelle . . . 15
- 3.7 Koordinatensysteme . . . 16

4 Installation **18**
- 4.1 Installation und Konfiguration . . . 18
- 4.2 Einschalten . . . 18
- 4.3 Netzwerkkonfiguration . . . 18
- 4.4 Aufspüren von *rc_visard*-Geräten . . . 19
- 4.5 Web GUI . . . 21

5 Der *rc_visard* auf einen Blick **23**
- 5.1 Stereovision . . . 23
- 5.2 Sensordynamik . . . 24
- 5.3 Kalibrierung zu einem Roboter . . . 25

6 Softwaremodule **27**
- 6.1 Stereokamera . . . 28
- 6.2 Stereo-Matching . . . 32
- 6.3 Sensordynamik . . . 38
- 6.4 Visuelle Odometrie . . . 42
- 6.5 Stereo-INS . . . 45
- 6.6 SLAM (optional) . . . 45
- 6.7 Kamerakalibrierung . . . 46
- 6.8 Hand-Auge-Kalibrierung . . . 52

7 Schnittstellen **65**
- 7.1 GigE Vision 2.0/GenICam-Schnittstelle . . . 65
- 7.2 REST-API-Schnittstelle . . . 71
- 7.3 Die rc_dynamics-Schnittstelle . . . 105

8 Wartung **109**

8.1 Reinigung der Kameralinsen . . . 109
8.2 Kamerakalibrierung . . . 109
8.3 Aktualisierung der Firmware . . . 109
8.4 Wiederherstellung der vorherigen Firmware-Version . . . 111
8.5 Neustart des *rc_visard* . . . 112
8.6 Aktualisierung der Softwarelizenz . . . 112
8.7 Download der Logdateien . . . 112

9 Zubehör **113**
9.1 Anschlussset . . . 113
9.2 Verkabelung . . . 113
9.3 Ersatzteile . . . 114

10 Fehlerbehebung **115**
10.1 LED-Farben . . . 115
10.2 Probleme mit der Hardware . . . 115
10.3 Probleme mit der Konnektivität . . . 116
10.4 Probleme mit den Kamerabildern . . . 117
10.5 Probleme mit Tiefen-/Disparitäts-, Fehler- oder Konfidenzbildern . . . 117
10.6 Probleme mit der Zustandsschätzung . . . 119
10.7 Probleme mit GigE Vision/GenICam . . . 119

11 Kontakt **120**
11.1 Support . . . 120
11.2 Downloads . . . 120
11.3 Adresse . . . 120

12 Anhang **121**
12.1 Formate für Posendaten . . . 121

HTTP Routing Table **123**

Index **124**

1 Einführung

Revisionen

Roboception kann dieses Produkt bei Bedarf jederzeit ohne Vorankündigung ändern, 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 angegebenen Revisionshinweise.

Revision 13. Juli 2017 Beta-Version

Copyright

Copyright © 2017 Roboception GmbH. Alle Rechte vorbehalten. 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 von Roboception 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.

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 *Achtung* gekennzeichnet werden. Zusätzliche Informationen sind als *Hinweis* gekennzeichnet.

Achtung: Die mit *Achtung* 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.

Hinweis: Hinweise werden in diesem Handbuch eingesetzt, um zusätzliche relevante Informationen zu vermitteln.

1.1 Überblick

Der 3D-Sensor *rc_visard* stellt Echtzeit-Kamerabilder und Disparitätsbilder bereit, die auch zur Berechnung von Tiefenbildern und 3D-Punktewolken verwendet werden können. Zudem erstellt er Konfidenz- und Fehlerbilder, mit denen sich die Qualität der Bilderfassung messen lässt. Der Sensor kann sich aufgrund von Bild- und Trägheitsdaten selbst lokalisieren. Mit dem integrierten SLAM-Modul lässt sich eine mobile Navigationslösung umsetzen. Der *rc_visard*, ein Gerät der Schutzklasse IP 54, umfasst eine intuitive Web- und eine standardisierte GenICam-Schnittstelle, wodurch er mit allen großen Bildverarbeitungsbibliotheken kompatibel ist. Der *rc_visard* wird in zwei 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 ist der *rc_visard* für jedermann der ideale 3D-Sensor.

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.

Hinweis: 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.

Hinweis: 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 des Produkts, die nicht ausdrücklich von Roboception genehmigt wurde, kann zum Verlust der Gewährleistungs- und Garantierechte führen.

Achtung: Der *rc_visard* arbeitet mit komplexer Hardware- und Software-Technologie, die ggf. nicht immer wie vorgesehen funktioniert. Der Käufer muss seine Anwendung so gestalten, dass eine Fehlfunktion des *rc_visard*-Sensors nicht zu Körperverletzungen, Sachschäden oder anderen Verlusten führt.

Achtung: 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 das Typenschild des *rc_visard* beschädigt, entfernt oder unkenntlich gemacht wurde.

Achtung: 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.

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

1.3 Schnittstellen, Zulassungen und Normen

1.3.1 Schnittstellen

Der *rc_visard* unterstützt folgende Standardinterfaces:

GEN<i>CAM

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

GiGE VISION

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

1.3.2 Zulassungen

Der *rc_visard* hat folgende Zulassungen erhalten:

CE

EG-Konformitätserklärung

Type tested · Production monitored · TÜV SÜD · C US

Zertifizierung durch den TÜV Süd

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

1.4 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 erhältlich und wird verwendet, wenn ‚.local' an einen Host-Namen angehängt wird.

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

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

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

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

INS Ein inertiales Navigationssystem (INS) ist ein 3D-Messsystem, das Positions- und Orientierungsdaten über Inertialsensoren (Beschleunigungs- und Drehratensensoren) berechnet. Wird das INS mit einem visuellen Messinstrument kombiniert, wird es auch VINS (Visual Inertial Navigation System) 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* 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.

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.

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 6.3.2) über die *rc_dynamics-Schnittstelle* (Abschnitt 7.3). Um diese Daten zu empfangen, kann einer Anwendung ein Datagram Socket (Endpunkt der Datenübertragung) zugeordnet werden, der aus einer Kombination aus *IP-Adresse* und einer Service-Port-Nummer besteht (z. B. `192.168.0.100:49500`). Dieser Endpunkt wird im vorliegenden Handbuch in der Regel als *Ziel* eines rc_dynamics-Datenstroms bezeichnet.

URI

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

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

XYZ+Quaternion Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *XYZ+Quaternion-Format* (Abschnitt 12.1.2).

XYZABC-Format Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *XYZABC-Format* (Abschnitt 12.1.1).

2 Sicherheit

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

Hinweis: 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

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

Achtung:

- Der *rc_visard* muss vor der Verwendung ordnungsgemäß montiert werden.
- Alle Kabel sind am *rc_visard* bzw. am Gestell anzuschließen.
- 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.

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

Der vom *rc_visard* verwendete Schnittstellenstandard GigE Vision® unterstützt weder Authentifizierung noch Verschlüsselung. Alle vom und an den Sensor 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.

Achtung: 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 Sensors gilt als nicht bestimmungsgemäße Verwendung. Roboception haftet nicht für Schäden, die aus unsachgemäßer oder nicht bestimmungsgemäßer Verwendung entstehen.

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

3 Hardware-Spezifikation

Hinweis: 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 4.5) oder über die Roboception-Homepage http://www.roboception.com/documentation aufrufen.

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

In Abschnitt *Zubehör* (Abschnitt 9) ist angegeben, welche Kabelanbieter Roboception empfiehlt.

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

Hinweis: 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.

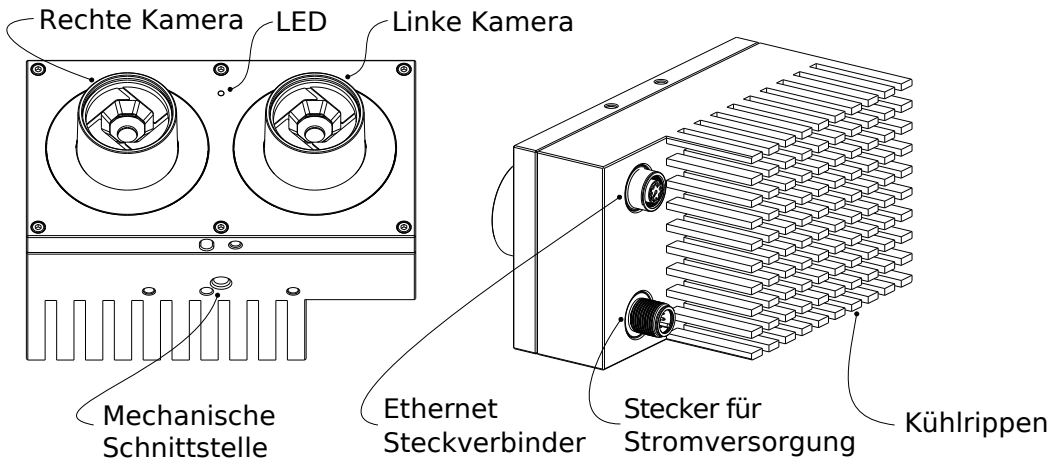


Abb. 3.1: Beschreibung der Bauteile

3.2 Technische Spezifikation

Tab. 3.1 enthält die gemeinsame technische Spezifikation für beide *rc_visard*-Varianten.

Tab. 3.1: Gemeinsame technische Spezifikation für beide *rc_visard*-Modelle

	***rc_visard* 65 / *rc_visard* 160**
Bildauflösung	1280 x 960 Pixel, farbig oder monochrom
Sichtfeld	Horizontal: 61°, Vertikal: 48°
Tiefenbild (Auflösung)	640 x 480 Pixel (High) bei 3 Hz 320 x 240 Pixel (Medium) bei 15 Hz 214 x 160 Pixel (Low) bei 25 Hz
Eigenbewegung	200 Hz, geringe Latenz
GPU/CPU	Nvidia Tegra K1
Stromversorgung	18–30 V
Kühlung	Passiv

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

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

	***rc_visard* 65**	***rc_visard* 160**
Basisabstand	65 mm	160 mm
Tiefenmessbereich	0,2 m bis unendlich	0,5 m bis unendlich
Tiefenauflösung	0,5 mm bei 0,2 m 15 mm bei 1,0 m	1,5 mm bei 0,5 m 6 mm bei 1,0 m 23 mm bei 2,0 m 50 mm bei 3,0 m
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

Der *rc_visard* kann für zusätzliche Funktionalitäten mit Onboard-Softwaremodulen, wie z. B. SLAM, ausgestattet werden. Diese Softwaremodule können bestellt werden und benötigen ein Lizenz-Update.

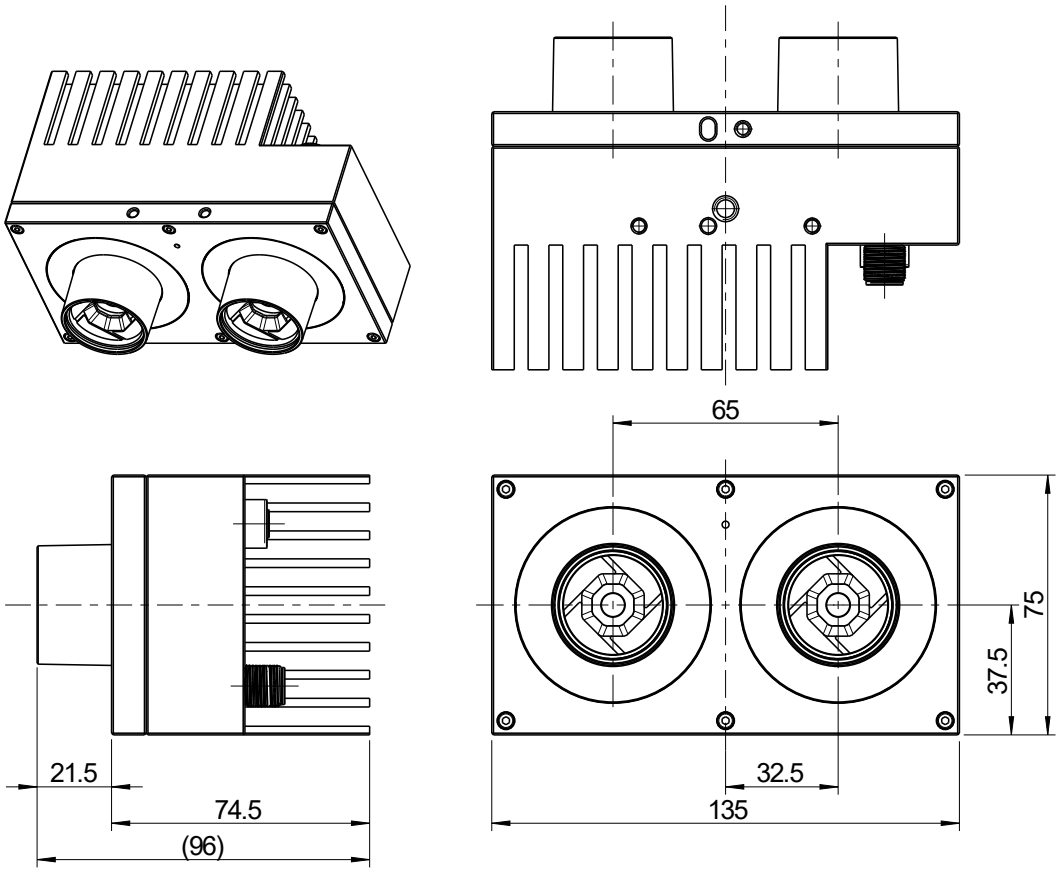


Abb. 3.2: Abmessungen des *rc_visard* 65

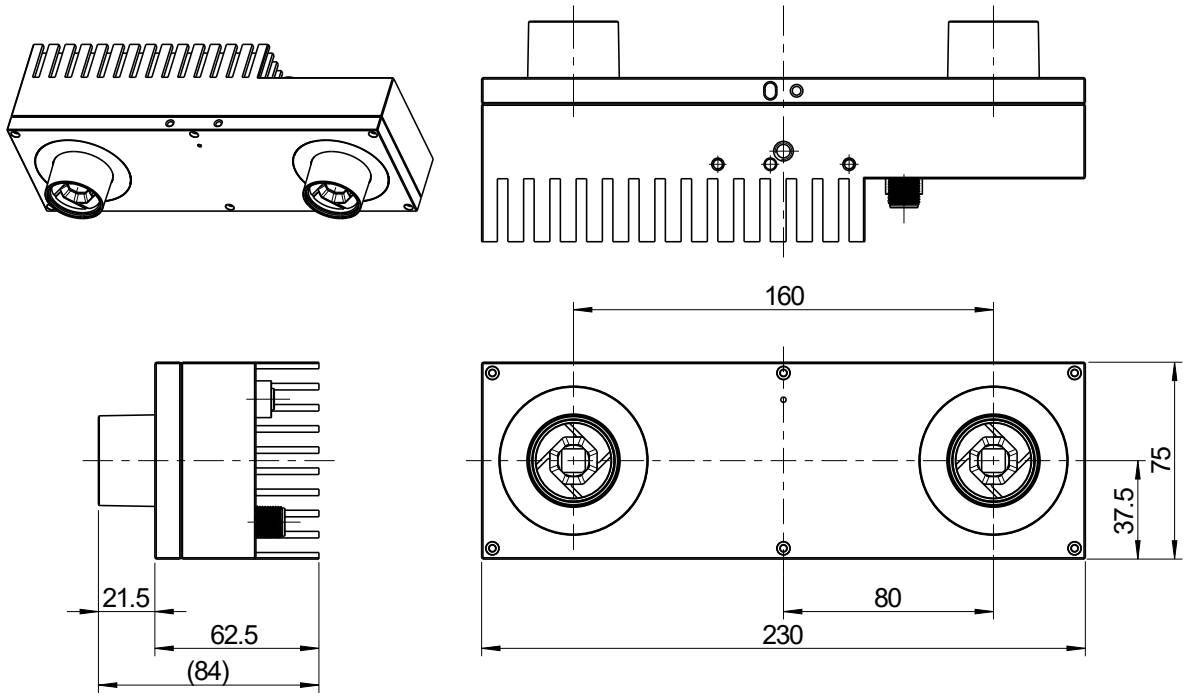


Abb. 3.3: Abmessungen des *rc_visard* 160

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: $2.76 \frac{g}{cm^3}$), 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. 3.3 angegebenen Umweltbedingungen für die Lagerung, den Transport und den Betrieb sind ausnahmslos einzuhalten.

Tab. 3.3: Umweltbedingungen

	***rc_visard* 65 / *rc_visard* 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	• Von korrosiven Flüssigkeiten oder Gasen fernhalten. • Von explosiven Flüssigkeiten oder Gasen fernhalten. • Von starken elektromagnetischen Störungen fernhalten.

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

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

Achtung: 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. Der Sensor entspricht den Anforderungen an industrielle Ausrüstung der Klasse A im Sinne der EN 55011. Daher muss jeder *rc_visard* 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. 3.4: Grenzwerte für die Stromversorgung

	Minimum	***Bemessungswert***	***Maximum***
Versorgungsspannung	18 V	24 V	30 V
Max. Leistungsaufnahme			25 W
Überstromschutz	Schutz der Stromversorgung mit einer 2-A-Sicherung		
Erfüllung der EMV-Anforderungen	Industrielles Gerät gemäß EN 55011 Klasse A		

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

Achtung: Jeder *rc_visard* muss von einem eigenen Netzabteil versorgt werden.

Achtung: 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.

Achtung: Die Richtlinien zum Kabelmanagement sind zwingend einzuhalten. Kabel sind immer mit einer Zugentlastung am Gestell 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.

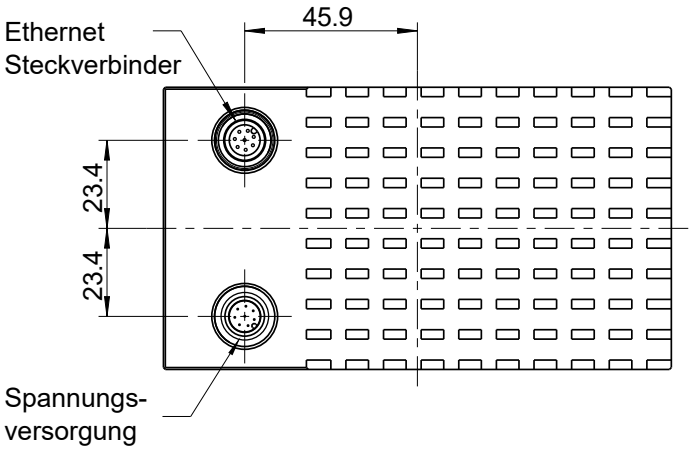


Abb. 3.4: Lage der elektrischen Anschlüsse des *rc_visard* 65 für die Ethernetverbindung (oben) und die Stromversorgung (unten)

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

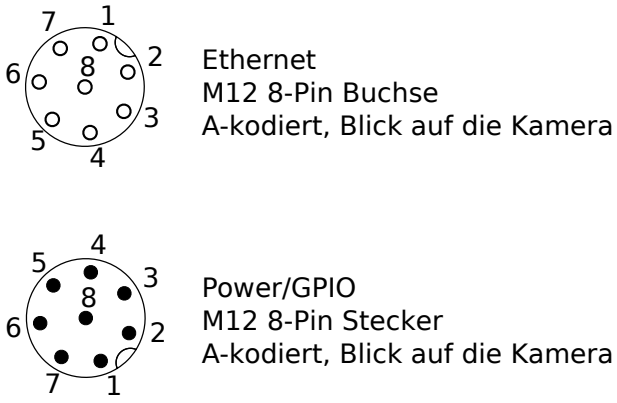


Abb. 3.5: Steckerbelegung für den Strom- und Ethernetanschluss

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

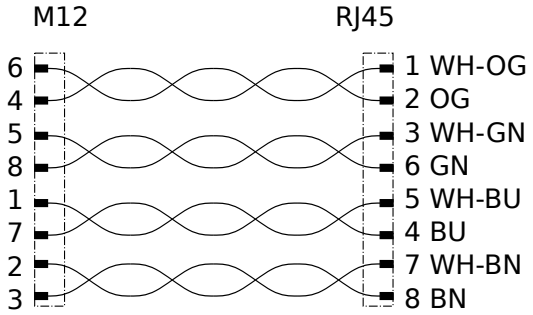


Abb. 3.6: Steckerbelegung für die M12/Ethernet-Verkabelung

Die Steckerbelegung für den Stromanschluss ist in Tab. 3.5 angegeben.

Tab. 3.5: 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 Eingang 1* und *2* sowie *GPIO Ausgang 2* sind für künftige Funktionen vorbehalten und haben derzeit keine für Benutzer zugänglichen Funktionen. Diese Pins sollten ungeerdet bleiben.

Achtung: 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.

GPIO Ausgang 1 bietet standardmäßig ein Signal zur Belichtungssynchronisierung und hat für die Dauer der Belichtung einen logischen HIGH-Pegel.

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

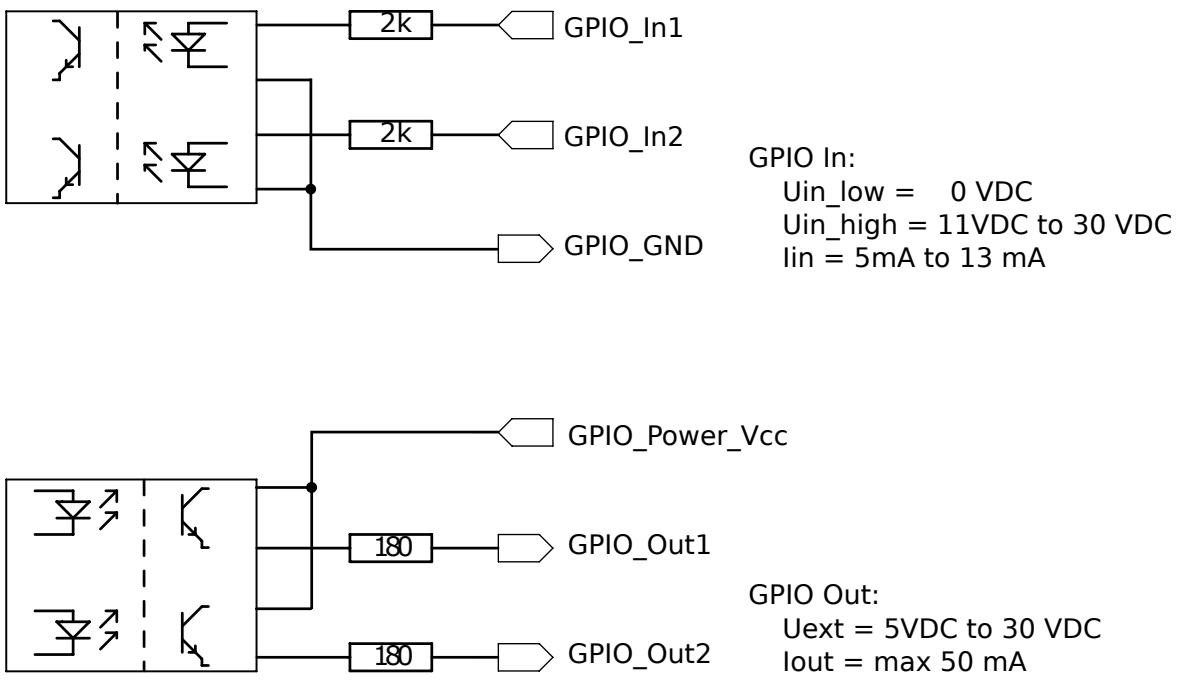


Abb. 3.7: GPIO-Schaltschema und zugehörige Spezifikationen: Keine Signale über 30 V anschließen!

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

3.6 Mechanische Schnittstelle

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

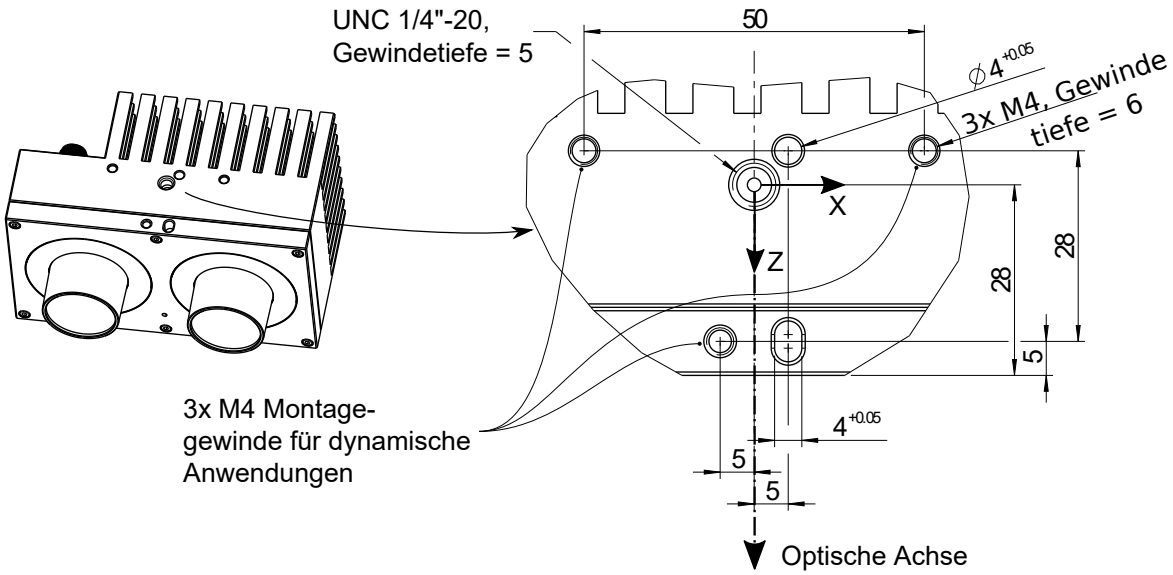


Abb. 3.8: Montagepunkt für den Anschluss des *rc_visard* an Roboter oder andere Vorrichtungen

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

Achtung: 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.

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.

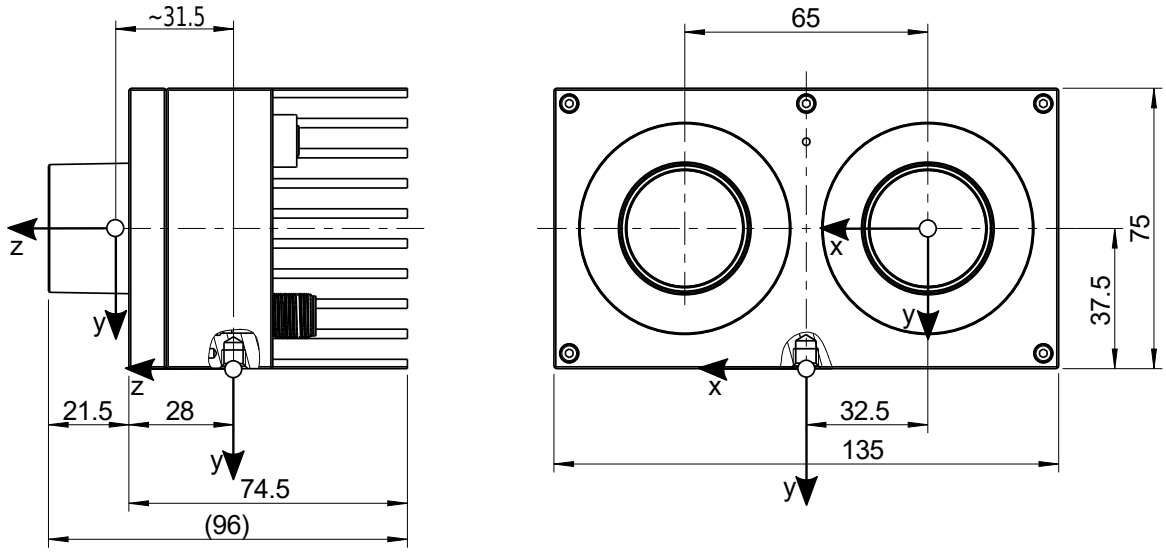


Abb. 3.9: Ungefähre Position des Sensor-/Kamerakoordinatensystems (in der linken Kameralinse) und des Montagepunkt-Koordinatensystems (am Stativgewinde) für den *rc_visard* 65

Die ungefähre Position des Sensor-/Kamerakoordinatensystems und des Montagepunkt-Koordinatensystems für den *rc_visard* 160 ist in Abb. 3.10 angegeben.

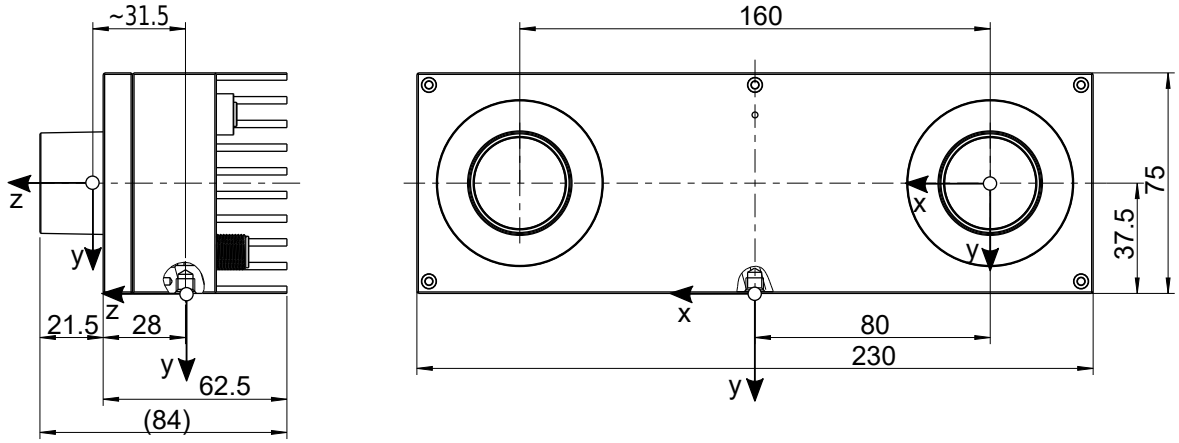


Abb. 3.10: Ungefähre Position des Sensor-/Kamerakoordinatensystems (in der linken Kameralinse) und des Montagepunkt-Koordinatensystems (am Stativgewinde) für den *rc_visard* 160

Hinweis: Der korrekte Versatz zwischen dem Sensor-/Kamerakoordinatensystem und einem Roboterkoordinatensystem kann über die *Hand-Auge Kalibrierung* (Abschnitt 6.8) bestimmt werden.

4 Installation

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

4.1 Installation und Konfiguration

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.

4.2 Einschalten

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

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

Ist kein Netzwerkkabel angeschlossen bzw. das Netzwerk nicht ordnungsgemäß konfiguriert, blinkt die LED alle fünf Sekunden rot. In diesem Fall muss die Netzwerkkonfiguration des Geräts überprüft werden. Für nähere Informationen zu den LED-Farbcodes siehe *LED-Farben* (Abschnitt 10.1).

4.3 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 automatisch oder manuell zugewiesen werden.

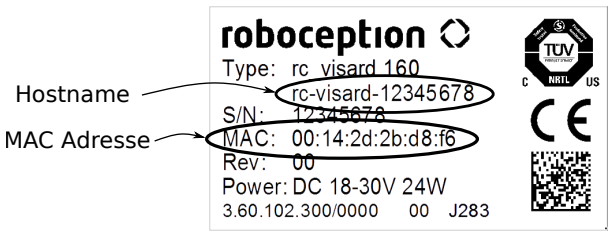


Abb. 4.1: Typenschild des *rc_visard*

4.3.1 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 Sensor aufgedruckte „Media Access Control"-Adresse, kurz *MAC-Adresse*, im DHCP-Server konfiguriert werden. Zudem ist der ebenfalls aufgedruckte Host-Name des Sensors im Domain Name Server *DNS* einzustellen. Der Host-Name im Format `rc-visard-<serial number>` ist ebenfalls auf den Sensor aufgedruckt. Sowohl die MAC-Adresse als auch der Host-Name sind zu Konfigurationszwecken an den Netzwerkadministrator zu übermitteln.

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

4.3.2 Manuelle Konfiguration

In einigen Fällen kann es nützlich sein, manuell eine feste IP-Adresse zu vergeben. Dies wird über die Standard *GigE Vision*® 2.0 Schnittstelle des Sensors vorgenommen und erfordert ein Konfigurationstool, welches auf dem Host-Computer zu installieren ist. Wir empfehlen hierfür IpConfigTool, das Teil des Baumer GAPI SDK ist. Für Windows und Linux kann das entsprechende *SDK* kostenlos von folgender Seite heruntergeladen werden: http://www.baumer.com.

Nach dem Start des Konfigurationstools wird 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. Lässt sich das Gerät nicht finden, kann es für die Konfiguration auch direkt an den Computer angeschlossen werden (siehe *Automatische Konfiguration (werkseitige Voreinstellung)*, Abschnitt 4.3.1).

Achtung: Die IP-Adresse muss eindeutig sein und innerhalb des Gültigkeitsbereichs des lokalen Netzwerks liegen. Zudem muss die Subnetz-Maske dem lokalen Netzwerk entsprechen, da anderenfalls möglicherweise nicht auf den *rc_visard* zugegriffen werden kann. Dieses Problem lässt sich vermeiden, indem die unter *Automatische Konfiguration (werkseitige Voreinstellung)* (Abschnitt 4.3.1) beschriebene automatische Konfiguration genutzt wird.

4.4 Aufspüren von *rc_visard*-Geräten

Alle Geräte, die eingeschaltet und mit dem lokalen Netzwerk oder direkt mit einem Computer verbunden sind (siehe *Netzwerkkonfiguration*, Abschnitt 4.3) 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 Daten, die ohne Installation direkt ausgeführt werden kann. Für Linux ist ein Installationspaket für Ubuntu 14.04 und 16.04 erhältlich. Nach dem Start wird 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 der Computer angehören. Ein Symbol im Discovery-Tool gibt an, ob ein Gerät richtig konfiguriert und damit auch über einen Webbrowser erreichbar ist.

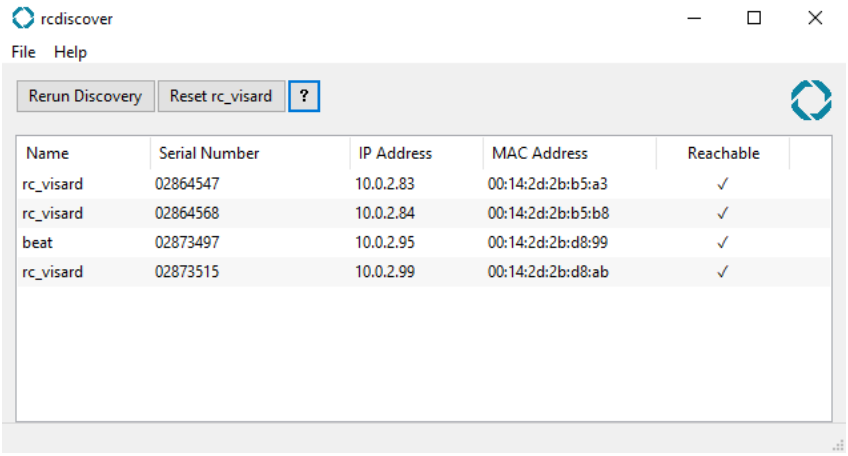


Abb. 4.2: `rcdiscover-gui`-Tool zum Aufspüren angeschlossener *rc_visard*-Geräte

Wurde das Gerät erfolgreich gefunden, öffnet sich nach einem Doppelklick auf den Geräteeintrag die *Web GUI* (Abschnitt 4.5) des Geräts im Standard-Browser des Betriebssystems. Wir empfehlen, Mozilla Firefox als Webbrowser zu verwenden.

4.4.1 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.

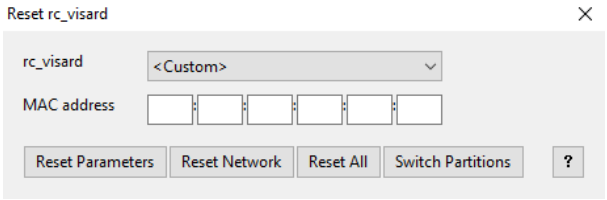


Abb. 4.3: Reset-Dialog des `rcdiscover-gui`-Tools

Wird ein *rc_visard* trotz falscher Konfiguration vom Discovery-Mechanismus erkannt, kann er aus der *rc_visard*-Drop-down-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 4.5) 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 8.4).

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.

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

4.5 Web GUI

Die Web GUI des *rc_visard* dient dazu, das Gerät zu testen, zu kalibrieren und seine On-Board-Verarbeitung zu konfigurieren. Auf die Web GUI kann über die IP-Adresse des Sensors 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 4.4) 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://rc-visard-<serial-number>`, wobei der Platzhalter `<serial-number>` gegen die auf dem Gerät aufgedruckte Seriennummer 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://rc-visard-<serial-number>.local`.

Die Überblicksseite der Web GUI enthält die wichtigsten Informationen zur On-Board-Verarbeitung.

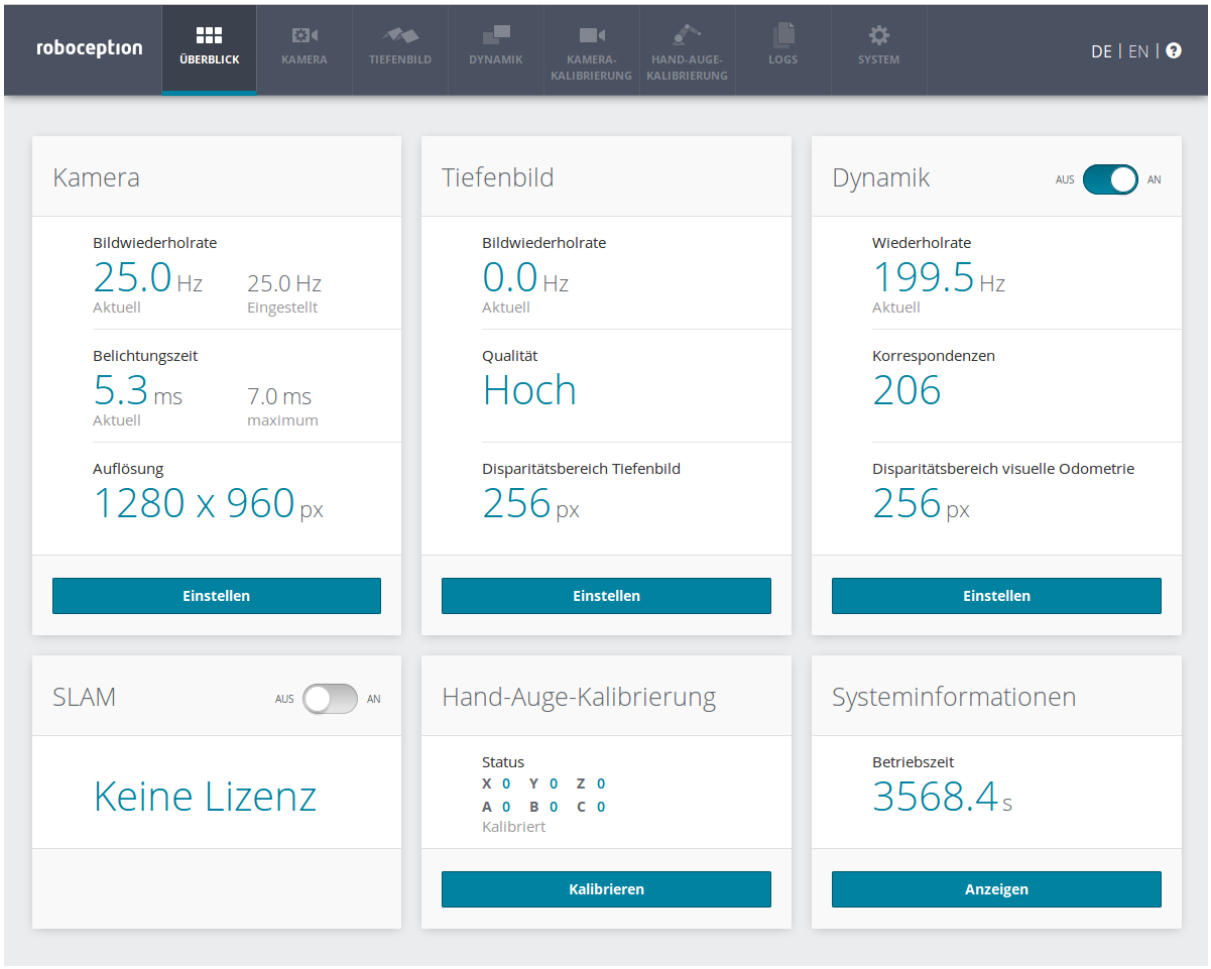


Abb. 4.4: Überblicksseite der Web GUI des *rc_visard*

Über Registerkarten im oberen Abschnitt der Seite kann auf einzelne *rc_visard*-Module zugegriffen werden:

- *Kamera* zeigt einen Live-Stream der rektifizierten linken und rechten 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 6.1.3).

- *Tiefenbild* bietet einen Live-Stream der rektifizierten Bilder der linken Kamera sowie Tiefen- und Konfidenzbilder. Auf der Seite lassen sich verschiedene Einstellungen zur Berechnung und Filterung von Tiefenbildern vornehmen. Für nähere Informationen siehe *Parameter* (Abschnitt 6.2.4).
- *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 6.4.1).
- *Kamerakalibrierung* dient dazu, die Kalibrierung der Kamera zu überprüfen. In seltenen Fällen kann es vorkommen, dass die Kamera nicht mehr ausreichend kalibriert ist. Dann kann die Kalibrierung über dieses Modul vorgenommen werden. Für nähere Informationen siehe *Kamerakalibrierung* (Abschnitt 6.7).
- *Hand-Auge-Kalibrierung* ermöglicht es, die statische Transformation zwischen dem *rc_visard* und einem im Robotersystem bekannten Koordinatensystem zu berechnen. Bei letzterem kann es sich um das Flansch-Koordinatensystem eines Roboterarms oder ein beliebiges statisches Koordinatensystem im Arbeitsraum des Roboters handeln, je nachdem, ob der *rc_visard* am Flansch oder statisch im Roboterumfeld installiert ist. Für nähere Informationen siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.8).
- Über *Logs* kann auf die Logdateien des *rc_visard* zugegriffen werden. Die Logdateien werden in der Regel überprüft, wenn Fehler vermutet werden.
- *System* erlaubt es, die Firmware oder Lizenzdatei zu aktualisieren, und bietet einige allgemeine Hinweise zum Gerät.

An Parametern vorgenommene Änderungen werden nicht dauerhaft übernommen und gehen verloren, wenn der *rc_visard* neu gestartet wird, es sei denn, die Schaltfläche *Speichern* wird betätigt, bevor die entsprechende Seite verlassen wird.

Weitere Informationen zu den einzelnen Parametern der Web GUI lassen sich über die jeweils daneben angezeigte Schaltfläche *Info* aufrufen.

5 Der *rc_visard* auf einen Blick

Der *rc_visard* ist eine selbstregistrierende 3D-Kamera. Sie 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.

5.1 Stereovision

Der *rc_visard* basiert auf *Stereovision* und nutzt hierzu das Verfahren *SGM* (*Semi-Global Matching*). 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.

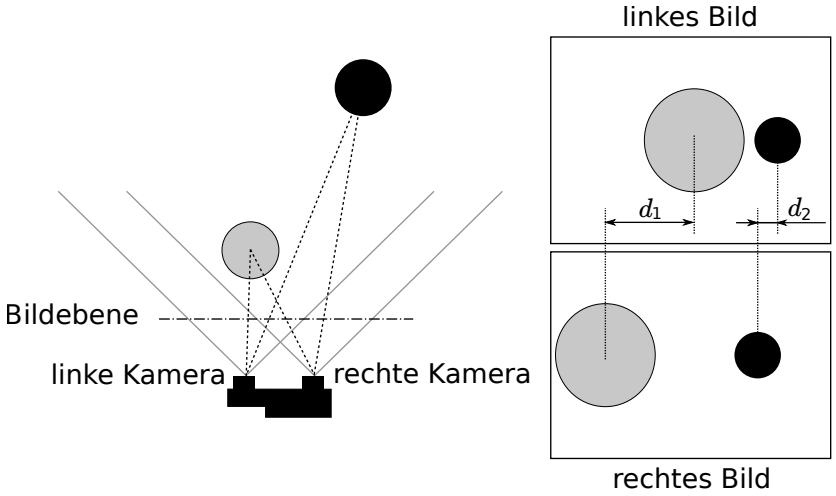


Abb. 5.1: Schematische Darstellung des Prinzips der Stereovision: Die Disparität d_2 des weiter entfernten (schwarzen) Objekts ist kleiner als die Disparität d_1 des nahe liegenden (grauen) Objekts.

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 kann der *rc_visard* sowohl im Innen- als auch im Außenbereich eingesetzt werden. Zudem können problemlos mehrere *rc_visard*-Sensoren störungsfrei 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 flachen weißen Wänden, keine Werte liefern. Das vom *rc_visard* genutzte Verfahren (*SGM*) bietet – selbst bei feineren Strukturen – den bestmöglichen Kompromiss aus Laufzeit und Genauigkeit.

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.

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

- *Stereokamera*: Dieses Modul dient dazu, synchronisierte Stereo-Bildpaare aufzunehmen und diese in Bilder umzuwandeln, die so weit es geht den Aufnahmen einer idealen Stereokamera entsprechen (Rektifizierung) (Abschnitt 6.1).
- *Stereo-Matching*: Dieses Modul errechnet mithilfe des Stereo-Matching-Verfahrens *SGM* die Disparitäten der rektifizierten Stereo-Bildpaare (Abschnitt 6.2).
- *Kamerakalibrierung*: Mit diesem Modul kann der Benutzer die Stereokamera des *rc_visard* neu kalibrieren (Abschnitt 6.7).

5.2 Sensordynamik

Neben 3D-Informationen zu einer Szene kann der *rc_visard* auch Echtzeit-Schätzungen zu seiner *Eigenbewegung* oder seinem *dynamischen Zustand* bereitstellen. Hierfür misst er seine aktuelle Pose, d. h. seine Position und Ausrichtung 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 (V*INS*) 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 errechnen. Eine statische Umgebung vorausgesetzt, entspricht die Bewegung, die eine Punktwolke in eine andere Punktwolke transformiert, der Kamerabewegung. Das Prinzip wird in Abb. 5.2 für einen vereinfachten 2D-Anwendungsfall dargestellt.

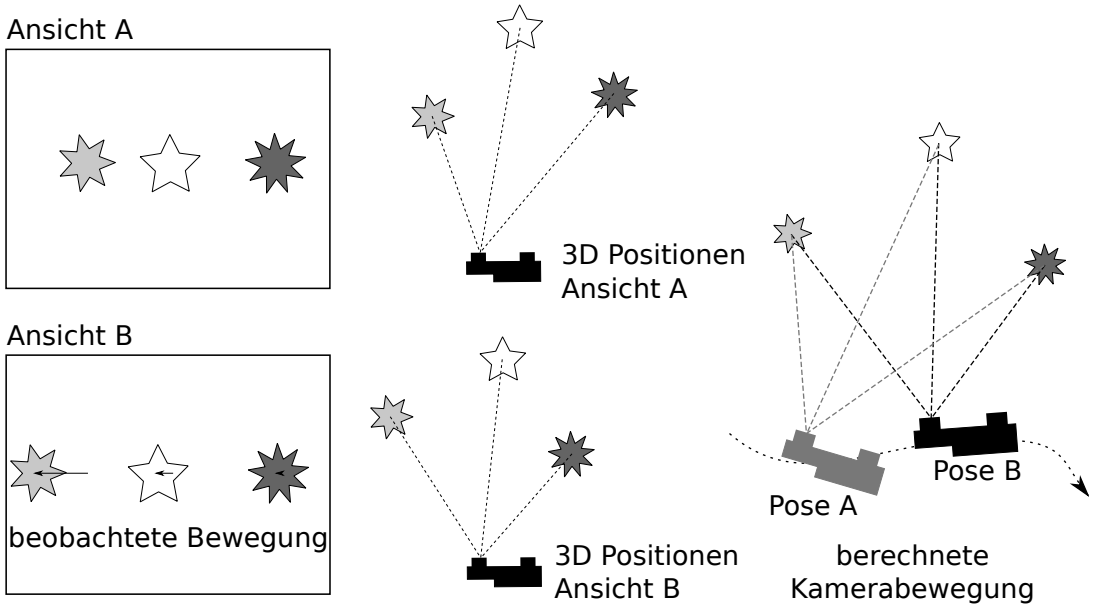


Abb. 5.2: Vereinfachte schematische Darstellung der stereobasierten visuellen Odometrie für 2D-Bewegungen: Die Kamerabewegung wird auf Grundlage der beobachteten Bewegung charakteristischer Bildpunkte berechnet.

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 inertiale Messeinheit (IMU), die Beschleunigungen und Winkelgeschwindigkeiten misst, die auftreten wenn sich der *rc_visard* bewegt. Das System misst zudem die Erdbeschleunigung, was die globale Ausrichtung in der Vertikale 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 6.3).
  - *Visuelle Odometrie*: Dieses Modul errechnet eine Bewegungsschätzung aufgrund von Kamerabildern (Abschnitt 6.4).
  - *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 6.5).
  - *SLAM (optional)*: Dieses Modul, das optional für den *rc_visard* erhältlich ist, erstellt eine interne Karte der Umgebung, auf deren Grundlage Posenfehler korrigiert werden (Abschnitt 6.6).

5.3 Kalibrierung zu einem Roboter

Der *rc_visard* wurde für den Einsatz im industriellen Umfeld konzipiert. Dies schließt auch Roboteranwendungen mit ein, für die der *rc_visard* entweder fest an einem Roboter oder auch statisch an einem Roboterarbeitsplatz montiert wird. Um die Ausgabewerte des *rc_visard* verwenden zu können, muss der Roboter wissen, wo im Roboter-Koordinatensystem der Sensor lokalisiert ist. Um die Lage des *rc_visard* im Roboter-Koordinatensystem

zu berechnen, bietet der Sensor die Möglichkeit einer *Hand-Auge Kalibrierung* (Abschnitt 6.8). Die Kalibrierroutine lässt sich entweder programmgesteuert über die *REST-API-Schnittstelle* (Abschnitt 7.2) oder manuell über die *Web GUI* (Abschnitt 4.5) ausführen.

6 Softwaremodule

Der *rc_visard* wird mit verschiedenen integrierten Softwaremodulen ausgeliefert, mit denen sich Kamerabilder, 3D-Informationen und Dynamik-Zustandsschätzungen bereitstellen und Kalibrierungen vornehmen lassen. Jedes Softwaremodul entspricht einem *node* in der *REST-API-Schnittstelle* (Abschnitt 7.2). Abb. 6.1 gibt einen Überblick über die Beziehungen zwischen den verschiedenen Softwaremodulen und den Daten, die sie über die verschiedenen *Schnittstellen* (Abschnitt 7) des *rc_visard* bereitstellen.

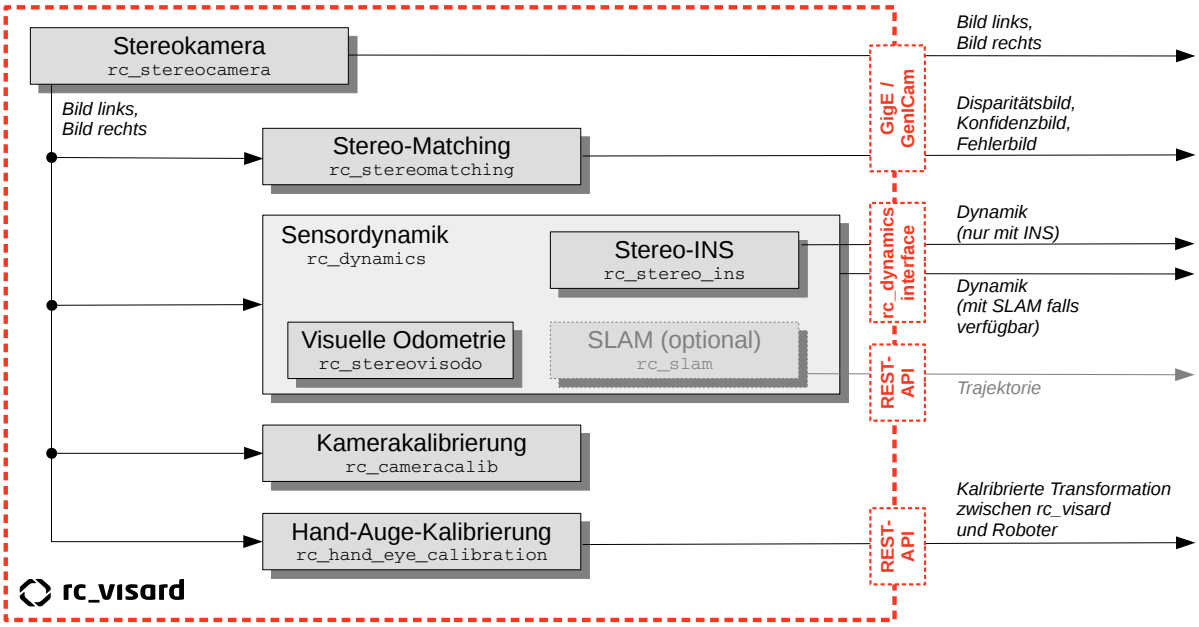


Abb. 6.1: Flussdiagramm der Softwaremodule mit den zugehörigen Modulnamen und den wichtigsten Ausgabedaten

Hinweis: Module, die mit *optional* gekennzeichnet sind, erweitern die Funktionalität des *rc_visard*. Kunden können ihre Lizenz erweitern, um zusätzliche Module zu erwerben.

Die integrierte Software des *rc_visard* umfasst folgende Module:

- ***Stereokamera*** **(`rc_stereocamera`, Abschnitt 6.1)** erfasst Stereo-Bildpaare und führt die planare Rektifizierung durch, wodurch die Stereokamera als Messinstrument verwendet werden kann. Bilder werden sowohl für die weitere interne Verarbeitung durch andere Module als auch als *GenICam-Bildstrom* für die externe Verwendung bereitgestellt.
- ***Stereo-Matching*** **(`rc_stereomatching`, Abschnitt 6.2)** nutzt das rektifizierte Stereo-Bildpaar, um 3D-Tiefeninformationen, z. B. für Disparitäts-, Fehler- und Konfidenzbilder, zu berechnen. Diese werden auch als GenICam-Bildströme bereitgestellt.
- ***Sensordynamik*** **(`rc_dynamics`, Abschnitt 6.3.)** erstellt Schätzungen zum dynamischen Zustand des *rc_visard*, z. B. zu seiner Pose, Geschwindigkeit und Beschleunigung. Diese Zustände werden als kon-

tinuierliche 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 6.4)** schätzt die Bewegung des *rc_visard* auf der Grundlage der Bewegungen charakteristischer Merkmale in den Bildern der linken Kamera.
  - ***Stereo-INS*** **(`rc_stereo_ins`, Abschnitt 6.5)** 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 (optional)*** **(`rc_slam`, Abschnitt 6.6)** übernimmt die simultane Lokalisierung und Kartenerstellung, um akkumulierte Posendaten zu korrigieren. Die Trajektorie des *rc_visard* lässt sich über die *REST-API-Schnittstelle* (Abschnitt 7.2) abfragen.

- ***Kamerakalibrierung*** **(`rc_cameracalib`, Abschnitt 6.7)** überprüft automatisch die Kalibrierung der Stereokamera des *rc_visard* und führt die Selbstkalibrierung durch, sofern sich diese verstellt hat. Mit diesem Modul kann der Benutzer zudem eine manuelle Neukalibrierung über die *Web GUI* (Abschnitt 4.5) vornehmen.
- ***Hand-Auge-Kalibrierung*** **(`rc_hand_eye_calibration`, Abschnitt 6.8)** ermöglicht dem Benutzer, den *rc_visard* entweder über die Web GUI oder die REST-API zu einem Roboter zu kalibrieren.

6.1 Stereokamera

Das Stereokamera-Modul umfasst Funktionen zur Erfassung von Stereo-Bildpaaren und zur planaren Rektifizierung, die nötig ist, um die Stereokamera als Messinstrument nutzen zu können.

6.1.1 Bilderfassung

Die Erfassung von Stereo-Bildpaaren ist der erste Schritt zur Stereovision. Da beide Kameras über Global Shutter verfügen und die Kamerachips per Hardware synchronisiert sind, werden alle Pixel beider Kameras immer zum exakt gleichen Zeitpunkt belichtet. GPIO-Ausgang 1 meldet dabei den jeweiligen Belichtungszeitpunkt (siehe *Verkabelung*, Abschnitt 3.5). Zudem wird die Zeit in der Mitte der Bildbelichtung den Bildern als Zeitstempel angeheftet. Dieser Zeitstempel ist für dynamische Anwendungen wichtig, bei denen sich der *rc_visard* oder die Szene bewegt.

Die Belichtungszeit lässt sich manuell auf einen festen Wert einstellen. Dies ist hilfreich in Umgebungen, in denen die Beleuchtung gesteuert wird, da die Lichtintensität so in allen Bildern gleich ist. Die Kamera ist standardmäßig auf automatische Belichtung eingestellt. In diesem Modus wählt der *rc_visard* die Belichtungszeit automatisch, bis zu einem benutzerdefinierten Höchstwert. Mit dem zulässigen Höchstwert soll eine mögliche Bewegungsunschärfe begrenzt werden: Hierzu kommt es, wenn Aufnahmen gemacht werden, während sich der *rc_visard* oder die Szene bewegt. Die maximale Belichtungszeit hängt also von der Anwendung ab. Ist die maximale Belichtungszeit erreicht, nutzt der Algorithmus eine Verstärkung (Gain), um die Bildhelligkeit zu erhöhen. Höhere Gain-Faktoren verstärken jedoch auch das Bildrauschen. Es gilt daher, die maximale Belichtungszeit bei schwacher Beleuchtung so zu wählen, dass ein guter Kompromiss zwischen Bewegungsunschärfe und Bildrauschen erzielt wird.

6.1.2 Planare Rektifizierung

Kameraparameter, wie die Brennweite, die Objektivverzeichnung und die Stellung der Kameras zueinander, müssen genau bekannt sein, soll die Stereokamera als Messinstrument eingesetzt werden. Die Parameter werden per Kalibrierung bestimmt (siehe *Kamerakalibrierung*, Abschnitt 6.7). Der *rc_visard* wird bereits ab Werk kalibriert und benötigt in der Regel keine Neukalibrierung. Die Kameraparameter beschreiben mit großer Präzision alle geometrischen Eigenschaften des Stereokamera-Systems, aber das daraus resultierende Modell ist komplex und kompliziert zu benutzen.

Rektifizierung bezeichnet den Prozess, Bilder auf Grundlage eines idealen Stereokamera-Modells zu reprojizieren. Dabei wird die Objektivverzeichnung korrigiert und die Bilder werden so ausgerichtet, dass ein Objektpunkt

in beiden Aufnahmen immer auf die gleiche Bildzeile projiziert wird. Die Sichtachsen der Kameras liegen genau parallel zueinander. Dies bedeutet, dass Objektpunkte in unendlicher Distanz in beiden Aufnahmen auf die gleiche Bildspalte projiziert werden. Je näher ein Objektpunkt liegt, desto größer ist der Unterschied zwischen den Bildspalten im rechten und linken Bild. Dieser Unterschied wird Disparität genannt.

Mathematisch lässt sich die Projektion des Objektpunkts $P = (P_x, P_y, P_z)$ auf den Bildpunkt $p_l = (p_{lx}, p_{ly}, 1)$ im linken rektifizierten Bild und auf den Bildpunkt $p_r = (p_{rx}, p_{ry}, 1)$ im rechten rektifizierten Bild wie folgt darstellen:

$$A = \begin{pmatrix} f & 0 & \frac{w}{2} \\ 0 & f & \frac{h}{2} \\ 0 & 0 & 1 \end{pmatrix}, \qquad T_s = \begin{pmatrix} t \\ 0 \\ 0 \end{pmatrix},$$

$$s_1 p_l = AP,$$
$$s_2 p_r = A(P - T_s).$$

Die Brennweite f ist der Abstand zwischen der gemeinsamen Bildebene und den optischen Zentren der linken und rechten Kamera. Sie wird in Pixeln gemessen. Als Basisabstand t wird der Abstand zwischen den optischen Zentren beider Kameras bezeichnet. Auch die Bildbreite w und Bildhöhe h werden in Pixeln gemessen. s_1 und s_2 sind Skalierungsfaktoren, die sicherzustellen, dass die dritten Koordinaten der Bildpunkte p_l und p_r 1 entsprechen.

Der *rc_visard* bietet über die GenICam-Schnittstelle zeitgestempelte rektifizierte Bilder der linken und rechten Kamera (siehe *Verfügbare Bild-Streams*, Abschnitt 7.1.2). Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 4.5) bereitgestellt.

Hinweis: 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.

6.1.3 Parameter

Das Stereokamera-Modul wird als `rc_stereocamera` bezeichnet und in der *Web GUI* (Abschnitt 4.5) auf der Registerkarte *Kamera* dargestellt. Der Benutzer kann die Kamera-Parameter entweder dort oder direkt über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) oder GigE Vision (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 7.1) ändern.

Hinweis: 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_stereocamera-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
exp_auto	bool	0	1	1	Umschalten zwischen automatischer und manueller Belichtung
exp_max	float64	6.6e-05	0.018	0.007	Maximale Belichtungszeit in Sekunden, wenn exp_auto auf TRUE gesetzt ist
exp_value	float64	6.6e-05	0.018	0.005	Maximale Belichtungszeit in Sekunden, wenn exp_auto auf FALSE gesetzt ist
fps	float64	1	25	25	Bildwiederholrate in Hertz
gain_value	float64	0	18	0	Manuelle Verstärkung in Dezibel, wenn exp_auto auf FALSE gesetzt ist
wb_auto	bool	0	1	1	Ein- und Ausschalten des manuellen Weißabgleichs (nur für Farbkameras)
wb_ratio_blue	float64	0.125	8	2.4	Blau zu Grün Verhältnis falls wb_auto auf False gesetzt ist (nur für Farbkameras)
wb_ratio_red	float64	0.125	8	1.2	Rot zu Grün Verhältnis falls wb_auto auf False gesetzt ist (nur für Farbkameras)

Dieses Modul meldet folgende Statuswerte:

Tab. 6.2: Statuswerte des rc_stereocamera-Moduls

Name	Beschreibung
baseline	Basisabstand t der Stereokamera in Metern
color	0 für Monochrom-Kameras, 1 für Farbkameras
exp	Tatsächliche Belichtungszeit in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Belichtungszeit (ms)* angezeigt.
focal	Brennweitenfaktor, normalisiert auf eine Bildbreite von 1
fps	Tatsächliche Bildwiederholrate der Kamerabilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Bildwiederholrate (Hz)* angezeigt.
gain	Tatsächlicher Verstärkungsfaktor in Dezibel. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Verstärkung (dB)* angezeigt.
height	Höhe des Kamerabildes in Pixeln
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
width	Breite des Kamerabildes in Pixeln

Beschreibung der Laufzeitparameter

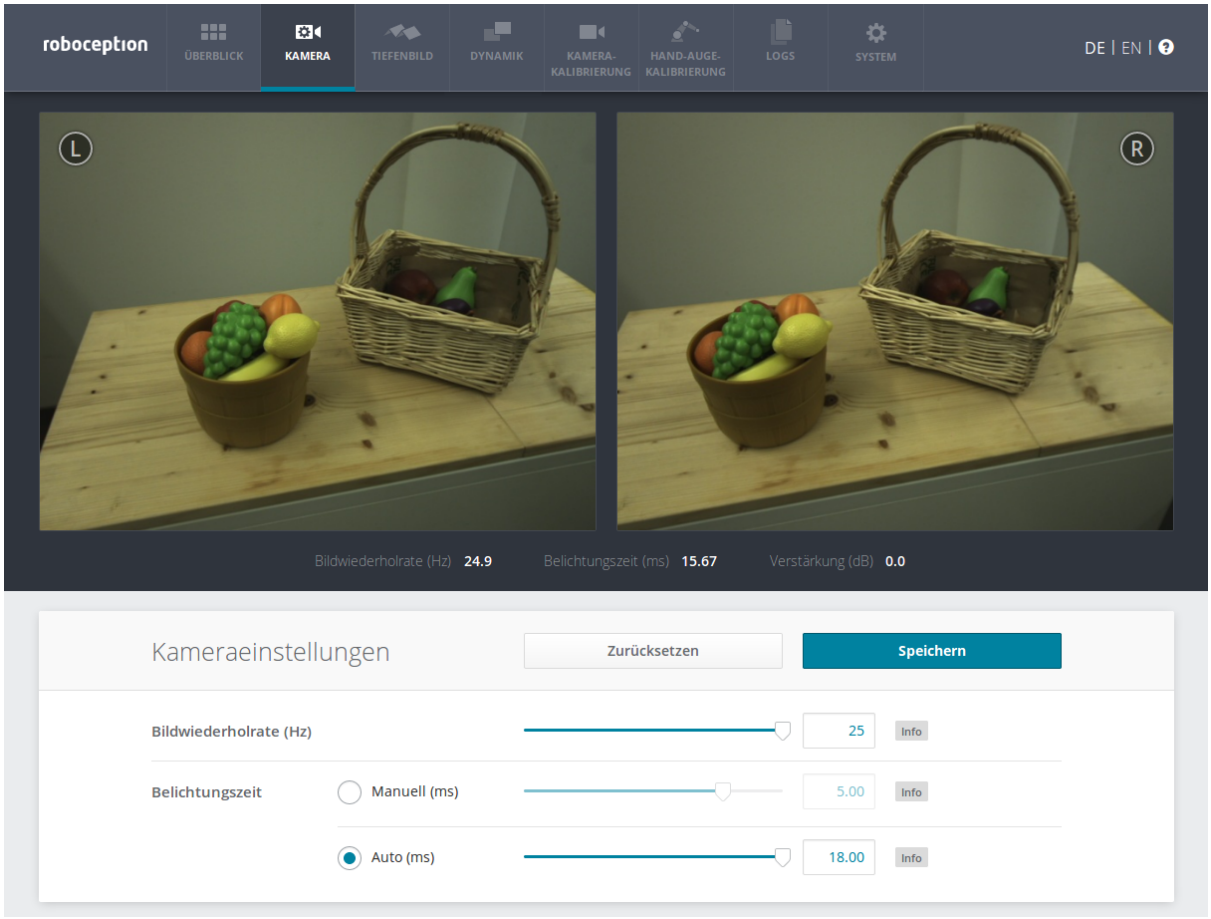


Abb. 6.2: Registerkarte *Kamera* in der Web GUI

`fps` (*Bildwiederholrate*) Dieser Wert bezeichnet die Bildwiederholrate der Kamera in Bildern pro Sekunde. Er gibt die obere Frequenz an, mit der Tiefenbilder berechnet werden können. Dies entspricht auch der Frequenz, mit der der *rc_visard* Bilder über GigE Vision bereitstellt. Wird diese Frequenz verringert, reduziert sich auch die zur Übertragung der Bilder benötigte Bandbreite des Netzwerkes.

`exp_auto` (*Belichtungszeit*) Dieser Wert lässt sich auf 1 – für den automatischen Belichtungsmodus – und auf 0 – für den manuellen Belichtungsmodus – stellen. Bei der manuellen Belichtung 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.

`exp_value` (*Manuell*) Dieser Wert gibt die Belichtungszeit im manuellen Modus in Sekunden an. Diese Belichtungszeit wird konstant gehalten, auch wenn die Bilder unterbelichtet sind. In der Web GUI lässt sich die Belichtungszeit der Einfachheit halber in Millisekunden angeben.

`gain_value` Dieser Wert gibt den Verstärkungsfaktor im manuellen Modus in Dezibel an. Höhere Verstärkungswerte reduzieren die Belichtungszeit, führen aber zu Rauschen. Dieser Wert kann nur in der REST-API oder GenICam, aber nicht in the Web GUI gesetzt werden.

`exp_max` (*Auto*) Dieser Wert gibt die maximale Belichtungszeit im automatischen Modus in Sekunden an. In der Web GUI lässt sich diese Belichtungszeit der Einfachheit halber in Millisekunden angeben. 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 Bildhelligkeit 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.

wb_auto Dieser Wert kann auf 1 gesetzt werden um den automatischen Weißabgleich anzuschalten. Bei 0 kann das Verhältnis der Farben manuell mit wb_ratio_red und wb_ratio_blue gesetzt werden. Der Weißabgleich ist bei monochromen Kameras ohne Funktion. Dieser Wert kann nur in der REST-API oder GenICam, aber nicht in der Web GUI gesetzt werden.

wb_ratio_red und wb_ratio_blue Mit diesen Werten kann das Verhältnis von Rot zu Grün bzw. Blau zu Grün für einen manuellen Weißabgleich gesetzt werden. Der Weißabgleich ist bei monochromen Kameras ohne Funktion. Dieser Wert kann nur in der REST-API oder GenICam, aber nicht in the Web GUI gesetzt werden.

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 7.1).

6.1.4 Services

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

save_parameters (***Speichern***) Bei Aufruf dieses Services werden die aktuellen Parametereinstellungen des Stereokamera-Moduls auf dem *rc_visard* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

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

> **Achtung:** Der Benutzer muss bedenken, dass bei Aufruf dieses Services die aktuellen Parametereinstellungen für das Stereokamera-Modul unwiderruflich verloren gehen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

6.2 Stereo-Matching

Das Stereo-Matching-Modul berechnet auf Grundlage des rektifizierten Stereobildpaars Disparitäts-, Fehler- und Konfidenzbilder.

6.2.1 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 Objektpunks im linken Bild. Der Begriff Disparität bezeichnet den Unterschied zwischen den Pixelspalten im rechten und linken Bild und gibt die Tiefe des Objektpunkts, d. h. dessen Abstand zum *rc_visard* 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 6.2.3). 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.

Wird der *rc_visard* in untexturierten Umgebungen eingesetzt, 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.

6.2.2 Berechnung von Tiefenbildern und Punktwolken

Die folgenden Gleichungen zeigen, wie sich die tatsächlichen 3D-Koordinaten P_x, P_y, P_z eines Objektpunkts im Sensor-Koordinatensystem (*Koordinatensysteme*, Abschnitt 3.7) aus den Pixelkoordinaten p_x, p_y des Disparitätsbilds und dem Disparitätswert d in Pixeln berechnen lassen:

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

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 7.1.1).

Hinweis: 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 Sie, dass für Gleichungen (6.1) davon ausgegangen wird, dass das Bildkoordinatensystem in der Bildmitte zentriert ist. In der folgenden Abbildung ist die Definition des Bildkoordinatensystems dargestellt.

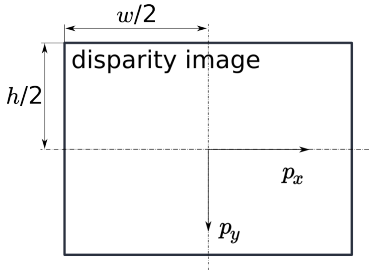


Abb. 6.3: Bildkoordinatensystem: Der Ursprung des Bildkoordinatensystems befindet sich in der Bildmitte – w ist die Bildbreite und h die Bildhöhe.

Die 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.

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

6.2.3 Konfidenz- und Fehlerbilder

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

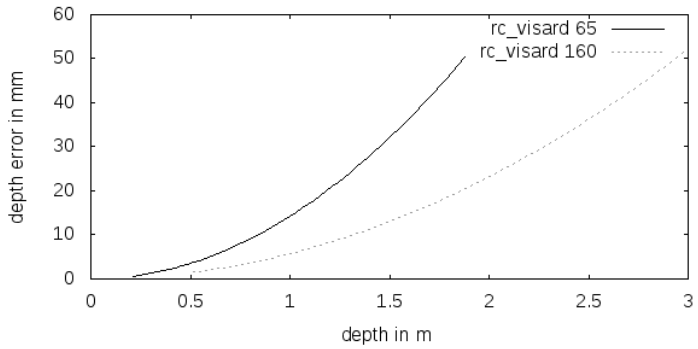
Der Disparitätsfehler d_{eps} (in Pixeln) lässt sich mithilfe der Brennweite f (in Pixeln), des Basisabstands t (in Metern) und des Disparitätswerts d (in Pixeln) desselben Pixels im Disparitätsbild in einen Tiefenfehler z_{eps} (in Metern) umrechnen:

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

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 Brennweite und Basisabstände beider *rc_visard*-Modelle sowie des typischen Disparitätsfehlers d_{eps} von 0,5 Pixeln lässt sich der Tiefenfehler wie folgt grafisch darstellen:



Der *rc_visard* stellt über die GenICam-Schnittstelle zeitgestempelte Disparitäts-, Fehler- und Konfidenzbilder zur Verfügung (siehe *Verfügbare Bild-Streams*, Abschnitt 7.1.2). Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 4.5) bereitgestellt.

6.2.4 Parameter

Das Stereo-Matching-Modul wird in der REST-API als `rc_stereomatching` bezeichnet und in der *Web GUI* (Abschnitt 4.5) auf der Registerkarte *Tiefenbild* dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) oder über GigE Vision (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 7.1) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter.

Tab. 6.3: Laufzeitparameter des `rc_stereomatching`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`disprange`	int32	32	512	256	Disparitätsbereich in Pixeln
`fill`	int32	0	4	3	Disparitätstoleranz (für das Füllen von Löchern) in Pixeln
`force_on`	bool	0	1	0	Verarbeitung auch ohne Empfänger erzwingen
`maxdepth`	float64	0.1	100	100	Maximale Tiefe in Metern
`maxdeptherr`	float64	0.01	100	100	Maximaler Tiefenfehler in Metern
`median`	int32	1	5	1	Größe des Medianfilter-Fensters in Pixeln
`minconf`	float64	0	1	0	Mindestkonfidenz
`mindepth`	float64	0.1	100	0.1	Minimaler Abstand in Metern
`quality`	string	-	-	H	S(taticHigh), H(igh), M(edium), oder L(ow)
`seg`	int32	0	4000	200	Mindestgröße der gültigen Disparitätssegmente in Pixeln

Dieses Modul meldet folgende Statuswerte:

Tab. 6.4: Statuswerte des `rc_stereomatching`-Moduls

Name	Beschreibung
`fps`	Tatsächliche Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Bildwiederholrate (Hz)* angezeigt.
`time_matching`	Zeit in Sekunden für die Durchführung des Stereo-Matchings mittels *SGM* auf der GPU
`time_postprocessing`	Zeit in Sekunden für die Nachbearbeitung des Matching-Ergebnisses auf der CPU

Da das SGM-Stereo-Matching-Verfahren und die Nachbearbeitung parallel ablaufen, entspricht die Gesamtverarbeitungszeit dieses Moduls dem Höchstwert aus `time_matching` und `time_postprocessing`. Diese Zeit wird unter der Bildvorschau in der Web GUI als *Verarbeitungszeit (s)* angezeigt.

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Registerkarte *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:

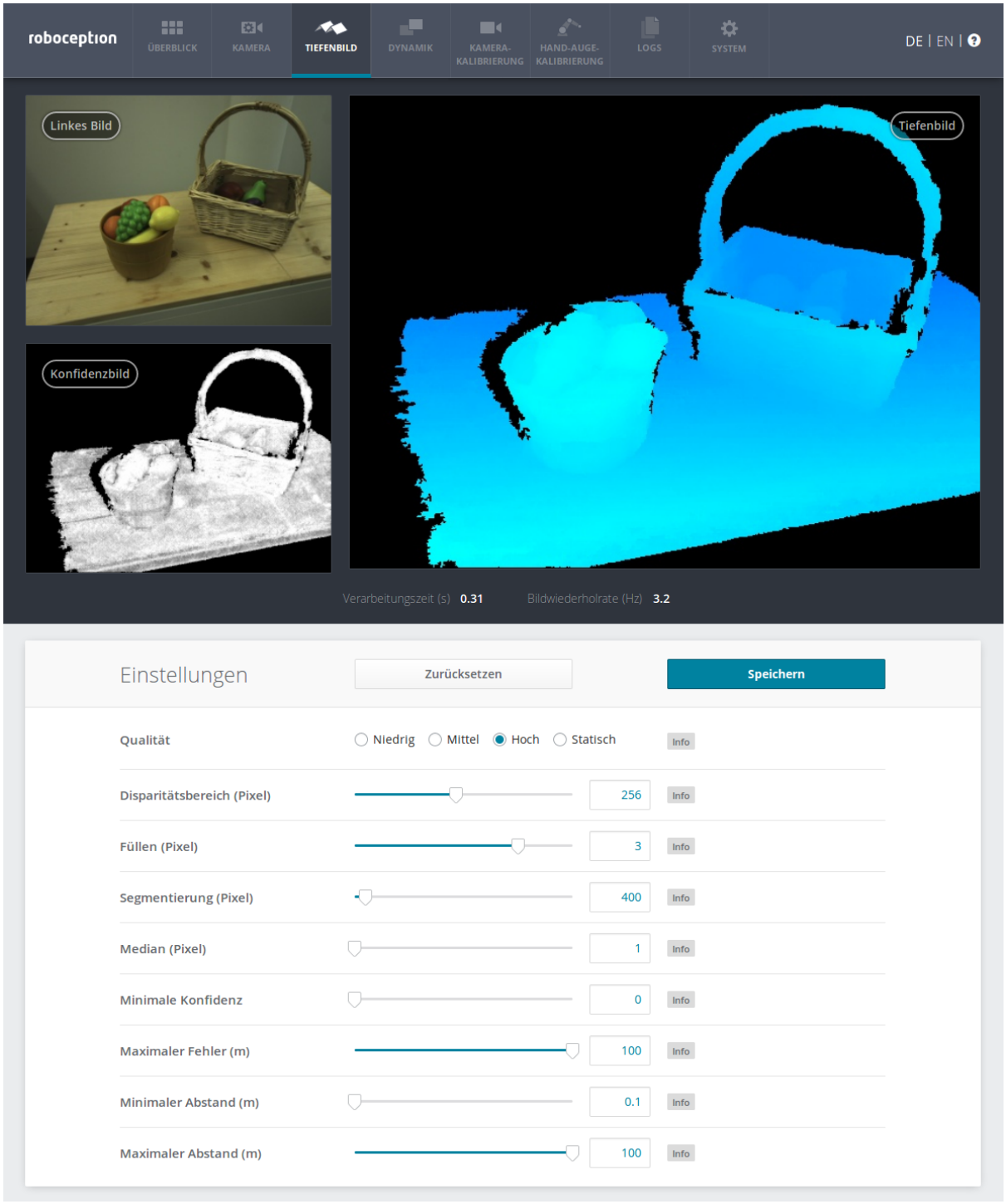


Abb. 6.4: Registerkarte *Tiefenbild* der Web GUI

`quality` (*Qualität*) Disparitätsbilder lassen sich in drei verschiedenen Auflösungen berechnen: Hoch (640 x 460 Pixel), Mittel (320 x 240 Pixel) und Niedrig (214 x 160 Pixel). Je niedriger die Auflösung, desto höher die Bildwiederholrate des Disparitätsbilds. Eine Bildwiederholrate von 25 Hz lässt sich nur bei niedriger Auflösung erreichen. Zudem kann die Qualitätsoption *Statisch* ausgewählt werden: Diese steht für eine hohe Auflösung bei einer maximalen Bildwiederholrate von 3 Hz und für die Akkumulation von Zwischenbildern. Durch diese Akkumulation wird das Bildrauschen reduziert. Die Option eignet sich jedoch nur, wenn sich die Szene nicht verändert. Es ist zu beachten, dass die Bildwiederholrate der Disparitäts-, Konfidenz- und Fehlerbilder immer höchstens der Bildwiederholrate der Kamera entspricht.

`disprange` (*Disparitätsbereich*) Der Disparitätsbereich geht immer von 0 bis zum maximalen Disparitätswert,

den ein Pixel in einem Disparitätsbild annehmen kann. Wird der Disparitätsbereich erhöht, wird der messbare Mindestabstand kleiner, da größere Disparitätswerte mit geringeren Messabständen einhergehen. Der Disparitätsbereich ist in Pixeln angegeben und kann auf einen Wert zwischen 32 und 512 Pixeln eingestellt werden. Da mit einem größeren Disparitätsbereich auch der Bereich für die Suche nach dem passenden Pixel im rechten rektifizierten Bild wächst, führt ein größerer Disparitätsbereich zu einer längeren Verarbeitungszeit und einer geringeren Bildwiederholrate. Der Wert des Disparitätsbereichs bezieht sich auf ein hochauflösendes Disparitätsbild (640 x 460 Pixel) und muss nicht skaliert werden, wenn eine niedrigere Auflösung gewählt wird. So gibt der Disparitätsbereich für jede Auflösungsoption den gleichen Mindestabstand an.

`fill` (*Füllen*) Diese Option wird verwendet, um Löcher im Disparitätsbild durch Interpolation einer Ebene zu füllen. Dabei werden lediglich Löcher, die kleiner als die Segmentierungsgröße (siehe unten) sind, für die Interpolation ausgewählt. Der Füllwert gibt die maximale Disparitätsabweichung an, die ein Rand-Pixel von der Interpolationsebene haben darf. Nur wenn alle Rand-Pixel um weniger als den Füllwert von der Ebene abweichen, wird ein Loch gefüllt. Größere Füllwerte verringern die Anzahl an Löchern, aber die interpolierten Werte können größere Fehler aufweisen. Die Konfidenz für die interpolierten Pixel wird auf einen geringen Wert von 0,5 eingestellt. Deren Fehler ist auf die mittlere Abweichung der Loch-Randpixel von der Interpolationsebene eingestellt. Das Auffüllen lässt sich deaktivieren, wenn der Wert auf 0 gesetzt wird.

`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. Dies eignet sich, um Disparitätsfehler zu entfernen. Bei größeren Werten kann es jedoch vorkommen, dass real vorhandene Objekte entfernt werden.

`median` (*Median*) Dieser Wert bestimmt die Seitenlänge der Filterregion (in Pixeln) für den Medianfilter, der das Disparitätsbild glättet. Größere Werte führen zu einer Überglättung und beanspruchen mehr Bearbeitungszeit. Der Medianfilter lässt sich effektiv abschalten, wenn das Filterfenster auf die Größe 1 gesetzt wird.

`minconf` (*Minimale Konfidenz*) Die Mindestkonfidenz lässt sich einstellen, um potenziell falsche Disparitätsmessungen herauszufiltern. Dabei werden alle Pixel, deren Konfidenz unter dem gewählten Wert liegt, im Disparitätsbild auf „ungültig“ gesetzt.

`maxdeptherr` (*Maximaler Fehler*) Der maximale Tiefenfehler 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 vom Sensor (siehe *Konfidenz- und Fehlerbilder*, Abschnitt 6.2.3).

`mindepth` (*Minimaler Abstand*) Der Mindestabstand bezeichnet den geringsten Abstand vom Sensor, ab dem Messungen möglich sind. Größere Werte verringern implizit den Disparitätsbereich, wodurch sich auch die Rechenzeit verkürzt. Der Mindestabstand wird in Metern angegeben.

`maxdepth` (*Maximaler Abstand*) Der Höchstabstand ist der größte Abstand vom Sensor, bis zu dem Messungen möglich sind. Pixel mit größeren Distanzwerten werden auf „ungültig“ gesetzt. Wird dieser Wert auf das Maximum gesetzt, so sind Werte bis zur Unendlichkeit möglich. Der Höchstabstand wird in Metern angegeben.

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 7.1).

6.2.5 Services

Das Stereo-Matching-Modul bietet folgende Services, um Parametereinstellungen zu speichern bzw. wiederherzustellen.

`save_parameters` (*Speichern*) Beim Aufruf dieses Services werden die aktuellen Parametereinstellungen des Stereo-Matching-Moduls auf dem *rc_visard* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

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

> **Achtung:** Der Benutzer muss bedenken, dass beim Aufruf dieses Services die aktuellen Parametereinstellungen für das Stereo-Matching-Modul unwiderruflich verloren gehen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

6.3 Sensordynamik

Das Dynamik-Modul 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 auf Grundlage der Bewegung von charakteristischen Bildpunkten im linken Kamerabild (Abschnitt 6.4).
- ***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 6.5).
- ***SLAM (optional)*** **(`rc_slam`)** übernimmt die simultane Lokalisierung und Kartenerstellung (*SLAM*), um akkumulierte Posendaten zu korrigieren (Abschnitt 6.6).

6.3.1 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.

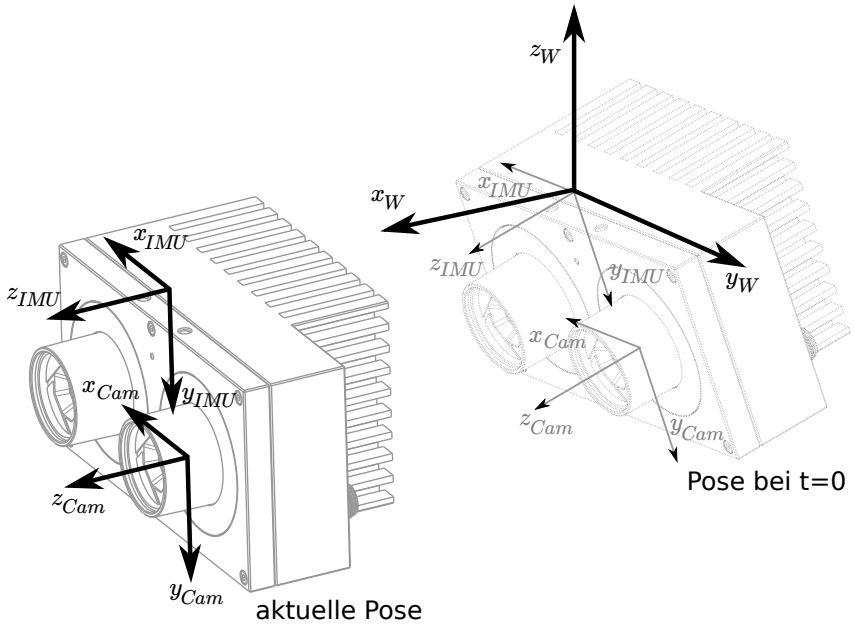


Abb. 6.5: Koordinatensysteme für die Zustandsschätzung: Das IMU-Koordinatensystem liegt im Gehäuse des *rc_visard*, das Kamera-Koordinatensystem in der Austrittspupille der linken Kamera (siehe auch *Koordinatensysteme*, Abschnitt 3.7)

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).

6.3.2 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 7.3):

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

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

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 `rc_slam` und `rc_stereo_ins` kombiniert werden, wenn eine Lizenz für das *SLAM*-Modul vorliegt. Liegt keine Lizenz vor, sind beide Datenströme gleichwertig. Die Posen werden in Weltkoordinaten angegeben und beziehen sich auf den Ursprung des Kamerakoordinatensystems (siehe *Koordinatensysteme für die Zustandsschätzung*, Abschnitt 6.3.1). 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 `rc_slam` und `rc_stereo_ins` kombiniert werden, wenn eine Lizenz für das SLAM-Modul vorliegt. Liegt keine Lizenz vor, 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 6.3.1). 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 `rc_slam` und `rc_stereo_ins` kombiniert werden, wenn eine Lizenz für das SLAM-Modul vorliegt. Liegt keine Lizenz vor, 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 $\omega = (\omega_x, \omega_y, \omega_z)^T$ 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 6.3.1). 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 (optional)*, Abschnitt 6.6) 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. Ist SLAM nicht aktiv, 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.

6.3.3 Services

Das Sensordynamik-Modul bietet folgende Services, um die Zustandsschätzung zu starten, stoppen und neu zu starten. Es lässt sich über die Rubrik *Dynamik* auf der Registerkarte *Überblick* der *Web GUI* starten und stoppen.

`start` startet das Modul.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:

```
{
  "accepted": "bool",
  "enteredState": "uint8"
}
```

`stop` stoppt das Modul.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:

```
{
  "accepted": "bool",
  "enteredState": "uint8"
}
```

`restart` startet das Modul neu.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:

```
{
  "accepted": "bool",
  "enteredState": "uint8"
}
```

`getstate` ruft den aktuellen Zustand des Moduls ab.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:

```
{
  "accepted": "bool",
  "enteredState": "uint8"
}
```

6.4 Visuelle Odometrie

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

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

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 Bildwiederholrate zu erhöhen und die Latenz zu verringern. Das Ergebnis der Sensordatenfusion wird in Form verschiedener Datenströme bereitgestellt (siehe *Stereo-INS*, Abschnitt 6.5).

6.4.1 Parameter

Das Odometrie-Modul heißt `rc_stereovisodo` und wird in der *Web GUI* (Abschnitt 4.5) auf der Registerkarte *Dynamik* dargestellt. Der Benutzer kann die Parameter der visuellen Odometrie entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter.

Tab. 6.5: Laufzeitparameter des `rc_stereovisodo`-Moduls

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

Dieses Modul meldet folgende Statuswerte:

Tab. 6.6: Statuswerte des `rc_stereovisodo`-Moduls

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

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.

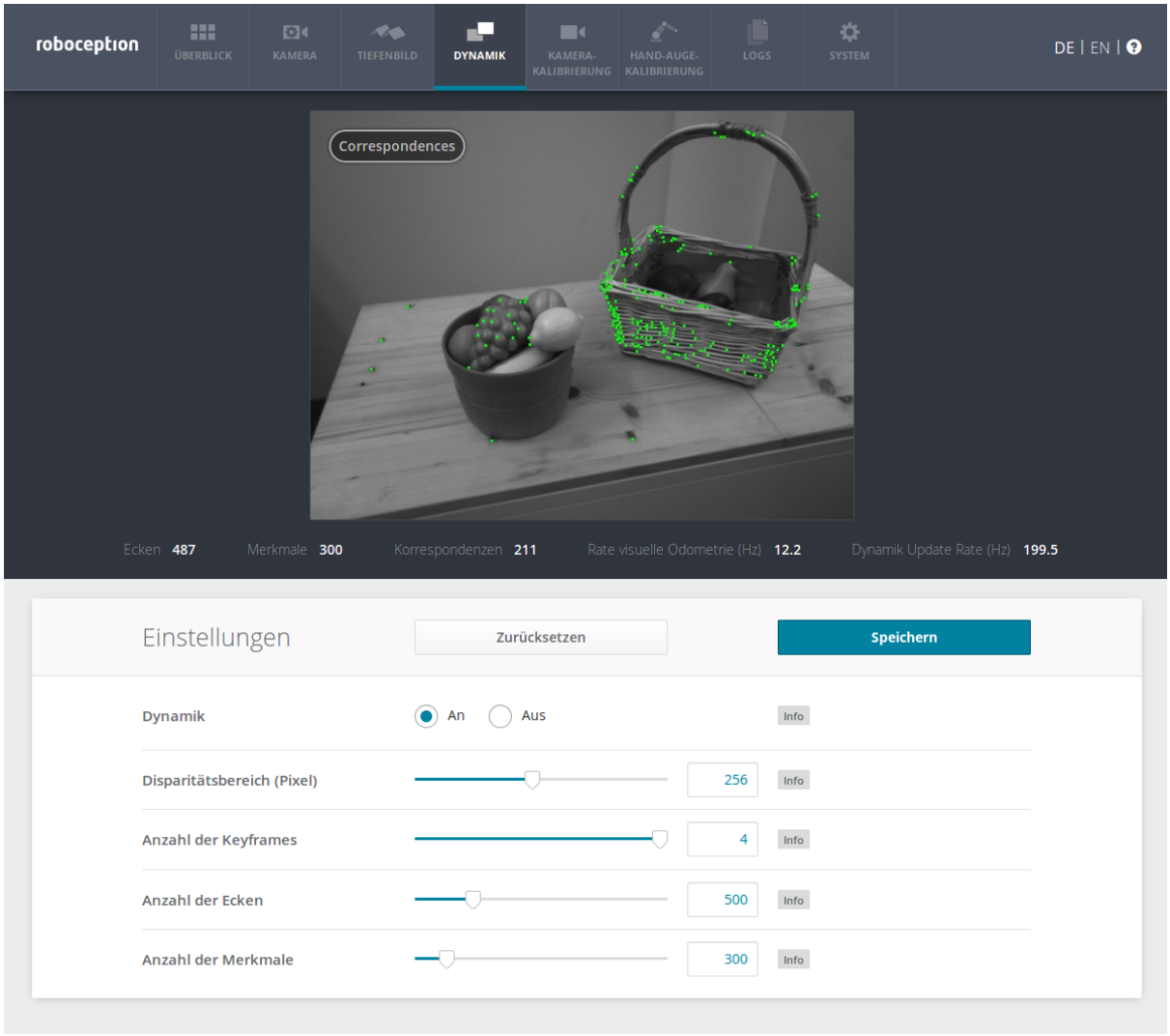


Abb. 6.6: *Dynamik*-Registerkarte der Web GUI

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 Registerkarte *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:

`start` (*Dynamik*) Dieser Parameter startet die Module für die Schätzungen zur Sensordynamik (siehe *Services*, Abschnitt 6.3.3).

`disprange` (*Disparitätsbereich*) Der Disparitätsbereich gibt den maximalen Disparitätswert an, den jedes Bildmerkmal im hochaufgelösten Disparitätsbild (640 x 460 Pixel) 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.

`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.

`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 Odometriefrequenzführen.

`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 ausgwählt 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.

Hinweis: Die Erhöhung der Anzahl an Keyframes, Ecken oder Merkmalen wird zwar die Robustheit erhöhen, erfordert aber 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.

6.4.2 Services

Das Visuelle-Odometrie-Modul bietet folgende Services, um Parametereinstellungen zu speichern bzw. wiederherzustellen. Die Namen der zugehörigen Web GUI-Schaltflächen sind in Klammern hinzugefügt.

`save_parameters` (*Speichern*) Beim Aufruf dieses Services werden die aktuellen Parametereinstellungen zur visuellen Odometrie auf dem *rc_visard* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

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

Achtung: Der Benutzer muss bedenken, dass beim Aufruf dieses Services die aktuellen Parametereinstellungen für die visuelle Odometrie unwiderruflich verloren gehen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

Das Modul bietet keine eigenen Funktionen zum Starten bzw. Stoppen, da es über das *Dynamik-Modul* (Abschnitt 6.3) gestartet bzw. gestoppt wird.

6.5 Stereo-INS

Die stereobildgestützte inertiale Navigation (*INS*) 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.

6.5.1 Parameter

Das Stereo-INS-Modul heißt `rc_stereo_ins`.

Dieses Modul besitzt keine Laufzeitparameter.

Dieses Modul meldet folgende Statuswerte:

Tab. 6.7: Statuswerte des `rc_stereo_ins`-Moduls

Name	Beschreibung
`freq`	Frequenz des Stereo-INS-Prozesses in Hertz. Dieser Wert wird auf der Registerkarte *Überblick* der Web GUI in der Rubrik *Dynamik* als *Wiederholrate* angezeigt.
`state`	Interner Zustand

6.6 SLAM (optional)

Das SLAM-Modul ist Teil des Dynamik-Moduls. Wenn sich der *rc_visard* bewegt, summieren sich mit der Zeit die bei der Posenschätzung auftretenden Fehler. Das SLAM-Modul kann diese Fehler korrigieren, indem es bereits besuchte Orte wiedererkennt.

Das Akronym SLAM steht für simultane Lokalisierung und Kartenerstellung (simultaneous localization and mapping). Das SLAM-Modul erstellt eine Karte aus den für die visuelle Odometrie genutzten Bildmerkmalen. Die Karte wird später verwendet, um kumulierte Posenfehler zu korrigieren. Am ehesten lässt sich dies in Anwendungen beobachten, bei denen der Roboter, nachdem er eine lange Strecke zurückgelegt hat, an einen besuchten Ort zurückkehrt (dies wird auch „Schleifenschluss“ oder „Loop Closure“ genannt). In diesem Fall kann der Roboter Bildmerkmale, die bereits in seiner Karte abgespeichert sind, wiedererkennen und seine Posenschätzung auf dieser Grundlage korrigieren. Kommt es zu einem Schleifenschluss, wird nicht nur die aktuelle, sondern auch die bisherige Posenschätzung (Trajektorie des *rc_visard*) korrigiert. Die kontinuierliche Korrektur der Trajektorie sorgt dafür, dass die Karte immer mehr an Genauigkeit gewinnt. Die Genauigkeit der Trajektorie ist auch wichtig, wenn diese zum Aufbau eines integrierten Weltmodells verwendet wird, indem beispielsweise die ermittelten 3D-Punktwolken in ein gemeinsames Koordinatensystem projiziert werden (siehe *Berechnung von Tiefenbildern und Punktwolken*, Abschnitt 6.2.2). Zu diesem Zweck kann die gesamte Trajektorie des *rc_visard* beim SLAM-Modul abgefragt werden.

Das SLAM-Modul, das optional für den *rc_visard* erhältlich sein wird, läuft intern auf dem Sensor. Sobald eine SLAM-Lizenz auf dem *rc_visard* hinterlegt wird, erscheint das SLAM-Modul in der Registerkarte *Überblick* der *Web GUI* als *Verfügbar*. Auch die Aktivierung und Deaktivierung des Moduls erfolgt über diese Seite.

6.7 Kamerakalibrierung

Um die Stereokamera 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 *Planare Rektifizierung*, Abschnitt 6.1.2). Der *rc_visard* ist bereits ab Werk kalibriert. Nichtsdestotrotz kann es vorkommen, dass die Kalibrierung überprüft und neu durchgeführt werden muss, wenn der *rc_visard* einer starken mechanischen Beanspruchung ausgesetzt war. Mit diesem Modul lassen sich die Kalibrierungsüberprüfung und Neukalibrierung vornehmen.

6.7.1 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 8.7):

> *„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.

6.7.2 Kalibriervorgang

Die manuelle Kalibrierung kann über die Registerkarte *Kamera-Kalibrierung* der Web GUI vorgenommen werden. Diese Registerkarte bietet einen Assistenten, der den Benutzer durch den Kalibriervorgang führt.

Hinweis: Da der *rc_visard* bereits ab Werk kalibriert ist, ist in aller Regel keine Neukalibrierung der Kameras nötig. Eine Neukalibrierung ist nur erforderlich, wenn das Gerät einer starken mechanischen Belastung ausgesetzt war, wenn es beispielsweise fallen gelassen wurde.

Schritt 1: Einstellung der Kalibrierparameter

Für die Kalibrierung bedarf es eines präzisen Kalibriermusters. Das Muster lässt sich ausdrucken und muss auf eine flache Oberfläche geklebt werden. Eine Vorlage mit zugehörigen Anweisungen kann `hier`: http://www.roboception.com/download heruntergeladen werden. Die Anweisungen sind genau zu befolgen, da die Kalibrierqualität in hohem Maße von der Genauigkeit des Kalibriermusters abhängt.

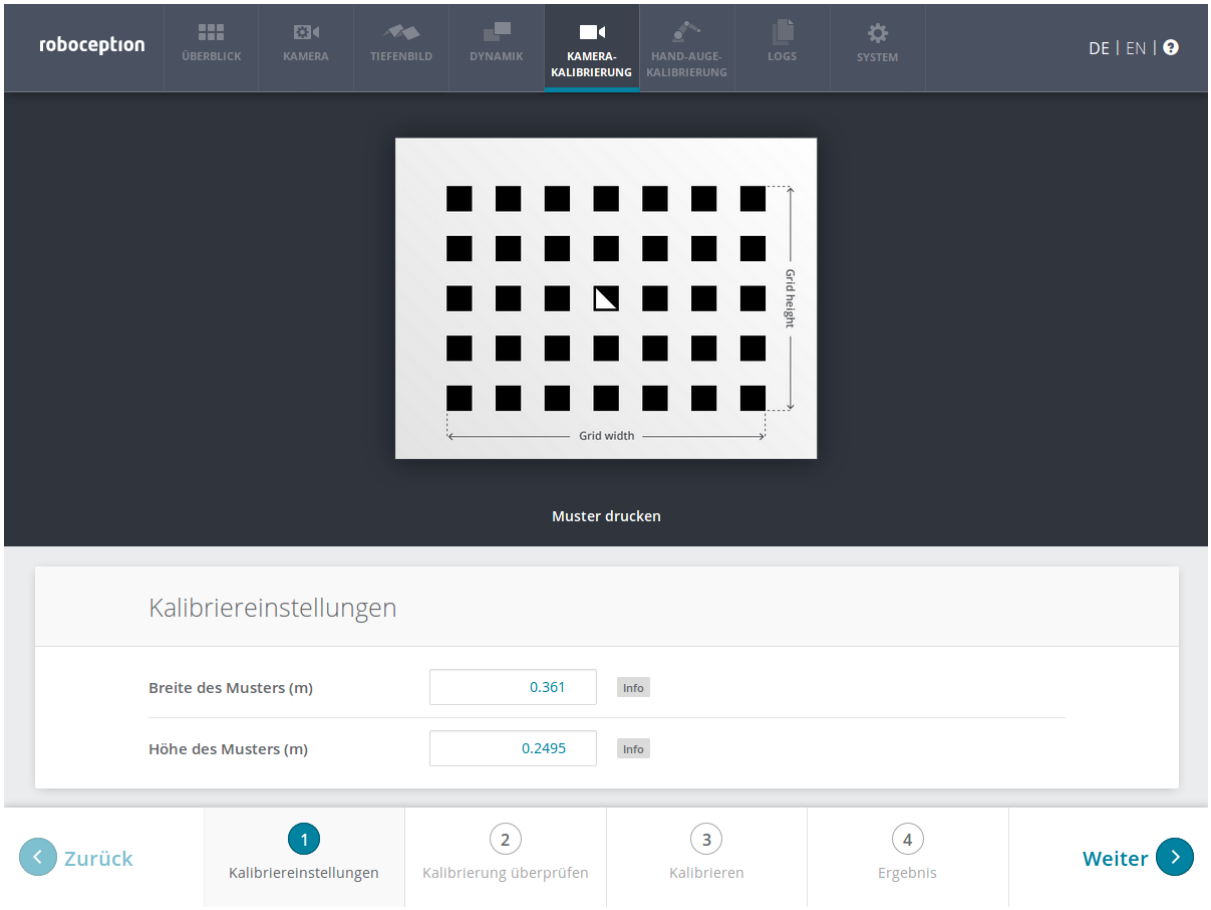


Abb. 6.7: Kalibriereinstellungen

Um die Kalibrierung der Kamera überprüfen bzw. neu durchführen zu können, muss in der *Web GUI* (Abschnitt 4.5) das Modul *Kamera-Kalibrierung* ausgewählt werden. Dafür sind im ersten Schritt, wie im Screenshot oben gezeigt, die Breite und Höhe des Kalibriermusters anzugeben. Mit Klick auf *Weiter* gelangt der Benutzer zum nächsten Schritt.

Schritt 2: Kalibrierung überprüfen

Im zweiten 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. Alle schwarzen Quadrate des Musters müssen komplett sichtbar und dürfen nicht verdeckt sein. Jedes korrekt erkannte Quadrat wird mit einem grünen Haken belegt. Das Muster kann nur dann korrekt erkannt werden, wenn alle schwarzen Quadrate erkannt werden. Nachdem das Muster vollständig erkannt wurde, wird der Kalibrierfehler automatisch berechnet und das Ergebnis auf dem Bildschirm angegeben.

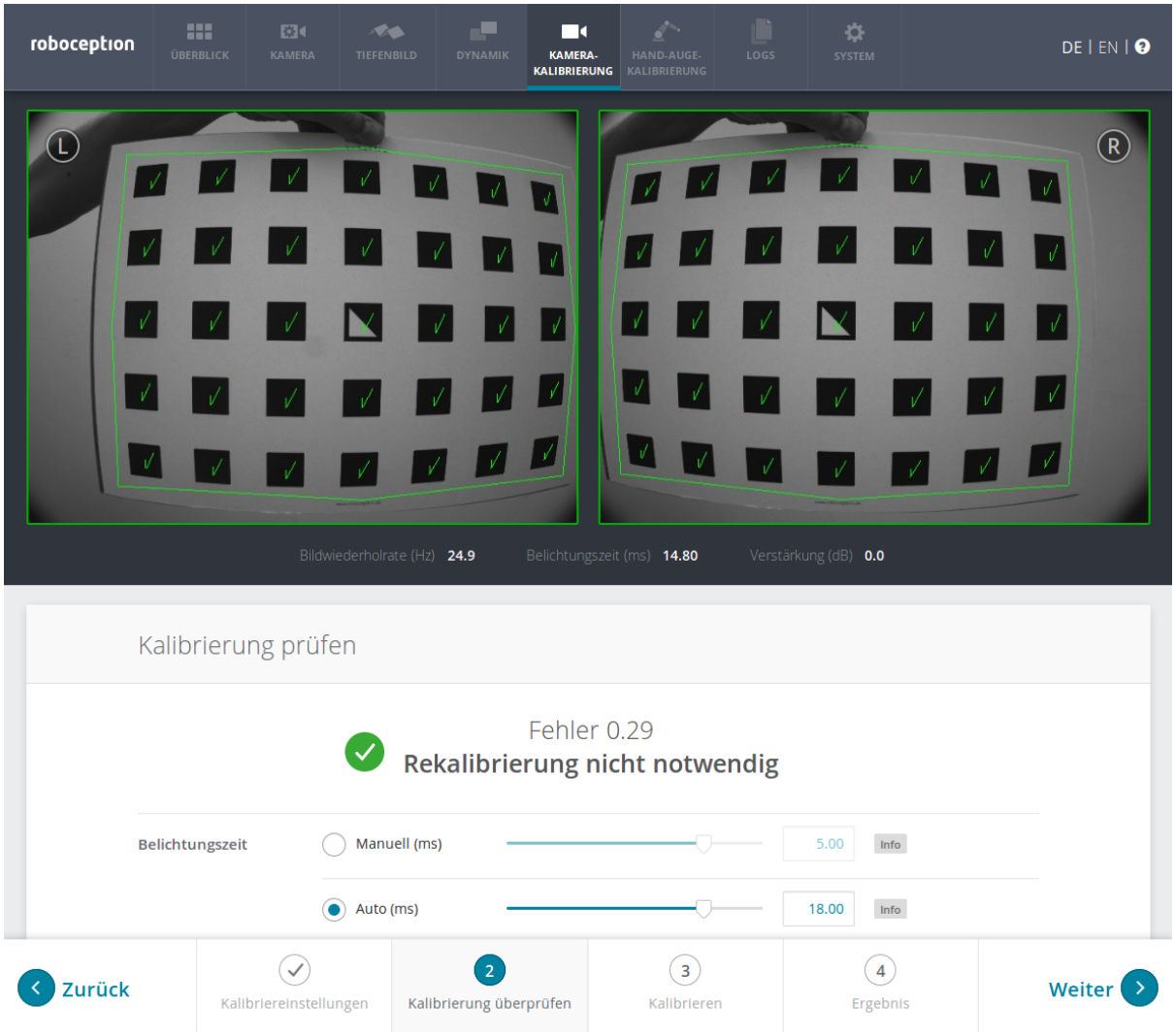


Abb. 6.8: Überprüfung der Kalibrierung

Werden einige der Quadrate nicht oder nur für kurze Zeit erkannt, so kann dies an einem unpräzisen oder beschädigten Kalibriermuster oder an schlechten Lichtverhältnissen liegen.

Hinweis: 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.

Der typische Kalibrierfehler beläuft sich auf etwa 0,3 Pixel. Liegt der Fehler unter einem Wert von 0,4 bis 0,5 Pixel, 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.

Achtung: Große Kalibrierfehler können durch falsch kalibrierte Kameras, ein unpräzises Kalibriermuster oder eine falsch eingetragene Musterbreite oder Musterhöhe verursacht werden. Der Benutzer muss sich daher vergewissern, dass das Muster präzise und die erfassten Breiten- und Höhendaten korrekt sind. Anderenfalls kann die manuelle Kalibrierung sogar dazu führen, dass die Kameras dekalibriert werden!

Schritt 3: Durchführung der Kalibrierung

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 Bildrauschen vermieden werden. Die maximale Belichtungszeit im automatischen Modus sollte groß genug sein, um einen sehr kleinen Verstärkungsfaktor zu erzielen, idealerweise um 0,0 dB. Die Verstärkung wird, wie in Abb. 6.9 gezeigt, unter den Kamerabildern angegeben.

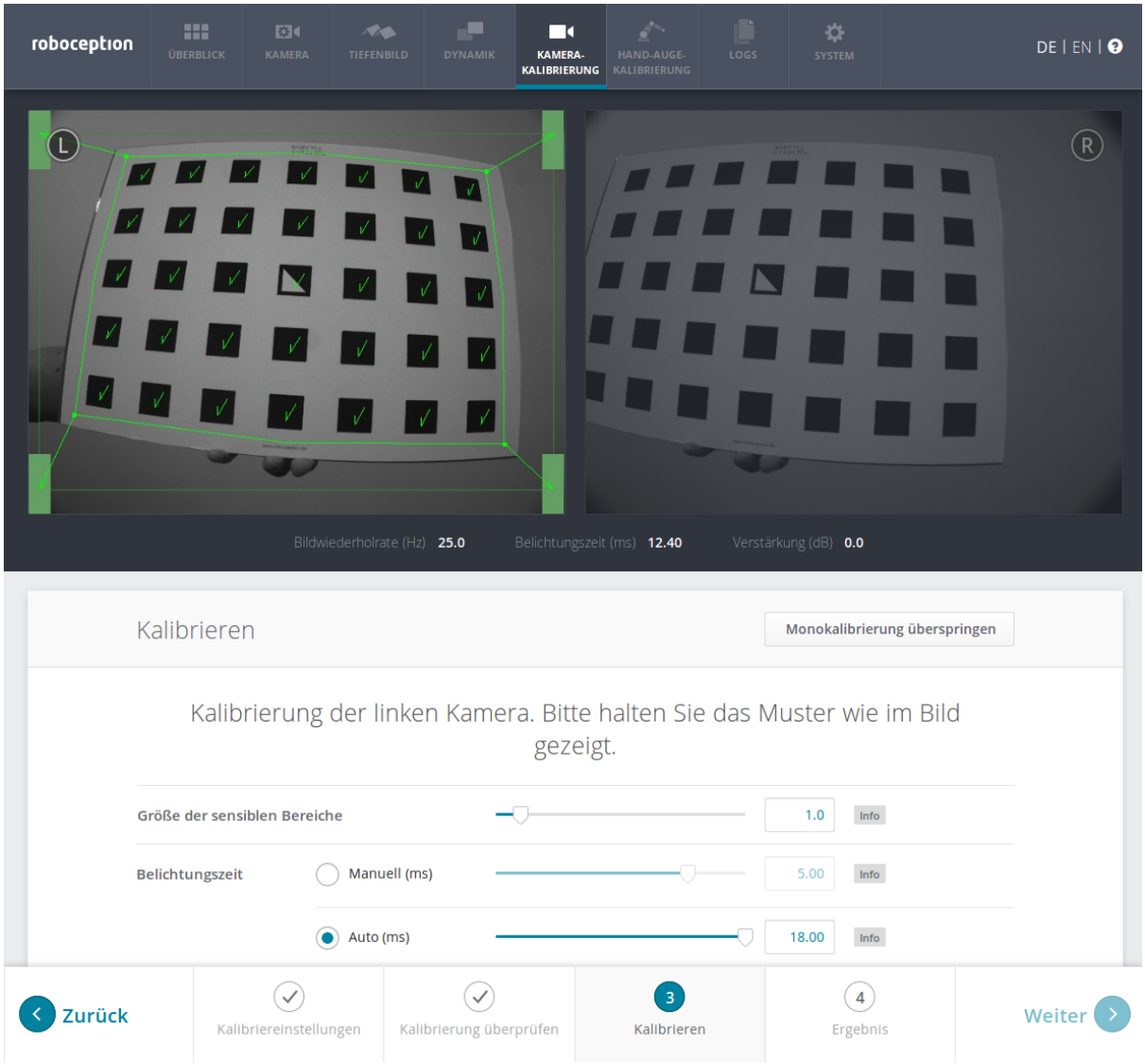


Abb. 6.9: Start des Kalibriervorgangs

Zur Kalibrierung muss das Kalibriermuster in verschiedenen Ausrichtungen vor die Kamera gehalten werden. Die Pfeile, die von den Ecken des Musters bis zu den grünen Bildschirmbereichen führen, geben an, dass alle Musterecken innerhalb der grünen Rechtecke platziert werden müssen. Diese grünen Rechtecke sind sensible Bereiche. Mit dem Schieberegler *Größe der sensiblen Bereiche* lässt sich, wie im Screenshot in Abb. 6.9 gezeigt, die Größe der Rechtecke einstellen, um die Kalibrierung zu vereinfachen. Es ist jedoch zu bedenken, dass die Größe nicht zu stark erhöht werden darf, da dies auf Kosten der Kalibriergenauigkeit gehen kann.

Häufig wird der Fehler begangen, das Muster bei der Kalibrierung falsch herum zu halten. Dieser Fehler lässt sich leicht erkennen, da sich die von den Musterecken zu den grünen Rechtecken verlaufenden Linien in diesen Fall kreuzen (siehe Abb. 6.10).

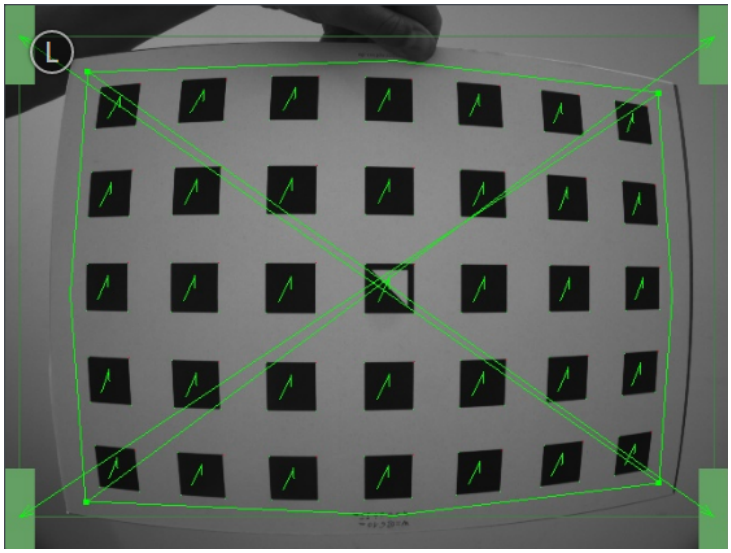


Abb. 6.10: Wird das Kalibriermuster falsch herum gehalten, kreuzen sich die grünen Linien.

Hinweis: Die Kalibrierung mag umständlich erscheinen, da das Muster hierfür in bestimmten vordefinierten Stellungen gehalten werden muss. Dieses Vorgehen ist jedoch zwingend einzuhalten, soll ein qualitativ hochwertiges Kalibrierergebnis erzielt werden.

Monokalibrierung

Um den *rc_visard* vollständig zu kalibrieren, müssen zunächst beide Kameras einzeln intrinsisch kalibriert werden. 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 sollte die Option *Monokalibrierung überspringen* auf der Registerkarte *Kalibrieren* ausgewählt werden, um die Monokalibrierung bei der ersten Neukalibrierung zu überspringen. Anschließend sind die in *Stereokalibrierung* angegebenen Schritte zu befolgen. Führt die Stereokalibrierung nicht zu einem akzeptablen Kalibrierfehler, sollte die Kalibrierung erneut vorgenommen werden, jedoch ohne die Monokalibrierung zu überspringen.

Für den Prozess der Monokalibrierung ist das Kalibriermuster für beide Kameras in den in Abb. 6.11 angegebenen Stellungen zu halten.

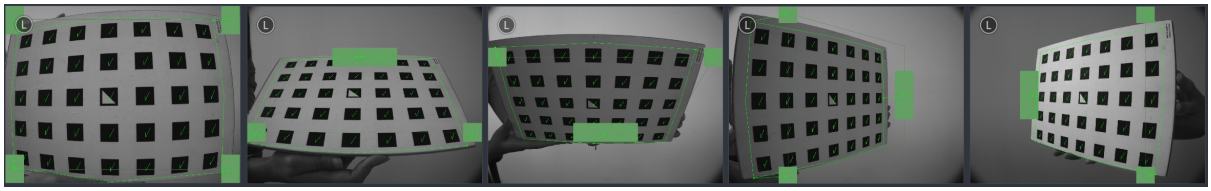


Abb. 6.11: Musterposen für die Monokalibrierung

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.

Stereokalibrierung

Nachdem die Monokalibrierung abgeschlossen ist bzw. wenn sie übersprungen wurde, beginnt der Prozess der Stereokalibrierung. Bei der Stereokalibrierung wird die relative Rotation und Translation der Kameras zueinander ermittelt.

Zunächst sollte das Muster in einem Abstand von weniger als 40 cm vor den Sensor gehalten werden. Es muss auf beiden Bildern vollständig sichtbar sein und die Blickrichtung der Kameras sollte senkrecht zum Muster stehen. Ist im Bild eine grüne Umrandung zu sehen, so bedeutet dies, dass die Bilder für die Kalibrierung akzeptiert wurden.

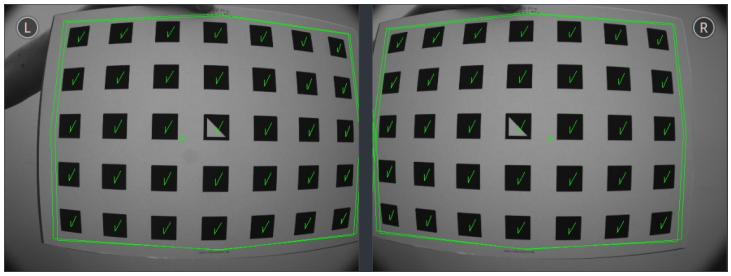


Abb. 6.12: Erster Schritt der Stereokalibrierung: Positionierung des Kalibriermusters in einem Abstand von weniger als 40 cm vor dem Sensor

Danach sollte das Muster wenigstens 1 m von den Kameras entfernt gehalten werden. Das kleine Kreuz in der Mitte der Bilder sollte sich innerhalb des Musters befinden und die Blickrichtung der Kameras sollte senkrecht zum Muster stehen. Ist im Bild eine grüne Umrandung zu sehen, so bedeutet dies, dass die Bilder für die Kalibrierung akzeptiert wurden.

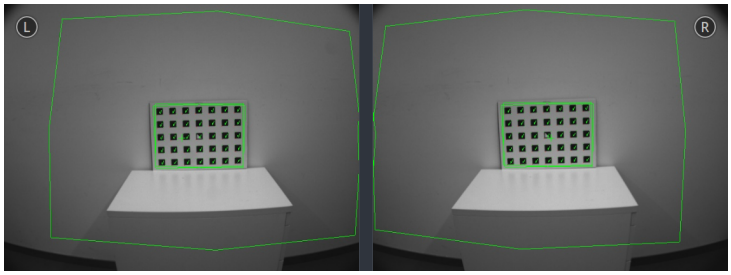


Abb. 6.13: Zweiter Schritt der Stereokalibrierung: Positionierung des Musters in einem Abstand von mehr als 1 m zum Sensor

Hinweis: Falls alle Häkchen auf dem Kalibriermuster verschwinden, liegt dies daran, dass die Kamerablickrichtung nicht senkrecht zum Muster steht, das grüne Kreuz sich nicht innerhalb des Musters befindet oder das Muster zu weit von der Kamera entfernt ist.

Schritt 4: 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 etwa 0,3.

Hinweis: 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.

Mit Klick auf *Kalibrierung speichern* wird das Kalibrierergebnis übernommen und auf dem Sensor gespeichert.

Achtung: 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.

6.7.3 Parameter

Dieses Modul wird in der REST-API als `rc_stereocalib` bezeichnet.

Hinweis: 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.

6.7.4 Services

Hinweis: 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.

6.8 Hand-Auge-Kalibrierung

Für Anwendungen, bei denen der *rc_visard* in eines oder mehrere Robotersysteme integriert wird, muss er zum jeweiligen Roboter-Koordinatensystem kalibriert werden. Zu diesem Zweck wird der *rc_visard* mit einer internen Kalibrierroutine, dem Modul zur *Hand-Auge-Kalibrierung*, ausgeliefert.

Hinweis: Für die Hand-Auge-Kalibrierung ist es völlig unerheblich, in Bezug auf welches benutzerdefinierte Roboter-Koordinatensystem der *rc_visard* kalibriert wird. Hierbei kann es sich um einen Endeffektor des Roboters (z. B. Flansch oder Tool Center Point (Werkzeugmittelpunkt)) oder um einen beliebigen anderen Punkt in der Roboterstruktur handeln. Einzige Voraussetzung für die Hand-Auge-Kalibrierung ist, dass die Pose (d. h. Positions- und Rotationswerte) dieses Roboter-Koordinatensystems in Bezug auf ein benutzerdefiniertes externes Koordinatensystem (z. B. Welt oder Roboter-Montagepunkt) direkt von der Robotersteuerung erfasst und an das Kalibriermodul übertragen werden kann.

Die *Kalibrierroutine* (Abschnitt 6.8.3) ist ein benutzerfreundliches dreistufiges Verfahren, für das mit einem Kalibriermuster gearbeitet wird. Das Muster lässt sich ausdrucken und muss auf eine flache Oberfläche geklebt werden. Eine Vorlage mit zugehörigen Anweisungen kann `hier`: http://www.roboception.com/download heruntergeladen werden. Die Anweisungen sind genau zu befolgen, da die Kalibrierqualität in hohem Maße von der Genauigkeit des Kalibriermusters abhängt.

6.8.1 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 7.2) des *rc_visard* enthalten. Der REST-API-Name dieses Moduls lautet `rc_hand_eye_calibration` und seine Services werden in *Services* (Abschnitt 6.8.5) erläutert.

   Hinweis: Für den beschriebenen Ansatz wird eine Netzwerkverbindung zwischen dem *rc_visard* und der Robotersteuerung benötigt, damit die Steuerung die Roboterposen an das Kalibriermodul des Sensors 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 Registerkarte *Hand-Auge-Kalibrierung* der *Web GUI* (Abschnitt 4.5) einen geführten Prozess vor, mit dem sich die Kalibrierroutine **manuell** durchführen lässt.

   Hinweis: 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.

6.8.2 Sensormontage

Wie in Abb. 6.14 und Abb. 6.15 dargestellt, ist für die Montage des *rc_visard* zwischen zwei unterschiedlichen Anwendungsfällen zu unterscheiden:

1. Der *rc_visard* wird **am Roboter montiert**, d. h. seine Montagepunkte sind mechanisch (siehe *Installation*, Abschnitt 4) mit einem Roboterpunkt (d. h. Flansch oder flanschmontiertes Werkzeug) verbunden und der *rc_visard* bewegt sich demnach mit dem Roboter.
2. Der *rc_visard* ist nicht am Roboter montiert, sondern an einem Tisch oder anderen Ort in der Nähe des Roboters befestigt und verbleibt daher verglichen mit dem Roboter in einer **statischen** Position.

Die allgemeine *Kalibrierroutine* (Abschnitt 6.8.3) ist in beiden Anwendungsfällen sehr ähnlich. Sie unterscheidet sich jedoch hinsichtlich der semantischen Auslegung der Ausgabedaten, d. h. der erhaltenen Kalibriertransformation, und hinsichtlich der Befestigung des Kalibriermusters.

Kalibrierung eines robotergeführten Sensors Soll ein robotergeführter *rc_visard* 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. 6.14).

Achtung: Es ist äußerst wichtig, dass sich das Kalibriermuster in Schritt 2 der *Kalibrierroutine* (Abschnitt 6.8.3) nicht bewegt. Daher wird dringend empfohlen, das Muster in seiner Position sicher zu fixieren, um unbeabsichtigte Bewegungen, wie sie durch Vibrationen, Kabelbewegungen oder Ähnliches ausgelöst werden, zu verhindern.

Das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 6.8.3) ist eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem des *rc_visard* und dem benutzerdefinierten *Roboter*-Koordinatensystems beschreibt, sodass Folgendes gilt:

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

wobei $\mathbf{p}_{\text{robot}} = (x, y, z)^T$ ein 3D-Punkt ist, dessen Koordinaten im *Roboter*-Koordinatensystem angegeben werden; $\mathbf{p}_{\text{camera}}$ den gleichem Punkt im *Kamera*-Koordinatensystem darstellt; und $\mathbf{R}(\mathbf{T})$ sowie $\mathbf{t}(\mathbf{T})$ die 3×3 Drehmatrix und den 3×1 Translationsvektor für eine Pose $\mathbf{T}$ angeben.

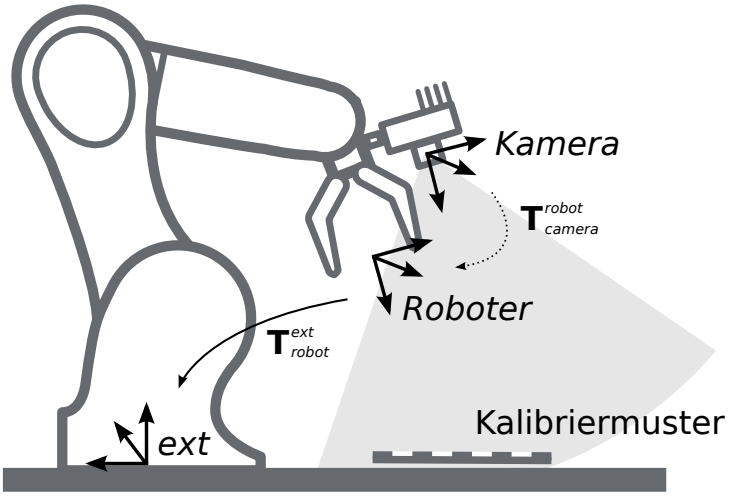


Abb. 6.14: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung eines robotergeführten *rc_visard*: Der Sensor wird mit einer festen relativen Position zu einem benutzerdefinierten *Roboter*-Koordinatensystem (z. B. Flansch oder Werkzeugmittelpunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des *Roboter*-Koordinatensystems in Bezug auf ein benutzerdefiniertes externes Referenzkoordinatensystem (*ext*) während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibriervorgangs ist die gewünschte Kalibriertransformation $\mathbf{T}^{\text{robot}}_{\text{camera}}$, d. h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten *Roboter*-Koordinatensystems.

Kalibrierung eines statisch montierten Sensors In Anwendungsfällen, bei denen der *rc_visard* statisch verglichen zum Roboter montiert wird, muss das Kalibriermuster, wie im Beispiel in Abb. 6.15 und Abb. 6.16 angegeben, angebracht werden.

Hinweis: Für das Modul zur Hand-Auge-Kalibrierung spielt es keine Rolle, wie das Kalibriermuster in Bezug auf das benutzerdefinierte *Roboter*-Koordinatensystem genau angebracht und positioniert wird. Das bedeutet, dass die relative Positionierung des Kalibriermusters zu diesem Koordinatensystem weder bekannt sein muss, noch für die Kalibrierroutine relevant ist (siehe in Abb. 6.16).

Achtung: 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 6.8.3) vom Benutzer definierte *Roboter*-Koordinatensystem nicht verändert.

Daher wird dringend empfohlen, unbeabsichtigte Bewegungen, wie sie beispielsweise durch Vibrationen verursacht werden, zu vermeiden. Hierfür kann das Kalibriermuster beispielsweise auf einer hölzernen Unterlage (empfohlene Mindestdicke: 1 cm) aufgebracht werden, die anschließend an die Roboterstruktur, z. B. seinen Flansch oder sein Werkzeug, angeschraubt wird.

In diesem Anwendungsfall ist das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 6.8.3) die Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$, die die (zuvor unbekannte) relative Transformation der zwischen dem *Kamera*-Koordinatensystem des *rc_visard* und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

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

wobei $\mathbf{p}_{\text{ext}} = (x, y, z)^T$ ein 3D-Punkt ist, dessen Koordinaten im externen Referenzkoordinatensystem *ext* angegeben werden; $\mathbf{p}_{\text{camera}}$ den gleichen Punkt im *Kamera*-Koordinatensystem darstellt; und $\mathbf{R}(\mathbf{T})$ sowie $\mathbf{t}(\mathbf{T})$ die 3×3 Drehmatrix und den 3×1 Translationsvektor für eine Position $\mathbf{T}$ angeben.

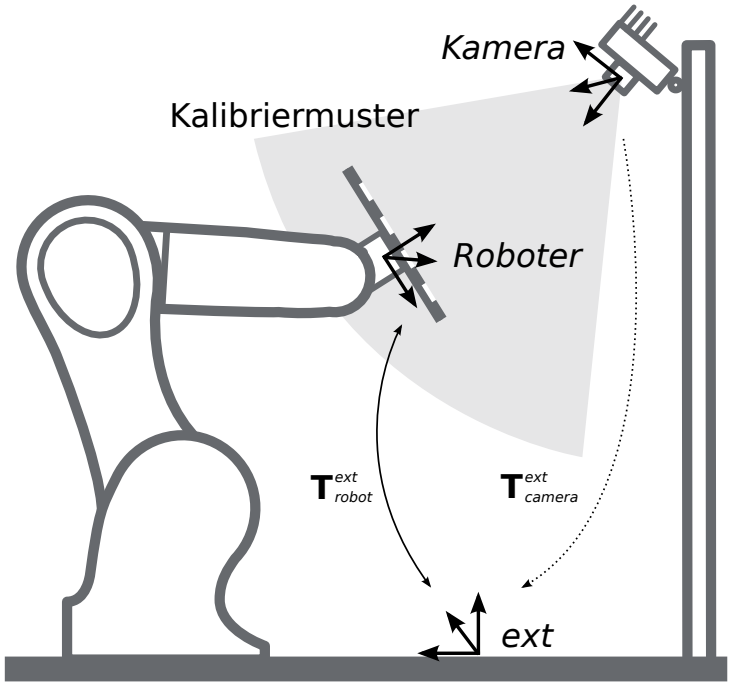


Abb. 6.15: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung eines statisch montierten *rc_visard*: Der Sensor wird mit einer festen Position relativ zu einem benutzerdefinierten externen Referenzkoordinatensystem *ext* (z. B. Weltkoordinatensystem oder Roboter-Montagepunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems in Bezug auf dieses Koordinatensystem während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibrierprozesses ist die gewünschte Kalibriertransformation $\mathbf{T}^{\text{ext}}_{\text{camera}}$, d. h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten externen Koordinatensystem *ext*.

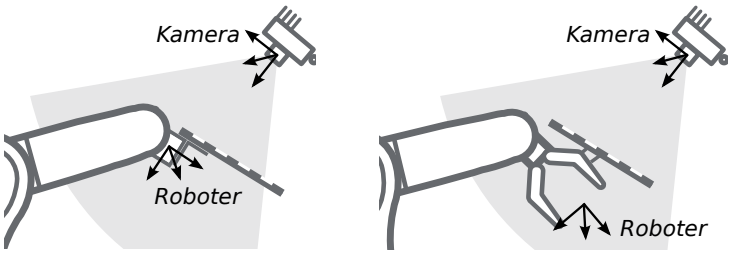


Abb. 6.16: Alternative Montageoptionen für die Befestigung des Kalibriermusters am Roboter

6.8.3 Kalibrierroutine

Die allgemeine Hand-Auge-Kalibrierroutine besteht aus den in Abb. 6.17 angegebenen drei Schritten. Auch der Hand-Auge-Kalibriervorgang der *Web GUI* (Abschnitt 4.5) greift diese drei Schritte auf.

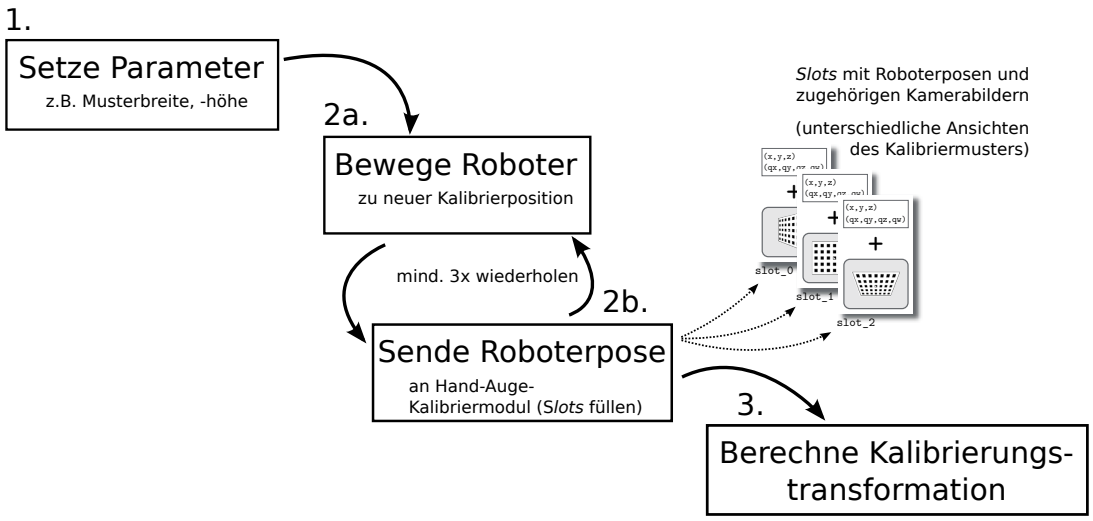


Abb. 6.17: Darstellung der drei Schritte der Hand-Auge-Kalibrierroutine

Schritt 1: Einstellung der Parameter

Bevor mit der eigentlichen Kalibrierroutine begonnen werden kann, müssen die Parameter für die Mustergröße und Sensormontage eingestellt werden. Für die REST-API sind die entsprechenden *Parameter* (Abschnitt 6.8.4) aufgelistet.

Web GUI-Beispiel: Die Web GUI bietet eine Oberfläche, über die sich diese Parameter im ersten Schritt der Kalibrierroutine, wie in Abb. 6.18 gezeigt, erfassen lassen. Neben den Angaben zur Mustergröße und Sensormontage kann der Benutzer in der Web GUI auch das Format der *Pose* einstellen. Dieses Format wird im nachfolgenden Schritt 2 des Kalibriervorgangs verwendet, um die Roboterposen zu übertragen. Folgende Formate sind möglich: *XYZABC* für Positionen und Eulersche Winkel oder *XYZ+Quaternion* für Positionen samt Quaternionen für die Darstellung von Drehungen. Für die genauen Definitionen siehe *Formate für Posendaten* (Abschnitt 12.1).

> **Hinweis:** Der Parameter *Pose* in der Web GUI wurde lediglich der Benutzerfreundlichkeit halber hinzugefügt. Für die programmgesteuerte Übertragung von Roboterposen über die REST-API ist die Verwendung des *XYZ+Quaternion*-Formats zwingend vorgeschrieben.

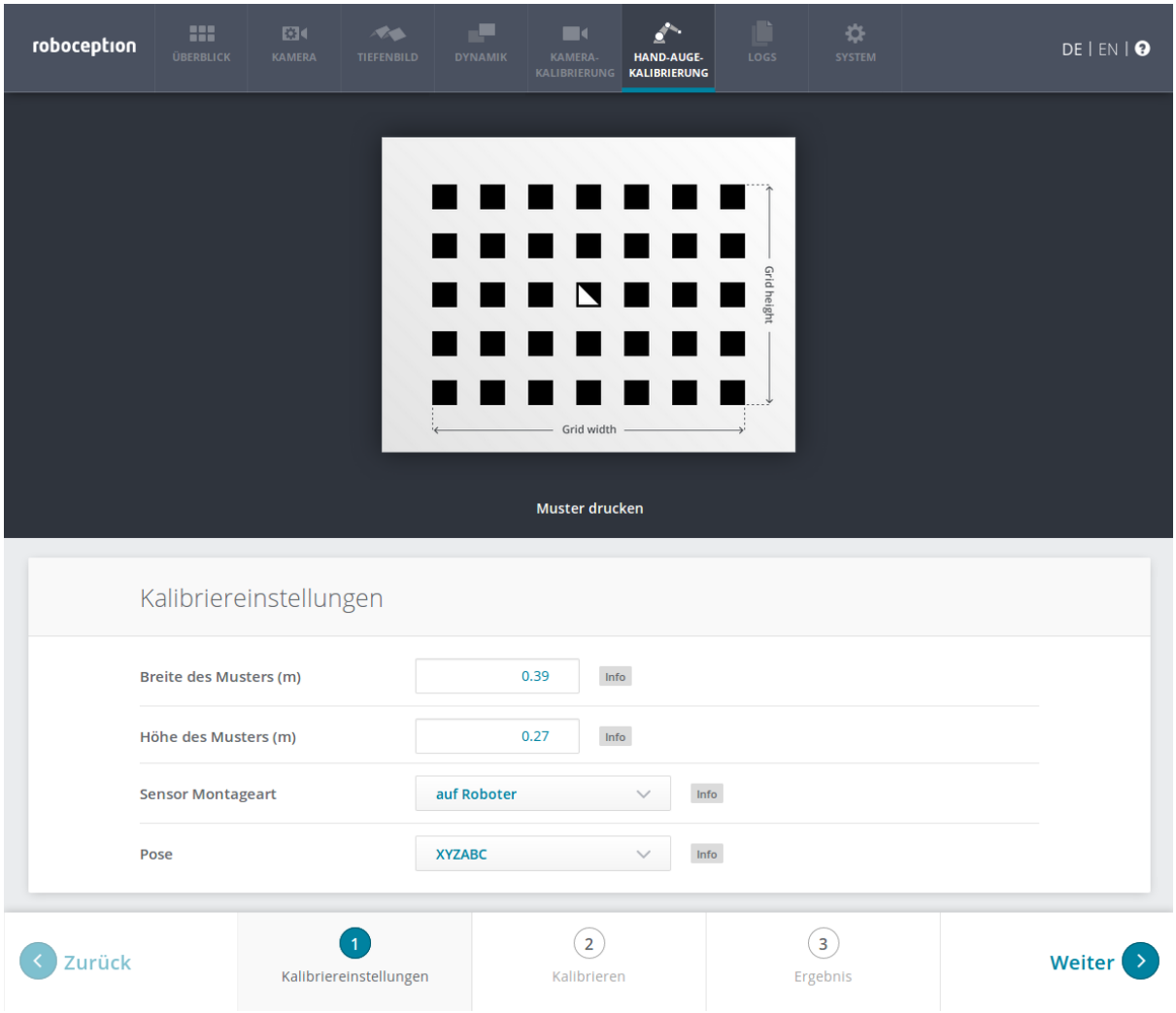


Abb. 6.18: Erfassung der Parameter zur Hand-Auge-Kalibrierung in der Web GUI des *rc_visard*

Schritt 2: Auswahl und Übertragung der Kalibrierpositionen des Roboters

In diesem Schritt (2a.) definiert der Benutzer verschiedene Kalibrierpositionen, die der Roboter anfahren muss. Dabei ist sicherzustellen, dass das Kalibriermuster bei allen Positionen im linken Kamerabild des *rc_visard* vollständig sichtbar ist. Zudem müssen die Roboterpositionen sorgsam ausgewählt werden, damit der *rc_visard* das Kalibriermuster aus unterschiedlichen Perspektiven aufnehmen kann. Abb. 6.19 zeigt eine schematische Darstellung der empfohlenen vier Ansichten.

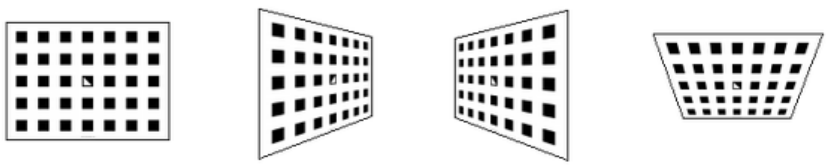


Abb. 6.19: Empfohlene Ansichten des Kalibriermusters während des Kalibriervorgangs

Achtung: Die Kalibrierqualität, d. h. die Genauigkeit des berechneten Kalibrierergebnisses, hängt von den Ansichten des Kalibriermuster ab. Je vielfältiger die Perspektiven sind, desto besser gelingt die Kalibrierung.

Werden sehr ähnliche Ansichten ausgewählt, d. h. werden die Positionen des Roboters bei den verschiedenen Wiederholungen von Schritt 2a nur leicht variiert, kann dies zu einer ungenauen Schätzung der gewünschten Kalibriertransformation führen.

Nachdem der Roboter die jeweilige Kalibrierposition erreicht hat, muss die entsprechende Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems im benutzerdefinierten externen Referenzkoordinatensystem *ext* an das Modul zur Hand-Auge-Kalibrierung übertragen werden (2b.). Hierfür bietet das Softwaremodul verschiedene *Slots*, in denen die gemeldeten Posen mit den zugehörigen Bildern der linken Kamera des *rc_visard* hinterlegt werden können. Alle gefüllten Slots werden dann verwendet, um die gewünschte Kalibriertransformation zwischen dem Kamera-Koordinatensystem des *rc_visard* und dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Sensoren) bzw. dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Sensoren) zu berechnen.

Hinweis: Um die Transformation für die Hand-Auge-Kalibrierung erfolgreich zu berechnen, müssen mindestens drei verschiedenen Roboter-Kalibrierposen übertragen und in Slots hinterlegt werden. Um Kalibrierfehler zu verhindern, die durch ungenaue Messungen entstehen können, sind mindestens **vier Kalibrierposen empfohlen**.

Um diese Posen programmgesteuert zu übertragen, bietet die REST-API den Service `set_pose` (siehe *Services*, Abschnitt 6.8.5).

Web GUI-Beispiel: Nachdem die Kalibriereinstellungen in Schritt 1 abgeschlossen und die Schaltfläche *Weiter* betätigt wurde, bietet die Web GUI vier verschiedene Slots (*Erste Ansicht*, *Zweite Ansicht*, usw.), in die der Benutzer die Posen manuell eintragen kann. Ganz oben wird ein Live-Stream der Kamera angezeigt, um nachverfolgen zu können, ob das Kalibriermuster aktuell erkannt wird oder nicht. Neben jedem Slot wird eine Empfehlung für die Ansicht des Kalibriermusters angezeigt. Der Roboter sollte für jeden Slot so bewegt werden, dass die empfohlene Ansicht erreicht wird.

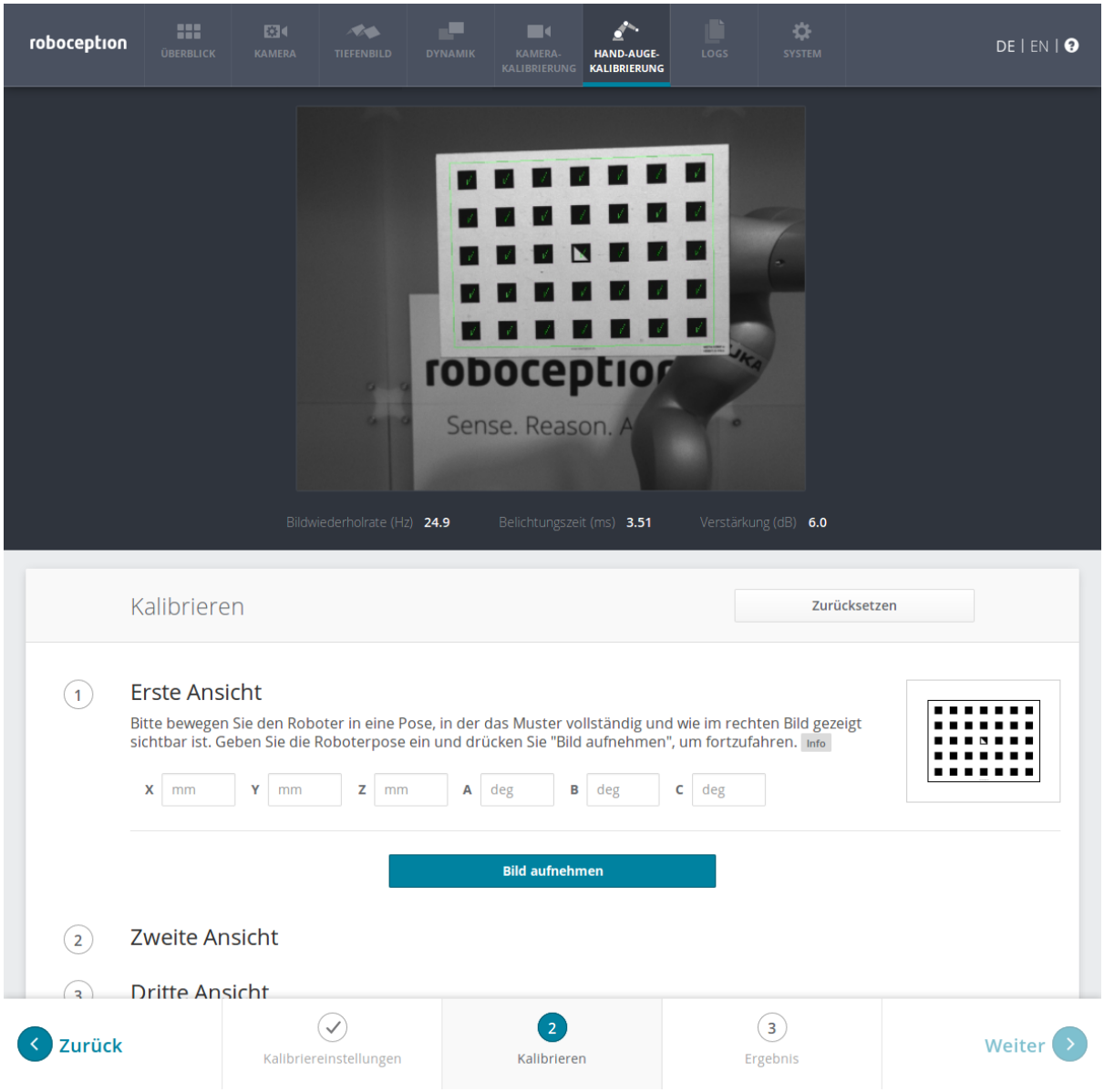


Abb. 6.20: Erstes Beispielbild für den Hand-Auge-Kalibriervorgang eines statisch montierten *rc_visard*

Sobald das tatsächliche Bild der empfohlenen Ansicht entspricht, sind die Posen des benutzerdefinierten *Roboter*-Koordinatensystems manuell in den entsprechenden Textfeldern zu erfassen und das Kamerabild mit der Schaltfläche *Bild aufnehmen* aufzunehmen.

Hinweis: Der Zugriff auf die Posendaten des Roboters hängt vom Modell des Roboters und seinem Hersteller ab. Möglicherweise lässt sich dies über ein im Lieferumfang des Roboters enthaltenes Teach-in- oder Handheld-Gerät vornehmen.

Achtung: Es ist sorgsam darauf zu achten, dass genaue und korrekte Werte eingegeben werden. Selbst kleinste Ungenauigkeiten oder Tippfehler können dazu führen, dass die Kalibrierung fehlschlägt.

Dieser Vorgang ist insgesamt viermal zu wiederholen. Vorausgesetzt, die in Abb. 6.19 dargestellten Empfehlungen zur Aufnahme des Kalibriermusters von oben, von links, von vorn und von rechts wurden eingehalten, werden die folgenden Kamerabilder mit den jeweiligen Roboterposen an das Softwaremodul zur Hand-Auge-Kalibrierung übertragen:

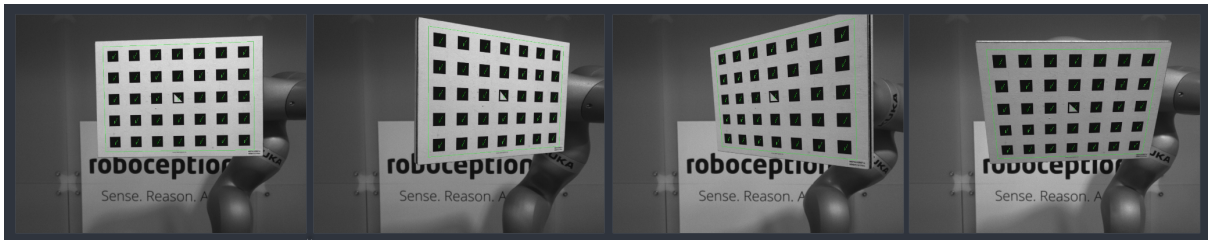


Abb. 6.21: Kamerabilder, die zum Zwecke der Kalibrierung aufgezeichnet wurden

Schritt 3: Berechnen und Speichern der Kalibriertransformation

Der letzte Schritt in der Hand-Auge-Kalibrierroutine besteht darin, die gewünschte Kalibriertransformation auf Grundlage der erfassten Posen und Kamerabilder zu berechnen. Die REST-API bietet hierfür den Service `calibrate` (siehe *Services*, Abschnitt 6.8.5). Je nachdem, wie der *rc_visard* montiert ist, wird dabei die Transformation (d. h. die Pose) zwischen dem *Kamera*-Koordinatensystem und entweder dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Sensoren) oder dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Sensoren) berechnet und ausgegeben (siehe *Sensormontage*, Abschnitt 6.8.2).

Damit der Benutzer die Qualität der resultierenden Kalibriertransformation beurteilen kann, gibt das Modul den Kalibrierfehler an. Dieser in Pixeln ausgedrückte Wert gibt den quadratischen Mittelwert des Reprojektionsfehlers über alle Kalibrierslots und alle Eckpunkte des Kalibriermusters an. Für eine intuitivere Darstellung lässt sich dieser Wert durch Verwendung der Brennweite f des *rc_visard* in Pixeln normalisieren:

$$E = \frac{E_{\text{camera}}}{f} \, .$$

> **Hinweis:** 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.

Der Wert E lässt sich nun als objektbezogener Fehler in Metern in der 3D-Welt auslegen. Angenommen, der Abstand zwischen dem Kalibriermuster und dem *rc_visard* beträgt einen Meter, dann liegt die *mittlere* Genauigkeit bei der Transformation der Kalibriermuster-Koordinaten vom *Kamera*-Koordinatensystem in das Zielkoordinatensystem bei $1 \cdot E$ Metern. Bei einem Abstand von 0,5 Metern gilt $0.5 \cdot E$ Meter usw.

Web GUI-Beispiel: Sofort nachdem die letzte der vier Aufnahmen getätigt wurde, löst die Web GUI automatisch die Berechnung des Kalibrierergebnisses aus. Der Benutzer muss nur auf die Schaltfläche *Weiter* klicken, um zum Ergebnis zu gelangen. Für dieses Beispiel mit einem statisch montierten *rc_visard* entspricht das Ergebnis der Position der linken Kamera des Sensors im Weltkoordinatensystem des Roboters – dargestellt in dem in Schritt 1 der Kalibrierroutine angegebenen Format *Pose*.

Aus dem in Abb. 6.22 angegebenen Fehler von $E_{\text{camera}} = 0.4$ Pixeln ergibt sich eine Kalibriergenauigkeit von $E = \frac{E_{\text{camera}}}{f} \approx \frac{0.4}{1081.46} \approx 0.00036$. Bei einem Abstand von einem Meter zum Sensor entspricht dies 0,36 mm; damit liegt der Fehler für diesen Kalibriervorgang im Submillimeterbereich.

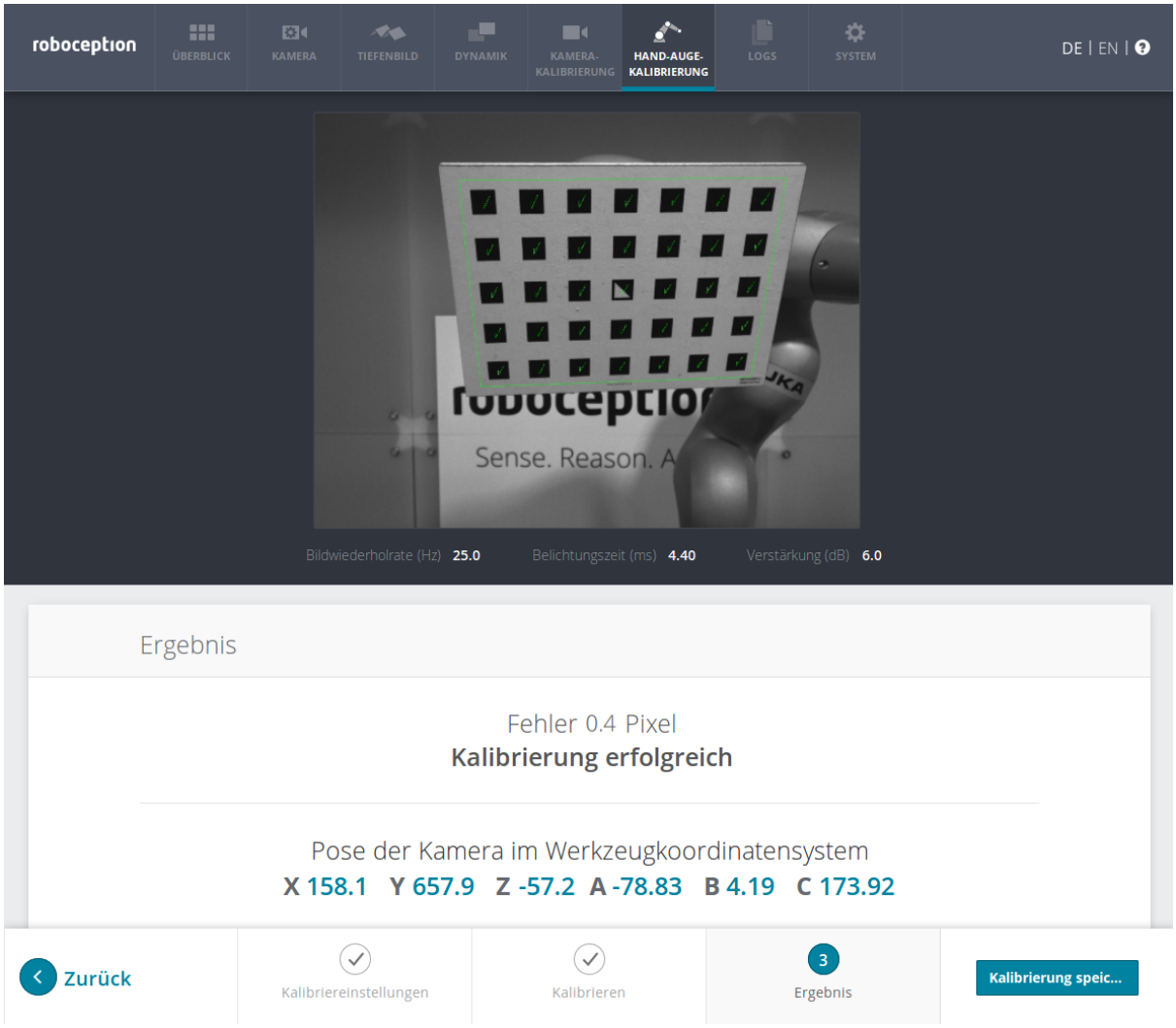


Abb. 6.22: Ergebnis der Hand-Auge-Kalibrierung, dargestellt in der Web GUI

6.8.4 Parameter

Das Modul zur Hand-Auge-Kalibrierung wird in der REST-API als `rc_hand_eye_calibration` bezeichnet und in der *Web GUI* (Abschnitt 4.5) auf der Registerkarte *Hand-Auge-Kalibrierung* dargestellt. Der Benutzer kann die Kalibrierparameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter.

Tab. 6.8: Laufzeitparameter des `rc_hand_eye_calibration`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`grid_height`	float64	0	10	0	Höhe des Kalibriermusters in Metern
`grid_width`	float64	0	10	0	Breite des Kalibriermusters in Metern
`robot_mounted`	bool	0	1	1	Angabe, ob der *rc_visard* auf einem Roboter montiert ist

Dieses Modul meldet keine Statuswerte.

Beschreibung der Laufzeitparameter

Für die Beschreibungen der Parameter sind die in der Web GUI gewählten Namen der Parameter in Klammern angegeben.

`grid_width` (*Breite des Musters (m)*) Breite des Kalibriermusters in Metern. Die Breite sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, ermittelt werden.

`grid_height` (*Höhe des Musters (m)*) Höhe des Kalibriermusters in Metern. Die Höhe sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, ermittelt werden.

`robot_mounted` (*Sensor Montageart*) Ist dieser Parameter auf 1 gesetzt, dann ist der *rc_visard* an einem Roboter montiert. Ist er auf 0 gesetzt, ist der *rc_visard* statisch montiert und das Kalibriermuster ist am Roboter angebracht.

(*Pose*) Der Benutzerfreundlichkeit halber kann der Benutzer die Kalibrierungsdaten in der Web GUI entweder im Format *XYZABC* oder im Format *XYZ+Quaternion* angeben (siehe *Formate für Posendaten*, Abschnitt 12.1). Wird die Kalibrierung über die REST-API vorgenommen, dann wird das Kalibrierergebnis immer im Format *XYZ+Quaternion* angegeben.

6.8.5 Services

Auf die Services, die die REST-API für die programmgesteuerte Durchführung der Hand-Auge-Kalibrierung und für die Speicherung oder Wiederherstellung der Modulparameter bietet, wird im Folgenden näher eingegangen.

`save_parameters` Mit diesem Service werden die aktuellen Parametereinstellungen zur Hand-Auge-Kalibrierung auf dem *rc_visard* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

`reset_defaults` Hiermit werden die Werkseinstellungen der Parameter dieses Moduls wieder hergestellt und angewandt („factory reset“). Dies hat keine Auswirkungen auf das Kalibrierergebnis oder auf die während der Kalibrierung gefüllten `Slots`. Es werden lediglich Parameter, wie die Maße des Kalibriermusters oder die Montageart des Sensors, zurückgesetzt.

> **Achtung:** Der Benutzer muss bedenken, dass beim Aufruf dieses Services die aktuellen Parametereinstellungen unwiderruflich verloren gehen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert keine Rückgabewerte.

`set_pose` Dieser Service setzt die Roboterpose als Kalibrierpose für die Hand-Auge-Kalibrierroutine.

Für diesen Service sind folgende Argumente nötig:

```
{
  "pose": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
```

```
  },
  "slot": "int32"
}
```

Dieser Service liefert folgenden Rückgabewert:

```
{
  "message": "string",
  "status": "int32",
  "success": "bool"
}
```

Das `slot`-Argument wird verwendet, um den verschiedenen Kalibrierpositionen Ziffern 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.

`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.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:

```
{
  "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.

> **Hinweis:** Zur Berechnung der Transformation der Hand-Auge-Kalibrierung werden mindestens drei Roboterposen benötigt (siehe `set_pose`). Empfohlen wird jedoch die Verwendung von vier Kalibrierposen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:

```
{
  "error": "float64",
  "message": "string",
  "pose": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  },
  "status": "int32",
  "success": "bool"
}
```

`save_calibration` Hiermit wird das Ergebnis der Hand-Auge-Kalibrierung auf dem *rc_visard* gespeichert und das vorherige Ergebnis überschrieben. Das gespeicherte Ergebnis lässt sich jederzeit über den Service `get_calibration` abrufen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:


```
{
  "message": "string",
  "status": "int32",
  "success": "bool"
}
```


get_calibration Hiermit wird die derzeit auf dem *rc_visard* gespeicherte Hand-Auge-Kalibrierung abgerufen.

Für diesen Service sind keine Argumente nötig.

Dieser Service liefert folgenden Rückgabewert:


```
{
  "error": "float64",
  "message": "string",
  "pose": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  },
  "status": "int32",
  "success": "bool"
}
```

7 Schnittstellen

Es stehen drei Schnittstellen zur Konfiguration und Datenübertragung des *rc_visard* zur Verfügung:

1. *GigE Vision 2.0/GenICam* (Abschnitt 7.1)

   Konfiguration bild- und kamerabezogener Einstellungen.

2. *REST API* (Abschnitt 7.2)

   Programmierschnittstelle zur Konfiguration des *rc_visard*, zur Abfrage von Statusinformationen, zum Anfordern von Datenströmen, usw.

3. *rc_dynamics streams* (Abschnitt 7.3)

   Echtzeit-Datenströme, die Zustandsschätzungen samt Posen-, Geschwindigkeits- und anderen Daten enthalten, werden über die *rc_dynamics*-Schnittstelle bereitgestellt. Sie sendet *ProtoBuf*-kodierte Nachrichten über UDP.

7.1 GigE Vision 2.0/GenICam-Schnittstelle

Gigabit Ethernet for Machine Vision (oder kurz „GigE Vision®“) ist eine industrieller Kameraschnittstellen-Standard basierend auf UDP/IP (siehe http://www.gigevision.com). Der *rc_visard* nutzt den GigE Vision®-Standard der Version 2.0 und ist damit mit allen GigE Vision®-2.0-Standard-konformen Frameworks und Bibliotheken kompatibel.

GigE Vision® verwendet GenICam (*Generic Interface for Cameras*), um die Eigenschaften der Kamera bzw. des Geräts zu beschreiben. Für nähere Informationen zu dieser generischen Programmierschnittstelle für Kameras siehe http://www.genicam.org/.

Über diese Schnittstelle stellt der *rc_visard* folgende Funktionen zur Verfügung:

- Discovery-Mechanismus,
- IP-Konfiguration,
- Konfiguration kamerabezogener Parameter,
- Bildaufnahme und
- Zeitsynchronisierung über das im Standard IEEE 1588-2008 definierte Precision Time Protocol (PTPv2).

Hinweis: 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.

Hinweis: Über seine Homepage stellt Roboception (http://www.roboception.com/download) Tools und eine C++-Programmierschnittstelle mit Beispielen zum Discovery-Mechanismus, zur Konfiguration und zum Bild-Streaming über die GigE Vision/GenICam-Schnittstelle zur Verfügung.

7.1.1 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 *Stereokamera* (Abschnitt 6.1) und *Stereo-Matching* (Abschnitt 6.2) beziehen.

Wichtige Standardparameter der GenICam-Schnittstelle

`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 7.1.2).

`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`, `WidthMax` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Maximale Breite eines Bildes (beträgt immer 1280 Pixel).

`Height`, `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 7.1.2).

`PixelFormat`

- Typ: Aufzählung, mögliche Werte: `Mono8` oder `YCbCr411_8` (nur bei Farbsensoren)
- Beschreibung: Pixelformat der linken und rechten rektifizierten Bilder die von den Komponenten `Intensity` und `IntensityCombined` zurückgegeben werden. Das `YCbCr411_8`-Format ist nur für Farbsensoren verfügbar.

`AcquisitionFrameRate`

- Typ: Float, Wertebereich: 1–25 Hz
- Voreinstellung: 25 Hz
- Beschreibung: Bildwiederholrate der Kamera (*FPS*, Abschnitt 6.1.3).

`ExposureAuto`

- Typ: Aufzählung, mögliche Werte: `Continuous` oder `Off`

- Voreinstellung: `Continuous`
- Beschreibung: Lässt sich für die manuelle Belichtung auf `Off` bzw. für die automatische Belichtung auf `Continuous` setzen (*Belichtung*, Abschnitt 6.1.3).

`ExposureTime`

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 5000 µs
- Beschreibung: Belichtungszeit der Kamera für den manuellen Belichtungsmodus, ausgedrückt in Mikrosekunden (*Manual*, Abschnitt 6.1.3).

`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 Kamera für den manuellen Belichtungsmodus, ausgedrückt in Dezibel (*Gain*, Abschnitt 6.1.3).

`BalanceWhiteAuto` (nur für Farbsensoren)

- 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 Farbsensoren verfügbar (*wb_auto*, Abschnitt 6.1.3).

`BalanceRatioSelector` (nur für Farbsensoren)

- 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 Farbsensoren verfügbar.

`BalanceRatio` (nur für Farbsensoren)

- 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. Dieses Merkmal ist nur für Farbkameras verfügbar (*wb_ratio*, Abschnitt 6.1.3).

`GevIEEE1588`

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

`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`.

`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. Der Wert beträgt jedoch 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.

Besondere Parameter der GenICam-Schnittstelle des *rc_visard*

`ExposureTimeAutoMax`

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 7000 µs
- Beschreibung: Maximale Belichtungszeit im automatischen Belichtungsmodus (*Auto*, Abschnitt 6.1.3).

`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.

`Baseline` (schreibgeschützt)

- Typ: Float
- Beschreibung: Abstand zwischen der linken und der rechten Kamera, ausgedrückt in Metern.

`DepthQuality`

- Typ: Aufzählung, mögliche Werte: `Low`, `Medium`, `High` oder `StaticHigh`
- Voreinstellung: `High`
- Beschreibung: Qualität der Disparitätsbilder. Eine geringere Tiefenqualität führt zu Disparitätsbildern mit einer geringeren Auflösung (*Qualität*, Abschnitt 6.2.4).

`DepthDispRange`

- Typ: Integer, Wertebereich: 32–512 Pixel
- Voreinstellung: 256 Pixel
- Beschreibung: Maximaler Disparitätswert in Pixeln (*Disparitätsbereich*, Abschnitt 6.2.4).

DepthFill

- Typ: Integer, Wertebereich: 0–4 Pixel
- Voreinstellung: 3 Pixel
- Beschreibung: Wert in Pixeln für *Füllen* (Abschnitt 6.2.4).

DepthSeg

- Typ: Integer, Wertebereich: 0–4000 Pixel
- Voreinstellung: 200 Pixel
- Beschreibung: Wert in Pixeln für *Segmentierung* (Abschnitt 6.2.4).

DepthMedian

- Typ: Integer, Wertebereich: 1–5 Pixel
- Voreinstellung: 1 Pixel
- Beschreibung: Wert in Pixeln für den *Median*-Filter (Abschnitt 6.2.4).

DepthMinConf

- Typ: Float, Wertebereich: 0.0–1.0
- Voreinstellung: 0.0
- Beschreibung: Wert für die *Minimale Konfidenz*-Filterung (Abschnitt 6.2.4).

DepthMinDepth

- Typ: Float, Wertebereich: 0.1–100.0 m
- Voreinstellung: 0.1 m
- Beschreibung: Wert in Metern für die *Minimale Abstands*-Filterung (Abschnitt 6.2.4).

DepthMaxDepth

- Typ: Float, Wertebereich: 0.1–100.0 m
- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Abstands*-Filterung (Abschnitt 6.2.4).

DepthMaxDepthErr

- Typ: Float, Wertebereich: 0.01–100.0 m
- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Fehler*-Filterung (Abschnitt 6.2.4).

7.1.2 Verfügbare Bild-Streams

Der *rc_visard* stellt über die GenICam-Schnittstelle die folgenden fünf Bild-Streams zur Verfügung:

Name der Komponente	PixelFormat	Breite × Höhe	Beschreibung
`Intensity`	`Mono8` (monochrome Sensoren) `YCbCr411_8` (Farbsensoren)	1280 × 960	Rektifiziertes Bild der linken Kamera
`IntensityCombined`	`Mono8` (monochrome Sensoren) `YCbCr411_8` (Farbsensoren)	1280 × 1920	Rektifiziertes Bild der linken Kamera, gestapelt auf das rektifizierte Bild der rechten Kamera
`Disparity`	`Coord3D_C16`	640 × 480 320 × 240 214 × 160	Disparitätsbild in gewünschter Auflösung, d. h. `High`, `Medium`, oder `Low`
`Confidence`	`Confidence8`	wie `Disparity`	Konfidenzbild
`Error`	Sonderformat: `0x81080001`	wie `Disparity`	Fehlerbild

Jedes Bild wird mit einem Zeitstempel und dem in der oben angegeben 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.

7.1.3 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 und der Basisabstand benötigt. Diese Parameter werden als GenICam-Parameter *FocalLengthFactor* und *Baseline* übertragen. Um die Brennweite f für die angegebene Disparitätsbildauflösung zu ermitteln, muss der *FocalLengthFactor* mit der Breite des Disparitätsbildes multipliziert werden. Sind diese Werte bekannt, können die Pixel-Koordinaten und die Disparitätswerte mithilfe der in *Berechnung von Tiefenbildern und Punktwolken* (Abschnitt 6.2.2) angegebenen Gleichungen in 3D-Objektkoordination im Sensor-Koordinatensystems umgerechnet werden (siehe *Koordinatensysteme*, Abschnitt 3.7).

Angenommen, dass es sich bei d_{ik} um den 16-Bit-Disparitätswert in der Spalte i und Zeile k eines Disparitätsbildes mit w Spalten und h Zeilen handelt, lässt sich die 3D-Rekonstruktion (in Metern) wie folgt mit den GenICam-Parametern durchführen:

$$
\begin{aligned}
P_x &= \left(i - \frac{w}{2}\right) \frac{\text{Baseline}}{d_{ik} \cdot \text{Scan3dCoordinateScale}}, \\
P_y &= \left(k - \frac{h}{2}\right) \frac{\text{Baseline}}{d_{ik} \cdot \text{Scan3dCoordinateScale}}, \\
P_z &= (\text{FocalLengthFactor} \cdot w) \frac{\text{Baseline}}{d_{ik} \cdot \text{Scan3dCoordinateScale}}.
\end{aligned}
$$

Das Konfidenzbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Diese Werte müssen durch 255 geteilt werden, um die zwischen 0 und 1 liegenden Konfidenzwerte zu berechnen.

Das Fehlerbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Der Fehler e_{ik} muss mit dem im GenICam-Parameter *Scan3dCoordinateScale* angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätsfehlerwerte d_{eps} in Pixeln zu ermitteln. Der Beschreibung in *Konfidenz- und Fehlerbilder* (Abschnitt 6.2.3) zufolge

lässt sich der Tiefenfehler z_{eps} (in Metern) mit den GenICam-Parametern wie folgt berechnen:

$$z_{eps} = \frac{e_{ik} \cdot \text{Scan3dCoordinateScale} \cdot \text{FocalLengthFactor} \cdot w \cdot \text{Baseline}}{(d_{ik} \cdot \text{Scan3dCoordinateScale})^2}.$$

Für nähere Informationen zu Disparitäts-, Fehler- und Konfidenzbildern siehe *Stereo-Matching* (Abschnitt 6.2).

7.2 REST-API-Schnittstelle

Neben der *GenICam-Schnittstelle* (Abschnitt 7.1) 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 4.5) 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, der Disparitätsberechnung und der visuellen Odometrie;
- Aufrufen von Services, z. B. zum Starten und Stoppen einzelner Softwaremodule, oder zum Nutzen spezieller Funktionen, wie der Hand-Auge-Kalibrierung;
- Konfiguration von Datenströmen, die, wie in *Die rc_dynamics-Schnittstelle* (Abschnitt 7.3) beschrieben, die *dynamischen Zustandsschätzungen* (Abschnitt 6.3.2) des *rc_visard* bereitstellen;
- Abruf des aktuellen Systemstatus und des Status einzelner Softwaremodule; sowie
- Aktualisierung der Firmware des *rc_visard* oder seiner Lizenz.

Hinweis: 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 der visuellen Odometrie.

7.2.1 Allgemeine Struktur der Programmierschnittstelle (API)

Der allgemeine **Einstiegspunkt** zur Programmierschnittstelle (API) des *rc_visard* ist `http://<rcvisard>/api/` wobei `<rcvisard>` entweder die IP-Adresse des Geräts ist oder sein dem jeweiligen DHCP-Server bekannter Host-Name (siehe *Netzwerkkonfiguration*, Abschnitt 4.3). Greift der Benutzer über einen Webbrowser auf diese Adresse zu, kann er die Programmierschnittstelle während der Laufzeit mithilfe der *Swagger UI* (Abschnitt 7.2.4) erkunden und testen.

Für die eigentlichen HTTP-Anfragen wird dem Einstiegspunkt der Programmierschnittstelle die **aktuelle Version der Schnittstelle als Postfix angehangen**, d. h. `http://<rcvisard>/api/v1`. 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 7.2.2) 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://<rcvisard>/api/v1/nodes/rc_stereomatching'
```

- Abruf einiger Parameterwerte eines Moduls über eine Abfragezeichenfolge

```
curl -X GET 'http://<rcvisard>/api/v1/nodes/rc_stereomatching/parameters?name=minconf&
↪name=maxdepth'
```

- Konfiguration eines neuen Datenstroms, wobei die Zielparameter als Formulardaten übertragen werden

```
curl -X PUT --header 'Content-Type: application/x-www-form-urlencoded' -d 'destination=10.0.
↪1.14%3A30000' 'http://<rcvisard>/api/v1/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://<rcvisard>/api/v1/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"
}
```

Hinweis: 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 7.2.2) des *rc_visard* und einzelnen Parameter und Services jedes *Softwaremoduls* (Abschnitt 6).

7.2.2 Verfügbare Ressourcen und Anfragen

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

- `/nodes`: Zugriff auf die *Softwaremodule* (Abschnitt 6) des *rc_visard* mit ihren jeweiligen Laufzeitzuständen, Parametern und verfügbaren Services.
- `/datastreams`: Zugriff auf die und Verwaltung der Datenströme der rc_dynamics-Schnittstelle (siehe *Die rc_dynamics-Schnittstelle*, Abschnitt 7.3) des *rc_visard*.
- `/logs`: Zugriff auf die im *rc_visard* hinterlegten Logdateien.
- `/system`: Zugriff auf den Systemzustand und Verwaltung der Lizenzen sowie der Firmware-Updates.

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 Softwaremodule der REST-API mit ihren jeweiligen Services und Parametern auflisten:

```
curl -X GET http://<rcvisard>/api/v1/nodes
```

Informationen zu einem bestimmten Modul (z. B. `rc_stereocamera`) lassen sich mit folgendem Befehl abrufen:

```
curl -X GET http://<rcvisard>/api/v1/nodes/rc_stereocamera
```

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_stereocamera`-Moduls lassen sich beispielsweise wie folgt abrufen:

```
curl -X GET http://<rcvisard>/api/v1/nodes/rc_stereocamera/status
```

Hinweis: Die zurückgegebenen **Statuswerte** sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Hinweis: 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://<rcvisard>/api/v1/nodes/rc_stereomatching/parameters
```

Der `median`-Parameter dieses Moduls könnte wie folgt auf den Wert 3 gesetzt werden:

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "value": 3 }' http://<rcvisard>/
↪api/v1/nodes/rc_stereomatching/parameters/median
```

Hinweis: Laufzeitparameter sind modulspezifisch und werden in dem jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Hinweis: Die meisten Parameter, die die Module über die REST-API anbieten, lassen sich auch über die benutzerfreundliche *Web GUI* (Abschnitt 4.5) des *rc_visard* erkunden und austesten.

Hinweis: Einige der Parameter, die über die REST-API des *rc_visard* bereitgestellt werden, sind auch über die *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 7.1) zugänglich. Die Einstellung dieser Parameter über die REST-API ist verboten, solange ein GenICam-Client verbunden ist.

Zudem bietet jedes Modul, das Laufzeitparameter bereitstellt, auch Services, um die aktuellen Parametereinstellungen zu speichern oder 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 Speichern und Wiederherstellen von Parametern oder auch das Starten und Stoppen von Modulen. Die Services der Zustandsschätzung (siehe *Stereo-INS*, Abschnitt 6.5) lassen sich beispielsweise wie folgt aufrufen:

```
curl -X GET http://<rcvisard>/api/v1/nodes/rc_stereo_ins/services
```

Um einen Service eines Moduls aufzurufen, wird eine `PUT`-Anfrage mit servicespezifischen Argumenten für die jeweilige Ressource gestellt (siehe das `"args"`-Feld der *Service-Datenmodell*, Abschnitt 7.2.3). Das Dynamik-Modul lässt sich beispielsweise wie folgt einschalten:

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "args": {}  }' http://<rcvisard>
↪/api/v1/nodes/rc_dynamics/services/start
```

Hinweis: Die Services und zugehörigen Argumente sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Die folgende Liste enthält alle REST-API-Anfragen zum Status des Moduls und seinen Parametern und Services:

`GET /nodes`

Abruf einer Liste aller verfügbaren Module.

Musteranfrage

```
GET /api/v1/nodes HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

[
  {
    "name": "rc_stereocalib",
    "parameters": [
      "grid_width",
      "grid_height",
      "snap"
    ],
    "services": [
      "save_parameters",
```

```
      "reset_defaults",
      "change_state"
    ],
    "status": "stale"
  },
  {
    "name": "rc_stereocamera",
    "parameters": [
      "fps",
      "exp_auto",
      "exp_value",
      "exp_max"
    ],
    "services": [
      "save_parameters",
      "reset_defaults"
    ],
    "status": "running"
  },
  {
    "name": "rc_hand_eye_calibration",
    "parameters": [
      "grid_width",
      "grid_height",
      "robot_mounted"
    ],
    "services": [
      "save_parameters",
      "reset_defaults",
      "set_pose",
      "reset",
      "save",
      "calibrate",
      "get_calibration"
    ],
    "status": "stale"
  },
  {
    "name": "rc_stereo_ins",
    "parameters": [],
    "services": [],
    "status": "stale"
  },
  {
    "name": "rc_stereomatching",
    "parameters": [
      "force_on",
      "quality",
      "disprange",
      "seg",
      "median",
      "fill",
      "minconf",
      "mindepth",
      "maxdepth",
      "maxdeptherr"
    ],
    "services": [
      "save_parameters",
      "reset_defaults"
    ],
    "status": "running"
  },
```

```
  {
    "name": "rc_stereovisodo",
    "parameters": [
      "disprange",
      "nkey",
      "ncorner",
      "nfeature"
    ],
    "services": [
      "save_parameters",
      "reset_defaults"
    ],
    "status": "stale"
  }
]
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.2.3)

GET /nodes/{node}

Abruf von Informationen zu einem einzelnen Modul.

Musteranfrage

```
GET /api/v1/nodes/<node> HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

{
  "name": "rc_stereocamera",
  "parameters": [
    "fps",
    "exp_auto",
    "exp_value",
    "exp_max"
  ],
  "services": [
    "save_parameters",
    "reset_defaults"
  ],
  "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 7.2.3)

`GET /nodes/{node}/parameters`

Abruf von Parametern eines Moduls.

Musteranfrage

```
GET /api/v1/nodes/<node>/parameters?name=<name> HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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": 1,
    "description": "Switching between auto and manual exposure",
    "max": 1,
    "min": 0,
    "name": "exp_auto",
    "type": "bool",
    "value": 1
  },
  {
    "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

- **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: Parameter-Array)*

- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 7.2.3)

PUT /nodes/{node}/parameters

Aktualisierung mehrerer Parameter.

Musteranfrage

```
PUT /api/v1/nodes/<node>/parameters HTTP/1.1
Host: <rcvisard>
Accept: application/json

[
  {
    "name": "string",
    "value": {}
  }
]
```

Musterantwort

```
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": 1,
    "description": "Switching between auto and manual exposure",
    "max": 1,
    "min": 0,
    "name": "exp_auto",
    "type": "bool",
    "value": 0
  },
  {
    "default": 0.005,
    "description": "Manual exposure time in s if exp_auto is false",
    "max": 0.018,
    "min": 6.6e-05,
    "name": "exp_value",
    "type": "float64",
    "value": 0.005
  }
]
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

JSON-Objekt-Array zur Anfrage

- **parameters** (*Parameter*) – Liste von Parametern *(obligatorisch)*

Anfrage-Header

- Accept – application/json

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter-Array)*
- 404 Not Found – Modul nicht gefunden
- 403 Forbidden – Aktualisierung des Parameters verboten, z. B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist.

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 7.2.3)

`GET /nodes/{node}/parameters/{param}`

Abruf eines bestimmten Parameters eines Moduls.

Musteranfrage

```
GET /api/v1/nodes/<node>/parameters/<param> HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

{
  "default": "H",
  "description": "Quality, i.e. H, M or L",
  "max": "",
  "min": "",
  "name": "quality",
  "type": "string",
  "value": "H"
}
```

Parameter

- **`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 7.2.3)

`PUT /nodes/{node}/parameters/{param}`

Aktualisierung eines bestimmten Parameters eines Moduls.

Musteranfrage

```
PUT /api/v1/nodes/<node>/parameters/<param> HTTP/1.1
Host: <rcvisard>
Accept: application/json

{
  "name": "string",
  "value": {}
}
```

Musterantwort

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

{
  "default": "H",
  "description": "Quality, i.e. H, M or L",
  "max": "",
  "min": "",
  "name": "quality",
  "type": "string",
  "value": "M"
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*
- **param** (*string*) – Name des Parameters *(obligatorisch)*

JSON-Objekt zur Anfrage

- **parameter** (*Parameter*) – 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)*
- 404 Not Found – Modul oder Parameter nicht gefunden
- 403 Forbidden – Aktualisierung des Parameters verboten, z. B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist.

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 7.2.3)

GET /nodes/{node}/services

Abruf von Beschreibungen aller von einem Modul angebotenen Services.

Musteranfrage

```
GET /api/v1/nodes/<node>/services HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

[
  {
    "args": {},
    "description": "Restarts the component.",
    "name": "restart",
    "response": {
      "accepted": "bool",
      "enteredState": "uint8"
    }
  },
  {
    "args": {},
    "description": "Starts the component.",
    "name": "start",
    "response": {
      "accepted": "bool",
      "enteredState": "uint8"
    }
  },
  {
    "args": {},
    "description": "Stops the component.",
    "name": "stop",
    "response": {
      "accepted": "bool",
      "enteredState": "uint8"
    }
  }
]
```

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 7.2.3)

GET /nodes/{node}/services/{service}

Abruf der Beschreibung eines modulspezifischen Services.

Musteranfrage

```
GET /api/v1/nodes/<node>/services/<service> HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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

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

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

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/v1/nodes/<node>/services/<service> HTTP/1.1
Host: <rcvisard>
Accept: application/json

{
  "args": {}
}
```

Musterantwort

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

{
```

```
  "name": "set_pose",
  "response": {
    "message": "Grid detected, pose stored.",
    "status": 1,
    "success": true
  }
}
```

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 – Erfolgreiche Verarbeitung *(Rückgabe: Funktion)*
- 404 Not Found – Modul oder Service nicht gefunden
- 403 Forbidden – Service-Aufruf verboten, z. B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist.

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

GET /nodes/{node}/status

Abruf des Status eines Moduls.

Musteranfrage

```
GET /api/v1/nodes/<node>/status HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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

- **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 7.2.3)

Datenströme

Über die folgenden Ressourcen und Anfragen ist es möglich, auf die Streams der *Die rc_dynamics-Schnittstelle* (Abschnitt 7.3) zuzugreifen und diese zu konfigurieren. Mit diesen REST-API-Anfragen ist es möglich,

- die verfügbaren und laufenden Datenströme anzuzeigen, z. B.

```
curl -X GET http://<rcvisard>/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://<rcvisard>/api/v1/datastreams/pose
```

- Datenströme zu stoppen, z. B.

```
curl -X DELETE http://<rcvisard>/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/v1/datastreams HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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 rc_visard (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: Stream-Array)*

Referenzierte Datenmodelle

- *Stream* (Abschnitt 7.2.3)

`GET /datastreams/{stream}`

Abruf der Datenstrom-Konfiguration.

Musteranfrage

```
GET /api/v1/datastreams/<stream> HTTP/1.1
Host: <rcvisard>
```

Musterantwort


```
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 Streams *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *Stream* (Abschnitt 7.2.3)

PUT /datastreams/{stream}

Aktualisierung einer Datenstrom-Konfiguration.

Musteranfrage

```
PUT /api/v1/datastreams/<stream> HTTP/1.1
Host: <rcvisard>
Accept: application/x-www-form-urlencoded
```

Musterantwort

```
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 Streams *(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: Stream)*
- 404 Not Found – Datenstrom nicht gefunden

Referenzierte Datenmodelle

- *Stream* (Abschnitt 7.2.3)

DELETE /datastreams/{stream}

Löschen eines Ziels aus der Datenstrom-Konfiguration.

Musteranfrage

```
DELETE /api/v1/datastreams/<stream>?destination=<destination> HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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 Streams *(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: Stream)*
- 404 Not Found – Datenstrom nicht gefunden

Referenzierte Datenmodelle

- *Stream* (Abschnitt 7.2.3)

System und Logs

Die folgenden Ressourcen und Anfragen sind für die System-Level-API des *rc_visard* verfügbar. Sie ermöglichen Folgendes:

- Zugriff auf Logdateien (systemweit oder modulspezifisch);
- Abruf von Informationen zum Gerät und zur Laufzeitstatistik, wie Datum, MAC-Adresse, Uhrzeitsynchronisierungsstatus und verfügbare Ressourcen;
- Verwaltung installierter Softwarelizenzen; und
- Aktualisierung des Firmware-Images des *rc_visard*.

GET /logs

Abruf einer Liste aller verfügbaren Logdateien.

Musteranfrage

```
GET /api/v1/logs HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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 7.2.3)

`GET /logs/{log}`

Abruf einer Logdatei: Die Art des Inhalts der Antwort richtet sich nach dem „format“-Parameter.

Musteranfrage

```
GET /api/v1/logs/<log>?format=<format>&limit=<limit> HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

{
  "date": 1503060035.2115774,
  "log": [
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Running rc_stereo_ins version 2.4.0",
      "timestamp": 1503060034.083
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Starting up communication interfaces",
      "timestamp": 1503060034.085
```

```
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Autostart disabled",
      "timestamp": 1503060034.098
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Initializing realtime communication",
      "timestamp": 1503060034.209
    },
    {
      "component": "rc_stereo_ins",
      "level": "INFO",
      "message": "Startet state machine in state IDLE",
      "timestamp": 1503060034.383
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "Init stereovisodo ...",
      "timestamp": 1503060034.814
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Using standard VO",
      "timestamp": 1503060034.913
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Playback mode: false",
      "timestamp": 1503060035.132
    },
    {
      "component": "rc_stereovisodo",
      "level": "INFO",
      "message": "rc_stereovisodo: Ready",
      "timestamp": 1503060035.212
    }
  ],
  "name": "dynamics.log",
  "size": 695
}
```


Parameter

- **log** (*string*) – Name der Logdatei *(obligatorisch)*

Anfrageparameter

- **format** (*string*) – Rückgabe des Logs im JSON- oder Rohdatenformat (mögliche Werte: `json` oder `raw`; Voreinstellung: `json`) *(optional)*
- **limit** (*integer*) – Beschränkung auf die letzten x Zeilen im JSON-Format (Voreinstellung: `100`) *(optional)*

Antwort-Headers

- Content-Type – text/plain application/json

Statuscodes

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

Referenzierte Datenmodelle

- *Log* (Abschnitt 7.2.3)

`GET /system`

Abruf von Systeminformationen zum *rc_visard*.

Musteranfrage

```
GET /api/v1/system HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

{
  "firmware": {
    "active_image": {
      "image_version": "rc_visard_v0.6.3"
    },
    "fallback_booted": true,
    "inactive_image": {
      "image_version": "rc_visard_v0.6.1"
    },
    "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": {
    "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 7.2.3)

GET /system/license
Abruf von Informationen zu den auf dem *rc_visard* installierten Lizenzen.

Musteranfrage

```
GET /api/v1/system/license HTTP/1.1
Host: <rcvisard>
```

Musterantwort

```
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 7.2.3)

POST /system/license
Aktualisierung der auf dem *rc_visard* installierten Lizenz mithilfe einer Lizenzdatei.

Musteranfrage

```
POST /api/v1/system/license HTTP/1.1
Host: <rcvisard>
Accept: multipart/form-data
```

Formularparameter

- **file** – Lizenzdatei *(obligatorisch)*

Anfrage-Header

- Accept – Multipart/Formulardaten

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – Keine gültige Lizenz

PUT /system/reboot
Neustart des *rc_visard*.

Musteranfrage

```
PUT /api/v1/system/reboot HTTP/1.1
Host: <rcvisard>
```

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/rollback

Abruf von Informationen zu Firmware/System-Images, die aktuell auf dem *rc_visard* aktiv oder inaktiv sind.

Musteranfrage

```
GET /api/v1/system/rollback HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

{
  "active_image": {
    "image_version": "rc_visard_v0.6.1"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "rc_visard_v0.6.0"
  },
  "next_boot_image": "active_image"
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 7.2.3)

PUT /system/rollback

Rollback auf vorherige Firmware-Version (inaktives System-Image).

Musteranfrage

```
PUT /api/v1/system/rollback HTTP/1.1
Host: <rcvisard>
```

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 500 Internal Server Error – Interner Fehler
- 400 Bad Request – Bereits auf die Verwendung der inaktiven Partition beim nächsten Boot-Vorgang gesetzt.

GET /system/update

Abruf von Informationen zu Firmware/System-Images, die aktuell auf dem *rc_visard* aktiv oder inaktiv sind.

Musteranfrage

```
GET /api/v1/system/update HTTP/1.1
Host: <rcvisard>
```

Musterantwort

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

{
  "active_image": {
    "image_version": "rc_visard_v0.6.1"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "rc_visard_v0.6.0"
  },
  "next_boot_image": "active_image"
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 7.2.3)

`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/v1/system/update HTTP/1.1
Host: <rcvisard>
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

7.2.3 Datentyp-Definitionen

Die REST-API definiert folgende Datenmodelle, die verwendet werden, um auf die *verfügbaren Ressourcen* (Abschnitt 7.2.2) 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.

LicenseComponents: Liste der Lizenzstatus-Angaben der einzelnen Sensor-Funktionen: Der zugehörige Statusindikator ist auf TRUE gesetzt, wenn die entsprechende Funktionalität mit einer installierten Softwarelizenz entsperrt wird.

Ein Objekt des Typs LicenseComponents besitzt folgende Eigenschaften:

- **calibration** (boolean): Funktionalität zur Kamerakalibrierung.
- **fusion** (boolean): Funktionalität zur Stereo-INS/Datenfusion.
- **hand_eye_calibration** (boolean): Funktionalität zur Hand-Auge-Kalibrierung.
- **rectification** (boolean): Funktionalität zur Bildrektifizierung.
- **self_calibration** (boolean): Funktionalität zur Selbstkalibrierung der Kamera.
- **slam** (boolean): SLAM-Funktionalität.
- **stereo** (boolean): Stereo-Matching-Funktionalität.
- **svo** (boolean): visuelle Odometrie-Funktionalität.

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.

LicenseInfo: Informationen zur aktuell auf dem *rc_visard* angewandten Softwarelizenz.

Ein Objekt des Typs LicenseInfo besitzt folgende Eigenschaften:

- **components** (*LicenseComponents*): siehe Beschreibung von *LicenseComponents*.
- **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
  },
  "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*

NodeInfo: Beschreibung eines auf dem *rc_visard* 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`, `stale` 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}*

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`, `stale` 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*

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 /nodes/{node}/parameters*
- *PUT /nodes/{node}/parameters*
- *GET /nodes/{node}/parameters/{param}*
- *PUT /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}*

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 *rc_visard*.

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.
- **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 *rc_visard*.
- **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",
  "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*

7.2.4 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://<rcvisard>/api/` oder auf `http://<rcvisard>/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 7.2.2). Auf dieser vereinfachten Benutzeroberfläche lassen sich alle Funktionen erkunden und austesten.

Hinweis: Der Benutzer muss bedenken, dass die Swagger UI des *rc_visard*, auch wenn sie zur Erprobung der REST-API bestimmt ist, ein 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 Anfrage des Typs `PUT`, `POST` und `DELETE`.

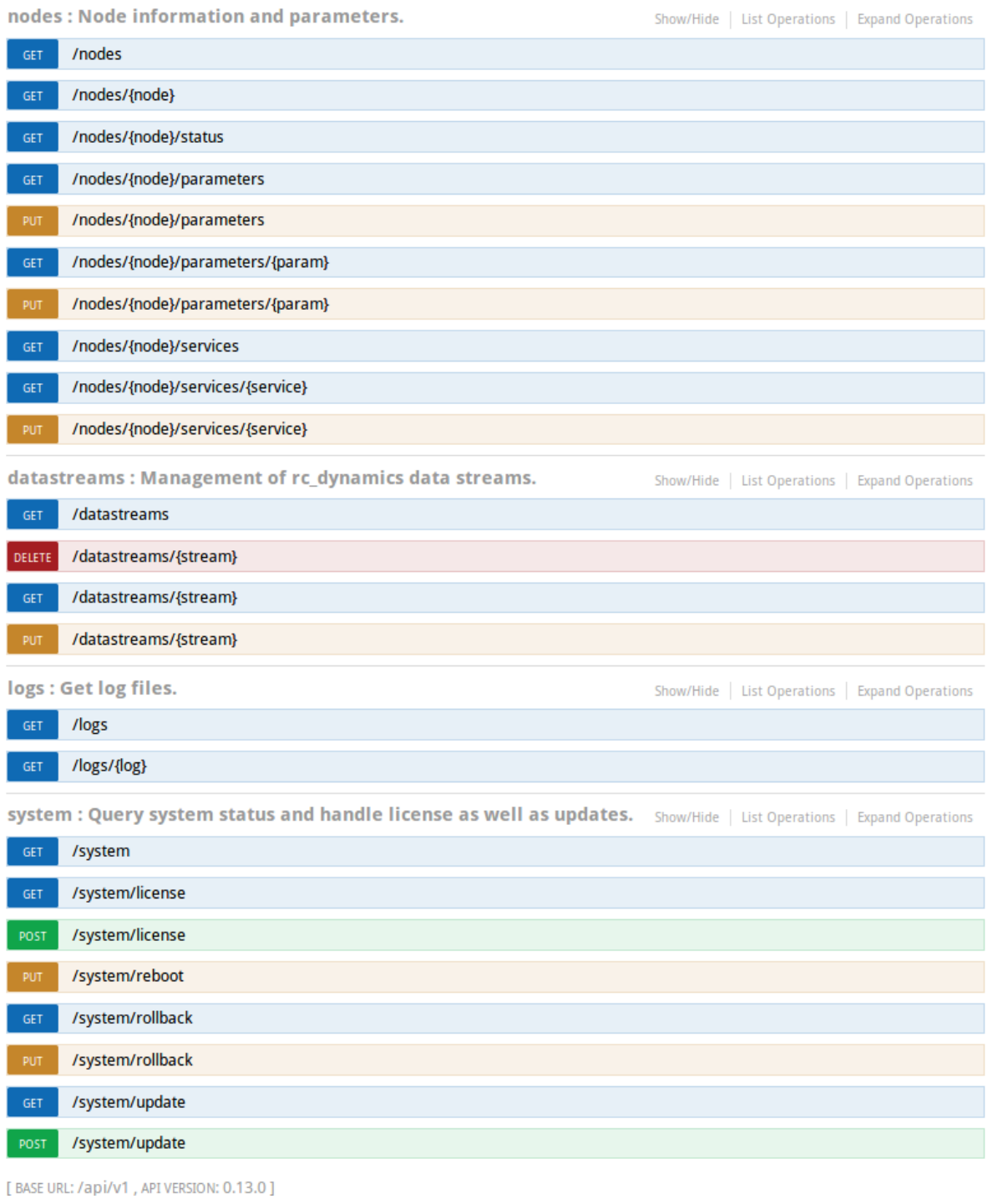


Abb. 7.1: Startansicht der Swagger UI des *rc_visard*, bei der die Ressourcen und Anfragen in `nodes`, `datastreams`, `logs` und `system` gruppiert sind.

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 der erforderliche Parameter (`node`-Name) ausgefüllt und anschließend die Schaltfläche *Try it out!* betätigt wird. Daraufhin zeigt die Swagger UI unter anderem den `curl`-Befehl an, der bei Auslösung der Anfrage ausgeführt wurde, sowie den Antworttext, in dem der aktuelle Status des angefragten Moduls in einer Zeichenfolge im JSON-Format enthalten ist.

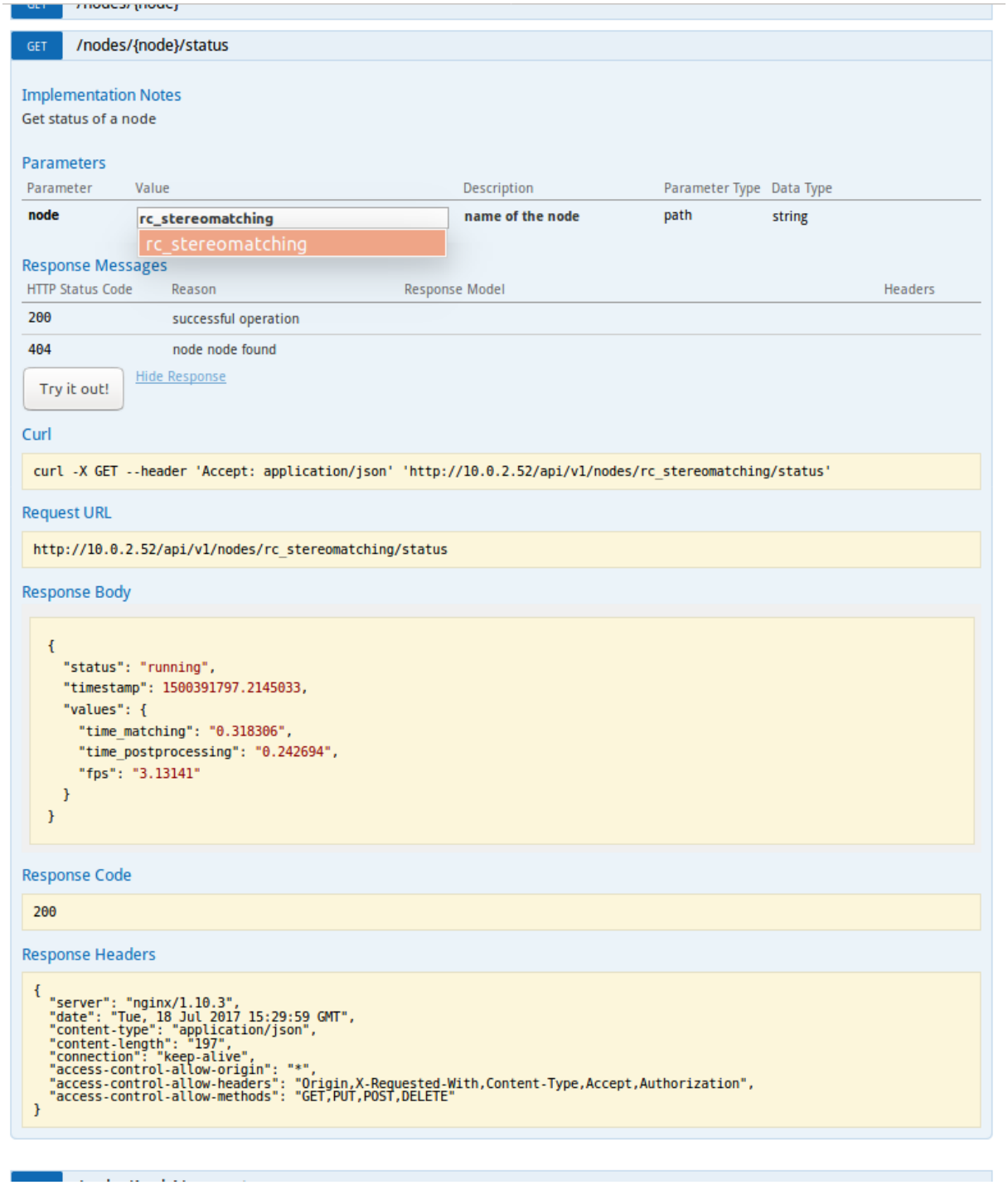


Abb. 7.2: Ergebnis nach Abfrage des Status des `rc_stereomatching`-Moduls

Einige Aktionen, wie das Setzen von Parametern oder der Aufruf von Services, bedürfen komplexerer Parameter als eine HTTP-Anfrage. Die Swagger UI erlaubt es Entwicklern, die für diese Aktionen benötigten Attribute, wie im nächsten Beispiel gezeigt, während der Laufzeit zu erkunden. In der folgenden Abbildung werden die Attribute, die für den `set_pose`-Service des `rc_hand_eye_calibration`-Moduls benötigt werden, erkundet, indem eine `GET`-Anfrage zu dieser Ressource durchgeführt wird. Die Antwort enthält eine vollständige Beschreibung des angebotenen Services, einschließlich aller erforderlichen Argumente mit ihren Namen und Typen in einer Zeichenfolge im JSON-Format.

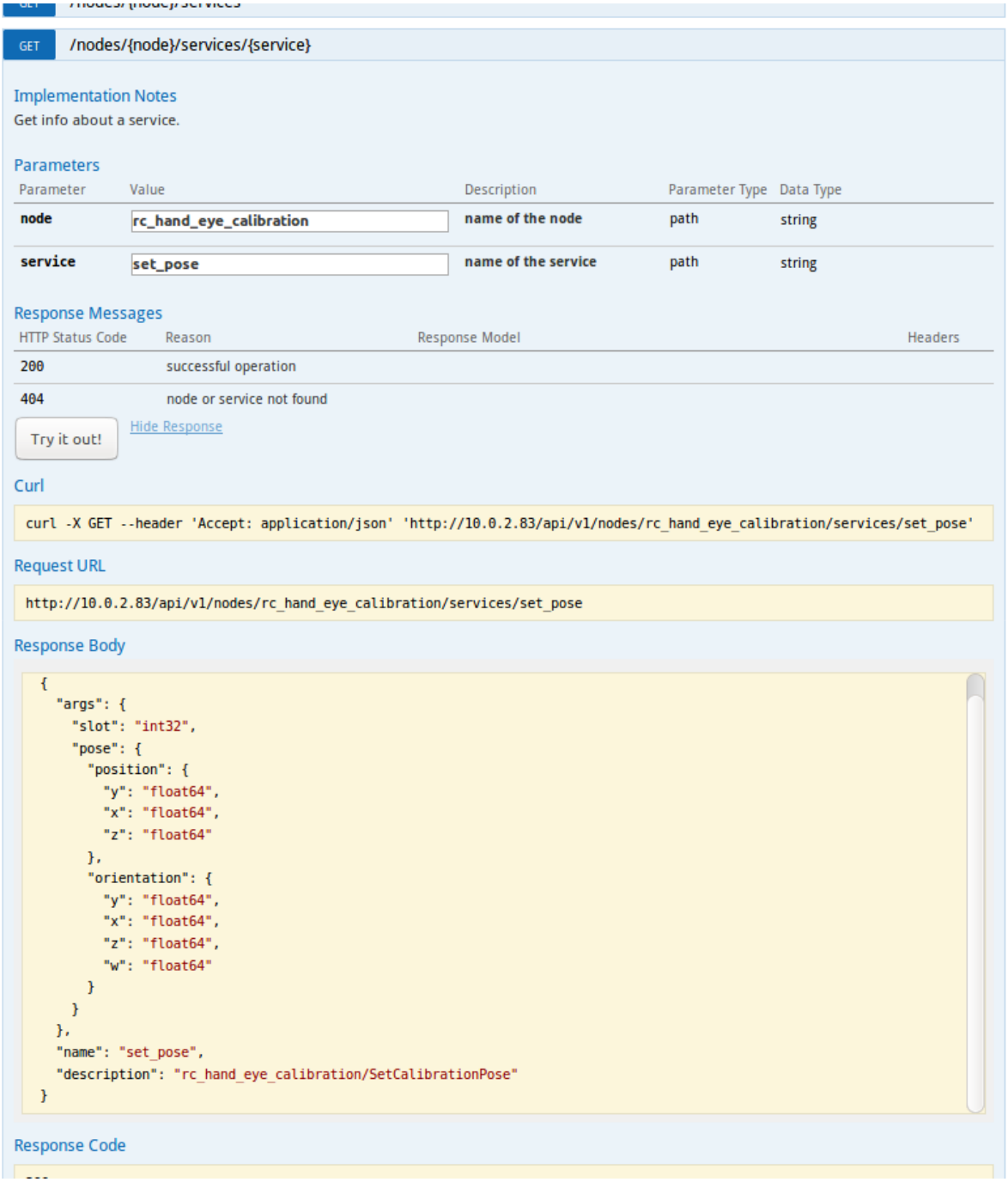


Abb. 7.3: Ergebnis der `GET`-Anfrage zum `set_pose`-Service zeigt die für diesen Service benötigten Argumente

Der Benutzer kann diesen vorformatierten JSON als Muster für die Argumente nutzen, um damit den Service tatsächlich aufzurufen:

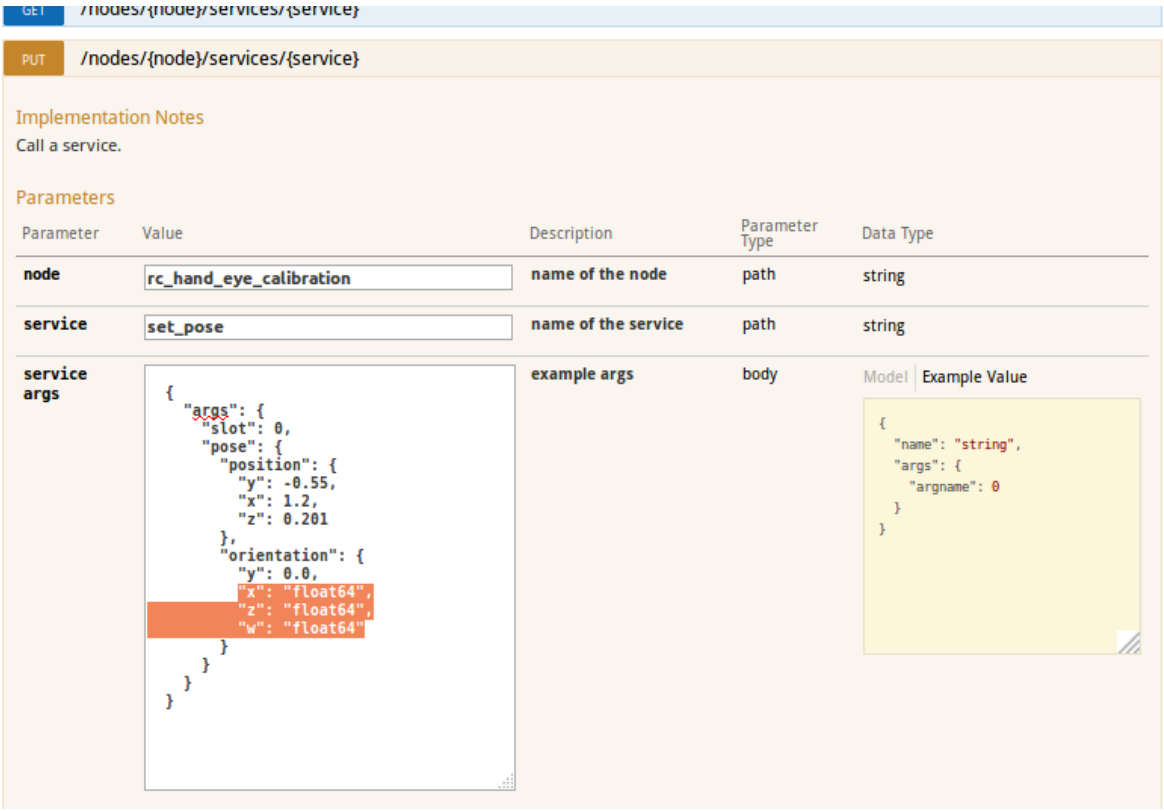


Abb. 7.4: Ausfüllen der Argumente des `set_pose`-Services

7.3 Die rc_dynamics-Schnittstelle

Die rc_dynamics-Schnittstelle bietet über Echtzeit-Datenströme kontinuierlichen Zugang zu verschiedenen *Dynamik-Zustandsschätzungen* (Abschnitt 6.3.2). Die Schnittstelle ermöglicht es, Zustandsschätzungen aller Art so zu konfigurieren, dass sie an einen beliebigen Host im Netzwerk gestreamt werden. Das dafür eingesetzte *Datenstromprotokoll* unterstützt alle gängigen Betriebssysteme und Programmiersprachen.

7.3.1 Starten/Stoppen der Dynamik-Zustandsschätzungen

Die Dynamik-Zustandsschätzungen des *rc_visard* sind nur verfügbar, wenn die zugehörige Komponente, z. B. das *Dynamik-Modul* (Abschnitt 6.3), eingeschaltet ist. Dies lässt sich entweder über die Web GUI – eine entsprechende Schaltfläche ist sowohl auf der Registerkarte *Überblick* als auch auf der Registerkarte *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://<rcvisard>/api/v1/nodes/rc_
↪dynamics/services/start'
```

Hinweis: Um Rechenressourcen zu sparen, wird empfohlen, die Dynamik-Zustandsschätzungen zu stoppen, wenn sie nicht länger benötigt werden.

7.3.2 Konfiguration von Datenströmen

Verfügbare Datenströme, d. h. Dynamik-Zustandsschätzungen, lassen sich über die *REST-API* (Abschnitt 7.2.2) des *rc_visard* auflisten und konfigurieren. So lässt sich beispielsweise mit dem Befehl `GET /datastreams` ei-

ne Liste aller verfügbaren Datenströme abrufen. Für eine detaillierte Beschreibung der im Folgenden benannten Datenströme siehe *Verfügbare Zustandsschätzungen* (Abschnitt 6.3.2).

Tab. 7.1: Datenströme, die über die rc_dynamics-Schnittstelle verfügbar sind

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

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://<rcvisard>/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 7.3.3) zufolge empfangen, deserialisieren und verarbeiten.

3. **Stoppen eines Datenstroms über die REST-API:** Der folgende Beispiel-`curl`-Befehl löst eine *`DELETE /datastreams/{stream}`*-Anfrage aus, mit der die zuvor beantragte Übertragung eines Datenstroms des Typs `pose_rt` mit dem Ziel `10.0.1.14:30000` gelöscht, d. h. gestoppt, wird:

   ```
   curl -X DELETE --header 'Accept: application/json' 'http://<rcvisard>/api/v1/
   ↪datastreams/pose_rt?destination=10.0.1.14:30000'
   ```

   Sollen alle Ziele für einen Datenstrom entfernt werden, ist lediglich der Zielparameter wegzulassen.

Achtung: Datenströme werden nicht automatisch gelöscht. Dies bedeutet, dass der *rc_visard* weiterhin Daten sendet, auch wenn der Client getrennt wird oder die gesandten Daten nicht länger verwendet. Es wird daher dringend empfohlen, Datenströme über die REST-API zu stoppen, wenn sie nicht länger verwendet werden. Anderenfalls können die akkumulierten Streaming-Prozesse einen beträchtlichen Anteil der Rechenressourcen des *rc_visard* in Beschlag nehmen und so die Gesamtleistung des Systems herabsetzen.

7.3.3 Datenstromprotokoll

Sobald ein Datenstrom eingerichtet ist, werden die Daten über das folgende Protokoll kontinuierlich an den angegebenen Client-Host und Port (`destination`) gesandt:

Netzwerkprotokoll: Derzeit wird ausschließlich das Netzwerkprotokoll *UDP* unterstützt, was bedeutet, dass Daten als UDP-Datagramme versandt werden.

Datenserialisierung: Die gesandten Daten werden über Google protocol buffers serialisiert. Dabei werden folgende Nachrichtentyp-Definitionen verwendet.

- Die *Kameraposen-Datenströme* und *Echtzeit-Datenströme der Kamerapose* (Abschnitt 6.3.2) 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
}
```

- Der *Echtzeit-Dynamik-Datenstrom* (Abschnitt 6.3.2) wird mithilfe des Nachrichtentyps `Dynamics` serialisiert:

```
message Dynamics
{
  optional Time timestamp                    = 1; // Time when the data was␣
↪captured
  optional Pose pose                         = 2;
  optional string pose_frame                 = 3; // Name of the frame that␣
↪the pose is given in
  optional Vector3d linear_velocity          = 4; // Linear velocity in m/s
  optional string linear_velocity_frame      = 5; // Name of the frame that␣
↪the linear_velocity is given in
  optional Vector3d angular_velocity         = 6; // Angular velocity in rad/s
  optional string angular_velocity_frame     = 7; // Name of the frame that␣
↪the angular_velocity is given in
  optional Vector3d linear_acceleration      = 8; // Gravity compensated␣
↪linear acceleration in m/s^2
  optional string linear_acceleration_frame  = 9; // Name of the frame that␣
↪the acceleration is given in
  repeated double covariance                 = 10 [packed=true]; // Row-major␣
↪representation of the 15x15 covariance matrix
  optional Frame cam2imu_transform           = 11; // pose of the left camera␣
↪wrt. the IMU frame
  optional bool possible_jump                = 12; // True if there possibly␣
↪was a jump in the pose estimation
}
```

- Der *IMU-Datenstrom* (Abschnitt 6.3.2) wird mithilfe des Nachrichtentyps `Imu` serialisiert:

```
message Imu
{
  optional Time timestamp                    = 1; // Time when the data was␣
↪captured
  optional Vector3d linear_acceleration      = 2; // Linear acceleration in m/
↪s^2 measured by the IMU
  optional Vector3d angular_velocity         = 3; // Angular velocity in rad/
↪s measured by the IMU
}
```

- Die enthaltenen Nachrichtentypen `PoseStamped`, `Pose`, `Time`, `Quaternion` und `Vector3D` werden wie folgt definiert:

```
message PoseStamped
{
  optional Time timestamp  = 1; // Time when the data was captured
  optional Pose pose       = 2;
```

```
}
```

```
message Pose
{
  optional Vector3d position      = 1; // Position in meters
  optional Quaternion orientation = 2; // Orientation as unit quaternion
  repeated double covariance      = 3 [packed=true]; // Row-major␣
↪representation of the 6x6 covariance matrix (x, y, z, rotation about X axis,␣
↪rotation about Y axis, rotation about Z axis)
}
```

```
message Time
{
  /// \brief Seconds
  optional int64 sec = 1;

  /// \brief Nanoseconds
  optional int32 nsec = 2;
}
```

```
message Quaternion
{
  optional double x = 2;
  optional double y = 3;
  optional double z = 4;
  optional double w = 5;
}
```

```
message Vector3d
{
  optional double x = 1;
  optional double y = 2;
  optional double z = 3;
}
```

7.3.4 Die rc_dynamics-Programmierschnittstelle

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

8 Wartung

Achtung: 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 6.7).

8.3 Aktualisierung der Firmware

Angaben zur aktuellen Firmware-Version sind auf der Registerkarte *System* in der Zeile *Systeminformationen* in der *Web GUI* (Abschnitt 4.5) angegeben. Diese Informationen lassen sich mithilfe einer `GET /system`-Anfrage über die die *REST-API-Schnittstelle* (Abschnitt 7.2) des *rc_visard* abrufen. Die Aktualisierung der Firmware kann entweder über die Web GUI oder über die REST-API vorgenommen werden.

Achtung: Nach einem Firmware-Update werden alle konfigurierten Parameter der Softwaremodule auf die Werkseinstellungen zurückgesetzt. Bevor das Update vorgenommen wird, sollten daher alle Einstellungen (über die *REST-API-Schnittstelle*, Abschnitt 7.2) abgefragt und in der Anwendung oder auf dem Client-PC gesichert werden.

Folgende Einstellungen sind davon ausgeschlossen und bleiben auch nach einem Firmware-Update erhalten:

- die Netzwerkkonfiguration des *rc_visard*, samt der ggf. vergebenen festen IP-Adresse und des benutzerdefinierten Gerätenamens;
- das letzte Ergebnis der *Hand-Auge-Kalibrierung* (Abschnitt 6.8), 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 6.7), 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-Artifact-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 (http://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 die Zeile *Software-Update* auf der Registerkarte *System* ausgewählt und die Schaltfläche *Update hochladen* betätigt werden (siehe Abb. 8.1). Nachdem die gewünschte Update-Image-Datei (Dateierweiterung. `.mender`) aus dem lokalen Dateisystem ausgewählt und geöffnet wurde, startet das Update.

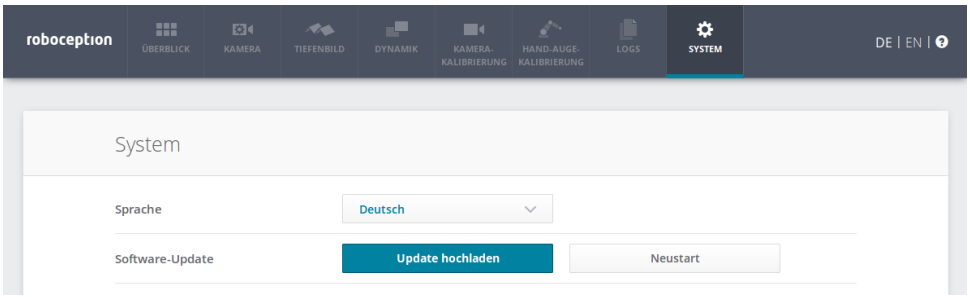


Abb. 8.1: Web GUI-Registerkarte *System*

Hinweis: Je nach Netzwerkarchitektur und Konfiguration kann das Hochladen mehrere Minuten in Anspruch nehmen. Während das Update über die Web GUI läuft, zeigt ein Statusbalken (siehe Abb. 8.2) an, wie weit das Update bereits vorangeschritten ist.

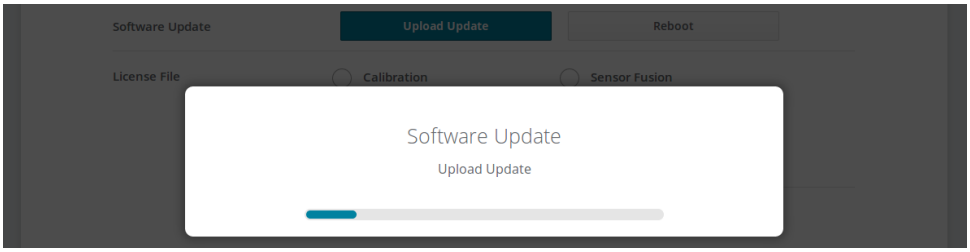


Abb. 8.2: Fortschrittsbalken für das Software-Update

Hinweis: Je nach Webbrowser kann es vorkommen, dass der in Abb. 8.2 gezeigte Statusbalken den Abschluss des Updates zu früh angibt. Es empfiehlt sich, zu warten, bis sich das in Abb. 8.3 gezeigte Kontextfenster öffnet. Insgesamt ist mit einer Update-Dauer von mindestens fünf Minuten zu rechnen.

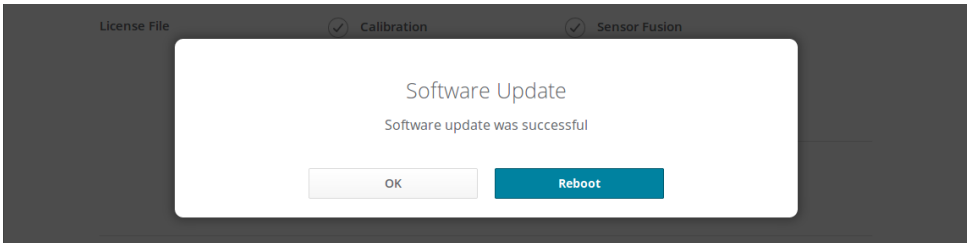


Abb. 8.3: Kontextfenster zum Neustart des Software-Updates

Achtung: 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.

Hinweis: 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 das in Abb. 8.3 gezeigte Kontextfenster, 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-Registerkarte *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 Registerkarte *System* 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.

8.4 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.

Hinweis: 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 7.2) des *rc_visard* aufgerufen werden – mithilfe der Anfrage `PUT /system/rollback`. Die Anfrage kann entweder mit einem HTTP-kompatiblen Client oder, wie in *Swagger UI* (Abschnitt 7.2.4) 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.

Achtung: Wie bei einem Firmware-Update werden alle Parameter der Softwaremodule auf die Werkseinstellungen zurückgesetzt. Bevor das Rollback ausgeführt wird, sollten daher alle Einstellungen (über die *REST-API-Schnittstelle*, Abschnitt 7.2) abgefragt und in der Anwendung oder auf dem Client-PC gesichert werden.

8.5 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 7.2) des *rc_visard* oder manuell auf der Registerkarte *System* der *Web GUI* (Abschnitt 4.5) vornehmen. Der Neustart ist abgeschlossen, wenn die LED wieder grün leuchtet.

8.6 Aktualisierung der Softwarelizenz

Lizenzen, die von Roboception zur Aktivierung zusätzlicher Funktionen erworben werden, können über die Registerkarte *System* der *Web GUI* (Abschnitt 4.5) installiert werden.

8.7 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 Registerkarte *Logs* auf der *Web GUI* (Abschnitt 4.5) 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 die Option *Alle Logs herunterladen* auf der Registerkarte *Logs* der Web GUI auszuwählen.

Die Logs sind nicht nur über die Web GUI, sondern auch über die *REST-API-Schnittstelle* (Abschnitt 7.2) 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. Es besteht aus folgenden Elementen:

- Netzwerkkabel mit gerader M12-Buchse und geradem RJ45-Stecker, Länge: 2 m oder 5 m;
- Netzteilkabel mit gerader M12-Buchse und DC-Stecker, Länge: 30 cm;
- Tischnetzteil: 24 V, 30 W.

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).

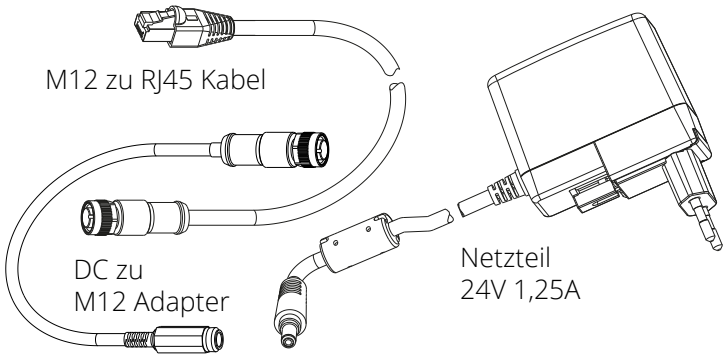


Abb. 9.1: Bestandteile des optional erhältlichen Anschlusssets

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 von Roboception 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 ein industrielles Gerät gemäß der Norm EN-55011 Klasse A klassifiziert. Um den Sensor an ein Gebäudenetz anzuschließen zu können, wird ein Netzteil gemäß EN 55011/55022 Klasse B benötigt.

Es ist Aufgabe des Kunden, ein Netzteil zu erwerben und zu installieren, das den Anforderungen der EN 61000-6-2 für die dauerhafte Installation in einem industriellen Umfeld entspricht. Ein Beispiel, das sowohl der EN 61000-6-2 als auch der EN 55011/55022 Klasse B entspricht, ist das Hutschienen-Netzteil PULS MiniLine ML60.241 (24 VDC; 2,5 A) der PULS GmbH (http://www.pulspower.com). Die Installation muss von einem qualifizierten Elektriker vorgenommen werden.

Es darf immer nur ein *rc_visard* an ein Netzteil angeschlossen werden. Die Länge der verwendeten Kabel darf 30 Meter nicht überschreiten.

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, *rc_visard* 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 Tab. 3.5 (Seite 14) 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.

Probleme mit der Zuverlässigkeit und/oder mechanische Schäden

Dies kann darauf hinweisen, dass die Umgebungsbedingungen (Vibrationen, Erschütterungen, Schwingungen und Temperatur) außerhalb der Spezifikation liegen (siehe Tab. 3.3, Seite 12).

- 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 Netzwerkkabel ordentlich mit dem *rc_visard* und dem Netzwerk verbunden ist.
- Ist kein Problem erkennbar, tauschen Sie das Ethernet-Kabel aus.

Der rc_visard 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 des *rc_visard* fehlerhaft. Siehe *Netzwerkkonfiguration* (Abschnitt 4.3).
- 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 übertragenen 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 der *rc_visard* im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verkürzen (siehe *Parameter*, Abschnitt 6.1.3) oder
- schalten Sie auf automatische Belichtung um (siehe *Parameter*, Abschnitt 6.1.3).

Kamerabild ist zu dunkel

- Wenn der *rc_visard* im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verlängern (siehe *Parameter*, Abschnitt 6.1.3) oder
- schalten Sie auf automatische Belichtung um (siehe *Parameter*, Abschnitt 6.1.3).

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 6.1.3).

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 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 (*Kontakt*, Abschnitt 11).

Kamerabild ist verschwommen

Schnelle Bewegungen können in Kombination mit langen Belichtungszeiten zu Unschärfe führen. Um Bewegungsunschärfe zu verringern,

- verringern Sie die Bewegungsgeschwindigkeit des *rc_visard*,
- verringern Sie die Bewegungsgeschwindigkeit von Objekten im Sichtfeld des *rc_visard* oder
- verkürzen Sie die Belichtungszeit der Kameras (siehe *Parameter*, Abschnitt 6.1.3).

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 Roboception (*Kontakt*, Abschnitt 11).

Bildwiederholrate ist zu niedrig

- Erhöhen Sie die Bildwiederholrate gemäß den Anweisungen in *Parameter* (Abschnitt 6.1.3).
- 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 6.2) und installieren Sie bei Bedarf einen Musterprojektor.
- Erhöhen Sie den *Disparitätsbereich* und senken Sie den *Minimalen Abstand* (Abschnitt 6.2.4).
- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 6.2.4).
- Überprüfen Sie, ob das Objekt zu nahe an den Kameras liegt. Berücksichtigen Sie die Tiefenmessbereiche der beiden *rc_visard*-Varianten (siehe Tab. 3.2, Seite 10).
- Senken Sie die *Minimale Konfidenz* (Abschnitt 6.2.4).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 6.2.4).
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 6.2.4). Disparitätsbilder mit einer gröberen 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 6.7).

Bildwiederholrate der Disparitätsbilder ist zu niedrig

- Überprüfen und erhöhen Sie die Bildwiederholrate der Kamerabilder (siehe *Parameter*, Abschnitt 6.1.3). Die Bildwiederholrate der Disparitätsbilder kann nicht größer sein als die Bildwiederholrate der Kamerabilder.
- Wählen Sie eine geringere *Qualität der Disparitätsbilder* (Abschnitt 6.2.4). Hochauflösende Disparitätsbilder sind nur mit einer Frequenz von etwa 3 Hz verfügbar. Die vollen 25 Hz lassen sich lediglich bei Disparitätsbildern mit niedriger Auflösung erreichen (siehe auch Tab. 3.1, Seite 10).
- Senken Sie den *Disparitätsbereich* und erhöhen Sie den *Minimalen Abstand* (Abschnitt 6.2.4), soweit dies für die Anwendung möglich ist.
- Senken Sie den *Median-Filter-Wert* (Abschnitt 6.2.4).

Disparitätsbild zeigt keine nahe liegenden Objekte

- Überprüfen Sie, ob das Objekt zu nahe an den Kameras liegt. Berücksichtigen Sie die Tiefenmessbereiche der beiden *rc_visard*-Varianten (siehe Tab. 3.2, Seite 10).
- Erhöhen Sie den *Disparitätsbereich* (Abschnitt 6.2.4).
- Senken Sie den *Minimalen Abstand* (Abschnitt 6.2.4).

Disparitätsbild zeigt keine weit entfernten Objekte

- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 6.2.4).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 6.2.4).
- Senken Sie die *Minimale Konfidenz* (Abschnitt 6.2.4).

Disparitätsbild rauscht zu stark

- Erhöhen Sie den *Segmentierungs-Wert* (Abschnitt 6.2.4).
- Erhöhen Sie den *Füllen-Wert* (Abschnitt 6.2.4).
- Erhöhen Sie den *Median filtering value* (Abschnitt 6.2.4).

Disparitätswerte oder resultierende Tiefenwerte sind zu ungenau

- Verringern Sie den Abstand zwischen dem *rc_visard* 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.
- Überprüfen Sie, ob sich die gewählte *rc_visard*-Variante für die Anwendung eignet (siehe Tab. 3.2, Seite 10).

Disparitätsbild ist zu glatt

- Senken Sie den *Median-Filter-Wert* (Abschnitt 6.2.4).
- Senken Sie den *Füllen-Wert* (Abschnitt 6.2.4).

Disparitätsbild zeigt keine feinen Strukturen

- Senken Sie den *Segmentierungs-Wert* (Abschnitt 6.2.4).
- Senken Sie den *Füllen-Wert* (Abschnitt 6.2.4).

10.6 Probleme mit der Zustandsschätzung

Keine Zustandsschätzungen verfügbar

- Kontrollieren Sie in der Web GUI, dass das Dynamik-Modul eingeschaltet ist (siehe *Parameter*, Abschnitt 6.4.1).
- 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 6.4.1) 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 6.4.1) an.

Posenfrequenz ist zu niedrig

- Verwenden Sie den Echtzeit-Datenstrom der Kamerapose mit einer Aktualisierungsrate im 200 Hz-Bereich. Siehe *Stereo-INS* (Abschnitt 6.5).

Verzögerung/Latenz der Posenschätzung ist zu groß

- Verwenden Sie den Echtzeit-Datenstrom der Kamerapose. Siehe *Stereo-INS* (Abschnitt 6.5).

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 7.1.1).

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 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
Telefon: +49 89 889 50 79-0

12 Anhang

12.1 Formate für Posendaten

12.1.1 XYZABC-Format

Das XYZABC-Format wird verwendet, um eine Pose mit sechs Werten auszudrücken. XYZ gibt die Positionskoordinaten in Millimetern an. ABC sind Eulersche Winkel in Grad. Die für Eulersche Winkel eingesetzte Konvention lautet ZYX, d. h. A rotiert um die Z-Achse, B rotiert um die Y-Achse und C rotiert um die X-Achse. Die Elemente der Drehmatrix lassen sich wie folgt berechnen:

$$
\begin{aligned}
r_{11} &= \cos B \cos A, \\
r_{12} &= \sin C \sin B \cos A - \cos C \sin A, \\
r_{13} &= \cos C \sin B \cos A + \sin C \sin A, \\
r_{21} &= \cos B \sin A, \\
r_{22} &= \sin C \sin B \sin A + \cos C \cos A, \\
r_{23} &= \cos C \sin B \sin A - \sin C \cos A, \\
r_{31} &= -\sin B, \\
r_{32} &= \sin C \cos B, \text{and} \\
r_{33} &= \cos C \cos B.
\end{aligned}
$$

Hinweis: Es wird davon ausgegangen, dass die trigonometrischen Funktionen $\sin$ und $\cos$ Werte in Grad akzeptieren. Das Argument muss mit dem Faktor $\frac{\pi}{180}$ multipliziert werden, wenn die Werte im Bogenmaß angegeben werden sollen.

Mithilfe dieser Werte lassen sich die Drehmatrix R und der Translationsvektor T wie folgt definieren:

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

Die Transformation lässt sich wie folgt auf einen Punkt P anwenden:

$$
P' = RP + T.
$$

12.1.2 XYZ+Quaternion-Format

Das XYZ+Quaternion-Format wird verwendet, um eine Position durch Positionskoordinaten und eine Einheitsquaternion auszudrücken. XYZ gibt die Positionskoordinaten in Metern an. Die Quaternion ist ein Vektor der Länge 1, der eine Rotation durch vier Werte definiert, d. h. $q = \begin{pmatrix} a & b & c & w \end{pmatrix}^T$ mit $||q|| = 1$. Hierfür lassen sich die Drehmatrix und der Translationsvektor wie folgt definieren:

$$
R = 2 \begin{pmatrix} \frac{1}{2} - b^2 - c^2 & ab - cw & ac + bw \\ ab + cw & \frac{1}{2} - a^2 - c^2 & bc - aw \\ ac - bw & bc + aw & \frac{1}{2} - a^2 - b^2 \end{pmatrix}, \qquad T = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.
$$

Die Transformation lässt sich wie folgt auf einen Punkt P anwenden:

$$P' = RP + T.$$

Hinweis: Im XYZ+Quaternion-Format werden die Posendaten in Metern, im XYZABC-Format in Millimetern angegeben.

HTTP Routing Table

/datastreams

GET /datastreams, 84
GET /datastreams/{stream}, 85
PUT /datastreams/{stream}, 86
DELETE /datastreams/{stream}, 86

/logs

GET /logs, 87
GET /logs/{log}, 88

/nodes

GET /nodes, 74
GET /nodes/{node}, 76
GET /nodes/{node}/parameters, 77
GET /nodes/{node}/parameters/{param}, 79
GET /nodes/{node}/services, 80
GET /nodes/{node}/services/{service}, 81
GET /nodes/{node}/status, 83
PUT /nodes/{node}/parameters, 78
PUT /nodes/{node}/parameters/{param}, 79
PUT /nodes/{node}/services/{service}, 82

/system

GET /system, 90
GET /system/license, 90
GET /system/rollback, 92
GET /system/update, 92
POST /system/license, 91
POST /system/update, 93
PUT /system/reboot, 91
PUT /system/rollback, 92

Index

Symbols

3D-Koordinaten, 33
    Disparitätsbild, 33
3D-Modellierung, 33, 40

A

Abmessungen
    rc_visard, 10
AcquisitionFrameRate
    GenICam, 66
aktive Partition, 111
Anschlussset, 113
automatische Belichtung, 31

B

Baseline
    GenICam, 68
Basisabstand, 28
    GenICam, 68
Baumer
    IpConfigTool, 19
Belichtung, 28
    automatisch, 31
    manuell, 31
Belichtungszeit, 29, 31
    Maximum, 31
Beschleunigung, 40, 41
    Sensordynamik, 24
Betriebsbedingungen, 12
Bewegungsunschärfe, 31
Bild
    Zeitstempel, 34, 70
Bildauflösung, 10
Bildmerkmale
    visuelle Odometrie, 42, 44
Bildrauschen, 31
Bildwiederholrate, 10
    Disparitätsbild, 36
    GenICam, 66
    Kamera, 31
    Pose, 39, 40
    visuelle Odometrie, 42
Brennweite, 28
Brennweitenfaktor
    GenICam, 68

C

CAD-Modell, 11
ComponentEnable
    GenICam, 66
ComponentIDValue
    GenICam, 66
ComponentSelector
    GenICam, 66
Confidence
    GenICam Bild-Stream, 69

D

Daten
    IMU, 41
    Inertialmesseinheit, 41
Datenmodell
    REST-API, 93
Datenstrom
    IMU, 41
    Pose, 40
    pose_rt, 40
    REST-API, 84
    Sensordynamik, 39, 40
Datentyp
    REST-API, 93
DepthDispRange
    GenICam, 68
DepthFill
    GenICam, 69
DepthMaxDepth
    GenICam, 69
DepthMaxDepthErr
    GenICam, 69
DepthMedian
    GenICam, 69
DepthMinConf
    GenICam, 69
DepthMinDepth
    GenICam, 69
DepthQuality
    GenICam, 68
DepthSeg
    GenICam, 69
DHCP, **5**, 19
discovery GUI, 19
Disparität, 23, 28, 32
    GenICam Bild-Stream, 69
Disparitätsbereich, 36
    GenICam, 68
    visuelle Odometrie, 44

Disparitätsbild, 23, 32
3D-Koordinaten, 33
Bildwiederholrate, 36
Parameter, 34
Qualität, 36
Web GUI, 34
Disparitätsfehler, 34
DNS, **5**
Download
Logdateien, 112
Dynamik
REST-API, 84
Web GUI, 42
Dynamik-Datenstrom, 39, 40
dynamischer Zustand, 24

E

Echtzeit-Pose, 39, 40
Ecken
visuelle Odometrie, 42, 44
Eigenbewegung, 24, 42
Error
GenICam Bild-Stream, 69
Ersatzteile, 114
Ethernet
Pin-Belegung, 14
ExposureAuto
GenICam, 66
ExposureTime
GenICam, 67
ExposureTimeAutoMax
GenICam, 68
externes Referenzkoordinatensystem
Hand-Auge-Kalibrierung, 52

F

Füllen, 37
GenICam, 69
Fehler, **34**
Hand-Auge-Kalibrierung, 60
Pose, 45
Feuchtigkeit, 12
Firmware
Mender, 109
Rollback, 111
Update, 109
Version, 109
FocalLengthFactor
GenICam, 68
fps, *siehe* Bildwiederholrate

G

Gehäusetemperatur
LED, 12
GenICam, **5**
AcquisitionFrameRate, 66
Baseline, 68
Basisabstand, 68
Bildwiederholrate, 66
Brennweitenfaktor, 68
ComponentEnable, 66
ComponentIDValue, 66
ComponentSelector, 66
DepthDispRange, 68
DepthFill, 69
DepthMaxDepth, 69
DepthMaxDepthErr, 69
DepthMedian, 69
DepthMinConf, 69
DepthMinDepth, 69
DepthQuality, 68
DepthSeg, 69
Disparitätsbereich, 68
ExposureAuto, 66
ExposureTime, 67
ExposureTimeAutoMax, 68
Füllen, 69
FocalLengthFactor, 68
GevIEEE1588, 67
Height, 66
HeightMax, 66
maximaler Abstand, 69
maximaler Fehler, 69
Median, 69
minimale Konfidenz, 69
minimaler Abstand, 69
PixelFormat, 66, 69
Qualität, 68
Scan3dCoordinateOffset, 68
Scan3dCoordinateScale, 68
Scan3dDistanceUnit, 67
Scan3dInvalidDataFlag, 68
Scan3dInvalidDataValue, 68
Scan3dOutputMode, 67
Segmentierung, 69
Width, 66
WidthMax, 66
Zeitstempel, 70
GenICam Bild-Stream
Confidence, 69
Disparität, 69
Error, 69
Intensity, 69
IntensityCombined, 69
Umwandlung, 70
Geschwindigkeit
linear, 40
Sensordynamik, 24
Winkel-, 40, 41
GevIEEE1588
GenICam, 67
GigE, **5**
GigE Vision, *siehe* GenICam
GigE Vision, **5**
GigEVision
IP-Adresse, 19

GPIO
Pin-Belegung, 14

H

Hand-Auge-Kalibrierung
externes Referenzkoordinatensystem, 52
Fehler, 60
Kalibrierung, 25, 55
Parameter, 61
Roboterkoordinatensystem, 52
Sensormontage, 53
Slot, 58
Height
GenICam, 66
HeightMax
GenICam, 66
Host-Name, 19

I

IMU, **5**, 24
Daten, 41
Datenstrom, 41
Inertialmesseinheit, 42
inaktive Partition, 111
Inertialmesseinheit
Daten, 41
IMU, 42
INS, **5**, 24
Installation
rc_visard, 18
Intensity
GenICam Bild-Stream, 69
IntensityCombined
GenICam Bild-Stream, 69
IP, **5**
IP-Adresse, **5**, 18
GigEVision, 19
IP 54, 12
IpConfigTool
Baumer, 19

K

Kühlung, 12
Kabel, 13, 113
Kalibriermuster, 46
Kalibrierung
Hand-Auge-Kalibrierung, 25, 55
Kamera, 46
Kamera-zu-IMU, 40
Rektifizierung, 28
Kamera
Bildwiederholrate, 31
Kalibrierung, 46
Parameter, 29, 31
Posen-Datenstrom, 40
Web GUI, 29
Kamera-zu-IMU
Kalibrierung, 40
Transformation, 40
Kamerakalibrierung
Monokalibrierung, 50
Parameter, 52
Services, 52
Stereokalibrierung, 50
Kameramodell, 28
Keyframes, 42
visuelle Odometrie, 42, 44
Komponenten
rc_visard, 9
Konfidenz, **34**
Minimum, 37
Koordinatensysteme
Montage, 16
Sensordynamik, 40
Zustandsschätzung, 38
Korrespondenzen
visuelle Odometrie, 42

L

LED, 18
Farben, 115
Gehäusetemperatur, 12
linear
Geschwindigkeit, 40
Link Local, 19
Link Local, **5**
Logdateien
Download, 112
Logs
REST-API, 87

M

MAC-Adresse, **5**, 19
manuelle Belichtung, 31
maximaler Abstand, 37
GenICam, 69
maximaler Fehler, 37
GenICam, 69
Maximum
Belichtungszeit, 31
Tiefenfehler, 37
mDNS, **5**
Median, 37
GenICam, 69
Mender
Firmware, 109
minimale Konfidenz, 37
GenICam, 69
minimaler Abstand, 37
GenICam, 69
Minimum
Konfidenz, 37
Monokalibrierung
Kamerakalibrierung, 50
Montage, 15

N

Netzteil, 114
Netzwerkkabel, 113
Netzwerkkonfiguration, 18
Neustart, 112
node
    REST-API, 73

P

Parameter
    Disparitätsbild, 34
    Hand-Auge-Kalibrierung, 61
    Kamera, 29, 31
    Kamerakalibrierung, 52
    REST-API, 73
    Services, 32
    visuelle Odometrie, 42
Pin-Belegung
    Ethernet, 14
    GPIO, 14
    Stromzufuhr, 14
PixelFormat
    GenICam, 66, 69
Pose
    Bildwiederholrate, 39, 40
    Datenstrom, 40
    Fehler, 45
    Sensordynamik, 24
    Zeitstempel, 39
pose_rt
    Datenstrom, 40
Posen-Datenstrom, 40
    Kamera, 40
Posenschätzung, *siehe* Zustandsschätzung
possible_jump
    Sensordynamik, 40
    SLAM, 40
PTP, 67
Punktwolke, 33

Q

Qualität
    Disparitätsbild, 36
    GenICam, 68
Quaternion
    Rotation, 40

R

rc_dynamics, 105
rc_visard
    Installation, 18
    Komponenten, 9
Rektifizierung, 28
REST-API, 71
    Datenmodell, 93
    Datenstrom, 84
    Datentyp, 93
    Dynamik, 84
    Einstiegspunkt, 71
    Logs, 87
    node, 73
    Parameter, 73
    Services, 74
    Statuswert, 73
    System, 87
    Version, 71
Roboterkoordinatensystem
    Hand-Auge-Kalibrierung, 52
Rollback
    Firmware, 111
Rotation
    Quaternion, 40

S

Scan3dCoordinateOffset
    GenICam, 68
Scan3dCoordinateScale
    GenICam, 68
Scan3dDistanceUnit
    GenICam, 67
Scan3dInvalidDataFlag
    GenICam, 68
Scan3dInvalidDataValue
    GenICam, 68
Scan3dOutputMode
    GenICam, 67
Schleifenschluss, 45
Schutzklasse, 12
SDK, **5**
Segmentierung, 37
    GenICam, 69
Selbstkalibrierung, 46
Semi-Global Matching, *siehe* SGM
Sensordatenfusion, 42
Sensordynamik
    Beschleunigung, 24
    Datenstrom, 39, 40
    Geschwindigkeit, 24
    Koordinatensysteme, 40
    Pose, 24
    possible_jump, 40
    Services, 41
Sensormontage
    Hand-Auge-Kalibrierung, 53
Services
    Kamerakalibrierung, 52
    Parameter, 32
    REST-API, 74
    Sensordynamik, 41
    visuelle Odometrie, 44
SGM, **5**, 23, **33**
Simultane Lokalisierung und Kartierung, *siehe* SLAM
SLAM, **5**, **45**
    possible_jump, 40
    Web GUI, 45
Slot

Hand-Auge-Kalibrierung, 58
Spezifikationen
rc_visard, 10
Stativ, 15
Statuswert
REST-API, 73
Stereo-Matching, 23
Stereokalibrierung
Kamerakalibrierung, 50
Stereokamera, 28
Stromkabel, 113, 114
Stromversorgung, **12**
Stromzufuhr
Pin-Belegung, 14
Swagger UI, 101
Synchronisierung, 67
System
REST-API, 87

T

Temperaturbereich, 12
Textur, 33
Tiefenbild, **33**, 33
Web GUI, 34
Tiefenfehler
Maximum, 37
Transformation
Kamera-zu-IMU, 40
Translation, 40

U

UDP, **5**
Umwandlung
GenICam Bild-Stream, 70
Update
Firmware, 109
URI, **6**
URL, **6**

V

Version
Firmware, 109
REST-API, 71
Verstärkung, 28
Verstärkungsfaktor, 29, 31
VINS, 24
visuelle Odometrie, 24, **42**
Bildmerkmale, 42, 44
Bildwiederholrate, 42
Disparitätsbereich, 44
Ecken, 42, 44
Keyframes, 42, 44
Korrespondenzen, 42
Parameter, 42
Services, 44
Web GUI, 42
VO, *siehe* visuelle Odometrie

W

Web GUI, **21**
Disparitätsbild, 34
Dynamik, 42
Kamera, 29
Logs, 112
SLAM, 45
Tiefenbild, 34
Update, 109
visuelle Odometrie, 42
Weißabgleich, 32
Width
GenICam, 66
WidthMax
GenICam, 66
Winkel-
Geschwindigkeit, 40, 41

X

XYZ+Quaternion, **6**
XYZABC-Format, **6**

Z

Zeitstempel, 28
Bild, 34, 70
GenICam, 70
Pose, 39
Zurücksetzen, 19
Zustandsschätzung, 39
Koordinatensysteme, 38