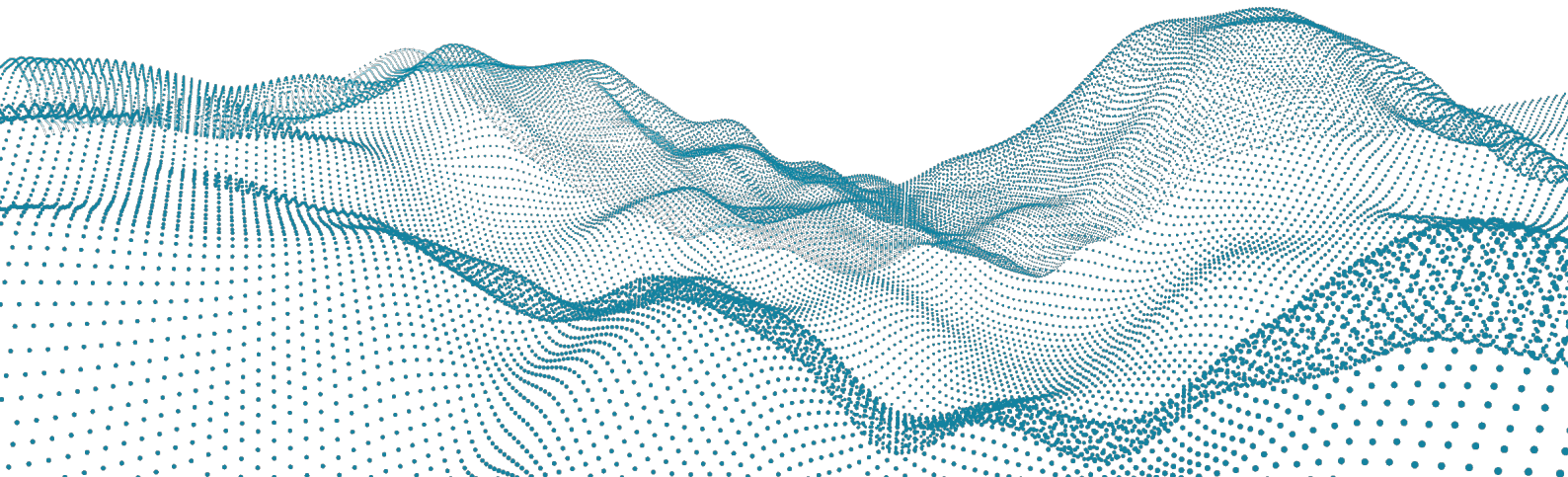
roboception

Roboception GmbH | April 2021

rc_cube Beschleuniger für den rc_visard

MONTAGE- UND BETRIEBSANLEITUNG



Revisionen

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

DOKUMENTATIONSVERSION 21.04.0 15.04.2021

Gültig für *rc_cube* Firmware 21.04.x

HERSTELLER

Roboception GmbH

Kaflerstraße 2

81241 München

Deutschland

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

Betriebsanleitung bitte vollständig lesen und produktnah aufbewahren.

Inhaltsverzeichnis

1 Einführung

Hinweise im Handbuch

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

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

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

1.1 Überblick

Der *rc_cube* ist ein Gerät zur performanten 3D-Bildverarbeitung, welches die Rechenkapazitäten der Roboception Stereokamera *rc_visard* erweitert.

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

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

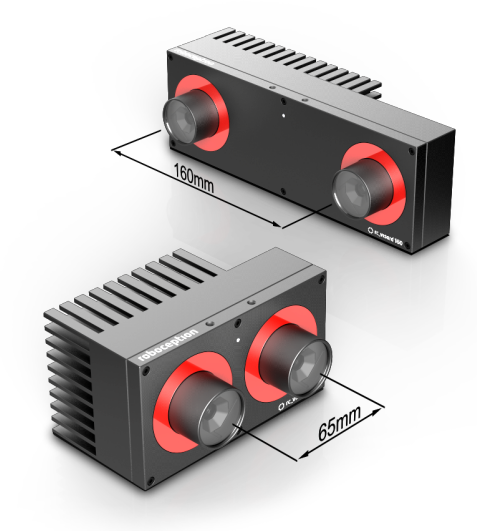


Abb. 1.1: *rc_visard* 65 und *rc_visard* 160

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

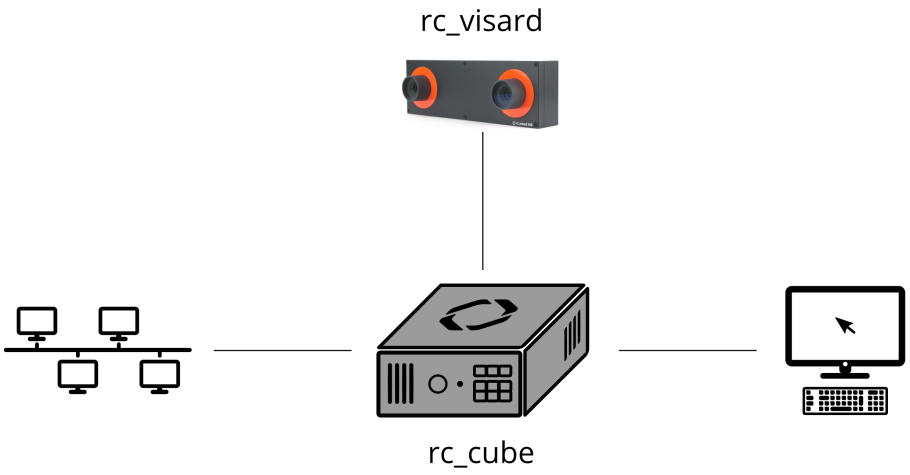


Abb. 1.2: Der *rc_cube*

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

1.2 Garantie

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

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

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

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

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

1.3 Schnittstellen, Zulassungen und Normen

1.3.1 Schnittstellen

Der *rc_cube* 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.

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

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

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

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

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

IP

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

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

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

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

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.

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

URI

URL Ein Uniform Resource Identifier (URI) ist eine Zeichenfolge, mit der sich Ressourcen in der REST-API des *rc_cube* 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://<ip>/api/v1/nodes/rc_stereocamera/parameters/fps` lokalisiert werden, wobei sich `<ip>` auf die *IP-Adresse* des *rc_cube* bezieht.

XYZ+Quaternion Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *Rotationsmatrix und Translationsvektor* (Abschnitt 10.1.1).

XYZABC Format zur Darstellung von Posen (Positionen und Orientierungen). Für eine Definition siehe *KUKA XYZ-ABC Format* (Abschnitt 10.1.5).

2 Sicherheit

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

Warnung: Wird der *rc_cube* zusammen mit *rc_visard*-Produkten betrieben, muss der Bediener alle Anweisungen zur Sicherheit, Inbetriebnahme und Wartung im *rc_visard*-Bedienhandbuch gelesen und verstanden haben.

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

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

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

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

2.1 Allgemeine Sicherheitshinweise

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

Warnung:

- Die zum *rc_cube* oder zugehöriger Ausrüstung angegebenen Sicherheitshinweise müssen stets eingehalten werden.
- Der *rc_cube* fällt nicht in den Anwendungsbereich der europäischen Maschinen- oder Medizinprodukterichtlinie.

2.2 Bestimmungsgemäße Verwendung

Der *rc_cube* ist für den gemeinsamen Betrieb mit einem Roboception *rc_visard* zur Datenerfassung (zum Beispiel von Stereo-Bildern) bestimmt. Darüber hinaus ist das System dazu bestimmt, die erfassten Daten mittels Algorithmen aus dem Bereich der 3D-Bildverarbeitung zu verarbeiten, um für Objekterkennung oder robotische Pick-and-Place-Anwendungen genutzt zu werden.

Warnung: Der *rc_cube* ist ausschließlich für den stationären Betrieb bestimmt.

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

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

Warnung: Der *rc_cube* muss an gesicherte interne Netzwerke angeschlossen werden.

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

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

3 Installation

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

3.1 Installation und Konfiguration

Für den Anschluss an ein Computernetzwerk, sowie die Verbindung mit einem *rc_visard*, verfügt der *rc_cube* über mindestens zwei Gigabit-Ethernet-Schnittstellen. Beide Steckplätze sind entsprechend beschriftet. Alle anderen Ethernet-Schnittstellen sind deaktiviert.

Für die Inbetriebnahme, bzw. während des Betriebs oder zur Fehlerbehandlung, können Eingabegeräte wie Maus und Tastatur sowie ein Computerbildschirm an den *rc_cube* angeschlossen werden. Dies ist jedoch nur optional, denn über das verbundene, lokale Netzwerk kann der Nutzer auf alle Funktionalitäten des *rc_cube* zugreifen.

Bemerkung: Wenn ein Bildschirm am *rc_cube* verwendet werden soll, dann muss er bereits beim Booten verbunden sein, oder der *rc_cube* muss neu gestartet werden, um den Bildschirm zu aktivieren.

3.2 Softwarelizenz

Jeder *rc_cube* wird mit einem USB-Dongle zur Lizenzierung und zum Schutz der installierten Softwarepakete ausgeliefert. Die erworbenen Lizenzen sind auf diesem Dongle installiert und somit an ihn und seine ID gebunden.

Die Funktionalität des *rc_cube* kann jederzeit durch ein *Upgrade der Lizenz* (Abschnitt 7.5) erweitert werden – zum Beispiel für zusätzlich erhältliche, optionale Softwaremodule.

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

Bemerkung: Die Dongle-ID und der Status der Softwarelizenz kann über die verschiedenen Schnittstellen des *rc_cube* abgefragt werden, zum Beispiel über die Seite *System* in der *Web GUI* (Abschnitt 6.1).

Bemerkung: Damit die Lizenzierung der Softwaremodule ordnungsgemäß funktioniert, muss der USB-Dongle an den *rc_cube* angesteckt werden, bevor dieser gestartet wird.

Bemerkung: Der *rc_cube* muss neu gestartet werden, sobald der Dongle eingesteckt oder abgezogen wurde.

3.3 Einschalten

Der *rc_cube* wird mit der Einschalttaste am Gerät gestartet. Ist ein Computerbildschirm angeschlossen, zeigt dieser die Web GUI an, sobald der Boot-Vorgang des *rc_cube* abgeschlossen ist.

Bemerkung: Zur Gewährleistung eines erfolgreichen Betriebs stellen Sie sicher, dass der mit dem *rc_cube* verbundene *rc_visard* mit Strom versorgt und dessen Boot-Vorgang abgeschlossen ist.

3.4 Aufspüren von *rc_cube*-Geräten

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

Das Open-Source-Tool `rcdiscover-gui` kann für Windows und Linux kostenlos von der Roboception-Homepage heruntergeladen werden: http://www.roboception.com/download. Dieses Tool besteht für Windows 7 und Windows 10 aus einer einzigen ausführbaren Datei, die ohne Installation direkt ausgeführt werden kann. Für Linux ist ein Installationspaket für Ubuntu erhältlich.

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

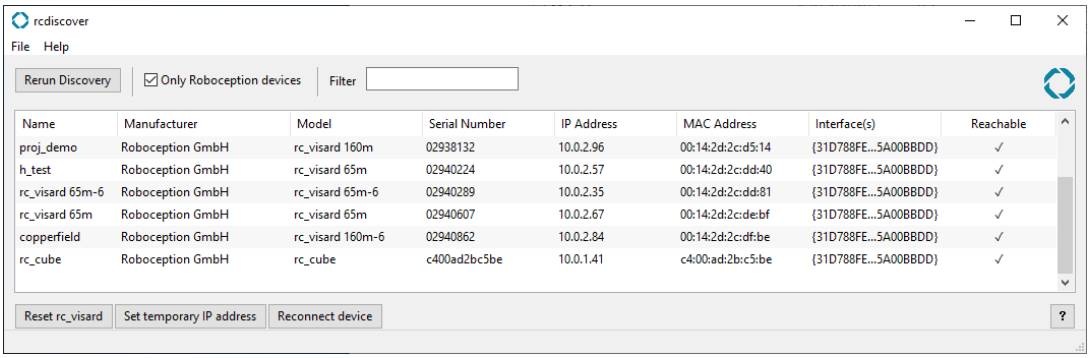


Abb. 3.1: `rcdiscover-gui`-Tool zum Aufspüren angeschlossener GigE Vision®-Geräte

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

3.4.1 Zurücksetzen der Konfiguration

Bemerkung: Der Zurücksetzmechanismums des `rcdiscover-gui`-Tools ist für *rc_cube*-Geräte im Moment nicht implementiert.

3.5 Netzwerkkonfiguration

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

Die Netzwerkeinstellungen des *rc_visard*, welcher mit dem *rc_cube* zusammen betrieben wird, werden automatisch passend konfiguriert, sobald der *rc_visard* mit dem *rc_cube* verbunden wird.

Warnung: Um Adresskonflikte mit dem internen Netzwerk zwischen dem *rc_cube* und dem verbundenen *rc_visard* zu vermeiden, muss die für den *rc_cube* gewählte IP-Adresse außerhalb des Adressbereichs `172.23.42.0/24` liegen.

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

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

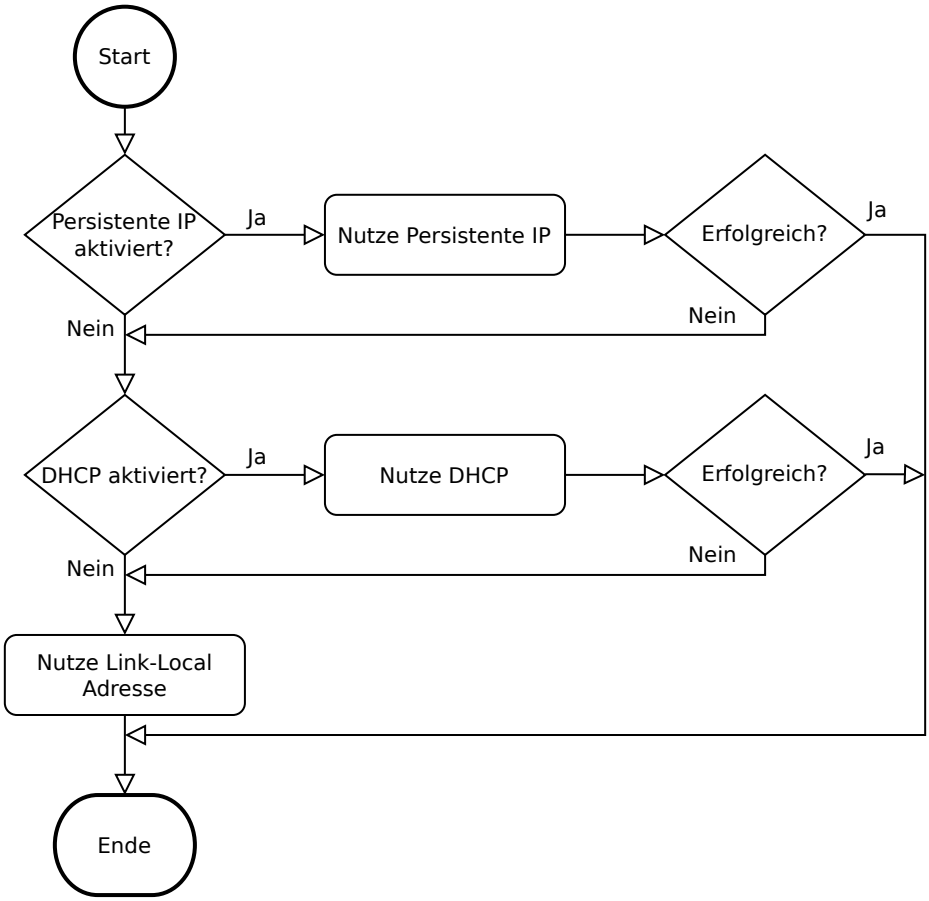


Abb. 3.2: Flussdiagram für die Auswahl der IP-Konfigurationsmethoden des *rc_cube*

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

- die Seite *System* der *rc_cube*-Web GUI – falls diese im lokalen Netzwerk bereits erreichbar ist, siehe *Web GUI* (Abschnitt 6.1)
- alle Konfigurationsprogramme, welche mit *GigE Vision*® 2.0 kompatibel sind, oder das Roboception Kommandozeilenprogramm `gc_config`. Üblicherweise wird nach dem Start dieser Programme das Netzwerk nach allen verfügbaren GigE Vision®-Sensoren durchsucht. Alle *rc_cube*-Geräte können über ihre Seriennummer und MAC-Adresse, die beide auf dem Gerät aufgedruckt sind, eindeutig identifiziert werden.
- das Roboception `rcdiscover-gui`-Tool, welches temporäre Änderungen oder das vollständige Zurücksetzen der Netzwerkkonfiguration des *rc_cube* erlaubt, siehe *Aufspüren von rc_cube-Geräten* (Abschnitt 3.4)

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

3.5.1 Host-Name

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

3.5.2 Automatische Konfiguration (werkseitige Voreinstellung)

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

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

Kann der *rc_cube* 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_cube* 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.

3.5.3 Manuelle Konfiguration

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

Warnung: Die IP-Adresse muss eindeutig sein und innerhalb des Gültigkeitsbereichs des lokalen Netzwerks liegen. Zudem muss die Subnetz-Maske dem lokalen Netzwerk entsprechen, da anderenfalls möglicherweise nicht auf den *rc_cube* zugegriffen werden kann. Dieses Problem lässt sich vermeiden, indem die unter *Automatische Konfiguration (werkseitige Voreinstellung)* (Abschnitt 3.5.2) beschriebene automatische Konfiguration genutzt wird.

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

4 Messprinzipien

Der *rc_cube* wird zusammen mit der 3D-Kamera *rc_visard* von Roboception betrieben und dient als performantes 3D-Bildverarbeitungssystem. Gemeinsam erstellen und verarbeiten sie rektifizierte Bilder, sowie Disparitäts-, Konfidenz- und Fehlerbilder, mit denen sich die Tiefenwerte der Aufnahme berechnen lassen.

Im Folgenden sind die zugrunde liegenden Messprinzipien genauer dargestellt.

4.1 Stereovision

Der *rc_cube* 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. 4.1 genauer dargestellt.

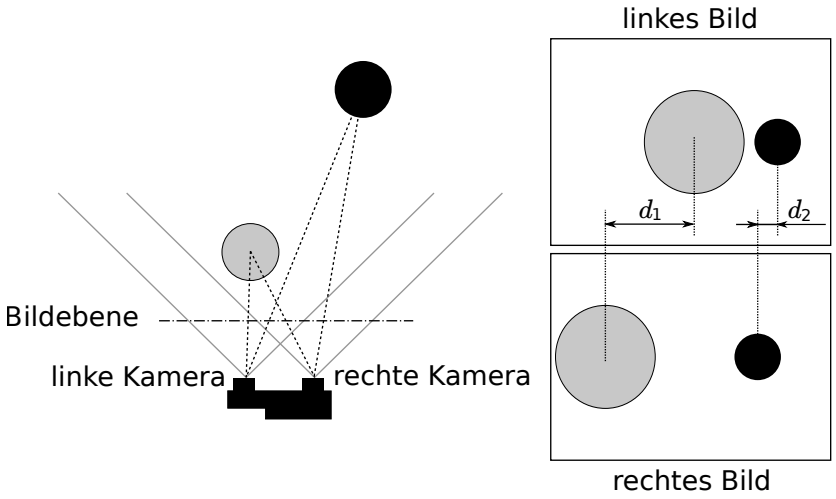


Abb. 4.1: Schematische Darstellung des Prinzips der Stereovision: Die Disparität d_2 des weiter entfernten (schwarzen) Objekts ist kleiner als die Disparität d_1 des nahe liegenden (grauen) Objekts.

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

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

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

- *Stereokamera*: Dieses Modul dient dazu, synchronisierte Stereo-Bildpaare aufzunehmen und diese in Bilder umzuwandeln, die weitestgehend den Aufnahmen einer idealen Stereokamera entsprechen (Rektifizierung) (Abschnitt 5.1.1).
- *Stereo-Matching*: Dieses Modul errechnet mithilfe des Stereo-Matching-Verfahrens *SGM* die Disparitäten der rektifizierten Stereo-Bildpaare (Abschnitt 5.1.2).

5 Softwaremodule

Der *rc_cube* wird mit einer Reihe von On-Board-Softwaremodulen mit verschiedenen Funktionalitäten ausgeliefert. Jedes Softwaremodul bietet über seinen zugehörigen *Node* eine *REST-API-Schnittstelle* (Abschnitt 6.3) als Programmierschnittstelle an.

Die Softwaremodule des *rc_cube* können unterteilt werden in

- ***3D-Kamera-Module*, Abschnitt 5.1** welche Stereobildpaare aufnehmen und 3D Tiefeninformationen bereitstellen, und auch über die *GigE Vision/GenICam-Schnittstelle* des *rc_cube* konfigurierbar sind.
- ***Detektionsmodule*, Abschnitt 5.2** welche eine Vielzahl verschiedener Detektionsfunktionen, wie Greifpunktberechnungen und Objekterkennung anbieten.
- ***Konfigurationsmodule*, Abschnitt 5.3** welche es dem Nutzer ermöglichen, den *rc_cube* für spezielle Anwendungen zu konfigurieren.

5.1 3D-Kamera-Module

Die 3D-Kamera-Software des *rc_cube* enthält die folgenden Module:

- ***Stereokamera* (`rc_stereocamera`, Abschnitt 5.1.1)** 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-Bild-Streams* für die externe Verwendung bereitgestellt.
- ***Stereo-Matching* (`rc_stereomatching`, Abschnitt 5.1.2)** nutzt die rektifizierten Stereo-Bildpaare, um 3D-Tiefeninformationen, z.B. für Disparitäts-, Fehler- und Konfidenzbilder, zu berechnen. Diese werden auch als GenICam-Bild-Streams bereitgestellt.

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

5.1.1 Stereokamera

Das Stereokamera-Modul ist ein Basismodul, das auf jedem *rc_cube* verfügbar ist, und beinhaltet Funktionen zur Erfassung von Stereo-Bildpaaren und zur planaren Rektifizierung, die nötig ist, um die Stereokamera als Messinstrument nutzen zu können.

5.1.1.1 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. Der Zeitpunkt in der Mitte der Bildbelichtung wird den Bildern als Zeitstempel angeheftet. Dieser Zeitstempel ist für dynamische Anwendungen wichtig, bei denen sich die Kamera oder die Szene bewegt.

Die Belichtungszeit lässt sich manuell auf einen festen Wert einstellen. Dies ist hilfreich in Umgebungen, in denen die Beleuchtung gesteuert werden kann, da die Lichtintensität so in allen Bildern gleich ist. Die Kamera ist standardmäßig auf automatische Belichtung eingestellt. In diesem Modus wird die Belichtungszeit automatisch, bis zu einem benutzerdefinierten Höchstwert, gewählt. Mit dem zulässigen Höchstwert soll eine mögliche Bewegungsunschärfe begrenzt werden: Hierzu kommt es, wenn Aufnahmen gemacht werden, während sich die Kamera 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.

5.1.1.2 Planare Rektifizierung

Kameraparameter, wie die Brennweite, die Objektivverzeichnung und die Stellung der Kameras zueinander, müssen genau bekannt sein, damit die Stereokamera als Messinstrument eingesetzt werden kann. Der *rc_visard* ist 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 sicherstellen, dass die dritten Koordinaten der Bildpunkte p_l und p_r 1 entsprechen.

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

5.1.1.3 Anzeigen und Herunterladen von Bildern

Der *rc_cube* bietet über die GenICam-Schnittstelle zeitgestempelte rektifizierte Bilder der linken und rechten Kamera (siehe *Verfügbare Bild-Streams*, Abschnitt 6.2.6). Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 6.1) bereitgestellt.

Die *Web GUI* (Abschnitt 6.1) bietet weiterhin die Möglichkeit, einen Schnappschuss der aktuellen Szene als .tar.gz-Datei zu speichern, wie in *Herunterladen von Stereo-Bildern* (Abschnitt 6.1.3) beschrieben wird.

5.1.1.4 Parameter

Das Stereokamera-Modul wird als `rc_stereocamera` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *Kamera* dargestellt. Der Benutzer kann die Kamera-Parameter entweder dort oder direkt über die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.3) oder GigE Vision (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.2) ändern.

Bemerkung: Wird der *rc_cube* 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. 5.1: Laufzeitparameter des `rc_stereocamera`-Moduls

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

Beschreibung der Laufzeitparameter

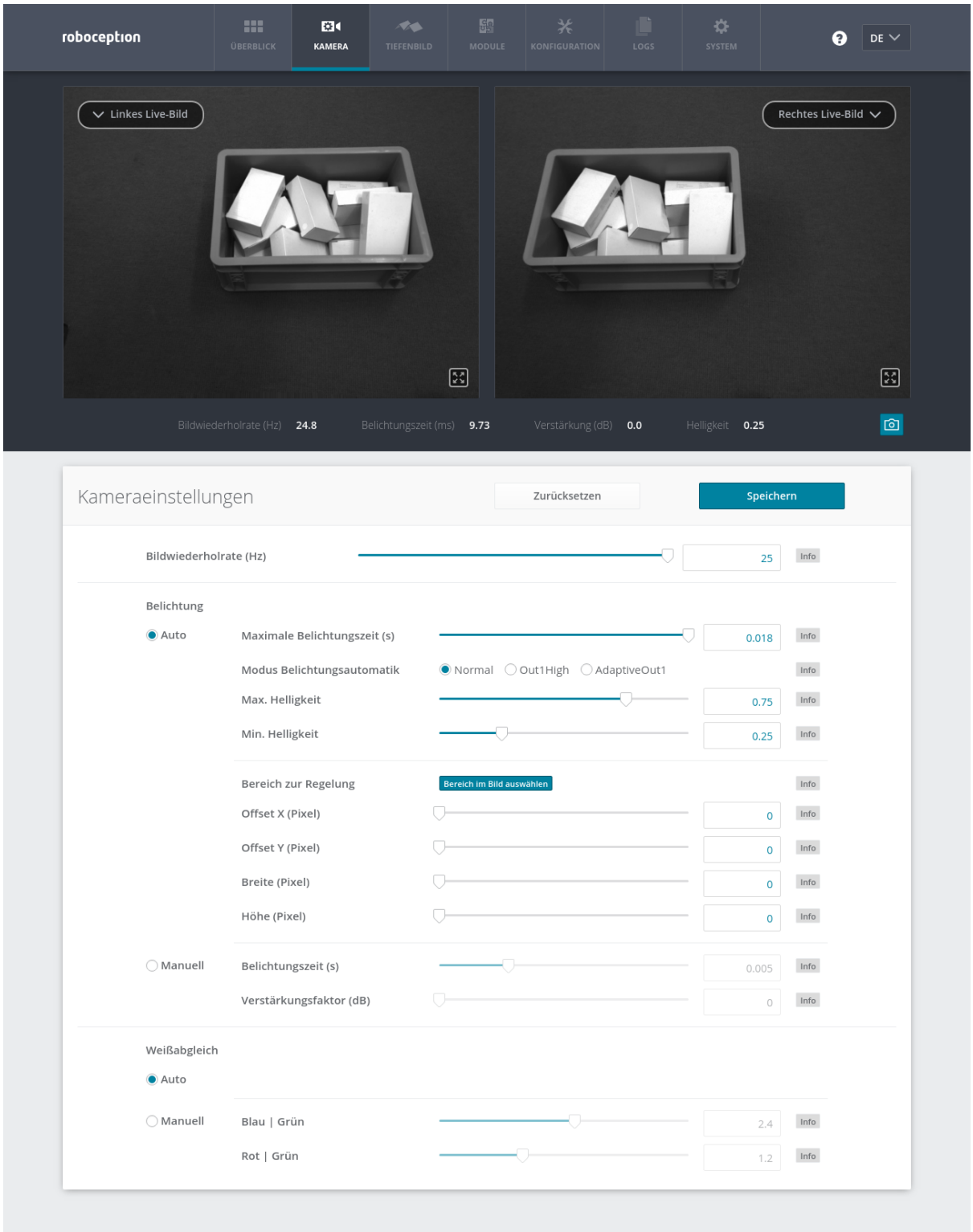


Abb. 5.1: Seite *Kamera* in der Web GUI

`fps` (*Bildwiederholrate*)

Dieser Wert bezeichnet die Bildwiederholrate der Kamera in Bildern pro Sekunde und begrenzt zugleich die Frequenz, mit der Tiefenbilder berechnet werden können. Die Bildwie-

derholrate entspricht auch der Frequenz, mit welcher der *rc_cube* Bilder über GigE Vision bereitstellt. Wird diese Frequenz verringert, reduziert sich auch die zur Übertragung der Bilder benötigte Bandbreite des Netzwerks.

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

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

`exp_auto` (*Belichtungszeit Auto* oder *Manuell*)

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

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

```
PUT http://<host>/api/v1/nodes/rc_stereocamera/parameters?exp_auto=<value>
```

`exp_auto_mode` (*Modus Belichtungsautomatik*)

Der Modus für automatische Belichtung kann auf *Normal*, *Out1High* oder *AdaptiveOut1* gesetzt werden. Diese Modi sind nur relevant, wenn der *rc_cube* mit einer externen Lichtquelle oder einem Projektor betrieben wird, der an den GPIO-Ausgang 1 des *rc_visard* angeschlossen ist. Dieser Ausgang kann durch das optionale IOControl-Modul (*IOControl und Projektor-Kontrolle*, Abschnitt 5.3.4) gesteuert werden.

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

Out1High: Die Belichtungszeit wird nur anhand der Bilder mit GPIO-Ausgang 1 HIGH angepasst. Bilder bei denen GPIO-Ausgang 1 LOW ist, werden für die Belichtungszeitregelung nicht berücksichtigt. Das bedeutet, die Belichtungszeit ändert sich nicht, solange nur Bilder mit GPIO-Ausgang 1 LOW aufgenommen werden. Dieser Modus wird für die Benutzung mit dem Single+Out1 Tiefenbild Aufnahmemodus (siehe *Stereo Matching Parameters*, 5.1.2.5 und externem Projektor empfohlen, wenn die Helligkeit der Szene nur zu den Zeitpunkten berücksichtigt werden soll, wenn GPIO-Ausgang 1 HIGH ist. Das ist zum Beispiel der Fall, wenn kurz vor einer Objekterkennung ein heller Teil des Roboters durch das Bild fährt, der die Belichtungseinstellungen jedoch nicht beeinflussen soll.

AdaptiveOut1: Dieser Modus nutzt alle Kamerabilder und speichert die Differenz der Belichtung zwischen Bildern mit GPIO Ausgang 1 HIGH und LOW. Während der IOControl-Modus für GPIO-Ausgang 1 LOW ist, werden die Bilder um diese Differenz unterbelichtet, um eine Überbelichtung zu verhindern, sobald der externe Projektor über GPIO-Ausgang 1 ausgelöst wird. Die Differenz der Belichtung wird als Out1 Reduktion unter den Livebildern angezeigt. Dieser Modus wird empfohlen, wenn im Stereo-Matching-Modul der Parameter `acquisition_mode` auf `SingleFrameOut1` (*Einzelbild+Out1*) gesetzt ist (*Parameter des Stereo-Matching-Moduls*, Abschnitt 5.1.2.5), und ein externer Projektor an den GPIO-Ausgang 1 angeschlossen ist, und wenn die Helligkeit der Szene zu jeder Zeit zur Belichtungszeitregelung berücksichtigt werden soll.Das ist zum Beispiel in Anwendungen mit veränderlichen äußeren Lichtbedingungen der Fall.

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

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

exp_max (*Maximale Belichtungszeit*)

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

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

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

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

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

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

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

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

```
PUT http://<host>/api/v1/nodes/rc_stereocamera/parameters?<exp_auto_average_max|exp_
↪auto_average_min>=<value>
```

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

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

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

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

```
PUT http://<host>/api/v1/nodes/rc_stereocamera/parameters?<exp_offset_x|exp_offset_
↪y|exp_width|exp_height>=<value>
```

`exp_value` (*Belichtungszeit*)

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

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

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

`gain_value` (*Verstärkungsfaktor*)

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

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

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

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

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

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

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

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

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

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

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

Die gleichen Parameter sind – mit leicht abweichenden Namen und teilweise mit anderen Einheiten oder Datentypen – auch über die GenICam-Schnittstelle verfügbar (siehe *GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.2).

5.1.1.5 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.2: Statuswerte des `rc_stereocamera`-Moduls

Name	Beschreibung
`out1_reduction`	Anteil der Helligkeits-Reduktion (0.0 - 1.0) für Bilder mit GPIO-Ausgang 1=LOW, wenn exp_auto_mode=AdaptiveOut1 oder exp_auto_mode=Out1High
`baseline`	Basisabstand t der Stereokamera in Metern
`brightness`	Aktuelle Helligkeit als Wert zwischen 0 und 1
`color`	0 für monochrome Kameras, 1 für Farbkameras
`exp`	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
`test`	0 for Live-Bilder und 1 für Test-Bilder
`time`	Verarbeitungszeit für die Bilderfassung in Sekunden
`width`	Breite des Kamerabildes in Pixeln

5.1.1.6 Services

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

`save_parameters`

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

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_stereocamera/services/save_parameters
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

`reset_defaults`

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

Dieser Service kann wie folgt aufgerufen werden.

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

Warnung: Durch den Aufruf dieses Services gehen die aktuellen Parametereinstellungen für das Stereokamera-Modul unwiderruflich verloren.

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.1.2 Stereo-Matching

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

Um Disparitäts-, Fehler- und Konfidenzbilder in voller Auflösung zu berechnen, wird eine gesonderte StereoPlus *Lizenz* (Abschnitt 7.5) benötigt. Diese Lizenz ist auf jedem *rc_cube* enthalten, der nach dem 31.01.2019 gekauft wurde.

5.1.2.1 Berechnung von Disparitätsbildern

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

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

Um Disparitätswerte zu berechnen, muss der Stereo-Matching-Algorithmus die zugehörigen Objektpunkte im linken und rechten Kamerabild finden. Diese Punkte stellen jeweils den gleichen Objektpunkt in der Szene dar. Für das Stereo-Matching nutzt der *rc_cube 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 5.1.2.3). Bei der Textur in der Szene sollte es sich nicht um ein künstliches, regelmäßig wiederkehrendes Muster handeln, da diese Strukturen zu Mehrdeutigkeiten und damit zu falschen Disparitätsmessungen führen können.

Für schwach texturierte Objekte oder in untexturierten Umgebungen lässt sich mithilfe eines externen Musterprojektors eine statische künstliche Struktur auf die Szene projizieren. Dieses projizierte Muster sollte zufällig sein und keine wiederkehrenden Strukturen enthalten. Der *rc_cube* bietet das IOControl-Modul als optionales Softwaremodul (siehe *IOControl und Projektor-Kontrolle*, Abschnitt 5.3.4), das einen Musterprojektor ansteuern kann.

5.1.2.2 Berechnung von Tiefenbildern und Punktwolken

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

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

wobei f die Brennweite nach der Rektifizierung (in Pixeln) und t der während der Kalibrierung ermittelte Stereo-Basisabstand (in Metern) ist. Diese Werte werden auch über die GenICam-Schnittstelle zur Verfügung gestellt (siehe *Besondere Parameter der GenICam-Schnittstelle des rc_cube*, Abschnitt 6.2.4).

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

Es ist zu beachten, dass für Gleichungen (5.1) davon ausgegangen wird, dass das Bildkoordinatensystem im Bildhauptpunkt zentriert ist, der üblicherweise in der Bildmitte liegt, und dass sich p_x, p_y auf die Mitte des Pixels bezieht, durch Addieren von 0.5 auf die ganzzahligen Pixelkoordinaten. In der folgenden Abbildung ist die Definition des Bildkoordinatensystems dargestellt.

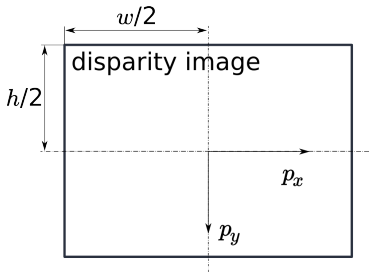


Abb. 5.2: Bildkoordinatensystem: Der Ursprung des Bildkoordinatensystems befindet sich in der Bildmitte – w ist die Bildbreite und h die Bildhöhe.

Die gleichen Formeln, aber mit den entsprechenden GenICam-Parametern, sind in *Umwandlung von Bild-Streams* (Abschnitt 6.2.7) angegeben.

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

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

5.1.2.3 Konfidenz- und Fehlerbilder

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

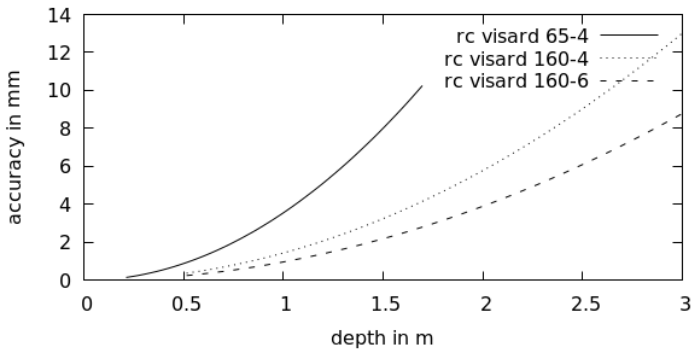
Der Disparitätsfehler d_{eps} (in Pixeln) lässt sich mithilfe der Brennweite f (in Pixeln), des Basisabstands t (in Metern) und des Disparitätswerts d (in Pixeln) desselben Pixels im Disparitätsbild in einen Tiefenfehler z_{eps} (in Metern) umrechnen:

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

Durch Kombination der Gleichungen (5.1) und (5.2) kann der Tiefenfehler zur Tiefe in Bezug gebracht werden:

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

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



5.1.2.4 Anzeigen und Herunterladen von Tiefenbildern und Punktwolken

Der *rc_cube* stellt über die GenICam-Schnittstelle zeitgestempelte Disparitäts-, Fehler- und Konfidenzbilder zur Verfügung (siehe *Verfügbare Bild-Streams*, Abschnitt 6.2.6). Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 6.1) bereitgestellt.

Die *Web GUI* (Abschnitt 6.1) bietet weiterhin die Möglichkeit, einen Schnappschuss der aktuellen Szene mit den Tiefen-, Fehler und Konfidenzbildern, sowie der Punktwolke als .tar.gz-Datei zu speichern, wie in *Herunterladen von Stereo-Bildern* (Abschnitt 6.1.4) beschrieben wird.

5.1.2.5 Parameter

Das Stereo-Matching-Modul wird in der REST-API als `rc_stereomatching` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *Tiefenbild* dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.3) oder über GigE Vision (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.3: Laufzeitparameter des `rc_stereomatching`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	Continuous	Acquisition mode: [Continuous, SingleFrame, SingleFrameOut1]
`double_shot`	bool	false	true	false	Kombination zweier Disparitätsbilder von zwei Stereobildpaaren
`fill`	int32	0	4	3	Disparitätstoleranz (für das Füllen von Löchern) in Pixeln
`maxdepth`	float64	0.1	100.0	100.0	Maximaler Abstand in Metern
`maxdeptherr`	float64	0.01	100.0	100.0	Maximaler Tiefenfehler in Metern
`minconf`	float64	0.5	1.0	0.5	Mindestkonfidenz
`mindepth`	float64	0.1	100.0	0.1	Minimaler Abstand in Metern
`quality`	string	-	-	High	Full (Voll), High (Hoch), Medium (Mittel), oder Low (Niedrig). Full benötigt eine ‚StereoPlus'-Lizenz.
`seg`	int32	0	4000	200	Mindestgröße der gültigen Disparitätssegmente in Pixeln
`smooth`	bool	false	true	true	Glättung von Disparitätsbildern (benötigt eine ‚StereoPlus'-Lizenz)
`static_scene`	bool	false	true	false	Mitteln von Bildern in statischen Szenen, um Rauschen zu reduzieren

Beschreibung der Laufzeitparameter

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

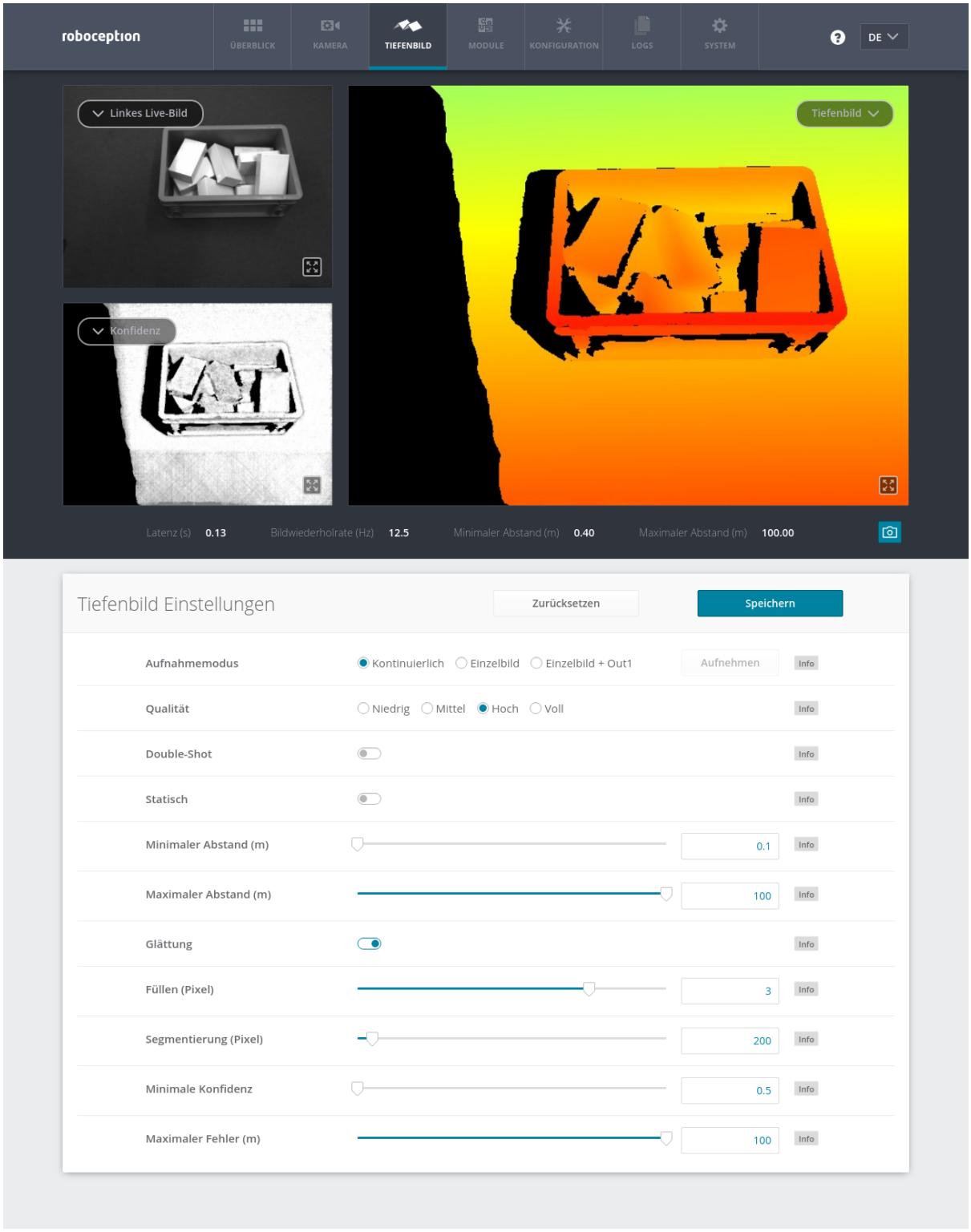


Abb. 5.3: Seite *Tiefenbild* der Web GUI

`acquisition_mode` (*Aufnahmemodus*)

Der Aufnahmemodus kann auf `Continuous` (*Kontinuierlich*), `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild + Out1*) eingestellt werden. *Kontinuierlich* ist die Standardeinstellung, bei der das Stereo-Matching kontinuierlich mit der vom Benutzer eingestellten Bildwie-

derholrate, entsprechend der verfügbaren Rechenressourcen, durchgeführt wird. Bei den beiden anderen Modi wird das Stereo-Matching bei jedem Drücken des *Aufnehmen*-Knopfes durchgeführt. Der *Einzelbild + Out1* Modus kontrolliert zusätzlich einen externen Projektor, falls dieser an GPIO-Ausgang 1 angeschlossen ist (*IOControl und Projektor-Kontrolle*, Abschnitt 5.3.4). In diesem Modus wird `out1_mode` des IOControl-Moduls automatisch bei jedem Trigger auf `ExposureAlternateActive` und nach dem Aufnehmen der Bilder für das Stereo-Matching auf `Low` gesetzt.

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

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

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

`quality` (*Qualität*)

Disparitätsbilder lassen sich in verschiedenen Auflösungen berechnen: `High` (*Hoch*, 640 x 480 Pixel), `Medium` (*Mittel*, 320 x 240 Pixel) und `Low` (*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. Es ist zu beachten, dass die Bildwiederholrate der Disparitäts-, Konfidenz- und Fehlerbilder immer höchstens der Bildwiederholrate der Kamera entspricht.

Zusätzlich steht mit einer gültigen StereoPlus-Lizenz das Stereo-Matching mit der vollen Auflösung (`Full`) von 1280 x 960 Pixeln zur Verfügung.

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

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

`double_shot` (*Double-Shot*)

Dieser Modus ist sinnvoll für Szenen, die mit einem Projektor im `Single + Out1` Modus oder im kontinuierlichen Modus mit der Projektoreinstellung `ExposureAlternateActive` aufgenommen werden. Löcher, die durch Projektor-Reflexionen verursacht werden, werden gefüllt mit Tiefeninfomationen aus den Bildern ohne Projektormuster. Dieser Modus darf nur verwendet werden, wenn sich die Szene während der Aufnahme der Bilder nicht verändert.

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

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

`static_scene` (*Statisch*)

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

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

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

`mindepth` (*Minimaler Abstand*)

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

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

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

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

`maxdepth` (*Maximaler Abstand*)

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

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

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

`smooth` (*Glättung*)

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

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

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

`fill` (*Füllen*)

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

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

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

`seg` (*Segmentierung*)

Der Segmentierungsparameter wird verwendet, um die Mindestanzahl an Pixeln anzugeben, die eine zusammenhängende Disparitätsregion im Disparitätsbild ausfüllen muss. Isolierte Regionen, die kleiner sind, werden im Disparitätsbild auf ungültig gesetzt. Der Wert bezieht sich immer auf ein Disparitätsbild mit hoher Qualität mit der Auflösung 640 x 480 Pixeln

und muss nicht verändert werden, wenn andere Qualitäten gewählt werden. Die Segmentierung eignet sich, um Disparitätsfehler zu entfernen. Bei größeren Werten kann es jedoch vorkommen, dass real vorhandene Objekte entfernt werden.

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

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

`minconf` (*Minimale Konfidenz*)

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

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

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

`maxdeptherr` (*Maximaler Fehler*)

Der maximale Fehler wird verwendet, um Messungen, die zu ungenau sind, herauszufiltern. Alle Pixel mit einem Tiefenfehler, der den gewählten Wert überschreitet, werden im Disparitätsbild auf „ungültig“ gesetzt. Der maximale Tiefenfehler wird in Metern angegeben. Der Tiefenfehler wächst in der Regel quadratisch mit dem Abstand eines Objekts zur Kamera (siehe *Konfidenz- und Fehlerbilder*, Abschnitt 5.1.2.3).

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

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

Die gleichen Parameter sind – mit leicht abweichenden Namen und teilweise mit anderen Einheiten oder Datentypen – auch über die GenICam-Schnittstelle verfügbar (siehe *GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.2).

5.1.2.6 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 5.4: 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.
`latency`	Zeit zwischen Bildaufnahme und Weitergabe des Disparitätsbildes in Sekunden
`mindepth`	Tatsächlicher minimaler Arbeitsabstand in Metern
`maxdepth`	Tatsächlicher maximaler Arbeitsabstand in Metern
`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`.

5.1.2.7 Services

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

`acquisition_trigger`

Der Aufruf signalisiert dem Modul, das Stereo-Matching auf den nächsten Stereobildern durchzuführen, falls `acquisition_mode` auf `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild+Out1*) eingestellt ist. Es wird ein Fehler zurückgegeben, falls `acquisition_mode` auf `Continuous` (*Kontinuierlich*) eingestellt ist.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Mögliche Rückgabewerte sind in der Tabelle unten aufgeführt.

Tab. 5.5: Mögliche Rückgabewerte des `acquisition_trigger` Serviceaufrufs.

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

`save_parameters`

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

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "save_parameters",
  "response": {
    "return_code": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

reset_defaults

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

Dieser Service kann wie folgt aufgerufen werden.

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

Warnung: Durch den Aufruf dieses Services gehen die aktuellen Parametereinstellungen für das Stereo-Matching-Modul unwiderruflich verloren.

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.2 Detektionsmodule

Der *rc_cube* bietet Softwaremodule für unterschiedliche Detektionsanwendungen:

- ***LoadCarrier*** **(`rc_load_carrier`, Abschnitt 5.2.1)** ermöglicht das Anlegen und Abrufen von Load Carriern (Behältern), sowie deren Erkennung und Füllstandserkennung.
- ***TagDetect*** **(`rc_april_tag_detect` und `rc_qr_code_detect`, Abschnitt 5.2.2)** ermöglicht die Erkennung von AprilTags und QR-Codes sowie die Schätzung von deren Pose.
- ***ItemPick und BoxPick*** **(`rc_itempick` und `rc_boxpick`, Abschnitt 5.2.3)** bietet eine Standardlösung für robotische Pick-and-Place-Anwendungen für Vakuum-Greifsysteme.
- ***SilhouetteMatch*** **(`rc_silhouettematch`, Abschnitt 5.2.4)** bietet eine Objekterkennungslösung für Objekte, die sich auf einer Ebene befinden.
- ***CADMatch*** **(`rc_cadmatch`, Abschnitt 5.2.5)** bietet eine Objekterkennungslösung für 3D-Objekte.

Diese Module sind optional und können durch Kauf einer separaten *Lizenz* (Abschnitt 7.5) aktiviert werden.

5.2.1 LoadCarrier

5.2.1.1 Einleitung

Das LoadCarrier Modul ist ein optionales Modul, welches intern auf dem *rc_cube* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module ItemPick, BoxPick oder SilhouetteMatch vorhanden ist. Andernfalls benötigt es eine gesonderte LoadCarrier *Lizenz* (Abschnitt 7.5), welche erworben werden muss.

Die Load Carrier Funktionalität wird vom LoadCarrier Modul selbst, aber auch den *ItemPick und BoxPick* (Abschnitt 5.2.3) Modulen, sowie vom *SilhouetteMatch* (Abschnitt 5.2.4) Modul und *CAD-Match* (Abschnitt 5.2.5) Modul angeboten.

5.2.1.2 Load Carrier

Ein sogenannter Load Carrier ist ein Behälter mit vier Wänden, einem Boden und einem rechteckigen Rand, der Objekte enthalten kann. Er kann genutzt werden, um das Volumen, in dem nach Objekten oder Greifpunkten gesucht wird, zu begrenzen.

Seine Geometrie ist durch die inneren und äußeren Abmessungen (`inner_dimensions` und `outer_dimensions`) definiert. Die maximalen `outer_dimensions` betragen 2.0 m in allen Dimensionen.

Der Ursprung des Load Carrier Koordinatensystems liegt im Zentrum des durch die *Außenmaße* definierten Quaders. Dabei ist die z-Achse senkrecht zum Boden des Load Carriers und zeigt aus dem Load Carrier heraus (siehe Abb. 5.4).

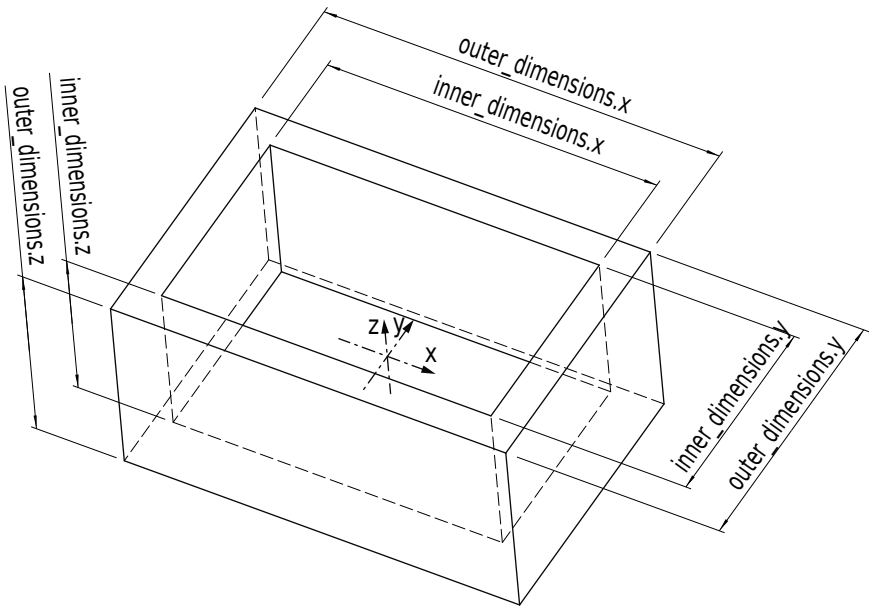


Abb. 5.4: Load Carrier mit Koordinatensystem und inneren und äußeren Abmessungen

Bemerkung: Die Innen- und Außenmaße eines Load Carriers sind typischerweise in den Angaben des jeweiligen Herstellers spezifiziert, und können im Produktblatt oder auf der Produktseite nachgeschlagen werden.

Das Innenvolumen eines Load Carriers ist durch seine Innenmaße definiert, aber enthält zusätzlich einen Bereich von 10 cm oberhalb des Load Carriers, damit Objekte, die aus dem Load Carrier herausragen, auch für die Detektion oder Greifpunktberechnung berücksichtigt werden. Weiterhin wird vom Innenvolumen in jeder Richtung ein zusätzlicher Sicherheitsabstand `crop_distance` abgezogen, welcher

als Laufzeitparameter konfiguriert werden kann (siehe *Parameter*, Abschnitt 5.2.1.7). Abb. 5.5 zeigt das Innenvolumen eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

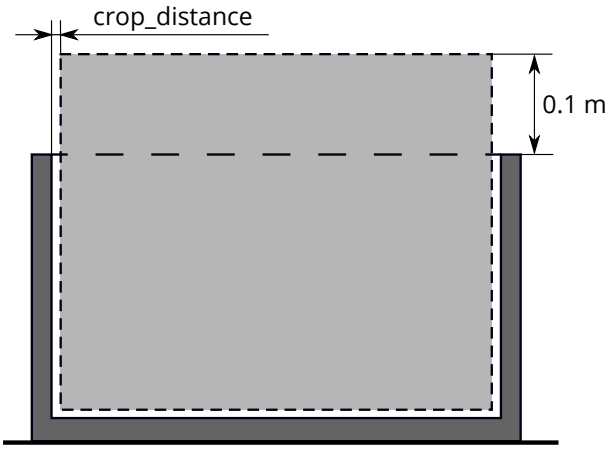


Abb. 5.5: Darstellung des Innenvolumens eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

Der *rc_cube* erlaubt das Speichern von bis zu 50 verschiedenen Load Carriern, von denen jeder mit einer `id` versehen ist. Die für eine spezifische Anwendung relevanten Load Carrier können mithilfe der *rc_cube* Web GUI oder der *REST-API-Schnittstelle* (Abschnitt 6.3) konfiguriert werden.

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

5.2.1.3 Load Carrier Abteil

Bei einigen Detektionsmodulen kann ein Load Carrier Abteil (`load_carrier_compartment`) angegeben werden, um das Volumen für die Erkennung zu begrenzen, zum Beispiel in *ItemPick's compute_grasps Service* (siehe 5.2.3.7). Ein Load Carrier Abteil ist eine Box, deren Pose `pose` als Transformation vom Load Carrier Koordinatensystem in das Abteilkoordinatensystem, welches im Zentrum der durch das Abteil definierten Box liegt, angegeben wird (siehe Abb. 5.6).

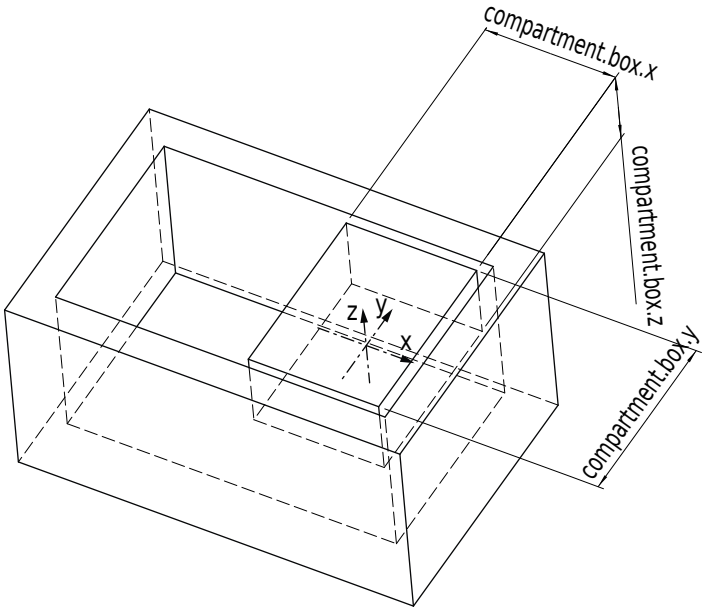


Abb. 5.6: Beispiel für ein Abteil innerhalb eines Load Carriers. Das gezeigte Koordinatensystem das Koordinatensystem des Load Carrier Abteils.

Als Volumen für die Detektion wird der Durchschnitt des Abteil-Volumens und des Load Carrier Innenraums verwendet. Wenn dieser Durchschnitt ebenfalls den Bereich von 10 cm oberhalb des Load Carriers enthalten soll, muss die Höhe der Box, die das Abteil definiert, entsprechend vergrößert werden.

5.2.1.4 Erkennung von Load Carriern

Der Erkennungsalgorithmus basiert auf der Erkennung des oberen, rechteckigen Randes (engl. rim) des Load Carriers. Für Standardbehälter wird dessen Stärke `rim_thickness` aus der Differenz von `inner_dimensions` und `outer_dimensions` berechnet. Für Nicht-Standardbehälter kann dieser Wert alternativ vom Nutzer explizit gesetzt werden.

Die Erkennung liefert die Pose des Load Carrier Koordinatensystems (siehe *Load Carrier*, Abschnitt 5.2.1.2) im gewünschten Referenzkoordinatensystem zurück.

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

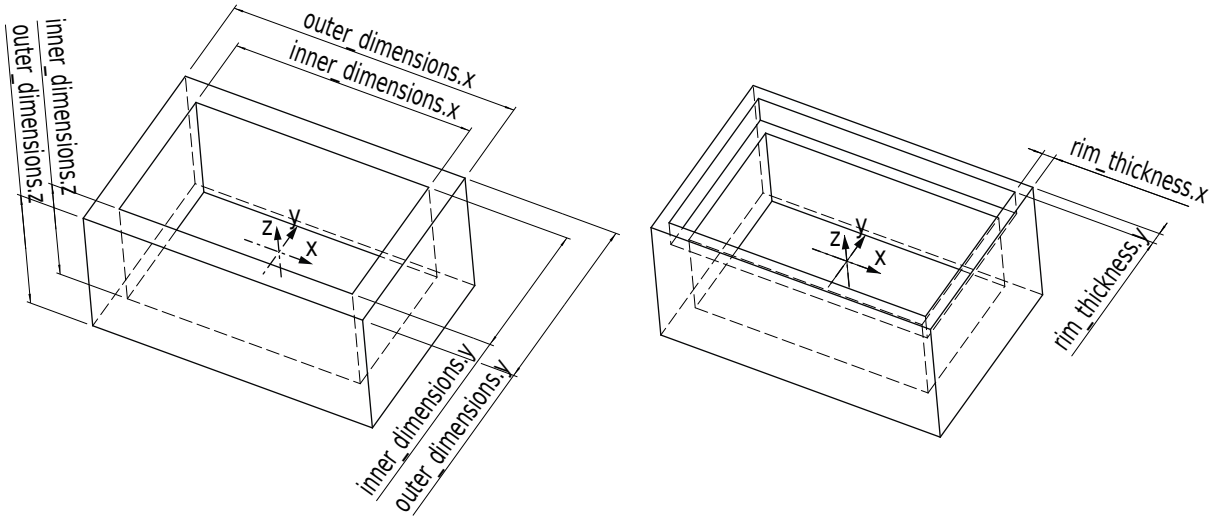


Abb. 5.7: Illustration verschiedener Load Carrier Modelle und deren Koordinatensysteme

Um Mehrdeutigkeiten bei der Lageschätzung der Load Carrier Erkennung zu umgehen, kann eine grobe Vorgabe für die Pose `pose` des Load Carriers spezifiziert werden. Wird keine angegeben, sucht der Algorithmus standardmäßig nach Load Carriern, die horizontal zum Gravitationsvektor stehen.

5.2.1.5 Füllstandserkennung

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

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

- `level_in_percent`: Minimum, Maximum und Mittelwert des Füllstands vom Boden in Prozent. Diese Werte können größer als 100% sein, falls die Zelle überfüllt ist.
- `level_free_in_meters`: Minimum, Maximum und Mittelwert in Metern des freien Teils der Zelle vom Rand des Load Carriers gemessen. Diese Werte können negativ sein, falls die Zelle überfüllt ist.
- `cell_size`: Abmessungen der 2D-Zelle in Metern.
- `cell_position`: Position des Mittelpunkts der Zelle in Metern (entweder im Koordinatensystem `camera` oder `external`, siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.2.3.4). Die z-Koordinate liegt auf der Ebene des Load Carrier Randes.
- `coverage`: Anteil der gültigen Pixel in dieser Zelle. Dieser Wert reicht von 0 bis 1 in Schritten von 0.1. Ein niedriger Wert besagt, dass die Zelle fehlende Daten beinhaltet (d.h. nur wenige Punkte konnten in der Zelle gemessen werden).

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

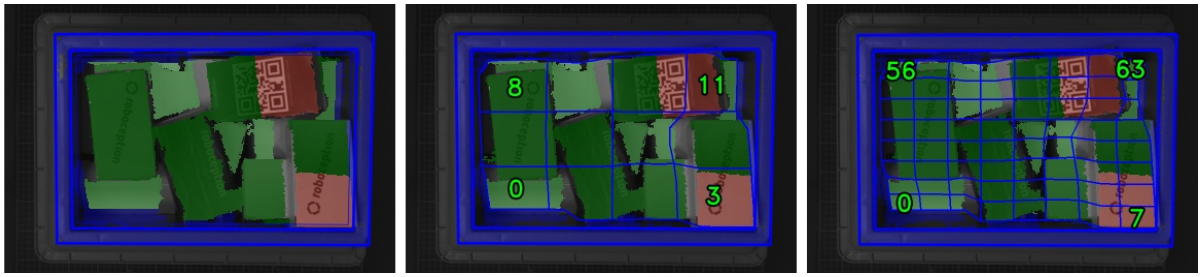


Abb. 5.8: Visualisierungen des Behälterfüllstands: Gesamtfüllstand ohne Raster (links), 4x3 Raster (Mitte), 8x8 Raster (rechts). Der Inhalt wird mit einem Farbverlauf von weiß (auf dem Boden) nach dunkelgrün dargestellt. Die überfüllten Bereiche sind rot dargestellt. Die Nummern stellen die Zell-IDs dar.

5.2.1.6 Wechselwirkung mit anderen Modulen

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

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

Stereokamera und Stereo-Matching

Folgende Daten werden vom LoadCarrier Modul verarbeitet:

- Die rektifizierten Bilder des *Stereokamera*-Moduls (`rc_stereocamera`, Abschnitt 5.1.1);
- Die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching*-Moduls (`rc_stereomatching`, Abschnitt 5.1.2).

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

Schätzung der Gravitationsrichtung

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

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

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_cube* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.3.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.1.2.5), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.3.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren (siehe *Stereokamera-Parameter*, Abschnitt 5.1.1.4).

Darüber hinaus sind keine weiteren Änderungen für diesen Anwendungsfall notwendig.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die Load Carrier Komponente automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.2.1.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht die Load Carrier Funktionalität alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

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

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

5.2.1.7 Parameter

Das LoadCarrier Modul wird in der REST-API als `rc_load_carrier` bezeichnet und wird intern von mehreren anderen Softwaremodulen genutzt. Seine Parameter und Services werden von diesen Modulen und von `rc_load_carrier` selbst zur Verfügung gestellt. Sie können auch in der *Web GUI* (Abschnitt 6.1) auf der Seite *LoadCarrier* (unter dem Menüpunkt *Module*) genutzt werden. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) ändern.

Übersicht über die Parameter

Dieses Modul bietet folgende Laufzeitparameter:

Tab. 5.6: Laufzeitparameter des `rc_load_carrier` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`crop_distance`	float64	0.0	0.05	0.005	Sicherheitsspielraum um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren
`model_tolerance`	float64	0.003	0.025	0.008	Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden in der Web GUI zeilenweise im Abschnitt *LoadCarrier Einstellungen* auf der Seite *LoadCarrier* dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet. Wenn die Parameter außerhalb des `rc_load_carrier` Moduls über die *REST-API-Schnittstelle* (Abschnitt 6.3) eines anderen Moduls verwendet werden, sind sie durch den Präfix `load_carrier_` gekennzeichnet.

`model_tolerance` (*Modelltoleranz*)

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

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

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

`crop_distance` (*Cropping*)

setzt den Sicherheitsspielraum, um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren (siehe Abb. 5.5).

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

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

5.2.1.8 Services

Die angebotenen Services des LoadCarrier Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) oder der *rc_cube Web GUI* (Abschnitt 6.1) auf der Seite *LoadCarrier* unter dem Menüpunkt *Module* ausprobiert und getestet werden.

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

Die folgende Tabelle führt die möglichen Rückgabe-Codes an:

Tab. 5.7: Rückgabecodes der Services des LoadCarrier Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-4	Die maximal erlaubte Zeitspanne von 5.0 Sekunden für die interne Akquise der Bilddaten wurde überschritten.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern überschritten wurde.
-302	Es wurde mehr als ein Load Carrier an den `detect_load_carriers` oder `detect_filling_level` Service übergeben, aber nur einer wird unterstützt.
10	Die maximal speicherbare Anzahl an Load Carriern wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
102	Der erkannte Load Carrier ist leer.
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.

Alle Softwaremodule, die die Load Carrier Funktionalität anbieten, stellen folgende Services zur Verfügung.

set_region_of_interest

siehe *set_region_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

get_regions_of_interest

siehe *get_regions_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_regions_of_interest

siehe *delete_regions_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

set_region_of_interest_2d

siehe *set_region_of_interest_2d* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

get_regions_of_interest_2d

siehe *get_regions_of_interest_2d* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_regions_of_interest_2d

siehe *delete_regions_of_interest_2d* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

set_load_carrier

speichert einen Load Carrier auf dem *rc_cube*. Alle Load Carrier sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "load_carrier": {
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

Die Definition des Typs `load_carrier` wird in *Erkennung von Load Carriern* (Abschnitt 5.2.1.4) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

get_load_carriers

gibt die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier zurück. Werden keine `load_carrier_ids` angegeben, enthält die Serviceantwort alle gespeicherten Load Carrier.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


delete_load_carriers

löscht die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier. Alle zu löschenden Load Carrier müssen explizit angegeben werden.

Dieser Service kann wie folgt aufgerufen werden.

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

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


```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```


Die Definition der *Response* mit jeweiligen Datentypen ist:


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


detect_load_carriers

löst die Erkennung von Load Carriern aus, wie in *Erkennung von Load Carriern* (Abschnitt 5.2.1.4) beschrieben.

Dieser Service kann wie folgt aufgerufen werden.

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

Request:

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

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

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.1.6).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.1.6).

Optionale Serviceargumente:

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

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

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

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_load_carriers",
  "response": {
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        }
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


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

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

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

detect_filling_level

löst eine Load Carrier Füllstandserkennung aus, wie in *Füllstandserkennung* (Abschnitt 5.2.1.5) beschrieben.

Dieser Service kann wie folgt aufgerufen werden.

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

Request:

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

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


Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.1.6).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.1.6).

Optionale Serviceargumente:

`filling_level_cell_count`: Anzahl der Zellen im Füllstandsraster.

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

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

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

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


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


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


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

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

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

5.2.2 TagDetect

5.2.2.1 Einführung

Die TagDetect-Module sind optionale Module, die intern auf dem *rc_cube* laufen, und benötigen gesonderte *Lizenzen* (Abschnitt 7.5), die erworben werden müssen. Diese Lizenzen sind auf jedem *rc_cube*, der nach dem 01.07.2020 gekauft wurde, vorhanden.

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

Die Markererkennung besteht aus drei Schritten:

1. Markererkennung auf dem 2D-Bildpaar (siehe *Markererkennung*, Abschnitt 5.2.2.2).
2. Schätzung der Pose jedes Markers (siehe *Posenschätzung*, Abschnitt 5.2.2.3).
3. Wiedererkennung von bisher gesehenen Markern (siehe *Marker-Wiedererkennung*, Abschnitt 5.2.2.4).

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

QR-Code

Abb. 5.9: Beispiel eines QR-Codes

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

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

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

AprilTag

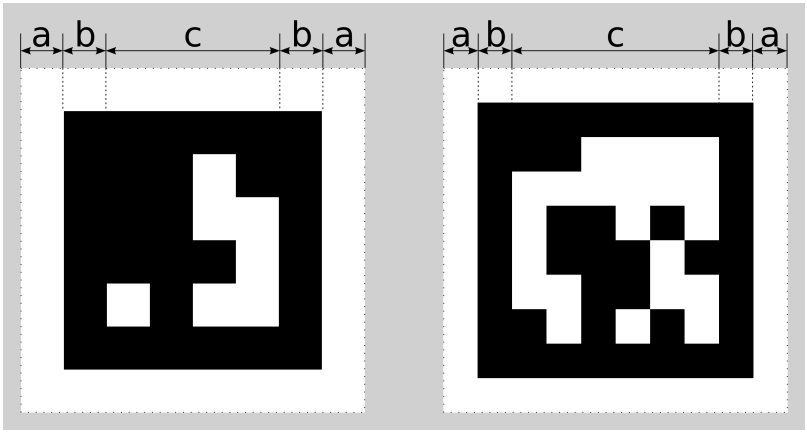


Abb. 5.10: Ein 16h5 Marker (links) und ein 36h11 Marker (rechts). AprilTags bestehen aus einem obligatorischen weißen (a) und schwarzen (b) Rahmen und einer variablen Menge an Datenmodulen (c).

AprilTags sind ähnlich zu QR-Codes. Sie wurden allerdings speziell zur robusten Identifikation auf weite Entfernungen entwickelt. Wie bei QR-Codes werden die „Pixel“ *Module* genannt. Abb. 5.10 veranschaulicht den Aufbau von AprilTags. Sie sind von einem obligatorischen weißen und schwarzen Rahmen

umschlossen, welcher jeweils ein Modul breit ist. Innen enthalten sie eine variable Menge an Datenmodulen. Anders als QR-Codes speichern sie keine benutzerdefinierten Informationen, sondern werden durch eine vordefinierte *Familie* und *ID* identifiziert. Die Tags in Abb. 5.10 sind zum Beispiel aus Familie 16h5 bzw. 35h11 und besitzen ID 0 bzw. 11. Alle unterstützten Familien werden in Tab. 5.8 aufgelistet.

Tab. 5.8: AprilTag-Familien

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

Die Zahl vor dem „h“ jeder Familie bezeichnet die Anzahl der Datenmodule, welche im Marker enthalten sind: Während ein 16h5 Marker 16 (4x4) Datenmodule enthält ((c) in Abb. 5.10), besteht ein 36h11 Marker aus 36 (6x6) Datenmodulen. Die Zahl hinter dem „h“ bezeichnet den Hamming-Abstand zwischen zwei Markern der Familie. Je höher, desto höher ist die Robustheit, aber desto weniger IDs stehen bei gleicher Anzahl an Datenmodulen zur Verfügung (siehe Tab. 5.8).

Der Vorteil von Familien mit weniger Datenmodulen (bspw. 16h5 im Vergleich zu 36h11) ist die niedrigere Auflösung der Marker. Jedes Modul ist somit größer, weshalb der Marker auf eine größere Distanz erkannt werden kann. Dies hat allerdings auch Nachteile: Zum einen stehen bei niedrigerer Zahl an Datenmodulen auch weniger IDs zur Verfügung. Wichtiger aber ist, dass die Robustheit der Markererkennung signifikant reduziert wird, da es zu einer höheren Falsch-Positiv-Rate kommt. Dies bedeutet, dass Marker verwechselt werden oder nicht existierende Marker in zufälliger Bildtextur oder im Bildrauschen erkannt werden.

Aus diesen Gründen empfehlen wir die Verwendung der 36h11-Familie und raten ausdrücklich von den Familien 16h5 und 25h7 ab. Letztgenannte Familien sollten nur benutzt werden, wenn eine große Erkennungsdistanz für die Anwendung unbedingt erforderlich ist. Jedoch ist die maximale Erkennungsdistanz nur ca. 25% größer, wenn anstelle der 36h11-Familie die 16h5-Familie verwendet wird.

Vorgenerierte AprilTags können von der AprilTag-Projektwebseite (https://april.eecs.umich.edu/software/apriltag.html) heruntergeladen werden. Jede Familie besteht aus mehreren PNGs, welche jeweils einen AprilTag enthalten, und einem PDF, welches jeden AprilTag auf einer eigenen Seite enthält. Jedes Pixel im PNG entspricht dabei einem Modul des AprilTags. Beim Drucken der Marker sollte darauf geachtet werden, den weißen Rand um den AprilTag mit einzuschließen – dieser ist sowohl in den PNGs also auch in den PDFs enthalten (siehe (a) in Abb. 5.10). Die Marker müssen außerdem ohne Interpolation auf die Druckgröße skaliert werden, sodass die scharfen Kanten erhalten bleiben.

Vergleich

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

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

5.2.2.2 Markererkennung

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

konfiguriert werden. Er hat ein Herunterskalieren des Stereo-Bildpaares vor der Markererkennung zur Folge. *Hoch* (`High`) ergibt die höchste maximale Erkennungssdistanz und Präzision, aber auch die längste Dauer der Erkennung. *Niedrig* (`Low`) führt zur kleinsten maximalen Erkennungsdistanz und Präzision, aber benötigt auch nur weniger als die halbe Zeit. *Mittel* (`Medium`) liegt dazwischen. Es sollte beachtet werden, dass dieser `quality`-Parameter keine Verbindung zum `quality`-Parameter des *Stereo-Matching* (Abschnitt 5.1.2) hat.

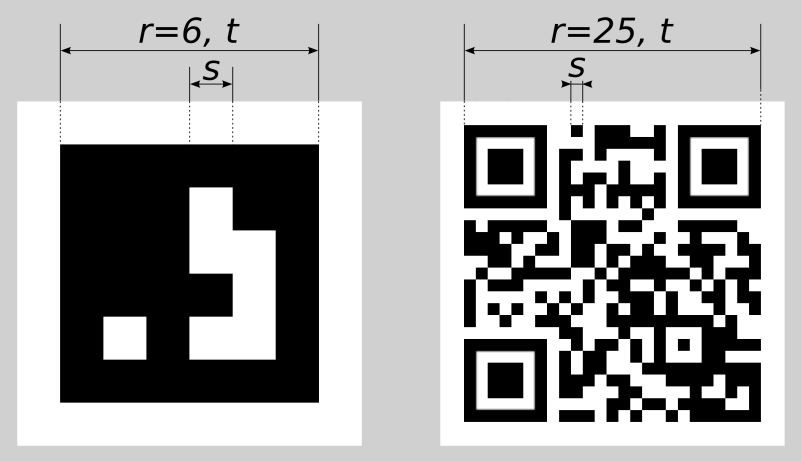


Abb. 5.11: Visualisierung der Modulgröße s, der Größe eines Markers in Modulen r und der Größe eines Markers in Metern t für QR-Codes (links) und AprilTags (rechts)

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

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

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

wobei f die *Brennweite* (Abschnitt 5.1.1.2) in Pixeln und s die Größe jedes Moduls in Metern bezeichnet. s kann leicht mit letztgenannter Formel berechnet werden, in welcher t der Markergröße in Metern und r der Breite des Markers in Modulen entspricht (bei AprilTags ohne den weißen Rahmen). Abb. 5.11 veranschaulicht diese Variablen. p bezeichnet die Zahl der Bildpixel pro Modul, welche für eine Erkennung erforderlich sind. Sie unterscheidet sich zwischen QR-Codes und AprilTags. Auch der Winkel des Markers zur Kamera und die Beleuchtung spielen eine Rolle. Ungefähre Werte für eine robuste Erkennung sind:

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

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

Tab. 5.9: Beispiele zur maximalen Erkennungsdistanz für QR-Codes mit einer Breite von $t = 4$ cm

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

Tab. 5.10: Beispiele zur maximalen Erkennungsdistanz für QR-Codes mit einer Breite von $t = 8$ cm

Markerbreite	Maximale Distanz
29 Module	0.49 m
21 Module	0.70 m

5.2.2.3 Posenschätzung

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

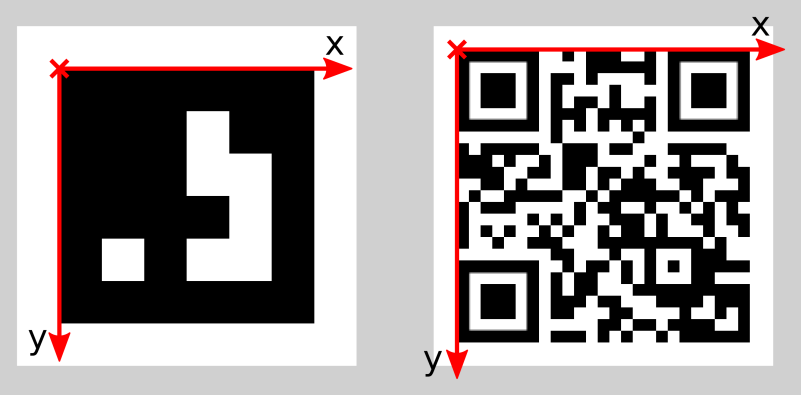


Abb. 5.12: Koordinatensysteme für AprilTags (links) bzw. QR-Codes (rechts)

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

Während der Posenschätzung wird auch die Größe des Markers geschätzt unter der Annahme, dass der Marker quadratisch ist. Bei QR-Codes bezieht sich die Größe auf den gesamten Marker, bei AprilTags dagegen nur auf den schwarzen Rand und nicht auf den äußeren weißen.

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

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

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

Unten stehende Tabellen enthalten grobe Angaben zur Präzision der geschätzten Posen von AprilTags und QR-Codes. Wir unterscheiden zwischen lateraler Präzision (also in x- und y-Richtung) und Präzision

in z-Richtung. Es wird angenommen, dass `quality` auf `High` gesetzt ist, und dass die Blickrichtung der Kamera parallel zur Normalen des Markers ist. Die Größe eines Markers hat keinen signifikanten Einfluss auf die Präzision in lateraler und z-Richtung. Im Allgemeinen verbessert ein größerer Marker allerdings die Präzision. Im Bezug auf die Präzision der Rotation, im speziellen um die x- und y-Achsen, übertreffen große Marker kleinere deutlich.

Tab. 5.11: Ungefähre Präzision der Pose von AprilTags

Distanz	*rc_visard* 65 - lateral	*rc_visard* 65 - z	*rc_visard* 160 - lateral	*rc_visard* 160 - z
0.3 m	0.4 mm	0.9 mm	0.4 mm	0.8 mm
1.0 m	0.7 mm	3.3 mm	0.7 mm	3.3 mm

Tab. 5.12: Ungefähre Präzision der Pose von QR-Codes

Distanz	*rc_visard* 65 - lateral	*rc_visard* 65 - z	*rc_visard* 160 - lateral	*rc_visard* 160 - z
0.3 m	0.6 mm	2.0 mm	0.6 mm	1.3 mm
1.0 m	2.6 mm	15 mm	2.6 mm	7.9 mm

5.2.2.4 Marker-Wiedererkennung

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

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

Die Marker-Wiedererkennung vergleicht die Positionen der Ecken der Marker im Kamera-Koordinatensystem, um identische Marker wiederzufinden. Marker werden als identisch angenommen, falls sie sich nicht oder nur geringfügig in diesem Koordinatensystem bewegt haben.

Über den `max_corner_distance`-Parameter kann der Benutzer festlegen, wie weit ein Marker sich zwischen zwei Erkennungsläufen bewegen darf, um als identisch zu gelten. Der Parameter definiert die maximale Distanz zwischen den Ecken zweier Marker, was in Abb. 5.13 dargestellt ist. Die euklidischen Abstände der vier zusammengehörenden Markerecken in 3D werden berechnet. Falls keiner dieser Abstände den Grenzwert überschreitet, gilt der Marker als wiedererkannt.

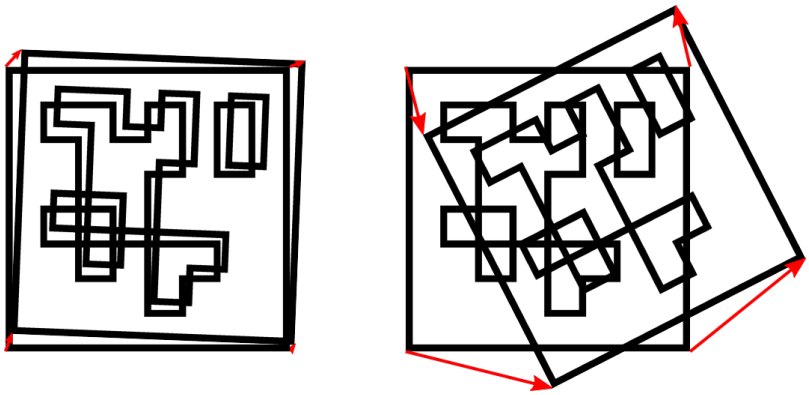


Abb. 5.13: Vereinfachte Darstellung der Marker-Wiedererkennung. Die euklidischen Abstände zwischen zusammengehörigen Markerecken in 3D werden berechnet (rote Pfeile).

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

5.2.2.5 Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das TagDetect-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.2.2.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene `pose_frame`-Werte können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben.
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

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

5.2.2.6 Parameter

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

Neben der *REST-API-Schnittstelle* (Abschnitt 6.3) stellen die TagDetect-Module außerdem Seiten in der Web GUI bereit, über welche sie manuell ausprobiert und konfiguriert werden können.

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

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.13: Laufzeitparameter des `rc_qr_code_detect`-Moduls

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

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

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

5.2.2.7 Statuswerte

Die TagDetect-Module melden folgende Statuswerte:

Tab. 5.14: Statuswerte der `rc_qr_code_detect`- und `rc_april_tag_detect`-Module

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`state`	Der aktuelle Zustand des Moduls
`tag_detection_time`	Berechnungszeit für die Markererkennung beim letzten Aufruf in Sekunden

Der Parameter `state` kann folgende Werte annehmen:

Tab. 5.15: Mögliche Zustände der TagDetect-Module

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

5.2.2.8 Services

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

`start`

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

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

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

stop

stoppt das Modul durch einen Übergang zu `IDLE`.

Dieser Übergang kann auf dem Zustand `RUNNING` und `FATAL` durchgeführt werden. Alle Marker-Wiedererkennungs-Informationen werden beim Stoppen gelöscht.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

restart

startet das Modul neu. Wenn im Zustand `RUNNING` oder `FATAL`, wird das Modul erst gestoppt und dann wieder gestartet. In `IDLE` wird das Modul nur gestartet.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

detect

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

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

Dieser Service kann wie folgt aufgerufen werden.

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

Request:

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Optionale Serviceargumente:

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

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

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

Zusätzlich zur ID kann auch die ungefähre Größe ($\pm 10\%$) eines Markers angegeben werden. Wie in *Posenschätzung* (Abschnitt 5.2.2.3) erklärt, verhilft dies Mehrdeutigkeiten aufzulösen, die in bestimmten Situationen auftreten können.

Das Feld `pose_frame` gibt an, ob die Posen im Kamera- oder im externen Koordinatensystem zurückgegeben werden (siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.2.2.5). Der Standardwert ist `camera`.

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


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


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

`tags` enthält alle erkannten Marker.

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

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

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

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

`size` wird auf die geschätzte Markergröße gesetzt. Bei AprilTags ist hier der weiße Rahmen nicht enthalten.

Das Feld `return_code` besteht aus einem Integer-Wert und einer optionalen Textnachricht. Erfolgreiche Service-Anfragen werden mit einem Wert von 0 quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet

wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Mögliche Werte für `return_code` sind in der folgenden Tabelle angegeben.

Code	Beschreibung
0	Erfolg
-1	Ein ungültiges Argument wurde übergeben.
-4	Die maximale Wartezeit auf ein Stereo-Bildpaar wurde überschritten.
-9	Die Lizenz ist ungültig.
-101	Ein interner Fehler trat während der Markererkennung auf.
-102	Ein Rückwärtssprung der Systemzeit trat auf
-103	Ein interner Fehler trat während der Posenschätzung auf.
-200	Ein schwerwiegender interner Fehler trat auf.
200	Mehrere Warnungen traten auf. Siehe die Auflistung in `message`.
201	Das Modul war nicht im Zustand `RUNNING`.

Marker können vom `detect`-Ergebnis aus mehreren Gründen ausgeschlossen werden, z.B. falls ein Marker nur in einem der Kamerabilder sichtbar war, oder falls die Posenschätzung fehlschlug. Diese herausgefilterten Marker werden im Log aufgelistet, auf welches wie in *Download der Logdateien* (Abschnitt 7.6) beschrieben zugegriffen werden kann.

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

Aufgrund von Änderungen der Systemzeit auf dem *rc_cube* können Zeitsprünge auftreten, sowohl vorwärts als auch rückwärts (siehe *Zeitsynchronisierung*, Abschnitt 6.6). Während Vorwärtssprünge keinen Einfluss auf die TagDetect-Module haben, invalidieren Rücksprünge die bereits empfangenen Bilder. Deshalb wird, wenn ein Rücksprung erkannt wird, Fehler -102 beim nächsten `detect`-Aufruf zurückgegeben. Dies geschieht auch, um den Benutzer darauf hinzuweisen, dass die Zeitstempel in der `detect`-Antwort ebenso zurückspringen werden.

save_parameters

Beim Aufruf dieses Services werden die aktuellen Parametereinstellungen des TagDetect-Moduls auf dem *rc_cube* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

reset_defaults

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

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.2.3 ItemPick und BoxPick

5.2.3.1 Einführung

Die ItemPick und BoxPick Module sind optional erhältliche Module, welche intern auf dem *rc_cube* laufen und gesonderte ItemPick- bzw. BoxPick-*Lizenzen* (Abschnitt 7.5) benötigen.

Die Module liefern eine gebrauchsfertige, modellfreie Perzeptionslösung, um robotische Pick-and-Place-Anwendungen für Vakuum-Greifsysteme zu realisieren. Dazu analysieren die Module die sichtbare 3D-Szene, extrahieren mittels Clustering-Verfahren ebene Greifflächen und berechnen daraus mögliche 3D-Greifposen für die Positionierung des Sauggreifers.

Darüber hinaus bieten beide Module:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_cube Web GUI* (Abschnitt 6.1)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *Region of Interest*, Abschnitt 5.3.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 5.2.1), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Fächern, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.
- einen Qualitätswert für jeden vorgeschlagenen Greifpunkt, der die Ebenheit der für das Greifen verfügbaren Oberfläche bewertet
- die Sortierung der berechneten Greifpunkte anhand des Gravitationsvektors und der Größe, sodass bei gestapelten Objekten zuerst die oberen gegriffen werden

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

5.2.3.2 Erkennung von Rechtecken (BoxPick)

Das BoxPick-Modul unterstützt die Erkennung von mehreren Objektmodellen (`item_models`) vom Typ (`type`) Rechteck (`RECTANGLE`). Jedes Rechteck ist durch seine minimale und maximale Größe definiert, wobei die minimale Größe kleiner als die maximale Größe sein muss. Die Abmessungen sollten relativ genau angegeben werden, um Fehldetektionen zu verhindern, jedoch eine gewisse Toleranz beinhalten, um Messunsicherheiten und mögliche Produktionsabweichungen zu berücksichtigen.

Optional können dem BoxPick-Modul folgende Informationen übergeben werden:

- die ID des Load Carriers, welcher die Objekte enthält
- ein Teilbereich innerhalb eines Load Carriers, in dem Objekte detektiert werden sollen
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, in der nach Objekten gesucht wird
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die gewählte Region of Interest im externen Koordinatensystem definiert ist

Die zurückgegebenen Objektposen sind relativ zum Mittelpunkt des Rechtecks definiert. Die z-Achse zeigt in Richtung der Kamera. Jedes erkannte Rechteck beinhaltet eine `uuid` (Universally Unique Identifier) und den Zeitstempel `timestamp` des ältesten Bildes, das für die Erkennung benutzt wurde.

5.2.3.3 Berechnung der Greifpunkte

Die ItemPick- und BoxPick-Module bieten einen Service, um Greifpunkte für Sauggreifer zu berechnen. Der Sauggreifer ist durch die Länge und Breite der Greiffläche definiert.

Das ItemPick-Modul identifiziert ebene Flächen in der Szene und unterstützt flexible und/oder deformierbare Objekte. Der Typ (`type`) dieser Objektmodelle (`item_models`) ist als unbekannt (`UNKNOWN`) definiert, da sie keine gebräuchliche geometrische Form aufweisen müssen. Optional kann eine minimale und maximale Größe angegeben werden.

Bei BoxPick werden die Greifpunkte auf den erkannten Rechtecken berechnet (siehe *Erkennung von Rechtecken (BoxPick)*, Abschnitt 5.2.3.2).

Optional können den Modulen zu einer Greifpunktberechnung weitere Informationen übergeben werden:

- die ID des Load Carriers, welcher die zu greifenden Objekte enthält
- ein Unterabteil (`load_carrier_compartment`) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteil*, Abschnitt 5.2.1.3).
- die ID der 3D Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die 3D Region of Interest, innerhalb der Greifpunkte berechnet werden
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 5.2.3.4) gegeben.

Ein vom ItemPick- oder BoxPick-Modul ermittelter Greifpunkt repräsentiert die empfohlene Pose des TCP (Tool Center Point) des Sauggreifers. Der Greifpunkt `type` ist immer auf `SUCTION` gesetzt. Für jeden Greifpunkt liegt der Ursprung der Greifpose `pose` im Mittelpunkt der größten von der jeweiligen Greiffläche umschlossenen Ellipse. Die Orientierung des Greifpunkts ist ein rechtshändiges Koordinatensystem, sodass die z-Achse orthogonal zur Greiffläche in das zu greifende Objekt zeigt und die x-Achse entlang der längsten Ausdehnung ausgerichtet ist.

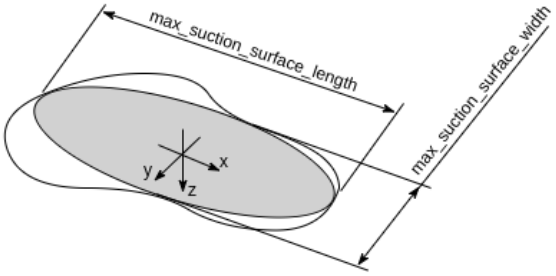


Abb. 5.14: Veranschaulichung eines berechneten Greifpunktes mit Greifpose und der zugehörigen Ellipse, welche die Greiffläche bestmöglich beschreibt.

Zusätzlich enthält jeder Greifpunkt die Abmessungen der maximal verfügbaren Greiffläche, die als Ellipse mit den Achslängen `max_suction_surface_length` und `max_suction_surface_width` beschrieben wird. Der Nutzer kann Greifpunkte mit zu kleinen Greifflächen herausfiltern, indem die minimalen Abmessungen der Greiffläche, die vom Sauggreifer benötigt wird, angegeben werden.

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

Jeder Greifpunkt enthält auch einen Qualitätswert (`quality`), der einen Hinweis auf die Ebenheit der Greiffläche gibt. Dieser Wert reicht von 0 bis 1, wobei höhere Werte für eine ebenere rekonstruierte Oberfläche stehen.

Jeder berechnete Greifpunkt lässt sich anhand einer `uuid` (Universally Unique Identifier) eindeutig identifizieren und enthält zusätzlich den Zeitstempel der ältesten Bildaufnahme, auf der die Greifpunktberechnung durchgeführt wurde.

Die Sortierung der Greifpunkte basiert auf einer Kombination von

- dem Abstand des Greifpunkts von der Kamera entlang der Gravitationsrichtung, und
- der Größe des zu greifenden Objekts.

Die Greifpunkte werden so zurückgeliefert, dass bei gestapelten Objekten zuerst die oberen gegriffen werden und größere Objekte bevorzugt werden.

5.2.3.4 Wechselwirkung mit anderen Modulen

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

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

Stereokamera und Stereo-Matching

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

- die rektifizierten Bilder des *Stereokamera*-Moduls (`rc_stereocamera`, Abschnitt 5.1.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching*-Moduls (`rc_stereomatching`, Abschnitt 5.1.2)

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

Schätzung der Gravitationsrichtung

Jedes Mal, wenn eine Load Carrier Erkennung oder Greifpunktberechnung durchgeführt wird, schätzt das ItemPick- bzw. BoxPick-Modul die Gravitationsrichtung basierend auf den IMU-Daten des *rc_visard*.

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

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_cube* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.3.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.1.2.5), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.3.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren (siehe *Stereokamera-Parameter*, Abschnitt 5.1.1.4).

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, können die ItemPick- und BoxPick-Module automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.2.3.7) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das ItemPick- oder BoxPick-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

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

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

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung der ItemPick und BoxPick Module aktiviert werden, indem die ID des benutzten Greifers und optional ein Greif-Offset an den `compute_grasps` Service übergeben werden. Der Greifer muss im CollisionCheck-Modul definiert werden (siehe *Erstellen*

eines Greifers (Abschnitt 5.3.3.2)) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3) gegeben.

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

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 5.3.3.4) beschrieben.

5.2.3.5 Parameter

Die ItemPick- und BoxPick-Module werden in der REST-API als `rc_itempick` und `rc_boxpick` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf den Seiten *BoxPick* bzw. *ItemPick* (unter dem Menüpunkt *Module*) dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) ändern.

Übersicht über die Parameter

Diese Softwaremodule bieten folgende Laufzeitparameter:

Tab. 5.16: Anwendungsspezifische Laufzeitparameter der `rc_itempick` und `rc_boxpick` Module

Name	Typ	Min.	Max.	Default	Beschreibung
`max_grasps`	int32	1	20	5	Maximale Anzahl von bereitgestellten Greifpunkten
`prefer_splits`	bool	false	true	false	**Nur für rc_boxpick**. Gibt an, ob Rechtecke in kleinere Rechtecke gesplittet werden sollen, falls möglich

Tab. 5.17: Laufzeitparameter der `rc_itempick` und `rc_boxpick` Module für die Load Carrier Erkennung

Name	Typ	Min.	Max.	Default	Beschreibung
`load_carrier_crop_distance`	float64	0.0	0.05	0.005	Sicherheitsspielraum um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren
`load_carrier_model_tolerance`	float64	0.003	0.025	0.008	Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen.

Tab. 5.18: Laufzeitparameter der `rc_itempick` und `rc_boxpick` Module für das Clustering-Verfahren

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

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf den *ItemPick*- bzw. *BoxPick*-Seiten in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_grasps` (*Anzahl Greifpunkte*)

ist die maximale Anzahl von bereitgestellten Greifpunkten.

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

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

`prefer_splits` (Nur für BoxPick, *Splitting bevorzugen*)

bestimmt, ob Rechtecke in kleinere Rechtecke aufgesplittet werden, falls die kleineren Rechtecke ebenfalls den angegebenen Objektmodellen entsprechen. Dieser Parameter sollte auf true gesetzt werden, wenn Boxen dicht beieinander liegen, und die Objektmodelle auch zu einem Rechteck der Größe von zwei angrenzenden Boxen passen. Wenn dieser Parameter auf False steht, werden in solch einem Fall die Rechtecke bevorzugt, die sich aus zwei angrenzenden Boxen ergeben.

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

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

`cluster_max_dimension` (Nur für ItemPick, *Maximale Größe*)

is the maximum allowed diameter for a cluster in meters. Clusters with a diameter larger than this value are not used for grasp computation.

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

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

`cluster_max_curvature` (*Maximale Krümmung*)

ist die maximal erlaubte Krümmung für Greifflächen. Je kleiner dieser Wert ist, desto mehr mögliche Greifflächen werden in kleinere Flächen mit weniger Krümmung aufgeteilt.

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

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

`clustering_patch_size` (Nur für ItemPick, *Patchgröße*)

ist die Pixelgröße der Patches für die Unterteilung des Tiefenbildes im ersten Clustering-Schritt.

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

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

`clustering_discontinuity_factor` (*Unstetigkeitsfaktor*)

beschreibt die erlaubte Unebenheit von Greifflächen. Je kleiner dieser Wert ist, umso mehr werden mögliche Greifflächen in kleinere Flächen mit weniger Unebenheiten aufgeteilt.

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

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

`clustering_max_surface_rmse` (*Maximaler RMSE*)

ist die maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern.

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

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

load_carrier_model_tolerance

siehe *Parameter der Load Carrier Funktionalität* (Abschnitt 5.2.1.7).

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

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

load_carrier_crop_distance

siehe *Parameter der Load Carrier Funktionalität* (Abschnitt 5.2.1.7).

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

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

5.2.3.6 Statuswerte

Statuswerte der `rc_itempick` und `rc_boxpick` Module:

Tab. 5.19: Statuswerte der `rc_itempick` und `rc_boxpick` Module

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

Folgende `state`-Werte werden gemeldet.

Tab. 5.20: Mögliche Werte für den Zustand der ItemPick und BoxPick Module

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier zu erkennen und Greifpunkte zu berechnen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

5.2.3.7 Services

Die angebotenen Services von `rc_itempick` bzw. `rc_boxpick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) oder der *rc_cube Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

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

Die folgende Tabelle führt die möglichen Rückgabe-Codes an:

Tab. 5.21: Rückgabecodes der Services des ItemPick- bzw. BoxPick-Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-4	Die maximal erlaubte Zeitspanne von 5.0 Sekunden für die interne Akquise der Bilddaten wurde überschritten.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern oder ROIs überschritten wurde.
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
-301	Für die Anfrage zur Greifpunktberechnung `compute_grasps` wurden mehrere Objektmodelle (`item_models`) vom Typ `UNKNOWN` übergeben.
-302	Mehr als ein Load Carrier wurde für die Anfrage `detect_load_carriers` oder `detect_filling_level` angegeben. Momentan wird nur ein Load Carrier gleichzeitig unterstützt.
10	Die maximal speicherbare Anzahl an Load Carriern oder ROIs wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` oder `set_region_of_interest` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Es wurden keine gültigen Greifflächen in der Szene gefunden.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision mit dem Load Carrier.
200	Das Modul ist im Zustand `IDLE`.
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.
400	Der Serviceanfrage `compute_grasps` wurden keine Objektmodelle (`item_models`) als Argumente mitgegeben.

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

start

versetzt das ItemPick-Modul in den Zustand `RUNNING`. Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `RUNNING` unterscheidenden Zustand zurück.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

stop

stoppt das Modul und versetzt es in den Zustand `IDLE`. Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

set_region_of_interest

siehe *set_region_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

get_regions_of_interest

siehe *get_regions_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_regions_of_interest

siehe *delete_regions_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

set_load_carrier

siehe *set_load_carrier* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

get_load_carriers

siehe *get_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_load_carriers

siehe *delete_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_load_carriers

siehe *detect_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_filling_level

siehe *detect_filling_level* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_items (nur BoxPick)

löst die Erkennung von Rechtecken aus, wie in *Erkennung von Rechtecken (BoxPick)* (Abschnitt 5.2.3.2) beschrieben.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/detect_items
```

Request:

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64"
          }
        },
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.3.4).

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

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.3.4).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Ob-

jekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteil*, Abschnitt 5.2.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, in der nach Objekten gesucht wird.

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_items",
  "response": {
    "items": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rectangle": {
          "x": "float64",
          "y": "float64"
        },
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


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

`items`: Liste von erkannten Rechtecken.

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

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

`compute_grasps` (für ItemPick)

löst die Erkennung von Greifpunkten für einen Sauggreifer aus, wie in *Berechnung der Greifpunkte* (Abschnitt 5.2.3.3) beschrieben.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/compute_grasps
```

Request:

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


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


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "unknown": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        }
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```


Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.3.4).

`suction_surface_length`: Länge der Greiffläche des verwendeten Vakuum-Greifsystems.

`suction_surface_width`: Breite der Greiffläche des verwendeten

Vakuum-Greifsystems.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.3.4).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu greifenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteil*, Abschnitt 5.2.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, innerhalb der Greifpunkte berechnet werden.

`item_models`: Liste von unbekannten Objekten mit minimaler und maximaler Größe, wobei die minimale Größe kleiner als die maximale Größe sein muss. Nur ein Objekt `item_model` vom Typ `UNKNOWN` wird aktuell unterstützt.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3)

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
      {
        "item_uuid": "string",
        "max_suction_surface_length": "float64",
        "max_suction_surface_width": "float64",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "quality": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        }
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


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

`grasps`: sortierte Liste von Sauggreifpunkten.

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

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

`compute_grasps` (für BoxPick)

löst die Erkennung von Rechtecken und Berechnung von Greifposen für diese Rechtecke aus, wie in *Berechnung der Greifpunkte* (Abschnitt 5.2.3.3) beschrieben.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/compute_grasps
```

Request:

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


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.3.4).

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

`suction_surface_length`: Länge der Greiffläche des verwendeten Vakuum-Greifsystems.

`suction_surface_width`: Breite der Greiffläche des verwendeten Vakuum-Greifsystems.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.3.4).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu greifenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteil*, Abschnitt 5.2.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, innerhalb der Greifpunkte berechnet werden.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3)

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
      {
        "item_uuid": "string",
        "max_suction_surface_length": "float64",
        "max_suction_surface_width": "float64",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "quality": "float64",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "items": [
    {
      "grasp_uuids": [
        "string"
      ],
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rectangle": {
        "x": "float64",
        "y": "float64"
      },
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


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

`items`: Liste von erkannten Rechtecken.

`grasps`: sortierte Liste von Sauggreifpunkten.

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

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

save_parameters

Beim Aufruf dieses Services werden die aktuellen Parametereinstellungen des ItemPick- oder BoxPick-Moduls auf dem *rc_cube* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden. Bei Firmware-Updates oder -Wiederherstellungen werden sie jedoch wieder auf den Standardwert gesetzt.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:


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


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

reset_defaults

Hiermit werden die Werkseinstellungen der Parameter dieses Moduls wiederhergestellt und angewandt („factory reset“). Dies betrifft nicht die konfigurierten ROIs und Load Carrier.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.2.4 SilhouetteMatch

5.2.4.1 Einführung

Das SilhouetteMatch-Modul ist ein optionales Modul, welches intern auf dem *rc_cube* läuft, und benötigt eine eigene *Lizenz* (Abschnitt 7.5), welche erworben werden muss.

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

Für jedes Objekt, das mit dem SilhouetteMatch-Modul erkannt werden soll, wird ein Template benötigt. Roboception bietet hierfür einen Template-Generierungsservice auf ihrer Website (https://roboception.com/de/template-request-de/) an, auf der der Benutzer CAD-Daten oder mit dem System aufgenommene Daten hochladen kann, um Templates generieren zu lassen.

Templates bestehen aus den prägnanten Kanten eines Objekts. Die Kanten des Templates werden mit den erkannten Kanten im linken und rechten Kamerabild abgeglichen, wobei die Größe der Objekte und deren Abstand zur Kamera mit einbezogen wird. Die Posen der erkannten Objekte werden zurückgegeben und können beispielsweise benutzt werden, um die Objekte zu greifen.

Das SilhouetteMatch-Modul bietet:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_cube Web GUI* (Abschnitt 6.1)
- eine *REST-API-Schnittstelle* (Abschnitt 6.3) und eine *KUKA Ethernet KRL Schnittstelle* (Abschnitt 6.4)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche des Kamerabilds auszuwählen (siehe *Setzen einer Region of Interest*, Abschnitt 5.2.4.3)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 5.2.1), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen

- die Definition von Greifpunkten für jedes Template über eine interaktive Visualisierung in der Web GUI
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern
- eine Sortierung der Greifpunkte nach ihrer Erreichbarkeit, sodass die Greifpunkte, die am dichtesten an der Kamera liegen und die geringste Rotation des TCP im Vergleich zu seiner bevorzugten Orientierung hervorrufen, zuerst zurückgeliefert werden

Taugliche Objekte

Das SilhouetteMatch-Modul ist für Objekte ausgelegt, die prägnante Kanten auf einer Ebene besitzen, welche parallel zu der Basisebene ist, auf der die Objekte liegen. Das trifft beispielsweise auf flache, nicht-transparente Objekte zu, wie gefräste, lasergeschnittene oder wasserstrahlgeschnittene Teile. Komplexere Objekte können auch erkannt werden, solange sie prägnante Kanten auf einer Ebene besitzen, z.B. ein gedrucktes Muster auf einer ebenen Fläche.

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

Taugliche Szene

Eine für das SilhouetteMatch-Modul taugliche Szene muss folgende Bedingungen erfüllen:

- Die zu erkennenden Objekte müssen, wie oben beschrieben, tauglich für das SilhouetteMatch-Modul sein.
- Nur Objekte, die zum selben Template gehören, dürfen gleichzeitig sichtbar sein (sortenrein). Falls auch andere Objekte sichtbar sind, muss eine passende Region of Interest (ROI) festgelegt werden.
- Alle sichtbaren Objekte befinden sich auf einer gemeinsamen Basisebene, welche kalibriert werden muss.
- Die Verkippung der Basisebene zur Blickrichtung der Kamera darf 10 Grad nicht übersteigen.
- Die Objekte sind weder teilweise noch komplett verdeckt.
- Alle sichtbaren Objekte liegen richtig herum.
- Die Objektkanten, welche abgeglichen werden sollen, sind sowohl im linken als auch im rechten Kamerabild zu sehen.

5.2.4.2 Kalibrierung der Basisebene

Bevor Objekte erkannt werden können, muss die Basisebene kalibriert werden. Hierbei wird die Distanz und der Winkel der Ebene, auf welcher die Objekte liegen, gemessen und persistent auf dem *rc_cube* gespeichert.

Durch die Trennung der Kalibrierung der Basisebene von der eigentlichen Objekterkennung werden beispielsweise Szenarien ermöglicht, in denen die Basisebene zeitweise verdeckt ist. Darüber hinaus wird die Berechnungszeit der Objekterkennung für Szenarien verringert, in denen die Basisebene für eine gewisse Zeit fixiert ist – die Basisebene muss in diesem Fall nicht fortlaufend neu detektiert werden.

Die Kalibrierung der Basisebene kann mit drei unterschiedlichen Verfahren durchgeführt werden, auf die im Folgenden näher eingegangen wird:

- AprilTag-basiert
- Stereo-basiert

- Manuell

Die Kalibrierung ist erfolgreich, solange der Normalenvektor der Basisebene höchstens 10 Grad gegen die Blickrichtung der Kamera verkippt ist. Eine erfolgreiche Kalibrierung wird persistent auf dem *rc_cube* gespeichert, bis sie entweder gelöscht wird oder eine neue Kalibrierung durchgeführt wird.

Bemerkung: Um Datenschutzproblemen entgegenzuwirken, wird die Visualisierung der Kalibrierung der Basisebene nach einem Neustart des *rc_cube* verschwommen dargestellt.

In Szenarien, in denen die Basisebene nicht direkt kalibriert werden kann, ist es auch möglich, zu einer zur Basisebene parallel liegenden Ebene zu kalibrieren. In diesem Fall kann der Parameter `offset` benutzt werden, um die geschätzte Ebene auf die eigentliche Basisebene zu verschieben. Der Parameter `offset` gibt die Distanz in Metern an, um welche die geschätzte Ebene in Richtung der Kamera verschoben wird.

In der REST-API ist eine Ebene durch eine Normale (`normal`) und einen Abstand (`distance`) definiert. `normal` ist ein normalisierter 3-Vektor, welcher die Normale der Ebene spezifiziert. Die Normale zeigt immer von der Kamera weg. `distance` repräsentiert den Abstand der Ebene von der Kamera in Richtung der Normale. `normal` und `distance` können auch als a, b, c, bzw. d der Ebenengleichung interpretiert werden:

$$ax + by + cz + d = 0$$

AprilTag-basierte Kalibrierung der Basisebene

Die AprilTag-Erkennung (siehe *TagDetect*, Abschnitt 5.2.2) wird benutzt, um AprilTags in der Szene zu finden und eine Ebene durch diese zu legen. Mindestens drei AprilTags müssen so auf der Basisebene platziert werden, dass sie im linken und rechten Kamerabild zu sehen sind. Die AprilTags sollten ein möglichst großes Dreieck aufspannen. Je größer das Dreieck ist, desto höher wird die Genauigkeit der Schätzung der Basisebene. Diese Methode sollte benutzt werden, wenn die Basisebene untexturiert und kein externer Projektor mit Zufallsmuster angeschlossen ist. Diese Kalibriermethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 6.3) als auch über die *rc_cube* Web GUI verfügbar.

Stereo-basierte Kalibrierung der Basisebene

Die 3D-Punktwolke, welche vom Stereo-Matching-Modul berechnet wird, wird benutzt um eine Ebene in den 3D-Punkten zu finden. Die Region of Interest (ROI) sollte für diese Methode deshalb so gewählt werden, dass nur die relevante Basisebene eingeschlossen wird. Der Parameter `plane_preference` erlaubt es auszuwählen, ob die zur Kamera am nächsten gelegene oder die von der Kamera am weitesten entfernte Ebene als Basisebene benutzt wird. Die am nächsten gelegene Ebene kann in Szenarien ausgewählt werden, in denen die Basisebene vollständig von Objekten verdeckt wird oder für die Kalibrierung nicht erreichbar ist. Diese Methode sollte benutzt werden, wenn die Basisebene texturiert ist oder ein Projektor mit Zufallsmuster angeschlossen ist. Diese Kalibriermethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 6.3) als auch über die *rc_cube* Web GUI verfügbar.

Manuelle Kalibrierung der Basisebene

Die Basisebene kann manuell gesetzt werden, falls die Parameter bekannt sind – beispielsweise von einer vorangegangenen Kalibrierung. Diese Kalibriermethode ist nur über die *REST-API-Schnittstelle* (Abschnitt 6.3) und nicht über die *rc_cube* Web GUI verfügbar.

5.2.4.3 Setzen einer Region of Interest

Falls Objekte nur in einem Teil des Sichtfelds der Kamera erkannt werden sollen, kann eine 2D Region of Interest (ROI) gesetzt werden, wie in *Region of Interest* (Abschnitt 5.3.2.2) beschrieben wird.

5.2.4.4 Setzen von Greifpunkten

Um das SilhouetteMatch-Modul direkt in einer Roboteranwendung zu nutzen, können für jedes Template Greifpunkte definiert werden. Ein Greifpunkt repräsentiert die gewünschte Position und Orientierung des Roboter-TCPs (Tool Center Point), mit der das Objekt gegriffen werden kann (siehe Abb. 5.15).

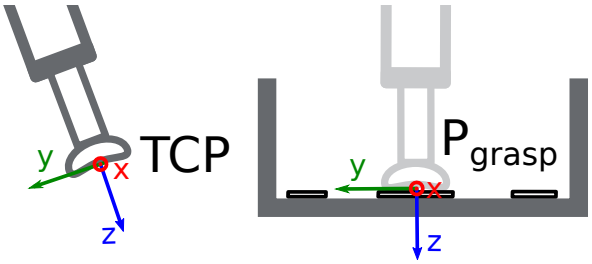


Abb. 5.15: Definition von Greifpunkten bezogen auf den Roboter-TCP

Jeder Greifpunkt enthält eine `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, die ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und die Greifpose (`pose`) im Koordinatensystem des Templates. Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 6.3), oder über die interaktive Visualisierung in der Web GUI definiert werden. Der *rc_cube* kann bis zu 50 Greifpunkte pro Template speichern.

Setzen von Greifpunkten in der Web GUI

Die *rc_cube* Web GUI bietet eine interaktive und intuitive Möglichkeit, Greifpunkte für Objekt-Templates zu setzen. Im ersten Schritt muss das Objekt-Template auf den *rc_cube* hochgeladen werden. Das kann über die *SilhouetteMatch*-Seite (unter dem Menüpunkt *Module*) in der Web GUI erfolgen, indem im Abschnitt *Templates und Greifpunkte* auf *neues Template hinzufügen* geklickt wird. Wenn der Upload abgeschlossen ist, erscheint ein Fenster mit einer 3D-Visualisierung des Templates, in dem Greifpunkte hinzugefügt oder existierende Greifpunkte bearbeitet werden können. Dasselbe Fenster erscheint, wenn ein vorhandenes Template bearbeitet wird. Wenn das Template ein Kollisionsmodell oder ein Visualisierungsmodell enthält, wird dieses Modell ebenfalls angezeigt.

Dieses Fenster bietet zwei Möglichkeiten, um Greifpunkte zu setzen:

1. **Greifpunkte manuell hinzufügen**: Durch Klicken auf das + Symbol wird ein neuer Greifpunkt im Ursprung des Templates angelegt. Diesem Greifpunkt kann ein eindeutiger Name gegeben werden, der seiner ID entspricht. Die gewünschte Pose des Greifpunkts im Koordinatensystem des Templates kann in den Feldern für *Position* und *Roll/Pitch/Yaw* eingegeben werden. Die Greifpunkte können frei platziert werden, auch außerhalb oder innerhalb des Templates, und werden mit ihrer Orientierung zur Überprüfung in der Visualisierung veranschaulicht.
2. **Greifpunkte interaktiv hinzufügen**: Greifpunkte können interaktiv zu einem Template hinzugefügt werden, indem zuerst auf den Button *Greifpunkt hinzufügen* oben links in der Visualisierung und anschließend auf den gewünschten Punkt auf dem Template geklickt wird. Wenn ein 3D-Modell angezeigt wird, wird er Greifpunkt wird an die Oberfläche des Modells angeheftet, andernfalls an die Template-Oberfläche. Die Orientierung des Greifpunkts entspricht einem rechtshändigen Koordinatensystem, sodass die z-Achse senkrecht auf der Template-Oberfläche steht und in das Template hineingerichtet ist. Die Position und Orientierung des Greifpunkts im Koordinatensystem des Templates ist auf der rechten Seite angezeigt. Die Position und Orientierung des Greifpunkts kann auch interaktiv verändert werden. Für den Fall, dass *An Oberfläche anheften* in der Visualisierung aktiv ist (das ist der Standardwert), kann der Greifpunkt durch Klicken auf *Verschieben* und anschließendes Klicken auf den Greifpunkt über die Oberfläche des Modells oder des Templates zur gewünschten Position bewegt werden. Die Orientierung des Greifpunkts um die Oberflächennormale kann ebenfalls interaktiv verändert werden, in dem auf *Rotieren* geklickt wird und anschließend der Greifpunkt mit der Maus rotiert wird. Wenn *An Oberfläche anheften*

nicht aktiv ist, kann der Greifpunkt mit der Maus frei in allen drei Raumrichtungen verschoben und rotiert werden.

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

Setzen von Greifpunkten über die REST-API

Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 6.3) mithilfe des `set_grasp` oder `set_all_grasps` Services gesetzt werden (siehe *Services*, Abschnitt 5.2.4.10). Im SilhouetteMatch-Modul besteht ein Greifpunkt aus der `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, der ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und der Greifpose (`pose`). Die Pose ist im Koordinatensystem des Templates angegeben und besteht aus einer Position (`position`) in Metern und einer Orientierung (`orientation`) als Quaternion.

5.2.4.5 Setzen der bevorzugten TCP-Orientierung

Das SilhouetteMatch-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des Greifers oder TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die *SilhouetteMatch*-Seite in der Web GUI gesetzt werden. Die resultierende Richtung der z-Achse des TCP wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann. Weiterhin wird die bevorzugte Orientierung genutzt, um die erreichbaren Greifpunkte zu sortieren. Die Sortierung basiert auf einer Kombination von

- dem Abstand von der Kamera entlang der z-Achse der bevorzugten TCP-Orientierung, und
- der Rotationsdifferenz zwischen der bevorzugten TCP-Orientierung und der Orientierung des Greifpunkts.

Greifpunkte, die am dichtesten an der Kamera liegen und die geringste TCP-Rotation hervorrufen, werden zuerst zurückgeliefert.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und der Sensor am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden, damit die bevorzugte Orientierung zur Filterung und Sortierung der Greifpunkte auf den erkannten Objekten genutzt werden kann. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera als die bevorzugte TCP-Orientierung genutzt.

5.2.4.6 Objekterkennung

Objekte können erst nach einer erfolgreichen Kalibrierung der Basisebene erkannt werden. Es muss sichergestellt werden, dass sich Position und Orientierung der Basisebene zwischen Kalibrierung und Objekterkennung nicht ändern. Anderenfalls muss die Kalibrierung erneuert werden.

Um eine Objekterkennung durchzuführen, müssen im Allgemeinen die folgenden Serviceargumente an das SilhouetteMatch-Modul übergeben werden:

- das Template des Objekts, welches in der Szene erkannt werden soll
- das Koordinatensystem, in dem die Posen der detektierten Objekte zurückgegeben werden sollen (siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.2.4.7)

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

- ein Versatz, falls Objekte nicht direkt auf der Basisebene liegen, sondern auf einer zu dieser parallelen Ebene. Der Versatz bezeichnet die Distanz beider Ebenen in Richtung der Kamera. Wenn dieser Wert nicht gesetzt wird, wird ein Versatz von 0 angenommen.
- die ID des Load Carriers, der die zu detektierenden Objekte enthält

- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, innerhalb der Objekte erkannt werden sollen. Wenn keine ROI gesetzt wird, werden Objekte im gesamten Kamerabild gesucht.
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die bevorzugte TCP-Orientierung im externen Koordinatensystem angegeben ist
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 5.2.4.7) gegeben.

Im *Ausprobieren*-Abschnitt der Seite *SilhouetteMatch* der Web GUI kann die Objektdetektion ausprobiert werden. Das Ergebnis wird, wie in Abb. 5.16 dargestellt, visualisiert.

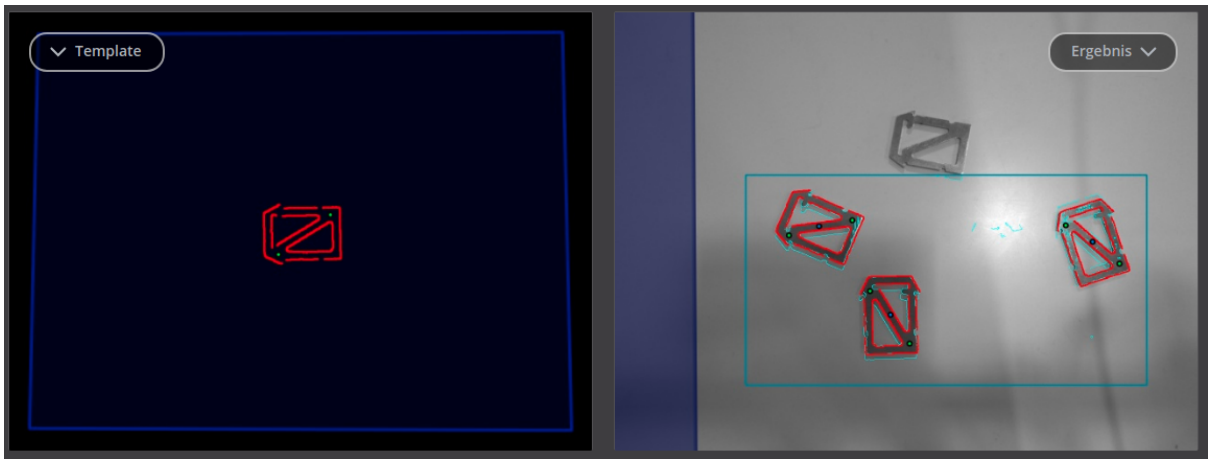


Abb. 5.16: Ergebnisbild des SilhouetteMatch-Moduls, wie über die Web GUI dargestellt

Das linke Bild zeigt die kalibrierte Basisebene in blau und das zu erkennende Template in rot mit den Greifpunkten (siehe *Setzen von Greifpunkten*, Abschnitt 5.2.4.4) in grün. Das Template wird passend zu Abstand und Verkippung der Basisebene verformt dargestellt.

Das rechte Bild zeigt das Detektionsergebnis. Die blauschattierte Fläche auf der linken Seite markiert den Teil des linken Kamerabilds, welcher nicht mit dem rechten Kamerabild überlappt. In diesem Bereich können keine Objekte erkannt werden. Die gewählte Region of Interest wird als petrolfarbenes Rechteck dargestellt. Erkannte Kanten im Bild werden in hellem Blau und erkannte Objekte (`instances`) in rot visualisiert. Blaue Punkte markieren jeweils den Ursprung der detektierten Objekte, wie im Template festgelegt. Erreichbare Greifpunkte sind als grüne Punkte dargestellt. Nicht erreichbare Greifpunkte werden als rote Punkte visualisiert (nicht im Bild dargestellt).

Die Posen der Objektursprünge werden im gewählten Koordinatensystem zurückgegeben. Wenn das ausgewählte Template auch Greifpunkte hat, dann wird zusätzlich zu den erkannten Objekten auch eine Liste von Greifpunkten (`grasps`) für alle erkannten Objekte zurückgegeben. Die Greifpunkte in dieser Liste sind nach ihrer Erreichbarkeit sortiert (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.2.4.5). Die Posen der Greifpunkte sind im gewünschten Koordinatensystem angegeben. Die erkannten Objekte und die Greifpunkte können einander über ihre UUIDs zugeordnet werden. Falls das Template eine kontinuierliche Rotationssymmetrie aufweist, besitzen alle Ergebnisposen die gleiche Orientierung. Für nicht-rotationssymmetrische Objekte richtet sich die Orientierung nach der Normalen der Basisebene.

Die Detektionsergebnisse und Berechnungszeiten werden durch Laufzeitparameter beeinflusst, welche weiter unten aufgezählt und beschrieben werden. Unsachgemäße Parameterwerte können zu Zeitüberschreitungen im Detektionsprozess des SilhouetteMatch-Moduls führen.

5.2.4.7 Wechselwirkung mit anderen Modulen

Die folgenden auf dem *rc_cube* laufenden Module liefern Daten für das SilhouetteMatch-Modul oder haben Einfluss auf die Datenverarbeitung.

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

Stereokamera und Stereo-Matching

Das SilhouetteMatch-Modul verarbeitet intern die rektifizierten Bilder des *Stereokamera*-Moduls (`rc_stereocamera`, Abschnitt 5.1.1). Es sollte deshalb auf eine passende Belichtungszeit geachtet werden, um optimale Ergebnisse zu erhalten.

Für die Kalibrierung der Basisebene mit der Stereo-Methode wird das Disparitätsbild des *Stereo-Matching*-Moduls (`rc_stereomatching`, Abschnitt 5.1.2) verarbeitet. Abgesehen davon sollte das Stereo-Matching-Modul nicht parallel zum SilhouetteMatch-Modul ausgeführt werden, da die Laufzeit der Objekterkennung sonst negativ beeinflusst wird.

Für beste Ergebnisse wird empfohlen, *Glättung* (Abschnitt 5.1.2.5) für *Stereo-Matching* zu aktivieren.

IOControl und Projektor-Kontrolle

Wenn der *rc_cube* in Verbindung mit einem externen Musterprojektor und dem Modul *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.3.4) betrieben wird, sollte der Projektor für die stereobasierte Kalibrierung der Basisebene benutzt werden.

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

Hand-Auge-Kalibrierung

Wenn die Kamera zu einem Roboter kalibriert ist, kann das SilhouetteMatch-Modul die Ergebnisposen automatisch im Roboterkoordinatensystem liefern. Für die *Services* (Abschnitt 5.2.4.10) des SilhouetteMatch-Moduls kann das Referenzkoordinatensystem aller Posen über das Argument `pose_frame` angegeben werden.

Es kann zwischen den folgenden zwei Werten für `pose_frame` gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen und Ebenenparameter werden im Kamera-Koordinatensystem angegeben.
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen und Ebenenparameter sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das SilhouetteMatch-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom internen Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

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

Bemerkung: Wurde keine Hand-Auge-Kalibrierung durchgeführt, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Bemerkung: Wird die Hand-Auge-Kalibrierung nach einer Kalibrierung der Basisebene verändert, wird die Kalibrierung der Basisebene als ungültig markiert und muss erneuert werden.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame` und der bevorzugten TCP-Orientierung nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die bevorzugte TCP-Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn `camera` als `pose_frame` ausgewählt ist und die bevorzugte TCP-Orientierung in `external` definiert ist, ist die Angabe der Roboterpose optional.

Wenn die aktuelle Roboterpose während der Kalibrierung der Basisebene angegeben wird, wird sie persistent auf dem *rc_cube* gespeichert. Falls für die Services `get_base_plane_calibration` oder `detect_objects` die dann aktuelle Roboterpose ebenfalls angegeben wird, wird die Basisebene automatisch zu der neuen Roboterpose transformiert. Das erlaubt dem Benutzer, die Roboterpose (und damit die Pose der Kamera) zwischen Kalibrierung der Basisebene und Objekterkennung zu verändern.

Bemerkung: Eine Objekterkennung kann nur durchgeführt werden, wenn die Verkippung der Basisebene zur Sichtachse der Kamera ein 10-Grad-Limit nicht übersteigt.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des SilhouetteMatch-Moduls aktiviert werden, indem das `collision_detection` Argument an den `detect_object` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im CollisionCheck-Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 5.3.3.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3) gegeben. Zusätzlich wird auf Kollisionen zwischen dem Greifer und der kalibrierten Basisebene geprüft, wenn der Laufzeitparameter `check_collisions_with_base_plane` auf true gesetzt ist. Wenn das ausgewählte Template ein Kollisionsmodell enthält und der Laufzeitparameter `check_collisions_with_matches` true ist, wird außerdem auch auf Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht begrenzt auf die Anzahl `max_number_of_detected_objects`) geprüft, wobei das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen ist.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in der Ergebnis-Visualisierung oben auf der *SilhouetteMatch*-Seite der Web GUI kollidierende Greifpunkte als rote Punkte dargestellt. Die Objekte, die bei der Kollisionsprüfung betrachtet werden, werden auch mit roten Kanten visualisiert.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 5.3.3.4) beschrieben.

5.2.4.8 Parameter

Das SilhouetteMatch-Modul wird in der REST-API als `rc_silhouettematch` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *SilhouetteMatch* (unter dem Menüpunkt *Module*) dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.22: Laufzeitparameter des rc_silhouettematch-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
check_collisions_with_base_plane	bool	false	true	true	Gibt an, ob die Kollisionen zwischen Greifer und der Basisebene geprüft werden
check_collisions_with_matches	bool	false	true	true	Gibt an, ob die Kollisionen zwischen Greifer und anderen Matches geprüft werden
edge_sensitivity	float64	0.1	1.0	0.6	Empfindlichkeit der Kantenerkennung
load_carrier_crop_distance	float64	0.0	0.05	0.005	Sicherheitsspielraum um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren
load_carrier_model_tolerance	float64	0.003	0.025	0.008	Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen
match_max_distance	float64	0.0	10.0	2.5	Der maximale tolerierte Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln
match_percentile	float64	0.7	1.0	0.85	Der Anteil der Template-Pixel, die innerhalb der maximalen Matchingdistanz liegen müssen, um ein Objekt erfolgreich zu detektieren
max_number_of_detected_objects	int32	1	20	10	Maximale Anzahl der zu detektierenden Objekte
quality	string	-	-	High	Quality: [Low, Medium, High]

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der SilhouetteMatch-Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

max_number_of_detected_objects (*Maximale Objektanzahl*)

Dieser Parameter gibt an, wie viele Objekte maximal in der Szene erkannt werden sollen. Falls mehr als die angegebene Zahl an Objekten gefunden wurden, werden nur die besten Ergebnisse zurückgeliefert.

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?max_number_of_detected_
↪objects=<value>
```

`quality` (*Qualität*)

Die Objekterkennung kann auf Bildern mit unterschiedlicher Auflösung durchgeführt werden: `High` (*Hoch*, 1280 x 960), `Medium` (*Mittel*, 640 x 480) oder `Low` (*Niedrig*, 320 x 240). Je niedriger die Auflösung ist, desto niedriger ist die Berechnungszeit der Objekterkennung, aber desto weniger Objektdetails sind erkennbar.

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

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

`match_max_distance` (*Maximale Matchingdistanz*)

Dieser Parameter gibt den maximal tolerierten Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln an. Falls das Objekt durch das Template nicht exakt genug beschrieben wird, wird es möglicherweise nicht erkannt, wenn dieser Wert zu klein ist. Höhere Werte können jedoch im Fall von komplexen Szenen und bei ähnlichen Objekten zu Fehldetektionen führen, und auch die Berechnungszeit erhöhen.

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_max_distance=<value>
```

`match_percentile` (*Matching Perzentil*)

Dieser Parameter kontrolliert, wie strikt der Detektionsprozess sein soll. Das Matching Perzentil gibt den Anteil der Template-Pixel an, die innerhalb der maximalen Matchingdistanz liegen müssen, um ein Objekt erfolgreich zu detektieren. Je höher der Wert, desto exakter muss ein Match sein, um als gültig angesehen zu werden.

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_percentile=<value>
```

`edge_sensitivity` (*Kantenempfindlichkeit*)

Der Parameter beeinflusst, wie viele Kanten in den Kamerabildern gefunden werden. Umso größer dieser Parameter gewählt wird, umso mehr Kanten werden für die Erkennung benutzt. Eine große Anzahl von Kanten im Bild kann die Erkennung verlangsamen.

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?edge_sensitivity=<value>
```

`check_collisions_with_base_plane` (*Kollisionsprüfung mit Basisebene*)

Wenn dieser Parameter auf true gesetzt ist und die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service aktiviert ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und der kalibrierten Basisebene geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit der Basisebene wäre, werden zurückgeliefert.

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪base_plane=<value>
```

`check_collisions_with_matches` (*Kollisionsprüfung mit Matches*)

Wenn dieser Parameter auf true gesetzt ist und die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service aktiviert ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht begrenzt auf die Anzahl `max_number_of_detected_objects`) geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit anderen detektierten Objekten wäre, werden zurückgeliefert.

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪matches=<value>
```

`load_carrier_model_tolerance`

siehe *Parameter der Load Carrier Funktionalität* (Abschnitt 5.2.1.7).

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?load_carrier_model_
↪tolerance=<value>
```

`load_carrier_crop_distance`

siehe *Parameter der Load Carrier Funktionalität* (Abschnitt 5.2.1.7).

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

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?load_carrier_crop_
↪distance=<value>
```

5.2.4.9 Statuswerte

Dieses Modul meldet folgende Statuswerte.

Tab. 5.23: Statuswerte des `rc_silhouettematch`-Moduls

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

5.2.4.10 Services

Die angebotenen Services des `rc_silhouettematch`-Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) oder der *rc_cube Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

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

Tab. 5.24: Fehlercodes und Warnung der Services des SilhouetteMatch-Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Objekterkennung.
-4	Die maximal erlaubte Zeitspanne von 5.0 Sekunden für die interne Akquise der Bilddaten wurde überschritten.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-8	Das Modul befindet sich in einem Zustand, in welchem dieser Service nicht aufgerufen werden kann. Beispielsweise kann `detect_object` nicht aufgerufen werden, solange keine Kalibrierung der Basisebene durchgeführt wurde.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an ROIs oder Templates überschritten wurde.
-100	Ein interner Fehler ist aufgetreten.
-101	Die Erkennung der Basisebene schlug fehl.
-102	Die Hand-Auge-Kalibrierung hat sich seit der letzten Kalibrierung der Basisebene verändert.
-104	Die Verkippung zwischen der Basisebene und der Sichtachse der Kamera überschreitet das 10-Grad-Limit.
10	Die maximale Anzahl an ROIs oder Templates wurde erreicht.
11	Ein bestehendes Element wurde überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Keiner der Greifpunkte ist erreichbar.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
107	Die Basisebene wurde nicht zur aktuellen Kamerapose transformiert, z.B. weil keine Roboterpose während der Kalibrierung der Basisebene angegeben wurde.
108	Das Template ist überholt.
151	Das Objekt-Template hat eine kontinuierliche Symmetrie.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

Das SilhouetteMatch-Modul bietet folgende Services.

`calibrate_base_plane`

führt die Kalibrierung der Basisebene durch, wie in *Kalibrierung der Basisebene* (Abschnitt 5.2.4.2) beschrieben. Eine erfolgreiche Kalibrierung der Basisebene wird persistent auf dem *rc_cube* gespeichert und vom Service zurückgegeben. Die Kalibrierung ist dauerhaft – auch über Firmware-Updates und -Wiederherstellungen hinweg – gespeichert.

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

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

Request:

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

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

Obligatorische Serviceargumente:

`plane_estimation_method`: Methode der Kalibrierung der Basisebene. Gültige Werte sind `STEREO`, `APRILTAG`, `MANUAL`.

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.4.7).

Potentiell obligatorische Serviceargumente:

`plane` wenn für `plane_estimation_method` `MANUAL` gewählt ist: Die Ebene, welche als Basisebene gesetzt wird.

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.4.7).

`region_of_interest_2d_id`: ID der Region of Interest für die Kalibrierung der Basisebene.

Optionale Serviceargumente:

`offset`: Versatz in Metern, um welchen die geschätzte Ebene in Richtung der Kamera verschoben wird.

`plane_preference` in `stereo`: Ob die der Kamera am nächsten (`CLOSEST`) gelegene oder die am weitesten entfernte (`FARTHEST`) Ebene als Basisebene benutzt wird. Diese Option kann nur gesetzt werden, falls `plane_estimation_method` auf `STEREO` gesetzt ist. Valide Werte sind

CLOSEST und FARTHEST. Falls der Wert nicht gesetzt ist, wird FARTHEST verwendet.

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

`plane`: kalibrierte Basisebene.

`timestamp`: Zeitstempel des Bildes, das für die Kalibrierung benutzt wurde.

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

get_base_plane_calibration

gibt die derzeitige Kalibrierung der Basisebene zurück.

Dieser Service kann wie folgt aufgerufen werden.

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

Request:

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.4.7).

Potentiell obligatorische Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.4.7).

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_base_plane_calibration",
  "response": {
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_base_plane_calibration

löscht die derzeitige Kalibrierung der Basisebene.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_base_plane_
↪calibration
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

set_region_of_interest_2d

siehe *set_region_of_interest_2d* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

get_regions_of_interest_2d

siehe *get_regions_of_interest_2d* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_regions_of_interest_2d

siehe *delete_regions_of_interest_2d* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

set_load_carrier

siehe *set_load_carrier* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

get_load_carriers

siehe *get_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_load_carriers

siehe *delete_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_load_carriers

siehe *detect_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_filling_level

siehe *detect_filling_level* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Sortierung und Filterung der vom `detect_object` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.2.4.5).

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Sortierung und Filterung der vom `detect_object` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.2.4.5).

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_grasp

speichert einen Greifpunkt für das angegebene Template auf dem *rc_cube*. Alle Greifpunkte sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Dieser Service kann wie folgt aufgerufen werden.

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

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


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


Die Definition der *Response* mit jeweiligen Datentypen ist:


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

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.2.4.4) beschrieben.

set_all_grasps

Ersetzt die gesamte Liste von Greifpunkten auf dem *rc_cube* für das angegebene Template.

Dieser Service kann wie folgt aufgerufen werden.

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

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

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

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.2.4.4) beschrieben.

get_grasps

gibt alle definierten Greifpunkte mit den angegebenen IDs (`grasp_ids`) zurück, die zu den Templates mit den angegebenen `template_ids` gehören. Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte zu den angegebenen `template_ids` zurückgeliefert. Wenn keine `template_ids` angegeben werden, werden alle Greifpunkte mit den geforderten `grasp_ids` zurückgeliefert. Wenn gar keine IDs angegeben werden, werden alle gespeicherten Greifpunkte zurückgeliefert.

Dieser Service kann wie folgt aufgerufen werden.

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

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


```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```


Die Definition der *Response* mit jeweiligen Datentypen ist:


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


delete_grasps

löscht alle Greifpunkte mit den angegebenen `grasp_ids`, die zu den Templates mit den angegebenen `template_ids` gehören. Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte gelöscht, die zu den Templates mit den angegebenen `template_ids` gehören. Die Liste `template_ids` darf nicht leer sein.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```


Die Definition der *Response* mit jeweiligen Datentypen ist:


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


get_symmetric_grasps

gibt alle Greifpunkte zurück, die symmetrisch zum angegebenen Greifpunkt sind. Der erste Greifpunkt in der Rückgabeliste ist derselbe, der dem Service übergeben wurde. Wenn das Template keine exakte Symmetrie hat, wird nur der übergebene Greifpunkt zurückgeliefert. Wenn das Template eine kontinuierliche Symmetrie hat (z.B. ein zylindrisches Objekt), werden nur 12 gleichverteilte Greifpunkte zurückgeliefert.

Dieser Service kann wie folgt aufgerufen werden.

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

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


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


Die Definition der *Response* mit jeweiligen Datentypen ist:

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


Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.2.4.4) beschrieben.

`detect_object`

führt eine Objekterkennung durch, wie in *Objekterkennung* (Abschnitt 5.2.4.6) beschrieben. Der Service gibt die Posen aller gefundenen Objektinstanzen zurück. Die maximale Anzahl der zurückgegebenen Instanzen kann über den `max_number_of_detected_objects`-Parameter kontrolliert werden.

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

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

Request:

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


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


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "object_id": "string",
      "region_of_interest_2d_id": "string"
    },
    "offset": "float64",
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Obligatorische Serviceargumente:

`object_id` in `object_to_detect`: ID des Templates, welches erkannt werden soll.

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.4.7).

Potentiell obligatorische Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.4.7).

Optionale Serviceargumente:

`offset`: Versatz in Metern, um welche die Basisebene in Richtung der Kamera verschoben werden soll.

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3)

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_object",
  "response": {
    "grasps": [
      {
        "id": "string",
        "instance_uuid": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "uuid": "string"
    }
  ],
  "instances": [
    {
      "grasp_uuids": [
        "string"
      ],
      "id": "string",
      "object_id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


`object_id`: ID des erkannten Templates.

`instances`: Liste der erkannten Objektinstanzen.

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind nach ihrer Erreichbarkeit sortiert, begonnen mit dem Greifpunkt, der am besten vom Roboter erreicht werden kann. Die `instance_uuid` gibt eine Referenz auf das detektierte Objekt in `instances` an, zu dem dieser Greifpunkt gehört.

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

`timestamp`: Zeitstempel des Bildes, das für die Erkennung benutzt wurde.

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

save_parameters

speichert die aktuellen Parametereinstellungen des SilhouetteMatch-Moduls auf dem *rc_cube*. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden. Bei Firmware-Updates oder -Wiederherstellungen werden sie jedoch wieder auf den Standardwert gesetzt.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:


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


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“). Dies betrifft nicht die konfigurierten ROIs und die Kalibrierung der Basisebene.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.2.4.11 Template Upload

Für den Upload, Download und das Auflisten von Templates werden spezielle REST-API-Endpunkte zur Verfügung gestellt. Bis zu 50 Templates können gleichzeitig auf dem *rc_cube* gespeichert werden.

GET /nodes/rc_silhouettematch/templates
listet alle rc_silhouettematch-Templates auf.

Musteranfrage

```
GET /api/v1/nodes/rc_silhouettematch/templates HTTP/1.1
```

Musterantwort

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

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.3.3)

GET /nodes/rc_silhouettematch/templates/{id}
ruft ein rc_silhouettematch-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v1/nodes/rc_silhouettematch/templates/<id> HTTP/1.1
```

Musterantwort

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

{
  "id": "string"
}
```

Parameter

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

Antwort-Header

- Content-Type – application/json application/octet-stream

Statuscodes

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

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.3.3)

PUT /nodes/rc_silhouettematch/templates/{id}
erstellt oder aktualisiert ein rc_silhouettematch-Template.

Musteranfrage

```
PUT /api/v1/nodes/rc_silhouettematch/templates/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

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

{
  "id": "string"
}
```

Parameter

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

Formularparameter

- **file** – Template-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das SilhouetteMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.3.3)

`DELETE /nodes/rc_silhouettematch/templates/{id}`
entfernt ein rc_silhouettematch-Template.

Musteranfrage

```
DELETE /api/v1/nodes/rc_silhouettematch/templates/<id> HTTP/1.1
Accept: application/json
```

Parameter

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

Anfrage-Header

- Accept – application/json

Antwort-Header

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das SilhouetteMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

5.2.5 CADMatch

5.2.5.1 Einleitung

Das CADMatch Modul ist ein optionales Modul des *rc_cube* und benötigt eine eigene *Lizenz* (Abschnitt 7.5), welche erworben werden muss.

Dieses Modul bietet eine gebrauchsfertige Lösung für die 3D-Objekterkennung anhand von CAD-Templates und liefert Greifpunkte für allgemeine Greifer. Die Objekte können sich in einer Kiste (Bin, Load Carrier) oder frei platziert im Erfassungsbereich der Kamera befinden.

Für jedes Objekt, das mit dem CADMatch-Modul erkannt werden soll, wird ein Template benötigt. Um Templates zu erhalten, setzen Sie sich bitte mit dem Roboception Support (*Kontakt*, Abschnitt 9) in Verbindung.

Das CADMatch-Modul bietet darüber hinaus:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_cube Web GUI* (Abschnitt 6.1)

- eine *REST-API-Schnittstelle* (Abschnitt 6.3) und eine *KUKA Ethernet KRL Schnittstelle* (Abschnitt 6.4)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *Region of Interest*, Abschnitt 5.3.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 5.2.1), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Abteilen, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- die Definition von Greifpunkten für jedes Template über eine interaktive Visualisierung in der Web GUI
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.
- eine Sortierung der Greifpunkte nach ihrer Erreichbarkeit, sodass die besten Greifpunkte zuerst zurückgeliefert werden

5.2.5.2 Setzen von Greifpunkten

Das CADMatch-Modul erkent 3D-Objekte in einer Szene basierend auf einem CAD-Template und liefert die Posen der Objektursprünge zurück. Um das CADMatch-Modul direkt in einer Roboteranwendung zu nutzen, können für jedes Template Greifpunkte definiert werden. Ein Greifpunkt repräsentiert die gewünschte Position und Orientierung des Roboter-TCPs (Tool Center Point), mit der das Objekt gegriffen werden kann (siehe Abb. 5.17).

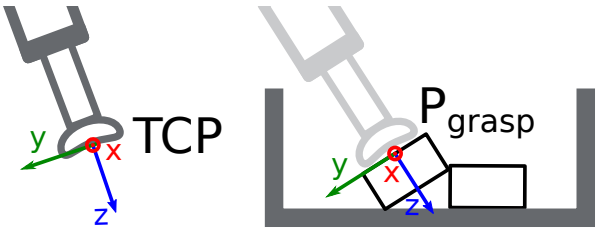


Abb. 5.17: Definition von Greifpunkten bezogen auf den Roboter-TCP

Jeder Greifpunkt enthält eine `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, die ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und die Greifpose (`pose`) im Koordinatensystem des Templates. Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 6.3), oder über die interaktive Visualisierung in der Web GUI definiert werden. Der *rc_cube* kann bis zu 50 Greifpunkte pro Template speichern.

Setzen von Greifpunken in der Web GUI

Die *rc_cube* Web GUI bietet eine interaktive und intuitive Möglichkeit, Greifpunkte für Objekt-Templates zu setzen. Im ersten Schritt muss das Objekt-Template auf den *rc_cube* hochgeladen werden. Das kann über die *CADMatch*-Seite (unter dem Menüpunkt *Module*) in der Web GUI erfolgen, indem im Abschnitt *Templates und Greifpunkte* auf *neues Template hinzufügen* geklickt wird. Wenn der Upload abgeschlossen ist, erscheint ein Fenster mit einer 3D-Visualisierung des Objekts, in dem Greifpunkte hinzugefügt oder existierende Greifpunkte bearbeitet werden können. Dasselbe Fenster erscheint, wenn ein vorhandenes Template bearbeitet wird.

Dieses Fenster bietet zwei Möglichkeiten, um Greifpunkte zu setzen:

1. **Greifpunkte manuell hinzufügen**: Durch Klicken auf das + Symbol wird ein neuer Greifpunkt im Ursprung des Templates angelegt. Diesem Greifpunkt kann ein eindeutiger Name gegeben

werden, der seiner ID entspricht. Die gewünschte Pose des Greifpunkts im Koordinatensystem des Templates kann in den Feldern für *Position* und *Roll/Pitch/Yaw* eingegeben werden. Die Greifpunkte können frei platziert werden, auch außerhalb oder innerhalb des Templates, und werden mit ihrer Orientierung zur Überprüfung in der Visualisierung veranschaulicht.

2. **Greifpunkte interaktiv hinzufügen**: Greifpunkte können interaktiv zu einem Template hinzugefügt werden, indem zuerst auf den Button *Greifpunkt hinzufügen* oben links in der Visualisierung und anschließend auf den gewünschten Punkt auf dem Template geklickt wird. Der Greifpunkt wird an die Oberfläche angeheftet und seine Orientierung entspricht einem rechtshändigen Koordinatensystem, sodass die z-Achse senkrecht auf der Template-Oberfläche steht und in das Template hineingerichtet ist. Die Position und Orientierung des Greifpunkts im Koordinatensystem des Templates ist auf der rechten Seite angezeigt. Die Position und Orientierung des Greifpunkts kann auch interaktiv verändert werden. Für den Fall, dass *An Oberfläche anheften* in der Visualisierung aktiv ist (das ist der Standardwert), kann der Greifpunkt durch Klicken auf *Verschieben* und anschließendes Klicken auf den Greifpunkt über die Oberfläche des Templates zur gewünschten Position bewegt werden. Die Orientierung des Greifpunkts um die Oberflächennormale kann ebenfalls interaktiv verändert werden, in dem auf *Rotieren* geklickt wird und anschließend der Greifpunkt mit der Maus rotiert wird. Wenn *An Oberfläche anheften* nicht aktiv ist, kann der Greifpunkt mit der Maus frei in allen drei Raumrichtungen verschoben und rotiert werden.

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

Setzen von Greifpunkten über die REST-API

Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 6.3) mithilfe des `set_grasp` oder `set_all_grasps` Services gesetzt werden (siehe *Services*, Abschnitt 5.2.5.8). Im CADMatch-Modul besteht ein Greifpunkt aus der `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, der ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und der Greifpose (`pose`) mit Position (`position`) in Metern und Orientierung (`orientation`) als Quaternion im Koordinatensystem des Templates.

5.2.5.3 Setzen der bevorzugten TCP-Orientierung

Das CADMatch-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des Greifers oder TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die *CADMatch*-Seite in der Web GUI gesetzt werden. Die resultierende Richtung der z-Achse des TCP wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann.

Weiterhin wird die bevorzugte Orientierung genutzt, um die erreichbaren Greifpunkte zu sortieren. Die Sortierung basiert auf einer Kombination von

- der Matching Score des Objekts, auf dem sich der Greifpunkt befindet, und
- dem Abstand des Greifpunkts von der Kamera entlang der z-Achse der bevorzugten TCP-Orientierung.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und der Sensor am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden, damit die bevorzugte Orientierung zur Filterung und Sortierung der Greifpunkte auf den erkannten Objekten genutzt werden kann. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera als die bevorzugte TCP-Orientierung genutzt.

5.2.5.4 Objekterkennung

Das CADMatch-Modul benötigt ein Objekt-Template für die Objekterkennung. Dieses Template enthält Informationen über die dreidimensionale Form des Objekts und markante Kanten, die im Kamerabild sichtbar sein können.

Die Objekterkennung ist ein zweistufiger Prozess bestehend aus einem initialen Schätzungsschritt und einem Posenverfeinerungsschritt. Als erstes wird die initiale Pose des Objekts anhand der Erscheinung des Objekts im Kamerabild berechnet. Als zweiter Schritt wird die geschätzte Pose anhand der 3D-Punktwolke und der Kanten im Kamerabild verfeinert. Damit das funktionieren kann, müssen die zu detektierenden Objekte im linken und rechten Kamerabild sichtbar sein.

Um eine Objekterkennung durchzuführen, müssen im Allgemeinen die folgenden Serviceargumente an das CADMatch-Modul übergeben werden:

- die ID des Objekt-Templates, welches in der Szene erkannt werden soll
- das Koordinatensystem, in dem die Posen der detektierten Objekte zurückgegeben werden sollen (siehe *Hand-Auge-Kalibrierung*, Abschnitt 5.2.5.5)

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

- die ID des Load Carriers, der die zu detektierenden Objekte enthält
- ein Unterabteil (`load_carrier_compartment`) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteil*, Abschnitt 5.2.1.3).
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, innerhalb der Objekte erkannt werden sollen
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die bevorzugte TCP-Orientierung im externen Koordinatensystem angegeben ist, oder die gewählte Region of Interest im externen Koordinatensystem definiert ist
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 5.2.5.5) gegeben.

In der Web GUI kann die Objekterkennung in Bereich *Ausprobieren* auf der *CADMatch*-Seite getestet werden.

Die erkannten Objekte werden in einer Liste von `matches` zurückgeliefert. Jedes erkannte Objekt enthält seine `uuid` (Universally Unique Identifier) und den Zeitstempel (`timestamp`) des ältesten Bildes, das zur Erkennung benutzt wurde. Die Pose (`pose`) eines erkannten Objekts entspricht der Pose des Ursprungs des Koordinatensystems des Objekt-Templates, das zur Detektion verwendet wurde. Weiterhin wird ein Matching-Score zurückgegeben, der die Qualität der Erkennung angibt.

Wenn das ausgewählte Template auch Greifpunkte hat (siehe *Setzen von Greifpunkten*, Abschnitt 5.2.5.2), dann wird zusätzlich zu den erkannten Objekten auch eine Liste von Greifpunkten (`grasps`) für alle erkannten Objekte zurückgegeben. Die Greifpunkte in dieser Liste sind nach ihrer Erreichbarkeit sortiert (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.2.5.3). Die Posen der Greifpunkte sind im gewünschten Koordinatensystem angegeben. Die erkannten Objekte und die Greifpunkte können einander über ihre UUIDs zugeordnet werden.

Bemerkung: Der erste Aufruf der Objekterkennung mit einem neuen Objekt-Template dauert etwas länger als die nachfolgenden Aufrufe, weil das Template erst in das CADMatch-Modul geladen werden muss.

5.2.5.5 Wechselwirkung mit anderen Modulen

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

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

Stereokamera und Stereo-Matching

Folgende Daten werden vom CADMatch-Modul verarbeitet:

- die rektifizierten Bilder des *Stereokamera*-Moduls (`rc_stereocamera`, Abschnitt 5.1.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching*-Moduls (`rc_stereomatching`, Abschnitt 5.1.2)

Der Parameter *Qualität* (`quality`) des Stereo-Matching-Moduls muss auf `Medium` oder höher gesetzt werden (siehe *Parameter*, Abschnitt 5.1.2.5). Die Einstellung `Full` oder `High` wird für CADMatch empfohlen.

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

Schätzung der Gravitationsrichtung

Jedes Mal, wenn eine Load Carrier Erkennung oder eine Objekterkennung innerhalb eines Load Carriers durchgeführt wird, schätzt das CADMatch-Modul die Gravitationsrichtung basierend auf den IMU-Daten des *rc_visard*.

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

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_cube* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 5.3.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 5.1.2.5), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 5.3.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren (siehe *Stereokamera-Parameter*, Abschnitt 5.1.1.4).

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das CADMatch-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 5.2.5.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).

2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das CADMatch-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

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

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

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des CADMatch-Moduls aktiviert werden, indem das `collision_detection` Argument an den `detect_object` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im CollisionCheck-Modul definiert werden (siehe *Erstellen eines Greifers* (Abschnitt 5.3.3.2)) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3) gegeben. Wenn das ausgewählte CADMatch Template eine Kollisionsgeometrie enthält und der Laufzeitparameter `check_collisions_with_matches` auf true gesetzt ist, werden auch Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht limitiert auf die Anzahl *max_matches*') geprüft. Dabei ist das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in der Ergebnis-Visualisierung oben auf der *CADMatch*-Seite der Web GUI kollidierende Greifpunkte als rote Punkte dargestellt. Die Objekte, die bei der Kollisionsprüfung betrachtet werden, werden auch mit roten Kanten visualisiert.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 5.3.3.4) beschrieben.

5.2.5.6 Parameter

Das CADMatch-Modul wird in der REST-API als `rc_cadmatch` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *CADMatch* (unter dem Menüpunkt *Module*) dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.25: Laufzeitparameter des rc_cadmatch-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
check_collisions_with_matches	bool	false	true	true	Gibt an, ob die Kollisionen zwischen Greifer und anderen Matches geprüft werden
edge_max_distance	float64	0.5	5.0	2.0	Der maximale tolerierte Abstand zwischen den Templatekanten und den detektierten Kanten im Bild in Pixeln
edge_sensitivity	float64	0.0	1.0	0.5	Empfindlichkeit des Kantendetektors
grasp_filter_-orientation_threshold	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
load_carrier_crop_distance	float64	0.0	0.05	0.005	Sicherheitsspielraum um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren
load_carrier_model_tolerance	float64	0.003	0.025	0.008	Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen
max_matches	int32	1	20	10	Maximale Anzahl von Matches
min_score	float64	0.05	1.0	0.3	Minimaler Score für Matches

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der *CADMatch*-Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

max_matches (*Maximale Matches*)

ist die maximale Anzahl der zu erkennenden Objekte.

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

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?max_matches=<value>
```

min_score (*Minimaler Score*)

ist der minimale Score für die Erkennung nach der Posenverfeinerung. Umso höher dieser Wert ist, umso besser müssen die 2D-Kanten und die 3D-Punktwolke mit dem angegebenen

Template übereinstimmen.

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

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?min_score=<value>
```

`edge_sensitivity` (*Kantenempfindlichkeit*)

ist die Empfindlichkeit des Kantendetektors. Umso höher dieser Wert ist, umso mehr Kanten werden für die Posenverfeinerung genutzt.

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

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?edge_sensitivity=<value>
```

`edge_max_distance` (*Maximale Kantendistanz*)

ist die maximal erlaubte Distanz in Pixeln zwischen den Templatekanten und den detektierten Kanten im Bild während der Posenverfeinerung.

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

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?edge_max_distance=<value>
```

`grasp_filter_orientation_threshold` (*Maximale Abweichung*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der bevorzugten TCP-Orientierung in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist.

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

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?grasp_filter_orientation_
↪threshold=<value>
```

`check_collisions_with_matches` (*Kollisionsprüfung mit Matches*)

Wenn dieser Parameter auf true gesetzt ist und die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service aktiviert ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und den anderen Matches (nicht begrenzt auf die Anzahl `max_matches`) geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit anderen Matches wäre, werden zurückgeliefert.

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

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

`load_carrier_model_tolerance`

siehe *Parameter der Load Carrier Funktionalität* (Abschnitt 5.2.1.7).

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

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

load_carrier_crop_distance

siehe *Parameter der Load Carrier Funktionalität* (Abschnitt 5.2.1.7).

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

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

5.2.5.7 Statuswerte

Das CADMatch-Modul meldet folgende Statuswerte.

Tab. 5.26: Statuswerte des `rc_cadmatch`-Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden
`object_detection_time`	Berechnungszeit für die letzte Objekterkennung in Sekunden
`state`	Aktueller Zustand des CADMatch-Moduls

Folgende `state`-Werte werden gemeldet.

Tab. 5.27: Mögliche Werte für den Zustand des CADMatch-Moduls

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier und Objekte zu erkennen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

5.2.5.8 Services

Die angebotenen Services von `rc_cadmatch` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) oder der *rc_cube Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

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

Die folgende Tabelle führt die möglichen Rückgabe-Codes an:

Tab. 5.28: Rückgabecodes der Services des CADMatch-Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-2	Ein interner Fehler ist aufgetreten.
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Objekterkennung.
-4	Die maximal erlaubte Zeitspanne von 5.0 Sekunden für die interne Akquise der Bilddaten wurde überschritten.
-8	Das Modul befindet sich in einem Zustand, in welchem dieser Service nicht aufgerufen werden kann. Beispielsweise muss die Stereo-Matching Qualität `quality` mindestens `Medium` sein.
-9	Ungültige Lizenz
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern oder ROIs überschritten wurde.
10	Die maximal speicherbare Anzahl an Load Carriern oder ROIs wurde erreicht.
11	Existierende Daten wurden überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Keiner der gefundenen Greifpunkte ist erreichbar.
102	Der erkannte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
151	Das Objekt-Template hat eine kontinuierliche Symmetrie.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

Das CADMatch-Modul stellt folgende Services zur Verfügung.

`start`

versetzt das CADMatch-Modul in den Zustand `RUNNING`. Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `RUNNING` unterscheidenden Zustand zurück.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

`stop`

stoppt das Modul und versetzt es in den Zustand `IDLE`. Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

set_region_of_interest

siehe *set_region_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

get_regions_of_interest

siehe *get_regions_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_regions_of_interest

siehe *delete_regions_of_interest* (Abschnitt 5.3.2.4).

Dieser Service kann wie folgt aufgerufen werden.

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

set_load_carrier

siehe *set_load_carrier* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

get_load_carriers

siehe *get_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

delete_load_carriers

siehe *delete_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_load_carriers

siehe *detect_load_carriers* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

detect_filling_level

siehe *detect_filling_level* (Abschnitt 5.2.1.8).

Dieser Service kann wie folgt aufgerufen werden.

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

set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Sortierung und Filterung der vom `detect_object` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.2.5.3).

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_preferred_orientation
```

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

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Sortierung und Filterung der vom `detect_object` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 5.2.5.3).

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_preferred_orientation
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_grasp

speichert einen Greifpunkt für das angegebene Template auf dem *rc_cube*. Alle Greifpunkte sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_grasp
```

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
          "y": "float64",
          "z": "float64"
        }
      },
      "template_id": "string"
    }
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.2.5.2) beschrieben.

set_all_grasps

Ersetzt die gesamte Liste von Greifpunkten auf dem *rc_cube* für das angegebene Template.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_all_grasps
```

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

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

Die Definition der *Response* mit jeweiligen Datentypen ist:

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


Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.2.5.2) beschrieben.

get_grasps

gibt alle definierten Greifpunkte mit den angegebenen IDs (`grasp_ids`) zurück, die zu den Templates mit den angegebenen `template_ids` gehören. Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte zu den angegebenen `template_ids` zurückgeliefert. Wenn keine `template_ids` angegeben werden, werden alle Greifpunkte mit den geforderten `grasp_ids` zurückgeliefert. Wenn gar keine IDs angegeben werden, werden alle gespeicherten Greifpunkte zurückgeliefert.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_grasps
```

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


```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```


Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_grasps",
  "response": {
    "grasps": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "template_id": "string"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


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


delete_grasps

löscht alle Greifpunkte mit den angegebenen `grasp_ids`, die zu den Templates mit den angegebenen `template_ids` gehören. Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte gelöscht, die zu den Templates mit den angegebenen `template_ids` gehören. Die Liste `template_ids` darf nicht leer sein.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/delete_grasps
```

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


```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```


Die Definition der *Response* mit jeweiligen Datentypen ist:


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


get_symmetric_grasps

gibt alle Greifpunkte zurück, die symmetrisch zum angegebenen Greifpunkt sind. Der erste Greifpunkt in der Rückgabeliste ist derselbe, der dem Service übergeben wurde. Wenn das Template keine exakte Symmetrie hat, wird nur der übergebene Greifpunkt zurückgeliefert. Wenn das Template eine kontinuierliche Symmetrie hat (z.B. ein zylindrisches Objekt), werden nur 12 gleichverteilte Greifpunkte zurückgeliefert.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_symmetric_grasps
```

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

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


Die Definition der *Response* mit jeweiligen Datentypen ist:


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


Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 5.2.5.2) beschrieben.

`detect_object`

führt eine Objekterkennung basierend auf einem Template durch, wie in *Objekterkennung* (Abchnitt 5.2.5.4) beschrieben.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/detect_object
```

Request:

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


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "template_id": "string"
  }
}
```


Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 5.2.5.5).

`template_id`: ID des Templates, welches erkannt werden soll.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: see *Hand-Auge-Kalibrierung* (Abschnitt 5.2.5.5).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteil*, Abschnitt 5.2.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, in der nach Objekten gesucht wird.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 5.3.3.3)

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_object",
  "response": {
    "grasps": [
      {
        "id": "string",
        "match_uuid": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        }
      }
    ],
    "matches": [
      {
        "grasp_uuids": [
          "string"
        ],
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "score": "float32",
        "template_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind nach ihrer Erreichbarkeit sortiert, begonnen mit dem Greifpunkt, der am besten

vom Roboter erreicht werden kann. Die `match_uuid` gibt eine Referenz auf das detektierte Objekt in `matches` an, zu dem dieser Greifpunkt gehört.

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

`matches`: Liste der erkannten Objekte für das angegebene Template. Der `score` gibt an, wie gut das Objekt mit dem Template übereinstimmt. Die `grasp_uuids` geben die Greifpunkte in der `grasps`-Liste an, die auf diesem Objekt erreichbar sind.

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

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

save_parameters

Beim Aufruf dieses Services werden die aktuellen Parametereinstellungen des CADMatch-Moduls auf dem *rc_cube* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden. Bei Firmware-Updates oder -Wiederherstellungen werden sie jedoch wieder auf den Standardwert gesetzt.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/save_parameters
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

reset_defaults

Hiermit werden die Werkseinstellungen der Parameter dieses Moduls wiederhergestellt und angewandt („factory reset“). Dies betrifft nicht die konfigurierten ROIs und Load Carrier.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.2.5.9 Template Upload

Für den Upload, Download und das Auflisten von Templates werden spezielle REST-API-Endpunkte zur Verfügung gestellt. Bis zu 30 Templates können gleichzeitig auf dem *rc_cube* gespeichert werden.

`GET /nodes/rc_cadmatch/templates`
listet alle rc_cadmatch-Templates auf.

Musteranfrage

```
GET /api/v1/nodes/rc_cadmatch/templates HTTP/1.1
```

Musterantwort

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

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.3.3)

`GET /nodes/rc_cadmatch/templates/{id}`
ruft ein rc_cadmatch-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v1/nodes/rc_cadmatch/templates/<id> HTTP/1.1
```

Musterantwort

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

{
  "id": "string"
}
```

Parameter

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

Antwort-Header

- Content-Type – application/json application/octet-stream

Statuscodes

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

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.3.3)

PUT /nodes/rc_cadmatch/templates/{id}
erstellt oder aktualisiert ein rc_cadmatch-Template.

Musteranfrage

```
PUT /api/v1/nodes/rc_cadmatch/templates/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

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

{
  "id": "string"
}
```

Parameter

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

Formularparameter

- **file** – Template-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 6.3.3)

DELETE /nodes/rc_cadmatch/templates/{id}
entfernt ein rc_cadmatch-Template.

Musteranfrage

```
DELETE /api/v1/nodes/rc_cadmatch/templates/<id> HTTP/1.1
Accept: application/json
```

Parameter

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

Anfrage-Header

- Accept – application/json

Antwort-Header

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

5.3 Konfigurationsmodule

Der *rc_cube* bietet mehrere verschiedene Konfigurationsmodule, welche es dem Nutzer ermöglichen, den *rc_cube* für spezielle Anwendungen zu konfigurieren.

Die Konfigurationsmodule sind:

- ***Hand-Auge-Kalibrierung*** **(`rc_hand_eye_calibration`, Abschnitt 5.3.1)** ermöglicht dem Benutzer, die Kamera entweder über die Web GUI oder die REST-API zu einem Roboter zu kalibrieren.
- ***Region of Interest*** **(Abschnitt 5.3.2)** ermöglicht das Setzen und Abrufen von 2D und 3D Regions of Interest
- ***CollisionCheck*** **(`rc_collision_check`, Abschnitt 5.3.3)** erlaubt, Greifer zu konfigurieren und abzufragen, und bietet eine einfache Möglichkeit zu prüfen, ob ein Greifer in Kollision ist.
- ***IOControl und Projektor-Kontrolle*** **(`rc_iocontrol`, Abschnitt 5.3.4)** bietet die Kontrolle über die Ein- und Ausgänge des *rc_visard* mit speziellen Betriebsarten zur Kontrolle eines externen Musterprojektors.

5.3.1 Hand-Auge-Kalibrierung

Für Anwendungen, bei denen die Kamera in eines oder mehrere Robotersysteme integriert wird, muss sie zum jeweiligen Roboter-Koordinatensystem kalibriert werden. Zu diesem Zweck wird der *rc_cube* mit einer internen Kalibrierroutine, dem Modul zur *Hand-Auge-Kalibrierung*, ausgeliefert. Dieses Modul ist ein Basismodul, welches auf jedem *rc_cube* verfügbar ist.

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

Die *Kalibrierroutine* (Abschnitt 5.3.1.3) ist ein benutzerfreundliches dreistufiges Verfahren, für das mit einem Kalibriermuster gearbeitet wird. Entsprechende Kalibriermuster können von Roboception bezogen werden.

5.3.1.1 Kalibrierschnittstellen

Für die Durchführung der Hand-Auge-Kalibrierung stehen die folgenden beiden Schnittstellen zur Verfügung:

1. Alle Services und Parameter dieses Moduls, die für eine **programmgesteuerte** Durchführung der Hand-Auge-Kalibrierung benötigt werden, sind in der *REST-API-Schnittstelle* (Abschnitt 6.3) des *rc_cube* enthalten. Der REST-API-Name dieses Moduls lautet `rc_hand_eye_calibration` und seine Services werden in *Services* (Abschnitt 5.3.1.5) erläutert.

   Bemerkung: Für den beschriebenen Ansatz wird eine Netzwerkverbindung zwischen dem *rc_cube* und der Robotersteuerung benötigt, damit die Steuerung die Roboterposen an das Kalibriermodul des *rc_cube* übertragen kann.

2. Für Anwendungsfälle, bei denen sich die Roboterposen nicht programmgesteuert an das Modul zur Hand-Auge-Kalibrierung des *rc_cube* übertragen lassen, sieht die Seite *Hand-Auge-Kalibrierung* unter dem Menüpunkt *Konfiguration* der *Web GUI* (Abschnitt 6.1) einen geführten Prozess vor, mit dem sich die Kalibrierroutine **manuell** durchführen lässt.

   Bemerkung: Während der Kalibrierung muss der Benutzer die Roboterposen, auf die über das jeweilige Teach-in- oder Handheld-Gerät zugegriffen werden muss, manuell in die Web GUI eingeben.

5.3.1.2 Kameramontage

Wie in Abb. 5.18 und Abb. 5.20 dargestellt, ist für die Montage der Kamera zwischen zwei unterschiedlichen Anwendungsfällen zu unterscheiden:

a. Die Kamera wird **am Roboter montiert**, d.h. sie ist mechanisch mit einem Roboterpunkt (d.h. Flansch oder flanschmontiertes Werkzeug) verbunden und bewegt sich demnach mit dem Roboter.

b. Die Kamera ist nicht am Roboter montiert, sondern an einem Tisch oder anderen Ort in der Nähe des Roboters befestigt und verbleibt daher verglichen mit dem Roboter in einer **statischen** Position.

Die allgemeine *Kalibrierroutine* (Abschnitt 5.3.1.3) ist in beiden Anwendungsfällen sehr ähnlich. Sie unterscheidet sich jedoch hinsichtlich der semantischen Auslegung der Ausgabedaten, d.h. der erhaltenen Kalibriertransformation, und hinsichtlich der Befestigung des Kalibriermusters.

Kalibrierung einer robotergeführten Kamera Soll eine robotergeführte Kamera zum Roboter kalibriert werden, so muss das Kalibriermuster in einer statischen Position zum Roboter, z.B. auf einem Tisch oder festen Sockel, befestigt werden (siehe Abb. 5.18).

Warnung: Es ist äußerst wichtig, dass sich das Kalibriermuster in Schritt 2 der *Kalibrierroutine* (Abschnitt 5.3.1.3) nicht bewegt. Daher wird dringend empfohlen, das Muster in seiner Position sicher zu fixieren, um unbeabsichtigte Bewegungen, wie sie durch Vibrationen, Kabelbewegungen oder Ähnliches ausgelöst werden, zu verhindern.

Das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 5.3.1.3) ist eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{robot}} = \mathbf{R}_{\text{camera}}^{\text{robot}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}_{\text{camera}}^{\text{robot}} , \tag{5.3}$$

wobei $\mathbf{p}_{\text{robot}} = (x, y, z)^T$ ein 3D-Punkt ist, dessen Koordinaten im *Roboter*-Koordinatensystem angegeben werden, $\mathbf{p}_{\text{camera}}$ denselben Punkt im *Kamera*-Koordinatensystem darstellt, und $\mathbf{R}_{\text{camera}}^{\text{robot}}$ sowie $\mathbf{t}_{\text{camera}}^{\text{robot}}$ die 3×3 Drehmatrix und den 3×1 Translationsvektor für eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$ angeben. In der Praxis wird die Rotation für das Kalibrierergebnis und die Roboterposen als Eulerwinkel oder Quaternion anstatt einer Rotationsmatrix definiert (siehe *Formate für Posendaten*, Abschnitt 10.1).

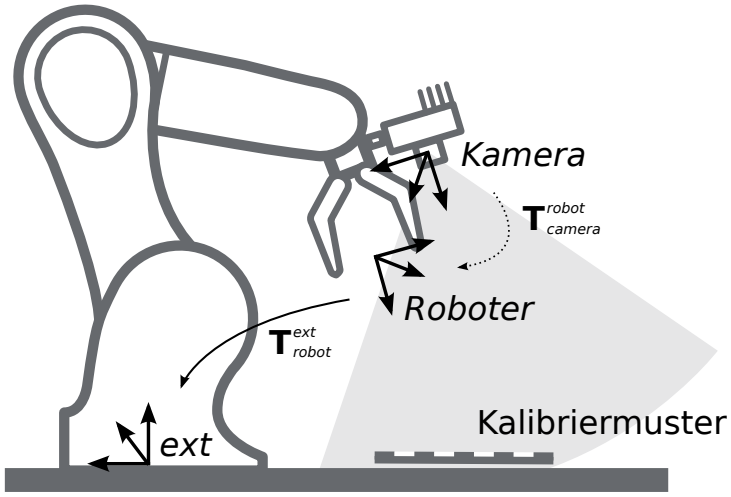


Abb. 5.18: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer robotergeführten Kamera: Sie wird mit einer festen relativen Position zu einem benutzerdefinierten *Roboter*-Koordinatensystem (z.B. Flansch oder Werkzeugmittelpunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{ext}_{robot}$ 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 $\mathbf{T}^{robot}_{camera}$, d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten *Roboter*-Koordinatensystem.

Zusätzliche Benutzereingaben werden benötigt, falls die Bewegung des Roboters so beschränkt ist, dass der Tool Center Point (TCP) nur um eine Achse rotieren kann. Das ist üblicherweise für Roboter mit vier Freiheitsgraden (4DOF) der Fall, welche häufig zur Palettierung eingesetzt werden. In diesem Fall muss der Benutzer angeben, welche Achse des Roboterkoordinatensystems der Rotationsachse des TCP entspricht. Außerdem muss der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Abb. 5.19 zeigt die Situation.

Für den *rc_visard* befindet sich der Ursprung des Kamerakoordinatensystems im optischen Zentrum der linken Kamera. Die ungefähre Position wird im Abschnitt Coordinate Frames angegeben.

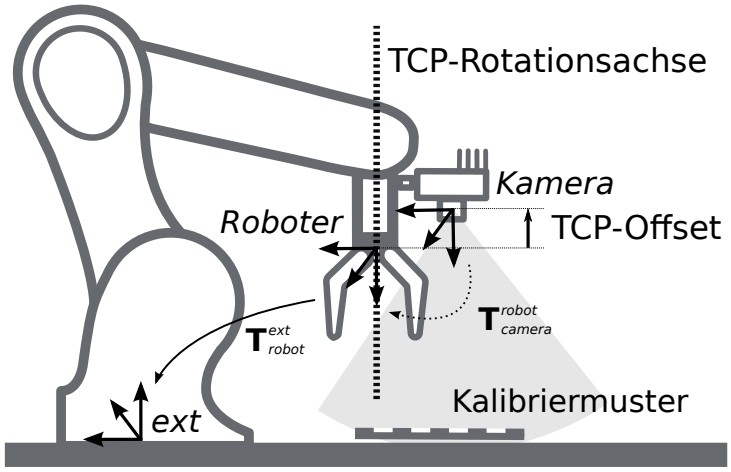


Abb. 5.19: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

Kalibrierung einer statisch montierten Kamera In Anwendungsfällen, bei denen die Kamera statisch verglichen zum Roboter montiert wird, muss das Kalibriermuster, wie im Beispiel in Abb. 5.20 und Abb. 5.21 angegeben, angebracht werden.

Bemerkung: Für das Modul zur Hand-Auge-Kalibrierung spielt es keine Rolle, wie das Kalibriermuster in Bezug auf das benutzerdefinierte *Roboter*-Koordinatensystem genau angebracht und positioniert wird. Das bedeutet, dass die relative Positionierung des Kalibriermusters zu diesem Koordinatensystem weder bekannt sein muss, noch für die Kalibrierroutine relevant ist (siehe in Abb. 5.21).

Warnung: Es ist äußerst wichtig, das Kalibriermuster sicher am Roboter anzubringen, damit sich seine relative Position in Bezug auf das in Schritt 2 der *Kalibrierroutine* (Abschnitt 5.3.1.3) vom Benutzer definierte *Roboter*-Koordinatensystem nicht verändert.

In diesem Anwendungsfall ist das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 5.3.1.3) die Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{ext}} = \mathbf{R}^{\text{ext}}_{\text{camera}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}^{\text{ext}}_{\text{camera}} \,, \tag{5.4}$$

wobei $\mathbf{p}_{\text{ext}} = (x, y, z)^T$ ein 3D-Punkt im externen Referenzkoordinatensystem *ext*, $\mathbf{p}_{\text{camera}}$ derselbe Punkt im Kamerakoordinatemsystem *camera* und $\mathbf{R}^{\text{ext}}_{\text{camera}}$ sowie $\mathbf{t}^{\text{ext}}_{\text{camera}}$ die 3×3 Rotationsmatrix und 3×1 Translationsvektor der Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$ sind. In der Praxis wird die Rotation für das Kalibrierergebnis und die Roboterposen als Eulerwinkel oder Quaternion anstatt einer Rotationsmatrix definiert (siehe *Formate für Posendaten*, Abschnitt 10.1).

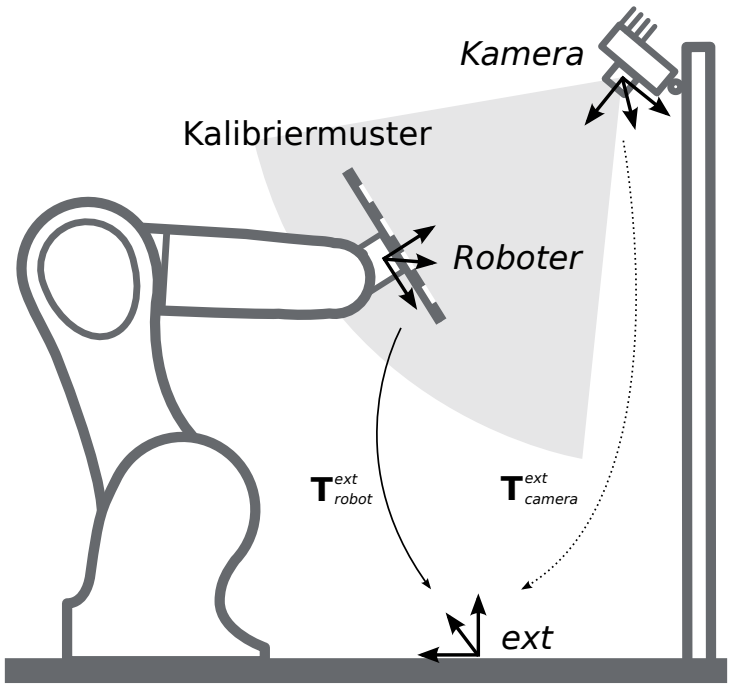


Abb. 5.20: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer statisch montierten Kamera: Sie wird mit einer festen Position relativ zu einem benutzerdefinierten externen Referenzkoordinatensystem *ext* (z.B. Weltkoordinatensystem oder Roboter-Montagepunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems in Bezug auf dieses Koordinatensystem während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibrierprozesses ist die gewünschte Kalibriertransformation $\mathbf{T}^{\text{ext}}_{\text{camera}}$, d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten externen Koordinatensystem *ext*.

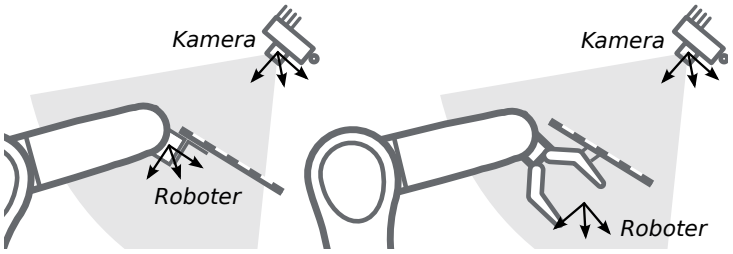


Abb. 5.21: Alternative Montageoptionen für die Befestigung des Kalibriermusters am Roboter

Zusätzliche Benutzereingaben werden benötigt, falls die Bewegung des Roboters so beschränkt ist, dass der Tool Center Point (TCP) nur um eine Achse rotieren kann. Das ist üblicherweise für Roboter mit vier Freiheitsgraden (4DOF) der Fall, welche häufig zur Palettierung eingesetzt werden. In diesem Fall muss der Benutzer angeben, welche Achse des Roboterkoordinatensystems der Rotationsachse des TCP entspricht. Außerdem muss der vorzeichenbehaftete Offset vom TCP zur sichtbaren Oberfläche des Kalibriermusters entlang der TCP-Rotationsachse angegeben werden. Das Kalibriermuster muss so angebracht werden, dass die TCP-Rotationsachse orthogonal zum Kalibriermuster verläuft. Abb. 5.22 zeigt die Situation.

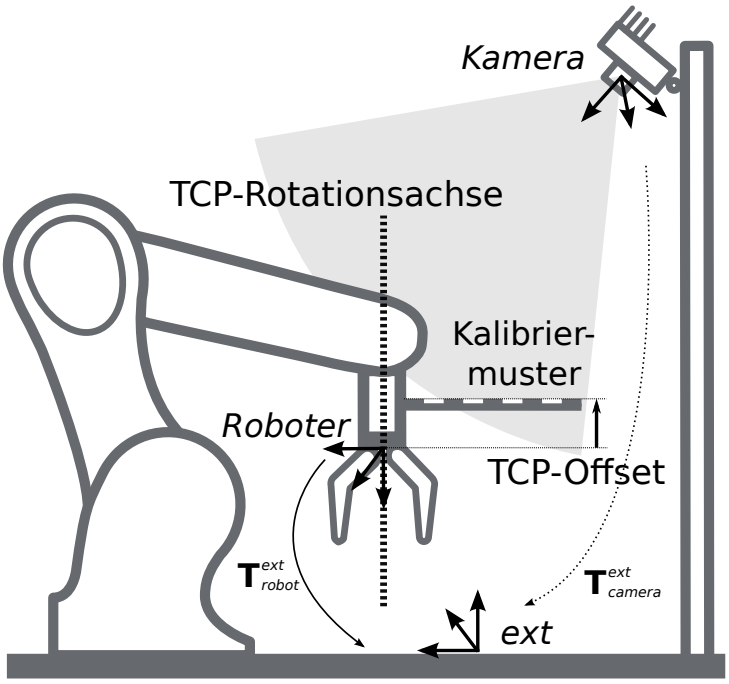


Abb. 5.22: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zur sichtbaren Oberfläche des Kalibriermusters entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

5.3.1.3 Kalibrierroutine

Die allgemeine Hand-Auge-Kalibrierroutine besteht aus den in Abb. 5.23 angegebenen drei Schritten. Auch der Hand-Auge-Kalibriervorgang der *Web GUI* (Abschnitt 6.1) greift diese drei Schritte auf.

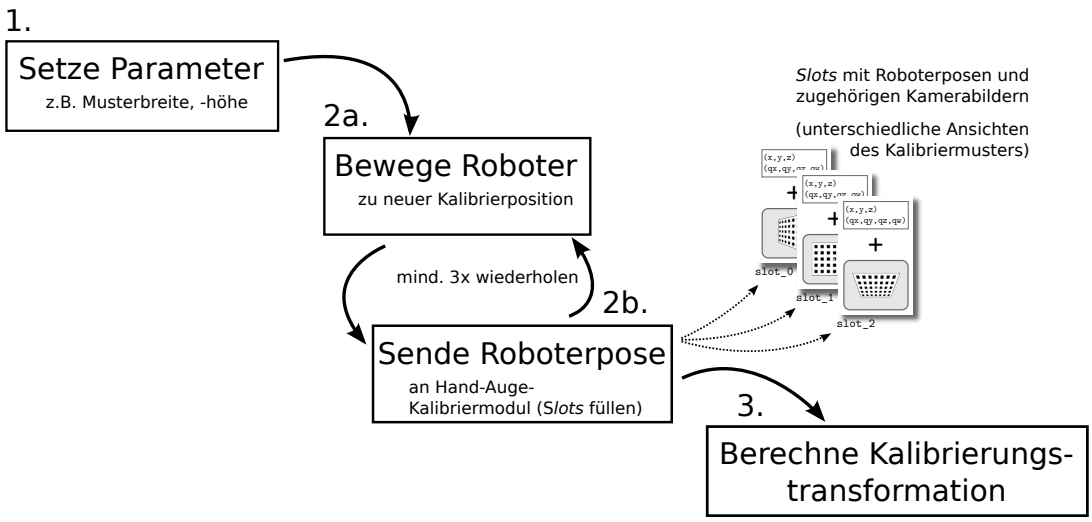


Abb. 5.23: Darstellung der drei Schritte der Hand-Auge-Kalibrierroutine

Schritt 1: Einstellung der Parameter

Bevor mit der eigentlichen Kalibrierroutine begonnen werden kann, müssen die Parameter für Kalibriermuster und Montage eingestellt werden. Für die REST-API sind die entsprechenden *Parameter* (Abschnitt 5.3.1.4) aufgelistet.

Web GUI-Beispiel: Die Web GUI bietet eine Oberfläche, über die sich diese Parameter im ersten Schritt der Kalibrierroutine, wie in Abb. 5.24 gezeigt, erfassen lassen. Neben den Angaben zur Mustergröße und Kameramontage kann der Benutzer in der Web GUI auch die Kalibrierung von 4DOF-Robotern einstellen. In diesem Fall müssen die Rotationsachse, sowie der Offset vom TCP zum Kamerakoordinatensystem (für Kameras am Roboter) oder zur Oberfläche des Kalibriermusters (für statische Kameras) angegeben werden. Außerdem kann das Format für Posen eingestellt werden. 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 10.1).

> **Bemerkung:** 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.

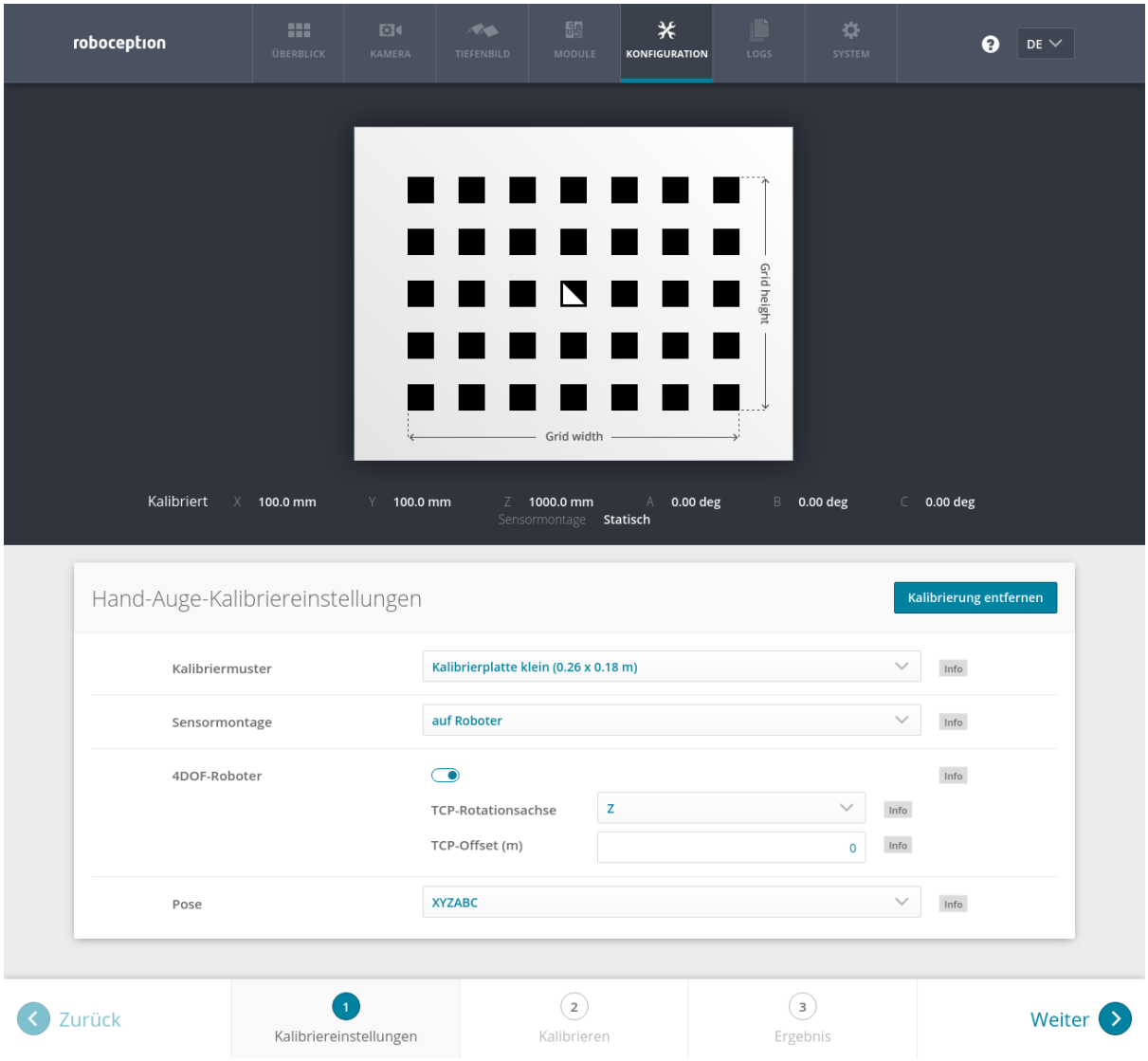


Abb. 5.24: Erfassung der Parameter zur Hand-Auge-Kalibrierung in der Web GUI des *rc_cube*

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 vollständig sichtbar ist. Zudem müssen die Roboterpositionen sorgsam ausgewählt werden, damit das Kalibriermuster aus unterschiedlichen Perspektiven aufgenommen wird. Abb. 5.25 zeigt eine schematische Darstellung der empfohlenen acht Ansichten.

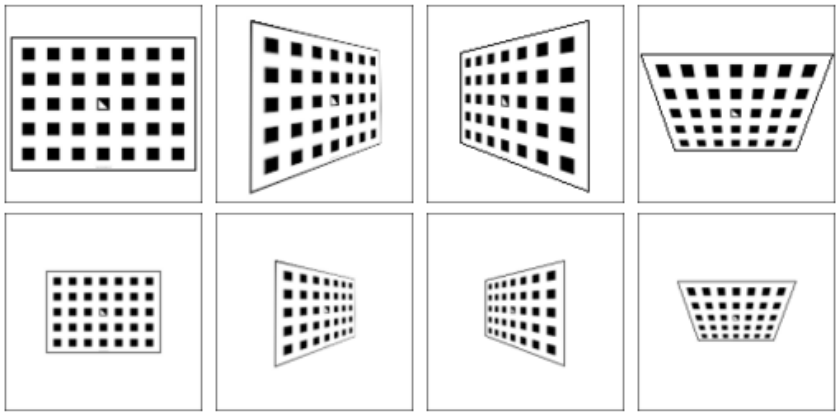


Abb. 5.25: Empfohlene Ansichten des Kalibriermusters während des Kalibriervorgangs. Im Fall von 4DOF-Robotern müssen andere Ansichten gewählt werden, welche so unterschiedlich wie möglich sein sollten.

Warnung: Die Kalibrierqualität, d.h. die Genauigkeit des berechneten Kalibrierergebnisses, hängt von den Ansichten des Kalibriermusters ab. Je vielfältiger die Perspektiven sind, desto besser gelingt die Kalibrierung. Werden sehr ähnliche Ansichten ausgewählt, d.h. 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 hinterlegt werden können. Alle gefüllten Slots werden dann verwendet, um die gewünschte Kalibriertransformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Kameras) bzw. dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Kameras) zu berechnen.

Bemerkung: 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 **acht Kalibrierposen empfohlen**.

Um diese Posen programmgesteuert zu übertragen, bietet die REST-API den Service `set_pose` (siehe *Services*, Abschnitt 5.3.1.5).

Web GUI-Beispiel: Nachdem die Kalibriereinstellungen in Schritt 1 abgeschlossen und die Schaltfläche *Weiter* betätigt wurde, bietet die Web GUI acht 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.

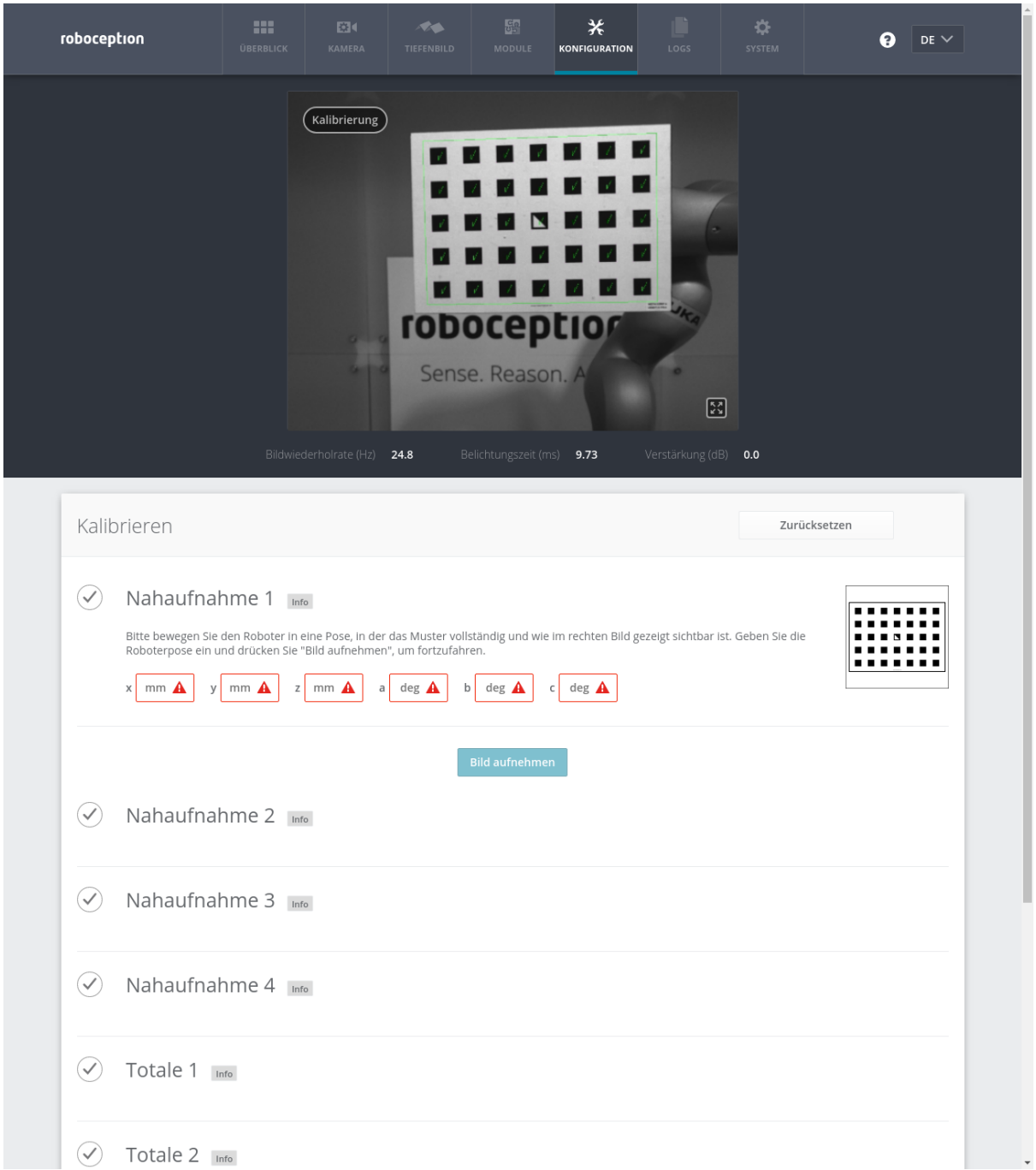


Abb. 5.26: Erstes Beispielbild für den Hand-Auge-Kalibriervorgang einer statisch montierten Kamera

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.

Bemerkung: 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.

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

rung fehlschlägt.

Dieser Vorgang ist insgesamt achtmal zu wiederholen. Vorausgesetzt, die in Abb. 5.25 dargestellten Empfehlungen zur Aufnahme des Kalibriermusters aus verschiedenen Blickwinkeln jeweils aus der Nähe und aus der Ferne wurden eingehalten, werden die folgenden Kamerabilder mit den jeweiligen Roboterposen an das Softwaremodul zur Hand-Auge-Kalibrierung übertragen:

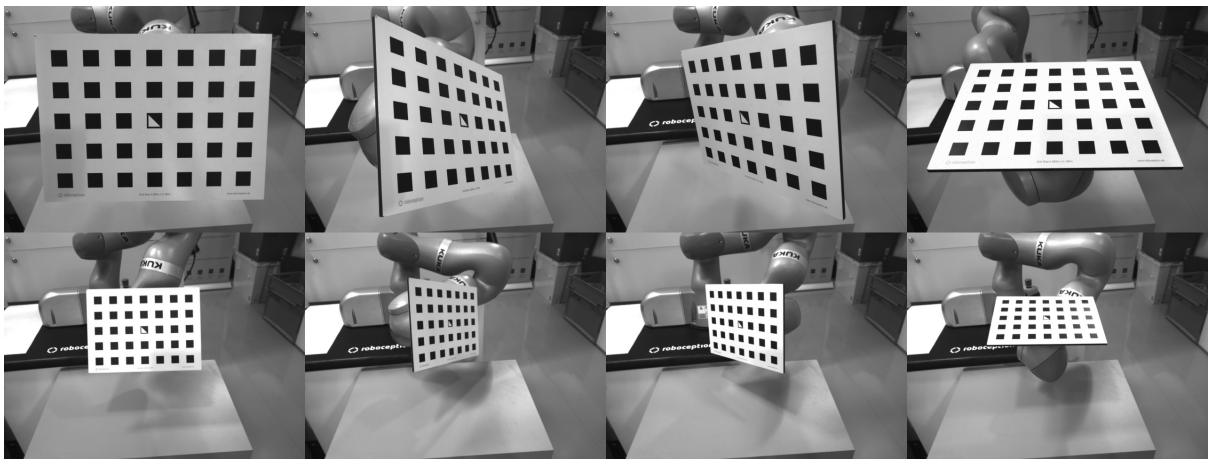


Abb. 5.27: Beispiel-Kamerabilder für die Hand-Auge-Kalibrierung

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 5.3.1.5). Je nachdem, wie die Kamera montiert ist, wird dabei die Transformation (d.h. die Pose) zwischen dem *Kamera*-Koordinatensystem und entweder dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Kameras) oder dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Kameras) berechnet und ausgegeben (siehe *Kameramontage*, Abschnitt 5.3.1.2).

Damit der Benutzer die Qualität der resultierenden Kalibriertransformation beurteilen kann, gibt das Modul die translatorischen und rotatorischen Kalibrierfehler an. Diese Werte werden aus der Varianz des Kalibrierergebnisses berechnet.

Web GUI-Beispiel: Nachdem die letzte der acht 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. Der Benutzer hat die Möglichkeit die Einstellungen für 4DOF-Roboter anzugeben oder zu korrigieren. Nach jeder Änderung muss die Schaltfläche *Neu berechnen* angeklickt werden.

Im Beispiel in Abb. 5.28 ist 4DOF ausgeschaltet und die Kamera statisch montiert. Die resultierende Ausgabe ist die Pose der linken Kamera im externen Koordinatensystem des Roboters. Der angegebene translatorische Fehler ist 0.67 mm, der rotatorische Fehler ist 0.83 Grad.

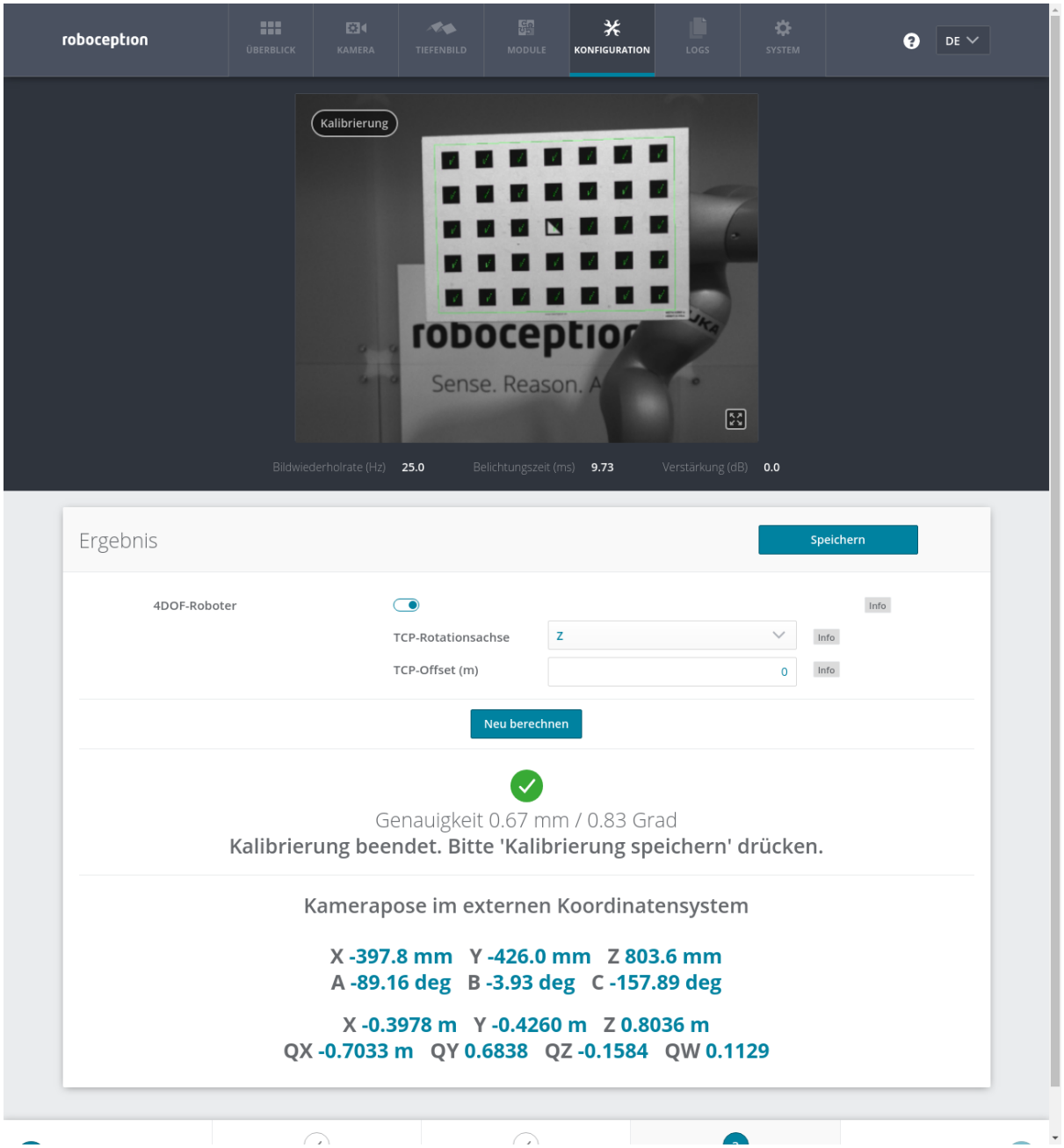


Abb. 5.28: Ergebnis der Hand-Auge-Kalibrierung, dargestellt in der Web GUI

5.3.1.4 Parameter

Das Modul zur Hand-Auge-Kalibrierung wird in der REST-API als `rc_hand_eye_calibration` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *Hand-Auge-Kalibrierung* unter dem Menüpunkt *Konfiguration* dargestellt. Der Benutzer kann die Kalibrierparameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.29: Laufzeitparameter des `rc_hand_eye_calibration`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`grid_height`	float64	0.0	10.0	0.0	Höhe des Kalibriermusters in Metern
`grid_width`	float64	0.0	10.0	0.0	Breite des Kalibriermusters in Metern
`robot_mounted`	bool	false	true	true	Angabe, ob der *rc_visard* auf einem Roboter montiert ist
`tcp_offset`	float64	-10.0	10.0	0.0	Offset vom TCP entlang tcp_rotation_axis
`tcp_rotation_axis`	int32	-1	2	-1	-1 für aus, 0 für x, 1 für y, 2 für z

Beschreibung der Laufzeitparameter

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

`grid_width` (*Breite des Musters (m)*)

Breite des Kalibriermusters in Metern. Die Breite sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, angegeben werden.

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

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?grid_width=<value>
```

`grid_height` (*Höhe des Musters (m)*)

Höhe des Kalibriermusters in Metern. Die Höhe sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, angegeben werden.

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

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?grid_height=<value>
```

`robot_mounted` (*Kamera Montageart*)

Ist dieser Parameter auf *true* gesetzt, dann ist die Kamera an einem Roboter montiert. Ist er auf *false* gesetzt, ist sie statisch montiert und das Kalibriermuster ist am Roboter angebracht.

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

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?robot_mounted=<value>
```

`tcp_offset` (*TCP Offset*)

Der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem (für Kameras auf dem Roboter) oder der sichtbaren Oberfläche des Kalibriermusters (für statische Kameras) entlang der TCP-Rotationsachse in Metern. Dies wird benötigt, falls die Roboterbewegung eingeschränkt ist, sodass der TCP nur um eine Achse gedreht werden kann (z.B. bei 4DOF-Robotern).

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

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?tcp_offset=<value>
```

`tcp_rotation_axis` (*TCP-Rotationsachse*)

Die Achse des Roboterkoordinatensystems, um die der Roboter seinen TCP drehen kann. 0 für X-, 1 für Y- und 2 für Z-Achse. Dies wird benötigt falls, die Roboterbewegung eingeschränkt ist, sodass der TCP nur um eine Achse gedreht werden kann (z.B. bei 4DOF-Robotern). -1 bedeutet, dass der Roboter seinen TCP um zwei unabhängige Achsen drehen kann. `tcp_offset` wird in diesem Fall ignoriert.

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

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

(*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 10.1). Wird die Kalibrierung über die REST-API vorgenommen, dann wird das Kalibrierergebnis immer im Format *XYZ+Quaternion* angegeben.

5.3.1.5 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_cube* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/save_parameters
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

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.

Warnung: Durch den Aufruf dieses Services gehen die aktuellen Parametereinstellungen für dieses Modul unwiderruflich verloren.

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

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.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/reset_calibration
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

set_pose

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

Das `slot`-Argument wird verwendet, um den verschiedenen Kalibrierpositionen Ziffern zuzuordnen. Wann immer der Service `set_pose` aufgerufen wird, wird ein Kamerabild aufge-

zeichnet. Dieser Service schlägt fehl, wenn das Kalibriermuster im aktuellen Bild nicht erkannt werden kann.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/set_pose
```

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

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "slot": "int32"
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Tab. 5.30: Rückgabewerte des `set_pose`-Services

`status`	`success`	Beschreibung
1	`true`	Pose erfolgreich gespeichert
3	`true`	Pose erfolgreich gespeichert. Es wurden genügend Posen für die Kalibrierung gespeichert, d.h. die Kalibrierung kann durchgeführt werden
4	`false`	das Kalibriermuster wurde nicht erkannt, z.B. weil es im Kamerabild nicht vollständig sichtbar ist
8	`false`	keine Bilddaten verfügbar
12	`false`	die angegebenen Orientierungswerte sind ungültig

`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. Damit die Kalibrierung für andere Module mit `get_calibration` verfügbar ist und persistent gespeichert wird, muss `save_calibration` aufgerufen werden.

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

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/calibrate
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "calibrate",
  "response": {
    "error": "float64",
    "message": "string",
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
    "status": "int32",
    "success": "bool",
    "translation_error_meter": "float64"
  }
}
```

Das Feld `error` gibt den Kalibrierfehler in Pixeln an, der aus dem translatorischen Fehler `translation_error_meter` und dem rotatorischen Fehler `rotation_error_degree` berechnet wird. Dieser Wert wird nur aus Kompatibilitätsgründen mit älteren Versionen angegeben. Die translatorischen und rotatorischen Fehler sollten bevorzugt werden.

Tab. 5.31: Rückgabewerte des `calibrate`-Services

`status`	`success`	Beschreibung
0	`true`	Kalibrierung erfolgreich, das Kalibrierergebnis wurde zurückgegeben.
1	`false`	Nicht genügend Posen gespeichert, um die Kalibrierung durchzuführen
2	`false`	Das berechnete Ergebnis ist ungültig, bitte prüfen Sie die Eingabewerte.
3	`false`	Die angegebenen Abmessungen des Kalibriermusters sind ungültig.
4	`false`	Ungenügende Rotation, `tcp_offset` and `tcp_rotation_axis` müssen angegeben werden
5	`false`	Genügend Rotation verfügbar, `tcp_rotation_axis` muss auf -1 gesetzt werden
6	`false`	Die Posen sind nicht unterschiedlich genug.

set_calibration

Hiermit wird die übergebene Transformation als Hand-Auge-Kalibrierung gesetzt. Die Kalibrierung wird im gleichen Format erwartet, in dem sie beim `calibrate` und `get_calibration` Aufruf zurückgegeben wird. Die gegebene Kalibrierung wird auch persistent gespeichert, indem intern `save_calibration` aufgerufen wird.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/set_calibration
```

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

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

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Tab. 5.32: Rückgabewerte des `set_calibration`-Services

status	success	Beschreibung
0	`true`	Setzen der Kalibrierung war erfolgreich
12	`false`	die angegebenen Orientierungswerte sind ungültig

save_calibration

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

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/save_calibration
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Tab. 5.33: Rückgabewerte des `save_calibration`-Services

status	success	Beschreibung
0	true	die Kalibrierung wurde erfolgreich gespeichert
1	false	die Kalibrierung konnte nicht im Dateisystem gespeichert werden
2	false	die Kalibrierung ist nicht verfügbar

remove_calibration

Dieser Service löscht die persistente Hand-Auge-Kalibrierung auf dem *rc_cube*. Nach diesem Aufruf gibt der `get_calibration` Service zurück, dass keine Hand-Auge-Kalibrierung vorliegt.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/remove_calibration
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Tab. 5.34: Rückgabewerte des `get_calibration`-Services

status	success	Beschreibung
0	true	persistente Kalibrierung gelöscht, Gerät nicht mehr kalibriert
1	true	keine persistente Kalibrierung gefunden, Gerät nicht kalibriert
2	false	die Kalibrierung konnte nicht gelöscht werden

get_calibration

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

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/get_calibration
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_calibration",
  "response": {
    "error": "float64",
    "message": "string",
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
    "status": "int32",
    "success": "bool",
    "translation_error_meter": "float64"
  }
}
```


Das Feld `error` gibt den Kalibrierfehler in Pixeln an, der aus dem translatorischen Fehler `translation_error_meter` und dem rotatorischen Fehler `rotation_error_degree` berechnet wird. Dieser Wert wird nur aus Kompatibilitätsgründen mit älteren Versionen angegeben. Die translatorischen und rotatorischen Fehler sollten bevorzugt werden.

Tab. 5.35: Rückgabewerte des `get_calibration`-Services

`status`	`success`	Beschreibung
0	`true`	eine gültige Kalibrierung wurde zurückgegeben
2	`false`	die Kalibrierung ist nicht verfügbar

5.3.2 Region of Interest

5.3.2.1 Einleitung

Die Region of Interest (ROI) Funktionalität wird von einer internen ROI-Komponente bereitgestellt und kann nur über die Softwaremodule genutzt werden, die diese Funktionalität anbieten.

Die 3D ROI-Funktionalität wird von den *ItemPick und BoxPick* (Abschnitt 5.2.3) Modulen, dem *CAD-Match* (Abschnitt 5.2.5) Modul und dem *LoadCarrier* (Abschnitt 5.2.1) Modul angeboten.

Die 2D ROI-Funktionalität wird vom *SilhouetteMatch* (Abschnitt 5.2.4) Modul und dem *LoadCarrier* (Abschnitt 5.2.1) Modul angeboten.

5.3.2.2 Region of Interest

Eine sogenannte Region of Interest (ROI) definiert ein abgegrenztes Raumvolumen (3D ROI, `region_of_interest`) oder eine rechteckige Region im linken Kamerabild (2D ROI, `region_of_interest_2d`), welche für eine spezifische Anwendung relevant sind.

Eine ROI kann das Volumen, in dem ein Load Carrier gesucht wird, einschränken, oder einen Bereich definieren, der nur die zu erkennenden oder zu greifenden Objekte enthält. Die Verarbeitungszeit kann sich deutlich verringern, wenn eine ROI genutzt wird.

Folgende Arten von 3D ROIs (`type`) werden unterstützt:

- `BOX`, für quaderförmige ROIs mit den Abmessungen `box.x`, `box.y`, `box.z`.
- `SPHERE`, für kugelförmige ROIs mit dem Radius `sphere.radius`.

Die Pose `pose` einer 3D ROI kann entweder relativ zum *Kamera*-Koordinatensystem `camera` oder mithilfe der Hand-Auge-Kalibrierung im *externen* Koordinatensystem `external` angegeben werden. Das externe Koordinatensystem steht nur zur Verfügung, wenn eine *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1) durchgeführt wurde. Wenn der *rc_visard* am Roboter montiert ist, und die ROI im externen Koordinatensystem definiert ist, dann muss jedem Detektions-Service, der diese ROI benutzt, die aktuelle Roboterpose übergeben werden.

Eine 2D ROI ist als rechteckiger Teil des linken Kamerabilds definiert und kann sowohl über die *REST-API-Schnittstelle* (Abschnitt 6.3) als auch über die *rc_cube Web GUI* (Abschnitt 6.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Konfiguration* gesetzt werden. Die Web GUI bietet hierfür ein einfach zu benutzendes Werkzeug an. Jeder ROI muss ein eindeutiger Name zugewiesen werden, um diese später adressieren zu können.

In der REST-API ist eine 2D-ROI über folgende Werte spezifiziert:

- `id`: Eindeutiger Name der ROI
- `offset_x`, `offset_y`: Abstand in Pixeln von der oberen rechten Bildecke entlang der x- bzw. y-Achse
- `width`, `height`: Breite und Höhe in Pixeln

Der *rc_cube* erlaubt das Speichern von bis zu 50 verschiedenen 3D ROIs und der gleichen Anzahl von 2D ROIs. Die Konfiguration von ROIs erfolgt in der Regel offline während der Einrichtung der gewünschten Anwendung. Die Konfiguration kann mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) des Moduls, das die Region of Interest Funktionalität anbietet, vorgenommen werden, oder über die *rc_cube Web GUI* (Abschnitt 6.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Konfiguration*.

Bemerkung: Die erstellten ROIs sind persistent auf dem *rc_cube* gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

5.3.2.3 Parameter

Die ROI-Komponente hat keine Laufzeitparameter.

5.3.2.4 Services

Die angebotenen Services der ROI-Funktionalität können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) des Moduls, das die ROI Funktionalität anbietet, oder über die *rc_cube Web GUI* (Abschnitt 6.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Konfiguration* ausprobiert und getestet werden.

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

Die folgende Tabelle führt die möglichen Rückgabe-Codes an:

Tab. 5.36: Rückgabe-Codes der Region of Interest Funktionalität

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an ROIs überschritten wurde.
10	Die maximal speicherbare Anzahl an ROIs wurde erreicht.
11	Mit dem Aufruf von `set_region_of_interest` oder `set_region_of_interest_2d` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.

Alle Softwaremodule, die die ROI-Funktionalität anbieten, stellen folgende Services zur Verfügung.

set_region_of_interest

speichert eine 3D ROI auf dem *rc_cube*. Alle ROIs sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "region_of_interest": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "sphere": {
        "radius": "float64"
      },
      "type": "string"
    }
  }
}
```

Die Definition des Typs `region_of_interest` wird in *Region of Interest* (Abschnitt 5.3.2.2) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

set_region_of_interest_2d

speichert eine 2D ROI auf dem *rc_cube*. Alle ROIs sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "region_of_interest_2d": {
      "height": "uint32",
      "id": "string",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    }
  }
}
```

Die Definition des Typs `region_of_interest_2d` wird in *Region of Interest* (Abschnitt 5.3.2.2) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_region_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_regions_of_interest

gibt die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D ROIs zurück. Werden keine `region_of_interest_ids` angegeben, enthält die Serviceantwort alle gespeicherten ROIs.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_regions_of_interest",
  "response": {
    "regions_of_interest": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "sphere": {
          "radius": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_regions_of_interest_2d

gibt die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D ROIs zurück. Werden keine `region_of_interest_2d_ids` angegeben, enthält die Serviceantwort alle gespeicherten ROIs.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_regions_of_interest_2d",
  "response": {
    "regions_of_interest": [
      {
        "height": "uint32",
        "id": "string",
        "offset_x": "uint32",
        "offset_y": "uint32",
        "width": "uint32"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_regions_of_interest

löscht die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D ROIs. Alle zu löschenden ROIs müssen explizit angegeben werden.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

delete_regions_of_interest_2d

löscht die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D ROIs. Alle zu löschenden ROIs müssen explizit angegeben werden.

Dieser Service kann wie folgt aufgerufen werden.

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

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

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_regions_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

5.3.3 CollisionCheck

5.3.3.1 Einleitung

Das CollisionCheck Modul ist ein optionales Modul, welches intern auf dem *rc_cube* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module ItemPick, BoxPick oder SilhouetteMatch vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 7.5).

Das Modul ermöglicht die Kollisionsprüfung zwischen dem Greifer und dem Load Carrier oder anderen detektierten Objekten (nur in Kombination mit *CADMatch*, Abschnitt 5.2.5, und *SilhouetteMatch*, Abschnitt 5.2.4). Es ist in die Module *ItemPick und BoxPick* (Abschnitt 5.2.3), *SilhouetteMatch* (Abschnitt 5.2.4) und *CADMatch* (Abschnitt 5.2.5) integriert, kann aber auch als eigenständiges Modul genutzt werden.

Warnung: Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch, anderen Objekten oder dem Objekt im Greifer. Nur in Kombination mit *CADMatch* (Abschnitt 5.2.5) und *SilhouetteMatch* (Abschnitt 5.2.4), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im jeweiligen Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und den anderen *detektierten* Objekten geprüft. Kollisionen mit Objekten, die nicht detektiert werden können, werden nicht geprüft.

5.3.3.2 Erstellen eines Greifers

Der Greifer ist eine Kollisionsgeometrie, die zur Prüfung auf Kollisionen zwischen dem geplanten Griff und dem Load Carrier verwendet wird. Der Greifer kann aus bis zu 15 miteinander verbundenen Elementen bestehen.

Es sind folgende Arten von Elementen möglich:

- Quader (`BOX`), mit den Abmessungen `box.x`, `box.y`, `box.z`.
- Zylinder (`CYLINDER`), mit dem Radius `cylinder.radius` und der Höhe `cylinder.height`.

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

Die Konfiguration des Greifers wird in der Regel während des Setups der Zielanwendung durchgeführt. Das kann über die *REST-API-Schnittstelle* (Abschnitt 6.3) oder die *rc_cube Web GUI* (Abschnitt 6.1) geschehen.

Flanschradius

Es werden standardmäßig nur Kollisionen mit dem Greifer, nicht aber mit der Robotergeometrie geprüft. Um Kollisionen zwischen dem Load Carrier und dem Roboter zu vermeiden, kann über den Laufzeitparameter `check_flange` (siehe *Übersicht der Parameter*, Abschnitt 5.3.3.4) ein zusätzlicher optionaler Test aktiviert werden. Dieser Test erkennt alle Griffe als Kollisionen, bei denen sich ein Teil des Roboterflanschs innerhalb des Load Carriers befinden würde (siehe Abb. 5.29). Der Test basiert auf der Greifergeometrie und dem Flanschradius.

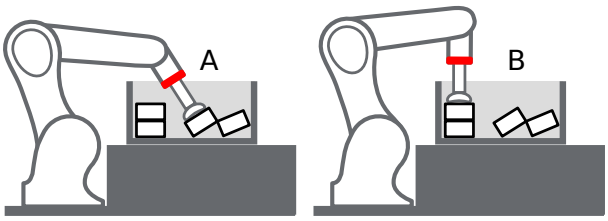


Abb. 5.29: Fall A: Der Griff wird nur als Kollision erkannt, wenn `check_flange` auf *true* gesetzt ist, denn der Flansch (rot) befindet sich im Load Carrier. Fall B: Der Griff ist in jedem Fall kollisionsfrei.

Erstellen eines Greifers über die REST-API

Bei der Greifererstellung über die *REST-API-Schnittstelle* (Abschnitt 6.3) hat jedes Greifer-Element ein *Parent*-Element, das die Verbindung zwischen den Elementen definiert. Der Greifer wird immer vom Roboterflansch ausgehend in Richtung TCP aufgebaut, und mindestens ein Element muss den Parent ‚flange' (Flansch) haben. Die IDs der Elemente müssen eindeutig sein und dürfen nicht ‚tcp' oder ‚flange' sein. Die Pose des Elements muss im Koordinatensystem des *Parent*-Elements angegeben werden. In der REST-API-Repräsentation ist das Koordinatensystem jedes Elements genau in seinem geometrischen Mittelpunkt. Damit ein Element also genau unterhalb seines *Parent*-Elements platziert wird, muss seine Position aus der Höhe des *Parent*-Elements und seiner eigenen Höhe berechnet werden (siehe Abb. 5.30).

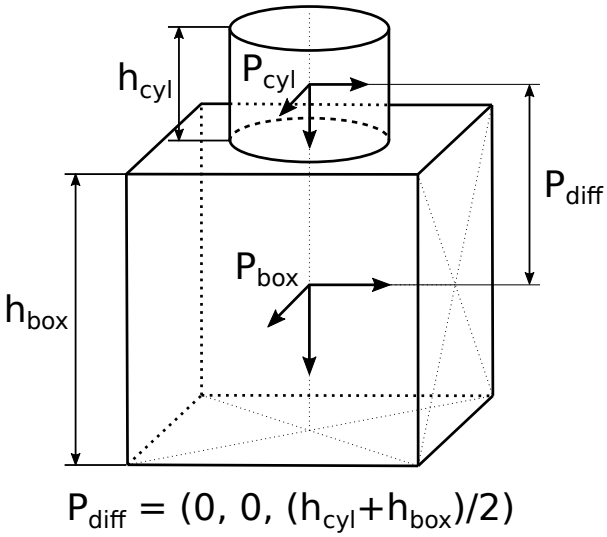


Abb. 5.30: Bezugskoordinatensysteme für das Erstellen von Greifern über die REST-API

Das Bezugskoordinatensystem für das erste Element liegt immer im Mittelpunkt des Roboterflanschs, wobei die z-Achse nach unten gerichtet ist. Über die REST-API können Greifer mit einer Baumstruktur erstellt werden, bei denen mehrere Elemente dasselbe *Parent*-Element haben.

Erstellen eines Greifers in der Web GUI

Die *CollisionCheck*-Seite in der *rc_cube Web GUI* (Abschnitt 6.1) bietet ein vereinfachtes Interface zum Erstellen von Greifern. Es ermöglicht die Auswahl des Typs, der Größe und der Position jedes Greiferelements. In der Web GUI-Repräsentation bezieht sich die Position jedes Elements auf die Unterseite des darüber liegenden *Parent*-Elements. Ein Element mit der Position (0, 0, 0) wird also genau unterhalb seines *Parent*-Elements platziert. Greifer mit einer Baumstruktur oder mit rotierten Elementen können nicht über die Web GUI erstellt werden.

Berechnete TCP-Position

Nach dem Erstellen des Greifers mit dem Service `set_gripper`, wird die TCP-Position im Flanschkoordinatensystem berechnet und als `tcp_pose_flange` zurückgegeben. Dieser Wert muss mit den tatsächlichen TCP-Koordinaten des Roboters übereinstimmen.

Nicht-rotationssymmetrische Greifer erstellen

Bei Greifern, die nicht rotationssymmetrisch um die z-Achse sind, muss sichergestellt werden, dass der Greifer so montiert wird, dass seine Ausrichtung mit der im CollisionCheck-Modul gespeicherten Darstellung übereinstimmt.

5.3.3.3 Kollisionsprüfung

Stand-Alone Kollisionsprüfung

Der Service `check_collisions` triggert die Kollisionsprüfung zwischen dem angegebenen Greifer und dem angegebenen Load Carrier für jeden der übergebenen Greifpunkte. Eine Kollisionsprüfung mit anderen Objekten ist nicht möglich. Das CollisionCheck-Modul überprüft, ob sich der Greifer in Kollision

mit mindestens einem Load Carrier befindet, wenn sich der TCP an der Greifposition befindet. Es können mehrere Load Carrier gleichzeitig getestet werden. Der Griff wird als Kollision markiert, wenn es mit mindestens einem der definierten Load Carriern zu einer Kollision kommen würde.

Das Argument `pre_grasp_offset` (Greif-Offset) kann für eine erweiterte Kollisionsprüfung genutzt werden. Der Greif-Offset P_{off} ist der Offset vom Greifpunkt P_{grasp} zur Vorgreifposition P_{pre} im Koordinatensystem des Greifpunkts (siehe Abb. 5.31). Wenn der Greif-Offset angegeben wird, werden Greifpunkte auch dann als Kollisionen erkannt, wenn der Greifer an einem beliebigen Punkt während der linearen Bewegung zwischen Vorgreifposition und Greifposition in Kollision geraten würde.

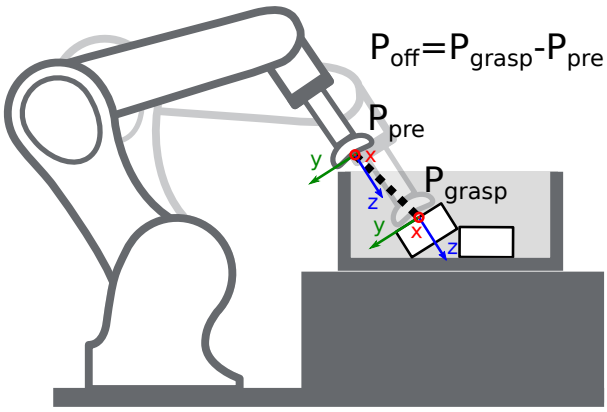


Abb. 5.31: Darstellung des Greif-Offsets für die Kollisionsprüfung. Im dargestellten Fall sind sowohl die Vorgreifposition als auch die Greifposition kollisionsfrei, aber die Trajektorie zwischen diesen Punkten hätte eine Kollision mit dem Load Carrier. Deswegen wird dieser Greifpunkt als Kollision erkannt.

Integrierte Kollisionsprüfung in anderen Modulen

Die Kollisionsprüfung ist in die Services der folgenden Softwaremodule integriert:

- *ItemPick und BoxPick* (Abschnitt 5.2.3): `compute_grasps` (siehe *compute_grasps für ItemPick*, Abschnitt 5.2.3.7 und *compute_grasps für BoxPick*, Abschnitt 5.2.3.7)
- *SilhouetteMatch* (Abschnitt 5.2.4): `detect_object` (siehe *detect_object*, Abschnitt 5.2.4.10)
- *CADMatch* (Abschnitt 5.2.5): `detect_object` (siehe *detect_object*, Abschnitt 5.2.5.8)

Jedem dieser Services kann ein `collision_detection`-Argument übergeben werden, das aus der ID des Greifers (`gripper_id`) und optional aus dem Greif-Offset (`pre_grasp_offset`, siehe *Stand-Alone Kollisionsprüfung*, Abschnitt 5.3.3.3) besteht. Wenn das `collision_detection` Argument übergeben wird, liefern diese Services nur die Greifpunkte zurück, an denen der Greifer nicht in Kollision mit dem erkannten Load Carrier ist. Dazu muss dem jeweiligen Service auch die ID des zu erkennenden Load Carriers übergeben werden. Nur für *CADMatch* (Abschnitt 5.2.5) und *SilhouetteMatch* (Abschnitt 5.2.4), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im jeweiligen Modul aktiviert ist, werden auch Greifpunkte, bei denen der Greifer in Kollision mit anderen *detektierten* Objekten wäre, herausgefiltert. Dabei ist das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen.

Warnung: Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch, anderen Objekten oder dem Objekt im Greifer. Nur in Kombination mit *CADMatch* (Abschnitt 5.2.5) und *SilhouetteMatch* (Abschnitt 5.2.4), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im jeweiligen Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und den anderen *detektierten* Objekten geprüft. Kollisionen mit Objekten, die nicht detektiert werden können, werden nicht geprüft.

Die Kollisionsprüfung wird von Laufzeitparametern beeinflusst, die weiter unten aufgeführt und beschrieben werden.

5.3.3.4 Parameter

Das CollisionCheck-Modul wird in der REST-API als `rc_collision_check` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *CollisionCheck* (unter der Seite *Konfiguration*) dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) ändern.

Übersicht der Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.37: Applikationsspezifische Laufzeitparameter des `rc_collision_check` Moduls

Name	Typ	Min	Max	Default	Beschreibung
`check_bottom`	bool	false	true	true	Kollisionsprüfung mit dem Boden des Load Carriers.
`check_flange`	bool	false	false	true	Die Position ist in Kollision, wenn der Flansch innerhalb des Load Carriers ist.
`collision_dist`	float64	0.0	0.1	0.01	Minimaler Abstand in Metern zwischen einem Greiferelement und einer Kollisionsgeometrie für einen kollisionsfreien Griff.

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite des CollisionCheck-Moduls der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben:

`collision_dist` (*Sicherheitsabstand*)

Minimaler Abstand zwischen einem Greiferelement und einer anderen Kollisionsgeometrie (Load Carrier und/oder detektiertes Objekt) für einen kollisionsfreien Griff.

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

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?collision_dist=<value>
```

`check_flange` (*Flansch-Check*)

Ermöglicht einen Sicherheitscheck mit dem Flansch, wie in *Flanschradius* (Abschnitt 5.3.3.2) beschrieben. Wenn dieser Parameter gesetzt ist, gelten alle Griffe, bei denen der Flansch innerhalb des Load Carriers wäre, als Kollisionen.

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

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_flange=<value>
```

`check_bottom` (*Boden-Check*)

Wenn dieser Check aktiviert ist, werden Kollisionen nicht nur mit den Load Carrier Wänden, sondern auch mit dem Boden geprüft. Falls der TCP innerhalb der Kollisionsgeometrie (z.B. innerhalb des Sauggreifers) liegt, ist es möglicherweise nötig, diesen Check zu deaktivieren.

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

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_bottom=<value>
```

5.3.3.5 Statuswerte

Statuswerte des `rc_collision_check`-Moduls:

Tab. 5.38: Statuswerte des `rc_collision_check`-Moduls

Name	Beschreibung
`last_evaluated_grasps`	Anzahl der ausgewerteten Griffe
`last_collision_free_grasps`	Anzahl der kollisionsfreien Griffe

5.3.3.6 Services

Die angebotenen Services von `rc_collision_check` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 6.3) oder der *rc_cube Web GUI* (Abschnitt 6.1) ausprobiert und getestet werden.

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

Die folgende Tabelle führt die möglichen Rückgabe-Codes an:

Tab. 5.39: Fehlercodes des CollisionCheck-Services

Code	Beschreibung
0	Erfolgreich
-1	Ein ungültiges Argument wurde übergeben.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-9	Lizenz für CollisionCheck ist nicht verfügbar.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Greifern überschritten wurde.
10	Die maximal speicherbare Anzahl an Greifern wurde erreicht.
11	Bestehender Greifer wurde überschrieben.

Das CollisionCheck-Modul stellt folgende Services zur Verfügung.

`set_gripper`

konfiguriert und speichert einen Greifer auf dem *rc_cube*. Alle Greifer sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/set_gripper
```

Request:

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


```
{
  "args": {
    "elements": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "cylinder": {
          "height": "float64",
          "radius": "float64"
        },
        "id": "string",
        "parent_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "type": "string"
      }
    ],
    "flange_radius": "float64",
    "id": "string",
    "tcp_parent_id": "string",
    "tcp_pose_parent": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Obligatorische Serviceargumente:

`elements`: Liste von geometrischen Elementen, aus denen der Greifer besteht. Jedes Element muss den `type` ‚CYLINDER' oder ‚BOX' haben, wobei die Dimensionen im Feld `cylinder` bzw. `box` angegeben werden. Die Pose jedes Elements muss im *Parent*-Koordinatensystem angegeben werden (siehe *Erstellen eines Greifers*, Abschnitt 5.3.3.2). Die `id` des Elements muss

eindeutig sein und darf nicht ‚tcp' oder ‚flange' sein. Die `parent_id` ist die ID des *Parent*-Elements, welche entweder ‚flange' ist oder der ID eines anderen Elements entsprechen muss.

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

`id`: Eindeutiger Name des Greifers.

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

`tcp_pose_parent`: Die Pose des TCP im Koordinatensystem des Elements, das in `tcp_parent_id` angegeben ist.

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_gripper",
  "response": {
    "gripper": {
      "elements": [
        {
          "box": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "cylinder": {
            "height": "float64",
            "radius": "float64"
          },
          "id": "string",
          "parent_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "type": "string"
        }
      ],
      "flange_radius": "float64",
      "id": "string",
      "tcp_parent_id": "string",
      "tcp_pose_flange": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "z": "float64"
        }
      },
      "tcp_pose_parent": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "type": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


`gripper`: Gibt den Greifer mit dem zusätzlichen Feld `tcp_pose_flange` zurück. Dieses Feld gibt die TCP-Koordinaten im Flanschkoordinatensystem an, um diese mit den Roboter-TCP-Koordinaten vergleichen zu können.

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

get_grippers

gibt die mit `gripper_ids` spezifizierten und gespeicherten Greifer zurück. Wenn keine `gripper_ids` angegeben werden, enthält die Serviceantwort alle gespeicherten Greifer.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/get_grippers
```

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


```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```


Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_grippers",
  "response": {
    "grippers": [
      {
        "elements": [
          {
            "box": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "cylinder": {
            "height": "float64",
            "radius": "float64"
          },
          "id": "string",
          "parent_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "type": "string"
        }
      ],
      "flange_radius": "float64",
      "id": "string",
      "tcp_parent_id": "string",
      "tcp_pose_flange": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "tcp_pose_parent": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "type": "string"
    }
  ],
  "return_code": {
    "message": "string",
    "value": "int16"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

delete_grippers

löscht die mit `gripper_ids` spezifizierten, gespeicherten Greifer. Alle zu löschenden Greifer müssen explizit angegeben werden.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/delete_grippers
```

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

```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

check_collisions

löst eine Kollisionsprüfung zwischen dem Greifer und einem Load Carrier aus.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/check_collisions
```

Request:

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

```
{
  "args": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "gripper_id": "string",
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        }
      }
    ],
    "pre_grasp_offset": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  }
}
```


Obligatorische Serviceargumente:

`grasps`: Liste von Griffen, die überprüft werden sollen.

`load_carriers`: Liste von Load Carriern, die auf Kollisionen überprüft werden sollen. Die Felder der Load Carrier Definition sind in *Erkennung von Load Carriern* (Abschnitt 5.2.1.4) beschrieben. Die Griffe und die Load Carrier Positionen müssen im selben Koordinatensystem angegeben werden.

`gripper_id`: Die ID des Greifers, der in der Kollisionsprüfung verwendet

werden soll. Der Greifer muss zuvor konfiguriert worden sein.

Optionale Serviceargumente:

`pre_grasp_offset`: Der Greif-Offset in Metern vom Greifpunkt zur Vorgreifposition. Wird ein Greif-Offset angegeben, dann wird die Kollisionsprüfung auf der gesamten linearen Trajektorie von der Vorgreifposition bis zur Greifposition durchgeführt.

Response:

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "check_collisions",
  "response": {
    "colliding_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "collision_free_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


`colliding_grasps`: Liste von Griffen, die in Kollision mit einem oder mehreren Load Carriern sind.

`collision_free_grasps`: Liste von kollisionsfreien Griffen.

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

`save_parameters`

speichert die aktuellen Parametereinstellungen des CollisionCheck-Moduls auf dem *rc_cube*. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden. Bei Firmware-Updates oder -Wiederherstellungen werden sie jedoch wieder auf den Standardwert gesetzt.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/save_parameters
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

`reset_defaults`

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

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

5.3.4 IOControl und Projektor-Kontrolle

Das IOControl Modul ist ein optionales Modul, welches intern auf dem *rc_visard* läuft und eine gesonderte IOControl *Lizenz* (Abschnitt 7.5) benötigt, die erworben werden muss. Diese Lizenz ist auf jedem *rc_visard*, der nach dem 01.07.2020 gekauft wurde, vorhanden.

Das IOControl-Modul ermöglicht das Lesen der digitalen Eingänge und die Kontrolle der digitalen Ausgänge (GPIOs) des *rc_visard*. Die Ausgänge können auf *aus* (LOW) oder *an* (HIGH) gesetzt werden. Sie

können auch so konfiguriert werden, dass sie genau für die Belichtungszeit jedes Bildes, oder auch nur jedes zweiten Bildes, *an* sind.

Das IOControl-Modul dient der Ansteuerung einer externen Lichtquelle oder eines Projektors, der an einen der GPIO-Ausgänge des *rc_visard* angeschlossen wird, und der mit der Bildaufnahme synchronisiert ist. Für den Fall, dass ein Musterprojektor für die Verbesserung des Stereo-Matchings verwendet wird, ist das projizierte Muster auch in den Intensitätsbildern sichtbar. Das kann für Bildverarbeitungs-Anwendungen, die auf dem Intensitätsbild basieren (z.B. Kantendetektion), von Nachteil sein. Aus diesem Grund erlaubt das IOControl-Modul auch das Setzen der Ausgänge für nur jedes zweite Kamerabild. Somit sind auch Intensitätsbilder ohne projiziertes Muster verfügbar.

5.3.4.1 Parameter

Das IOControl-Modul wird in der REST-API als `rc_iocontrol` bezeichnet und in der *Web GUI* (Abschnitt 6.1) auf der Seite *IOControl* (unter dem Menüpunkt *Konfiguration*) dargestellt. Der Benutzer kann die Parameter über die Web GUI, die REST-API (*REST-API-Schnittstelle*, Abschnitt 6.3), oder die GenICam-Schnittstelle mit den DigitalIOControl-Parametern `LineSelector` und `LineSource` ändern (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.2).

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 5.40: Laufzeitparameter des `rc_iocontrol`-Moduls

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

Beschreibung der Laufzeitparameter

`out1_mode` und `out2_mode` (*Ausgang 1 (Out1)* und *Ausgang 2 (Out2)*)

Die Betriebsarten für GPIO-Ausgang 1 und GPIO-Ausgang 2 können individuell gesetzt werden:

`Low` schaltet den GPIO-Ausgang permanent *aus* (LOW). Das ist die Standardeinstellung.

`High` schaltet den GPIO-Ausgang permanent *an* (HIGH).

`ExposureActive` schaltet den GPIO-Ausgang für die Belichtungszeit jedes Bildes *an* (HIGH).

`ExposureAlternateActive` schaltet den GPIO-Ausgang für die Belichtungszeit jedes zweiten Bildes *an* (HIGH).

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

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_mode|out2_mode>=<value>
```

Bemerkung: Die Parameter können nur verändert werden, wenn eine IOControl-Lizenz auf dem *rc_cube* verfügbar ist. Sonst gelten die Voreinstellungen für die Parameter, d.h. `out1_mode` = `Low` und `out2_mode` = `Low`.

Abb. 5.32 zeigt, welche Bilder für das Stereo-Matching und die GigE Vision-Übertragung in der Betriebsart `ExposureActive` mit einer benutzerdefinierten Bildwiederholrate von 8 Hz benutzt werden.

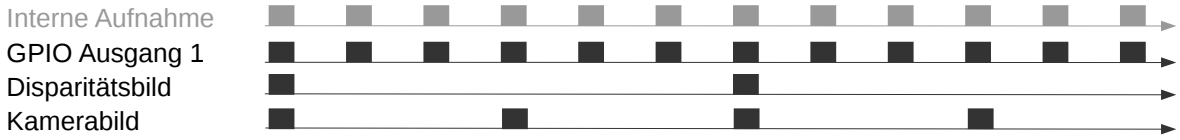


Abb. 5.32: Beispiel für die Nutzung der Betriebsart `ExposureActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes Bildes *an* (HIGH). Disparitätsbilder werden für Kamerabilder berechnet, die auch per GigE Vision in der benutzerdefinierten Bildwiederholrate versendet werden.

Die Betriebsart `ExposureAlternateActive` ist gedacht, um einen externen Musterprojektor anzusteuern, der am GPIO-Ausgang 1 des *rc_visard* angeschlossen ist. In diesem Fall nutzt das *Stereo-Matching-Modul* (Abschnitt 5.1.2) nur Bilder, bei denen GPIO-Ausgang 1 *an* (HIGH) ist, d.h. der Projektor ist an. Die maximale Bildwiederholrate, welche für das Stereo-Matching genutzt wird, ist hierbei die halbe vom Benutzer konfigurierte Bildwiederholrate (siehe *FPS*, Abschnitt 5.1.1.4). Alle Module, die Intensitätsbilder benutzen, wie z.B. *TagDetect* (Abschnitt 5.2.2) und *ItemPick* (Abschnitt 5.2.3), benutzen die Intensitätsbilder, bei denen der GPIO-Ausgang 1 *aus* (LOW) ist, d.h. der Projektor ist aus. Abb. 5.33 zeigt ein Beispiel.

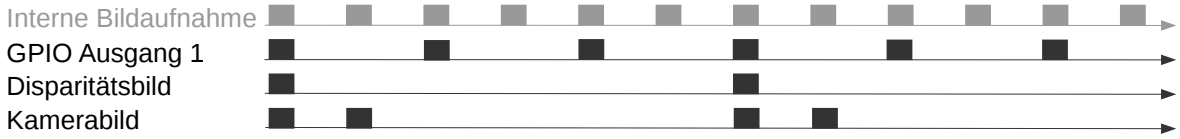


Abb. 5.33: Beispiel für die Nutzung der Betriebsart `ExposureAlternateActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes zweiten Bildes *an* (HIGH). Disparitätsbilder werden für Kamerabilder berechnet, bei denen GPIO-Ausgang 1 *an* (HIGH) ist, und die auch per GigE Vision in der benutzerdefinierten Bildwiederholrate versendet werden. In der Betriebsart `ExposureAlternateActive` werden Intensitätsbilder immer paarweise versendet: ein Bild mit GPIO-Ausgang 1 *an* (HIGH), für das ein Disparitätsbild verfügbar sein kann, und ein Bild mit GPIO-Ausgang 1 *aus* (LOW).

Bemerkung: In der Betriebsart `ExposureAlternateActive` gibt es zu einem Intensitätsbild mit angeschaltetem GPIO-Ausgang 1 (HIGH), d.h. mit projiziertem Muster, immer in 40 ms Abstand ein Intensitätsbild mit ausgeschaltetem GPIO-Ausgang 1 (LOW), d.h. ohne projiziertes Muster. Dies ist unabhängig von der benutzerdefinierten Bildwiederholrate und sollte in dieser speziellen Betriebsart für die Synchronisierung von Disparitäts- und projektionsfreien Kamerabildern berücksichtigt werden.

Die Funktionalität kann auch über die DigitalIOControl-Parameter der GenICam Schnittstelle kontrolliert werden (*GigE Vision 2.0/GenICam-Schnittstelle*, Abschnitt 6.2).

5.3.4.2 Services

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

Dieses Softwaremodul bietet folgende Services.

get_io_values

Mit diesem Aufruf kann der aktuelle Zustand der Ein- und Ausgänge (GPIOs) des *rc_visard* abgefragt werden.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/get_io_values
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_io_values",
  "response": {
    "in1": "bool",
    "in2": "bool",
    "out1": "bool",
    "out2": "bool",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Das Feld `timestamp` ist der Zeitpunkt der Messung.

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

Code	Beschreibung
0	Erfolgreich
-2	Interner Fehler
-9	Lizenz für *IOControl* ist nicht verfügbar

save_parameters

Beim Aufruf dieses Services werden die aktuellen Parametereinstellungen des Moduls auf dem *rc_cube* gespeichert. Das bedeutet, dass diese Werte selbst nach einem Neustart angewandt werden.

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/save_parameters
```

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

reset_defaults

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

Dieser Service kann wie folgt aufgerufen werden.

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

Dieser Service hat keine Argumente.

Die Definition der *Response* mit jeweiligen Datentypen ist:

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

Warnung: Durch den Aufruf dieses Services gehen die aktuellen Parametereinstellungen für das IOControl-Modul unwiderruflich verloren.

6 Schnittstellen

Es stehen die folgenden Schnittstellen zur Konfiguration und Datenübertragung des *rc_cube* zur Verfügung:

- *Web GUI* (Abschnitt 6.1)

  Leicht zu bedienendes grafisches Interface zum Konfigurieren und Kalibrieren des *rc_cube*, zum Anzeigen von Livebildern, Aufrufen von Services, Visualisieren von Ergebnissen, usw.

- *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 6.2)

  Konfiguration bild- und kamerabezogener Einstellungen.

- *REST-API-Schnittstelle* (Abschnitt 6.3)

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

- *Ethernet KRL Schnittstelle (EKI)* (Abschnitt 6.4)

  API zum Konfigurieren des *rc_cube* und Ausführen von Service-Anfrage von KUKA KSS Robotern aus.

- *gRPC Bilddatenschnittstelle* (Abschnitt 6.5)

  Synchronisierte Bilddaten per gRPC.

- *Zeitsynchronisation* (Abschnitt 6.6)

  Zeitsynchronisation zwischen dem *rc_cube* und dem Anwendungs-PC.

6.1 Web GUI

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

6.1.1 Zugriff auf die Web GUI

Auf die Web GUI kann über die IP-Adresse des *rc_cube* 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_cube-Geräten* (Abschnitt 3.4) beschrieben, ein Doppelklick auf das gewünschte Gerät vorgenommen wird.

Alternativ konfigurieren einige Netzwerkumgebungen den eindeutigen Host-Namen des *rc_cube* automatisch in ihrem Domain Name Server (*DNS*). In diesem Fall kann die Web GUI auch direkt über folgende *URL* aufgerufen werden: `http://<host-name>`, wobei der Platzhalter `<host-name>` gegen den Host-Namen des Geräts auszutauschen ist.

Für Linux und macOS funktioniert das ohne DNS über das Multicast-DNS-Protokoll (*mDNS*), das automatisch aktiviert wird, wenn `.local` zum Host-Namen hinzugefügt wird. So wird die URL einfach zu: `http://<host-name>.local`.

6.1.1.1 Zugriff auf die *rc_visard* Web GUI

Zur Fehlersuche oder Störungsbehebung können Nutzer auch direkt auf die Web GUI des mit dem *rc_cube* verbundenen *rc_visard* zugreifen. Diese ist unter dem Port 2342 des *rc_cube* und damit unter der *URL* `http://<host>:2342` erreichbar, wobei `<host>` durch die IP-Adresse bzw. den Host-Namen des *rc_cube* zu ersetzen ist, an den der *rc_visard* angeschlossen ist.

So kann zum Beispiel auf die Geräteinformationen und Log-Dateien des *rc_visard* zugegriffen werden.

Bemerkung: Für den Fall, dass ein Computer-Bildschirm an den *rc_cube* angeschlossen ist, zeigt dieser die Web GUI mit einem kleinen zusätzlichen Menü an, über welches auch die *rc_visard* Web GUI zu erreichen ist.

6.1.2 Kennenlernen der Web GUI

Die Überblicksseite der Web GUI enthält die wichtigsten Informationen über das Gerät und die Softwaremodule.

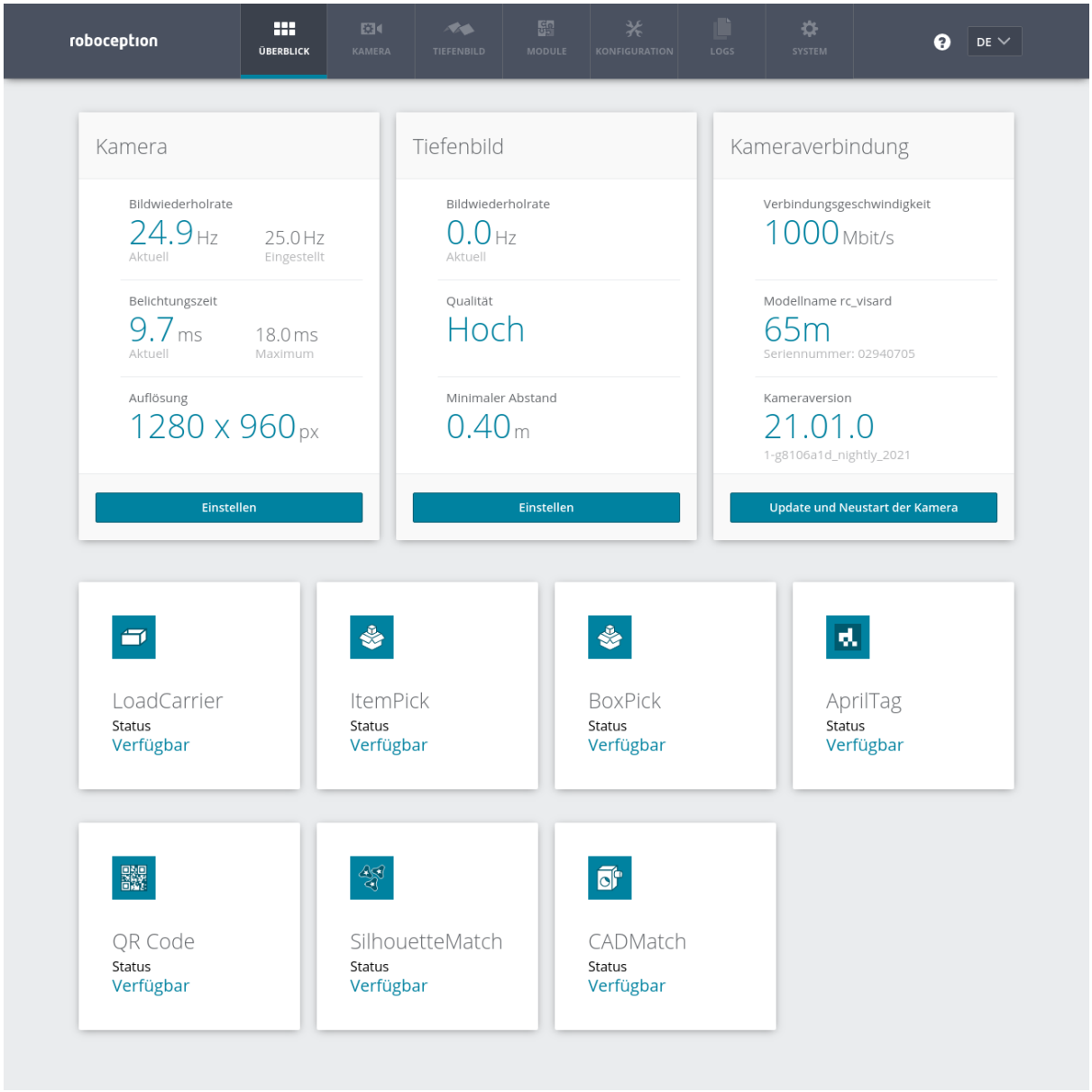


Abb. 6.1: Überblicksseite der Web GUI des *rc_cube*

Über Registerkarten im oberen Abschnitt der Seite kann auf die einzelnen Seiten der Web GUI des *rc_cube* 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 5.1.1.4).

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 5.1.2.5).

Module ermöglicht den Zugriff auf die Detektionsmodule des *rc_cube* (siehe *Detektionsmodule*, Abschnitt 5.2).

Konfiguration ermöglicht den Zugriff auf die Konfigurationsmodule des *rc_cube* (siehe *Konfigurationsmodule*, Abschnitt 5.3).

Logs ermöglicht den Zugriff auf die Logdateien des *rc_cube*. Die Logdateien werden in der Regel über-

prüft, wenn Fehler vermutet werden.

System ermöglicht dem Nutzer den Zugriff auf allgemeine Systemeinstellungen und Informationen zum Gerät, sowie die Möglichkeit, die Firmware oder Lizenzdatei zu aktualisieren.

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

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

6.1.3 Herunterladen von Stereo-Bildern

Die Web GUI bietet eine einfache Möglichkeit, einen Schnappschuss der aktuellen Szene als .tar.gz-Datei zu speichern. Dazu dient das Kamerasymbol unterhalb der Live-Streams auf der Seite *Kamera*. Dieser Schnappschuss beinhaltet:

- die rektifizierten linken und rechten Kamerabilder in voller Auflösung als .png-Dateien,
- eine Kameraparameter-Datei mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras.
- die aktuellen IMU-Messungen als imu.csv-Datei,
- eine nodes.json-Datei mit Informationen aller Module auf dem *rc_cube*,
- eine system_info.json-Datei mit Systeminformationen des *rc_cube*.

Die Dateinamen enthalten die Zeitstempel.

6.1.4 Herunterladen von Tiefenbildern und Punktwolken

Die Web GUI bietet eine einfache Möglichkeit, die Tiefendaten der aktuellen Szene als .tar.gz-Datei zu speichern. Dazu dient das Kamerasymbol unterhalb der Live-Streams auf der Seite *Tiefenbild*. Dieser Schnappschuss beinhaltet:

- die rektifizierten linken und rechten Kamerabilder in voller Auflösung als .png-Dateien,
- eine Parameterdatei für das linke Kamerabild mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras,
- die Disparitäts-, Fehler- und Konfidenzbilder in der Auflösung, die der aktuell eingestellten Qualität entspricht, als .png-Dateien,
- eine Parameterdatei zum Disparitätsbild mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras, sowie Informationen über die Disparitätswerte (ungültige Werte, Skalierung, Offset),
- eine Punktwolke in der aktuell eingestellten Auflösung (Qualität) as .ply-Datei,
- die aktuellen IMU-Messungen als imu.csv-Datei,
- eine nodes.json-Datei mit Informationen aller Module auf dem *rc_cube*,
- eine system_info.json-Datei mit Systeminformationen des *rc_cube*.

Die Dateinamen enthalten die Zeitstempel.

Bemerkung: Das Herunterladen der Tiefenbilder löst eine Bildaufnahme aus, in der gleichen Weise wie ein Klick auf den „Aufnehmen“-Button auf der *Tiefenbild*-Seite der Web GUI. Dies kann einen Einfluss auf laufende Anwendungen haben.

6.2 GigE Vision 2.0/GenICam-Schnittstelle

Gigabit Ethernet for Machine Vision (oder kurz „GigE Vision®“) ist ein industrieller Standard für Kameraschnittstellen, der auf UDP/IP basiert (siehe http://www.gigevision.com). Der *rc_cube* 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_cube* folgende Funktionen zur Verfügung:

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

Bemerkung: Der *rc_cube* unterstützt Jumbo-Frames mit einer Größe bis 9000 Byte. Für höchste Leistung wird empfohlen, die maximale Übertragungseinheit (MTU) des GigE-Vision-Clients auf 9000 zu stellen.

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

6.2.1 GigE Vision Ports

GigE Vision ist ein UDP basiertes Protokoll. Auf dem *rc_cube* sind die UDP Ports statisch und bekannt:

- UDP Port 3956: GigE Vision Control Protocol (GVCP). Zum Auffinden, steuern und konfigurieren des Geräts.
- UDP Port 50010: Stream channel source port. Nutzt das GigE Vision Stream Protocol (GVSP) zum transferieren der Bilder.

6.2.2 Wichtige Parameter der GenICam-Schnittstelle

Die folgende Liste enthält einen Überblick über relevante GenICam-Parameter des *rc_cube*, 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_cube* zudem eigene Parameter, die sich auf spezielle Eigenschaften der Module *Stereokamera* (Abschnitt 5.1.1) und *Stereo-Matching* (Abschnitt 5.1.2) beziehen.

6.2.3 Wichtige Standardparameter der GenICam-Schnittstelle

6.2.3.1 Kategorie: ImageFormatControl

`ComponentSelector`

- Typ: Aufzählung, mögliche Werte: `Intensity`, `IntensityCombined`, `Disparity`, `Confidence` oder `Error`
- Voreinstellung: -
- Beschreibung: Erlaubt dem Benutzer, einen der fünf Bild-Streams zur Konfiguration auszuwählen (siehe *Verfügbare Bild-Streams*, Abschnitt 6.2.6).

`ComponentIDValue` (schreibgeschützt)

- Typ: Integer
- Beschreibung: ID des vom `ComponentSelector` ausgewählten Bild-Streams.

`ComponentEnable`

- Typ: Boolean
- Voreinstellung: -
- Beschreibung: Ist der Parameter auf `true` gesetzt, aktiviert er den im `ComponentSelector` ausgewählten Bild-Stream. Anderenfalls deaktiviert er diesen Stream. Über `ComponentSelector` und `ComponentEnable` lassen sich einzelne Bild-Streams ein- und ausschalten.

`Width` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Bildbreite des vom `ComponentSelector` ausgewählten Bild-Streams in Pixeln.

`Height` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Bildhöhe des vom `ComponentSelector` ausgewählten Bild-Streams in Pixeln.

`WidthMax` (schreibgeschützt)

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

`HeightMax` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Maximale Höhe eines Bildes im Stream. Der Wert beträgt aufgrund der gestapelten Bilder der linken und rechten Kamera im `IntensityCombined`-Stream immer 1920 Pixel (siehe *Verfügbare Bild-Streams*, Abschnitt 6.2.6).

`PixelFormat`

- Typ: Aufzählung, mögliche Werte: `Mono8` oder `YCbCr411_8` (nur bei Farbkameras), `Coord3D_C16`, `Confidence8` und `Error8`
- Beschreibung: Pixelformat der selektierten Komponente. Die Aufzählung erlaubt nur aus Pixelformaten auszuwählen, die für die ausgewählte Komponente möglich sind. Bei einer Farbkamera kann man bei den Komponenten `Intensity` und `IntensityCombined` zwischen den Pixelformaten `Mono8` oder `YCbCr411_8` wählen.

6.2.3.2 Kategorie: AcquisitionControl

`AcquisitionFrameRate`

- Typ: Float, Wertebereich: 1–25 Hz
- Voreinstellung: 25 Hz
- Beschreibung: Bildwiederholrate der Kamera (*FPS*, Abschnitt 5.1.1.4).

`ExposureAuto`

- Typ: Aufzählung, mögliche Werte: `Continuous`, `Out1High`, `AdaptiveOut1` oder `Off`
- Voreinstellung: `Continuous`
- Beschreibung: Lässt sich für die manuelle Belichtung auf `Off` bzw. für die *automatische Belichtung* (Abschnitt 5.1.1.4) auf `Continuous`, `Out1High` oder `AdaptiveOut1` setzen. Der Wert `Continuous` entspricht dem Wert `Normal` für `exp_auto_mode` (*Modus Belichtungsautomatik*, Abschnitt 5.1.1.4) und `Out1High` und `AdaptiveOut1` den gleichnamigen Modi.

`ExposureTime`

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 5000 µs
- Beschreibung: Belichtungszeit der Kameras für den manuellen Belichtungsmodus, ausgedrückt in Mikrosekunden (*Belichtungszeit*, Abschnitt 5.1.1.4).

6.2.3.3 Kategorie: AnalogControl

`GainSelector` (schreibgeschützt)

- Typ: Aufzählung, Wert: ist immer `All`
- Voreinstellung: `All`
- Beschreibung: Der *rc_cube* unterstützt aktuell nur einen globalen Verstärkungsfaktor.

`Gain`

- Typ: Float, Wertebereich: 0–18 dB
- Voreinstellung: 0 dB
- Beschreibung: Verstärkungsfaktor der Kameras für den manuellen Belichtungsmodus, ausgedrückt in Dezibel (*Verstärkungsfaktor*, Abschnitt 5.1.1.4).

`BalanceWhiteAuto` (nur für Farbkameras)

- Typ: Aufzählung, mögliche Werte: `Continuous` oder `Off`
- Voreinstellung: `Continuous`
- Beschreibung: Lässt sich für den manuellen Weißabgleich auf `Off` bzw. für den automatischen Weißabgleich auf `Continuous` setzen. Dieser Parameter ist nur für Farbkameras verfügbar (*Weißabgleich*, Abschnitt 5.1.1.4).

`BalanceRatioSelector` (nur für Farbkameras)

- Typ: Aufzählung, mögliche Werte: `Red` oder `Blue`
- Voreinstellung: `Red`
- Beschreibung: Auswahl des Verhältnisses welches mit `BalanceRatio` einstellbar ist. `Red` bedeutet Verhältnis von Rot zu Grün, und `Blue` bedeutet Verhältnis von Blau zu Grün. Diese Einstellung ist nur für Farbkameras verfügbar.

`BalanceRatio` (nur für Farbkameras)

- Typ: Float, Wertebereich: 0.125 – 8
- Voreinstellung: 1.2 wenn `Red` und 2.4 wenn `Blue` im `BalanceRatioSelector` eingestellt sind
- Beschreibung: Gewichtung vom roten oder blauen zum grünen Farbkanal. Diese Einstellung ist nur für Farbkameras verfügbar (*wb_ratio*, Abschnitt 5.1.1.4).

6.2.3.4 Kategorie: DigitalIOControl

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

`LineSelector`

- Typ: Aufzählung, mögliche Werte: `Out1`, `Out2`, `In1` oder `In2`
- Voreinstellung: `Out1`
- Beschreibung: Wählt den Ein- oder Ausgang, um den aktuellen Zustand abzufragen oder die Betriebsart zu setzen.

`LineStatus` (schreibgeschützt)

- Typ: Boolean
- Beschreibung: Aktueller Zustand des mit `LineSelector` ausgewählten Ein- oder Ausgangs.

`LineStatusAll` (schreibgeschützt)

- Typ: Integer
- Beschreibung: Aktueller Zustand der Ein- und Ausgänge, welche in den unteren vier Bits angegeben werden.

Tab. 6.1: Bedeutung der Bits des `LineStatusAll` Parameters.

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

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

- Typ: Aufzählung, mögliche Werte: `ExposureActive`, `ExposureAlternateActive`, `Low` oder `High`
- Voreinstellung: `Low`
- Beschreibung: Betriebszustand des mit `LineSelector` gewählten Ausgangs, wie im Abschnitt zum IOControl Modul beschrieben (*out1_mode und out2_mode*, Abschnitt 5.3.4.1). Siehe auch den Parameter `AcquisitionAlternateFilter` zum Filtern von Bildern im Betriebszustand `ExposureAlternateActive`.

6.2.3.5 Kategorie: TransportLayerControl / PtpControl

`PtpEnable`

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

6.2.3.6 Kategorie: Scan3dControl

`Scan3dDistanceUnit` (schreibgeschützt)

- Typ: Aufzählung, Wert: ist immer `Pixel`
- Beschreibung: Einheit für die Disparitätsmessungen, ist immer `Pixel`.

`Scan3dOutputMode` (schreibgeschützt)

- Typ: Aufzählung, Wert: ist immer `DisparityC`

- Beschreibung: Modus für die Tiefenmessungen, ist immer `DisparityC`.

`Scan3dFocalLength` (schreibgeschützt)

- Typ: Float
- Beschreibung: Brennweite des mit `ComponentSelector` ausgewählten Bild-Streams in Pixeln. Im Fall der Komponenten `Disparity`, `Confidence` und `Error` hängt der Wert auch von der Auflösung ab, die implizit über `DepthQuality` eingestellt wurde.

`Scan3dBaseline` (schreibgeschützt)

- Typ: Float
- Beschreibung: Basisabstand der Stereokamera in Metern.

`Scan3dPrinciplePointU` (schreibgeschützt)

- Typ: Float
- Beschreibung: Horizontale Position des Bildhauptpunktes des mit `ComponentSelector` ausgewählten Bild-Streams. Im Fall der Komponenten `Disparity`, `Confidence` und `Error` hängt der Wert auch von der Auflösung ab, die implizit über `DepthQuality` eingestellt wurde.

`Scan3dPrinciplePointV` (schreibgeschützt)

- Typ: Float
- Beschreibung: Vertikale Position des Bildhauptpunktes des mit `ComponentSelector` ausgewählten Bild-Streams. Im Fall der Komponenten `Disparity`, `Confidence` und `Error` hängt der Wert auch von der Auflösung ab, die implizit über `DepthQuality` eingestellt wurde.

`Scan3dCoordinateScale` (schreibgeschützt)

- Typ: Float
- Beschreibung: Der Skalierungsfaktor, der mit den Disparitätswerten im Disparitätsbild-Stream zu multiplizieren ist, um die tatsächlichen Disparitätswerte zu erhalten. Der Wert beträgt immer 0,0625.

`Scan3dCoordinateOffset` (schreibgeschützt)

- Typ: Float
- Beschreibung: Der Versatz, der zu den Disparitätswerten im Disparitätsbild-Stream addiert werden muss, um die tatsächlichen Disparitätswerte zu erhalten. Für den *rc_cube* beträgt der Wert immer 0 und kann daher ignoriert werden.

`Scan3dInvalidDataFlag` (schreibgeschützt)

- Typ: Boolean
- Beschreibung: Ist immer `true`, was bedeutet, dass ungültige Daten im Disparitätsbild mit einem spezifischen Wert markiert werden, der durch den Parameter `Scan3dInvalidDataValue` definiert wird.

`Scan3dInvalidDataValue` (schreibgeschützt)

- Typ: Float
- Beschreibung: Ist der Wert, der für ungültige Disparität steht. Der Wert ist immer 0, was bedeutet, dass Disparitätswerte von 0 immer ungültigen Messungen entsprechen. Um zwischen ungültigen Disparitätsmessungen und Messungen, bei denen die Disparität aufgrund der unendlich weit entfernten Objekte 0 beträgt, unterscheiden zu können, wird der Disparitätswert für den letztgenannten Fall auf 0,0625 gesetzt. Dies entspricht noch immer einer Objektentfernung von mehreren hundert Metern.

6.2.3.7 Kategorie: ChunkDataControl

`ChunkModeActive`

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

6.2.4 Besondere Parameter der GenICam-Schnittstelle des *rc_cube*

6.2.4.1 Kategorie: AcquisitionControl

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

- Typ: Aufzählung, mögliche Werte: `Off`, `OnlyHigh` oder `OnlyLow`
- Voreinstellung: `Off`
- Beschreibung: Falls dieser Parameter auf `OnlyHigh` (oder entsprechend `OnlyLow`) und die `LineSource` für mindestens einen Ausgang auf `ExposureAlternateActive` eingestellt wird, dann werden nur die Kamerabilder übertragen, welche aufgenommen wurden, während der konfigurierte Ausgang an war, d.h. ein potentiell angeschlossener Projektor war an (oder bei `OnlyLow` entsprechend aus). Dieser Parameter ist ein einfaches Mittel um nur Bilder ohne ein projiziertes Muster zu bekommen. Der minimale Zeitunterschied zwischen einem Kamera- und einem Disparitätsbild ist in diesem Fall etwa 40 ms (siehe *IOControl*, Abschnitt 5.3.4.1).

`AcquisitionMultiPartMode`

- Typ: Aufzählung, mögliche Werte: `SingleComponent` oder `SynchronizedComponents`
- Voreinstellung: `SingleComponent`
- Beschreibung: Nur wirksam im MultiPart-Modus. Ist dieser Parameter auf `SingleComponent` gesetzt, werden die Bilder jeweils sofort als einzelne Komponente pro Frame/Puffer geschickt, sobald sie verfügbar sind. Dies entspricht dem Verhalten von Clients, die MultiPart nicht unterstützen. Ist dieser Parameter auf `SynchronizedComponents` gesetzt, werden die aktivierten Komponenten auf dem *rc_cube* zeitlich synchronisiert und in einem gemeinsamen Frame/Puffer versendet – allerdings nur, falls alle für diesen Zeitpunkt verfügbar sind.

`ExposureTimeAutoMax`

- Typ: Float, Wertebereich: 66–18000 µs
- Voreinstellung: 18000 µs
- Beschreibung: Maximale Belichtungszeit im automatischen Belichtungsmodus (*Maximale Belichtungszeit*, Abschnitt 5.1.1.4).

`ExposureRegionOffsetX`

- Typ: Integer, Wertebereich 0-1280
- Voreinstellung: 0
- Beschreibung: Horizontaler Offset des *Bereichs für die Belichtungszeitregelung* (Abschnitt 5.1.1.4) in Pixeln.

`ExposureRegionOffsetY`

- Typ: Integer, Wertebereich: 0-960
- Voreinstellung: 0
- Beschreibung: Vertikaler Offset des *Bereichs für die Belichtungszeitregelung* (Abschnitt 5.1.1.4) in Pixeln.

`ExposureRegionWidth`

- Typ: Integer, Wertebereich 0-1280
- Voreinstellung: 0
- Beschreibung: Breite des *Bereichs für die Belichtungszeitregelung* (Abschnitt 5.1.1.4) in Pixeln.

`ExposureRegionHeight`

- Typ: Integer, Wertebereich: 0-960
- Voreinstellung: 0
- Beschreibung: Höhe des *Bereichs für die Belichtungszeitregelung* (Abschnitt 5.1.1.4) in Pixeln.

`RcExposureAutoAverageMax`

- Typ: Float, Wertebereich 0-1
- Voreinstellung: 0.75
- Beschreibung: Maximale Helligkeit der *automatischen Belichtungszeitsteuerung* (Abschnitt 5.1.1.4) als Wert zwischen 0 (dunkel) und 1 (hell).

`RcExposureAutoAverageMin`

- Typ: Float, Wertebereich 0-1
- Voreinstellung: 0.25
- Beschreibung: Minimale Helligkeit der *automatischen Belichtungszeitsteuerung* (Abschnitt 5.1.1.4) als Wert zwischen 0 (dunkel) und 1 (hell).

6.2.4.2 Kategorie: Scan3dControl

`FocalLengthFactor` (schreibgeschützt)

- Typ: Float
- Beschreibung: Brennweite skaliert auf eine Bildbreite von einem Pixel. Um die Brennweite für ein bestimmtes Bild in Pixeln zu ermitteln, muss dieser Wert mit der Breite des empfangenen Bilds multipliziert werden. Siehe auch den Parameter `Scan3dFocalLength`.

`Baseline` (schreibgeschützt)

- Typ: Float
- Besschreibung: Dieser Parameter ist überholt. Der Parameter `Scan3dBaseline` sollte stattdessen benutzt werden.

6.2.4.3 Kategorie: DepthControl

`DepthAcquisitionMode`

- Typ: Aufzählung, mögliche Werte: `SingleFrame`, `SingleFrameOut1` oder `Continuous`
- Voreinstellung: `Continuous`
- Beschreibung: Im Modus `SingleFrame` wird das Stereo-Matching mit jedem Aufruf von `DepthAcquisitionTrigger` durchgeführt. Der Modus `SingleFrameOut1` kann zum Kontrollieren eines externen Projektors genutzt werden. Dabei wird bei jedem Trigger `Out1` auf `ExposureAlternateActive` und nach dem Empfangen der Stereobilder auf `Low` gesetzt. Dies ist jedoch nur möglich, wenn die IOControl-Lizenz verfügbar ist. Im Modus `Continuous` wird das Stereo-Matching kontinuierlich durchgeführt.

`DepthAcquisitionTrigger`

- type: Command

- Beschreibung: Dieses Kommando triggert das Stereo-Matching auf den nächsten verfügbaren Stereobildern, falls `DepthAcquisitionMode` auf `SingleFrame` oder `SingleFrameOut1` eingestellt ist.

`DepthQuality`

- Typ: Aufzählung, mögliche Werte: `Low`, `Medium`, `High` oder `Full` **(nur mit StereoPlus-Lizenz)**
- Voreinstellung: `High`
- Beschreibung: Qualität der Disparitätsbilder. Eine geringere `DepthQuality` führt zu Disparitätsbildern mit einer geringeren Auflösung (*Qualität*, Abschnitt 5.1.2.5).

`DepthDoubleShot`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: `True` zum Verbessern des Stereo-Matching-Resultats bei Szenen mit Projektor. Löcher im Tiefenbild werden gefüllt mit Tiefendaten aus dem Stereo Matching des Bildpaars ohne Projektormuster (*Double-Shot*, Abschnitt 5.1.2.5).

`DepthStaticScene`

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: `True` zum Mitteln über acht aufeinanderfolgende Kamerabilder zur Verbesserung des Stereo-Matching-Resultats (*Statisch*, Abschnitt 5.1.2.5).

`DepthSmooth` (schreibgeschützt ohne StereoPlus-Lizenz)

- Typ: Boolean
- Voreinstellung: `false`
- Beschreibung: `True` um Disparitätswerte zu glätten (*Glättung*, Abschnitt 5.1.2.5).

`DepthFill`

- Typ: Integer, Wertebereich: 0–4 Pixel
- Voreinstellung: 3 Pixel
- Beschreibung: Wert in Pixeln für *Füllen* (Abschnitt 5.1.2.5).

`DepthSeg`

- Typ: Integer, Wertebereich: 0–4000 Pixel
- Voreinstellung: 200 Pixel
- Beschreibung: Wert in Pixeln für *Segmentierung* (Abschnitt 5.1.2.5).

`DepthMinConf`

- Typ: Float, Wertebereich: 0.0–1.0
- Voreinstellung: 0.0
- Beschreibung: Wert für die *Minimale Konfidenz*-Filterung (Abschnitt 5.1.2.5).

`DepthMinDepth`

- Typ: Float, Wertebereich: 0.1–100.0 m
- Voreinstellung: 0.1 m
- Beschreibung: Wert in Metern für die *Minimale Abstands*-Filterung (Abschnitt 5.1.2.5).

`DepthMaxDepth`

- Typ: Float, Wertebereich: 0.1–100.0 m

- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Abstands*-Filterung (Abschnitt 5.1.2.5).

`DepthMaxDepthErr`

- Typ: Float, Wertebereich: 0.01–100.0 m
- Voreinstellung: 100.0 m
- Beschreibung: Wert in Metern für die *Maximale Fehler*-Filterung (Abschnitt 5.1.2.5).

6.2.5 Chunk-Daten

Der *rc_cube* unterstützt Chunk-Parameter, die mit jedem Bild mitgeschickt werden. Chunk-Parameter haben alle den Präfix `Chunk`. Ihre Bedeutung entspricht den gleichlautenden Nicht-Chunk-Parametern. Sie passen jedoch immer zu dem zugehörigen Bild. Zum Beispiel hängt `Scan3dFocalLength` von `ComponentSelector` und `DepthQuality` ab, da die Bildauflösung von beiden Parametern abhängt. Der Parameter `ChunkScan3dFocalLength`, welcher zu einem Bild geliefert wird, passt hingegen zu der Auflösung dieses Bildes.

Nützliche Chunk-Parameter:

- `ChunkComponentSelector` selektiert, für welche Komponente Chunk-Daten aus dem MultiPart-Puffer gelesen werden.
- `ChunkComponentID` und `ChunkComponentIDValue` dienen der eindeutigen Zuordnung des Bildes zu seiner Komponente (z.B. Kamerabild oder Disparitätsbild), ohne dies vom Bildformat oder der Bildgröße ableiten zu müssen.
- `ChunkLineStatusAll` bietet den Status der Ein- und Ausgänge zum Zeitpunkt der Bildaufnahme. Siehe `LineStatusAll` für eine Beschreibung der Bits.
- `ChunkScan3d...` sind nützlich zur 3D-Rekonstruktion wie im Abschnitt *Umwandlung von Bild-Streams* (Abschnitt 6.2.7) beschrieben.
- `ChunkPartIndex` gibt den Index des Bild-Parts im MultiPart-Block für die ausgewählte Komponente (`ChunkComponentSelector`) zurück.
- `ChunkRcOut1Reduction` gibt den Anteil der Bildhelligkeit an, um den Bilder mit GPIO Ausgang 1 (Out1) LOW dunkler sind als Bilder mit Out1 HIGH. Ein Wert von beispielsweise 0.2 bedeutet, dass die Bilder mit GPIO Out1 LOW 20% weniger Helligkeit haben als Bilder mit GPIO Out1 HIGH. Dieser Wert ist nur verfügbar, wenn `exp_auto_mode` der Stereokamera auf `AdaptiveOut1` oder `Out1High` gesetzt ist (*auto exposure mode*, Abschnitt 5.1.1.4).

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

6.2.6 Verfügbare Bild-Streams

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

Name der Komponente	PixelFormat	Breite × Höhe	Beschreibung
`Intensity`	`Mono8` (monochrome Kameras) `YCbCr411_8` (Farbkameras)	1280 × 960	Rektifiziertes Bild der linken Kamera
`IntensityCombined`	`Mono8` (monochrome Kameras) `YCbCr411_8` (Farbkameras)	1280 × 1920	Rektifiziertes Bild der linken Kamera, gestapelt auf das rektifizierte Bild der rechten Kamera
`Disparity`	`Coord3D_C16`	1280 × 1920 640 × 480 320 × 240 214 × 160	Disparitätsbild in gewünschter Auflösung, d.h. `DepthQuality` in `Full`, `High`, `Medium` oder `Low`
`Confidence`	`Confidence8`	wie `Disparity`	Konfidenzbild
`Error`	`Error8` (Sonderformat: 0x81080001)	wie `Disparity`	Fehlerbild

Jedes Bild wird mit einem Zeitstempel und dem in der Tabelle angegebenen *PixelFormat* ausgegeben. Dieses *PixelFormat* sollte verwendet werden, um zwischen den verschiedenen Bildtypen zu unterscheiden. Bilder, die den gleichen Aufnahmezeitpunkt haben, können durch Vergleich der GenICam-Zeitstempel einander zugeordnet werden.

6.2.7 Umwandlung von Bild-Streams

Das Disparitätsbild enthält vorzeichenlose 16-Bit-Ganzzahlwerte. Diese Werte müssen mit dem im GenICam-Parameter *Scan3dCoordinateScale* angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätswerte d in Pixeln zu ermitteln. Um die 3D-Objektkoordinaten aus den Disparitätswerten berechnen zu können, werden die Brennweite, der Basisabstand und der Bildhauptpunkt benötigt. Diese Parameter werden als GenICam-Parameter *Scan3dFocalLength*, *Scan3dBaseline*, *Scan3dPrincipalPointU* und *Scan3dPrincipalPointV* zur Verfügung gestellt. Die Brennweite und der Bildhauptpunkt hängen von der Bildauflösung der mit dem *ComponentSelector* selektierten Komponente ab. Sind diese Werte bekannt, können die Pixel-Koordinaten und die Disparitätswerte mithilfe der im Abschnitt *Berechnung von Tiefenbildern und Punktwolken* (Abschnitt 5.1.2.2) angegebenen Gleichungen in 3D-Objektkoordination im Kamera-Koordinatensystem umgerechnet werden.

Unter der Annahme, dass es sich bei d_{ik} um den 16-Bit-Disparitätswert in der Spalte i und Zeile k eines Disparitätsbildes handelt, ist der Fließkomma-Disparitätswert in Pixeln d_{ik} gegeben durch

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

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

$$P_x = (i + 0.5 - \text{Scan3dPrincipalPointU}) \frac{\text{Scan3dBaseline}}{d_{ik}},$$
$$P_y = (k + 0.5 - \text{Scan3dPrincipalPointV}) \frac{\text{Scan3dBaseline}}{d_{ik}},$$
$$P_z = \text{Scan3dFocalLength} \frac{\text{Scan3dBaseline}}{d_{ik}}.$$

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

Das Fehlerbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Der Fehler e_{ik} muss mit dem im GenICam-Parameter *Scan3dCoordinateScale* angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätsfehlerwerte d_{eps} in Pixeln zu ermitteln. Der Beschreibung in *Konfidenz- und Fehlerbilder* (Abschnitt 5.1.2.3) zufolge lässt sich der Tiefenfehler z_{eps} (in Metern) mit den GenICam-Parametern wie folgt berechnen:

$$d_{ik} = d16_{ik} \cdot \text{Scan3dCoordinateScale},$$
$$z_{eps} = \frac{e_{ik} \cdot \text{Scan3dCoordinateScale} \cdot \text{Scan3dFocalLength} \cdot \text{Scan3dBaseline}}{(d_{ik})^2}.$$

Bemerkung: Chunk-Daten sollten nach Möglichkeit mit dem Parameter *ChunkModeActive* angeschaltet und die zum Bild zugehörigen Parameter *ChunkScan3dCoordinateScale*, *ChunkScan3dFocalLength*, *ChunkScan3dBaseline*, *ChunkScan3dPrincipalPointU* und *ChunkScan3dPrincipalPointV* genutzt werden, denn deren Werte passen zu der Auflösung des zugehörigen Bildes.

Für nähere Informationen zu Disparitäts-, Fehler- und Konfidenzbildern siehe *Stereo-Matching* (Abschnitt 5.1.2).

6.3 REST-API-Schnittstelle

Neben der *GenICam-Schnittstelle* (Abschnitt 6.2) bietet der *rc_cube* eine umfassende RESTful-Web-Schnittstelle (REST-API), auf die jeder HTTP-Client und jede HTTP-Bibliothek zugreifen kann. Während die meisten Parameter, Services und Funktionen auch über die benutzerfreundliche *Web GUI* (Abschnitt 6.1) zugänglich sind, dient die REST-API eher als Maschine-Maschine-Schnittstelle für folgende programmgesteuerte Aufgaben:

- Setzen und Abrufen der Laufzeitparameter der Softwaremodule, z.B. der Stereokamera oder von Bildverarbeitungsmodulen,
- Aufrufen von Services, z.B. zum Starten und Stoppen einzelner Softwaremodule, oder zum Nutzen spezieller Funktionen, wie der Hand-Auge-Kalibrierung,
- Abruf des aktuellen Systemstatus und des Status einzelner Softwaremodule, sowie
- Aktualisierung der Firmware des *rc_cube* oder seiner Lizenz.

Bemerkung: In der REST-API des *rc_cube* bezeichnet der Begriff *Node* ein Softwaremodul, das gewisse algorithmische Funktionen bündelt und eine ganzheitliche Benutzeroberfläche (Parameter, Services, aktueller Status) besitzt. Beispiele für solche Module sind das Stereo-Matching-Modul oder das Modul zur Hand-Auge-Kalibrierung.

6.3.1 Allgemeine Struktur der Programmierschnittstelle (API)

Der allgemeine **Einstiegspunkt** zur Programmierschnittstelle (API) des *rc_cube* ist `http://<host>/api/` wobei `<host>` entweder die IP-Adresse des Geräts ist oder sein dem jeweiligen DHCP-Server bekannter

Host-Name (siehe *Netzwerkkonfiguration*, Abschnitt 3.5). Greift der Benutzer über einen Webbrowser auf diese Adresse zu, kann er die Programmierschnittstelle während der Laufzeit mithilfe der *Swagger UI* (Abschnitt 6.3.4) erkunden und testen.

Für die eigentlichen HTTP-Anfragen wird dem Einstiegspunkt der Programmierschnittstelle die **aktuelle Version der Schnittstelle als Postfix angehangen**, d.h. `http://<host>/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 6.3.2) 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://<host>/api/v1/nodes/rc_stereomatching'
```

- Abruf einiger Parameterwerte eines Moduls über eine Abfragezeichenfolge

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

- Setzen eines Modulparameters als JSON-formatierter Text im Body der Anfrage

```
curl -X PUT --header 'Content-Type: application/json' -d '[{"name": "mindepth", "value": 0.
↪1}]' 'http://<host>/api/v1/nodes/rc_stereomatching/parameters'
```

Zur Beantwortung solcher Anfragen greift die Programmierschnittstelle des *rc_cube* auf übliche Rückgabecodes zurück:

Statuscode	Beschreibung
`200 OK`	Die Anfrage war erfolgreich. Die Ressource wird im JSON-Format zurückgegeben.
`400 Bad Request`	Ein für die API-Anfrage benötigtes Attribut oder Argument fehlt oder ist ungültig.
`404 Not Found`	Auf eine Ressource konnte nicht zugegriffen werden. Möglicherweise kann die ID einer Ressource nicht gefunden werden.
`403 Forbidden`	Der Zugriff ist (vorübergehend) verboten. Möglicherweise sind einige Parameter gesperrt, während eine GigE Vision-Anwendung verbunden ist.
`429 Too many requests`	Die Übertragungsrate ist aufgrund einer zu hohen Anfragefrequenz begrenzt.

Der folgende Eintrag zeigt eine Musterantwort auf eine erfolgreiche Anfrage, mit der Informationen zum `minconf`-Parameter des `rc_stereomatching`-Moduls angefordert werden:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 157

{
    "name": "minconf",
    "min": 0,
    "default": 0,
    "max": 1,
    "value": 0,
    "type": "float64",
    "description": "Minimum confidence"
}
```

Bemerkung: Das tatsächliche Verhalten, die zulässigen Anfragen und die speziellen Rückgabecodes hängen in hohem Maße von der gewählten Ressource, vom Kontext und von der Aktion ab. Siehe die *verfügbaren Ressourcen* (Abschnitt 6.3.2) des *rc_cube* und einzelnen Parameter und Services jedes *Softwaremoduls* (Abschnitt 5).

6.3.2 Verfügbare Ressourcen und Anfragen

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

- **`/nodes`** Zugriff auf die *Softwaremodule* (Abschnitt 5) des *rc_cube* mit ihren jeweiligen Laufzeitzuständen, Parametern und verfügbaren Services.
- **`/logs`** Zugriff auf die im *rc_cube* hinterlegten Logdateien.
- **`/system`** Zugriff auf den Systemzustand, Netzwerkkonfiguration und Verwaltung der Lizenzen sowie der Firmware-Updates.

6.3.2.1 Module, Parameter und Services

Die *Softwaremodule* (Abschnitt 5) des *rc_cube* 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://<host>/api/v1/nodes
```

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

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

Bemerkung: Die zurückgegebenen **Statuswerte** sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Bemerkung: Statuswerte werden nur gemeldet, wenn sich das jeweilige Modul im Zustand `running` befindet.

Parameter: Die meisten Module stellen Parameter über die REST-API des *rc_cube* zur Verfügung, damit ihr Laufzeitverhalten an den Anwendungskontext oder die Anforderungen angepasst werden kann. Die REST-API ermöglicht es, den Wert eines Parameters zu setzen und abzufragen. Darüber hinaus stellt sie weitere Angaben, wie z.B. den jeweiligen Standardwert und zulässige Minimal- bzw. Maximalwerte von Parametern, zur Verfügung.

Die `rc_stereomatching`-Parameter lassen sich beispielsweise wie folgt abrufen:

```
curl -X GET http://<host>/api/v1/nodes/rc_stereomatching/parameters
```

Der `quality`-Parameter dieses Moduls könnte wie folgt auf den Wert `Full` gesetzt werden:

```
curl -X PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?quality=Full
```

oder äquivalent

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

Bemerkung: Laufzeitparameter sind modulspezifisch und werden in dem jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Bemerkung: Die meisten Parameter, die die Module über die REST-API anbieten, lassen sich auch über die benutzerfreundliche *Web GUI* (Abschnitt 6.1) des *rc_cube* erkunden und austesten.

Bemerkung: Einige der Parameter, die über die REST-API des *rc_cube* bereitgestellt werden, sind auch über die *GigE Vision 2.0/GenICam-Schnittstelle* (Abschnitt 6.2) zugänglich. Die Einstellung dieser Parameter über die REST-API und die Web GUI ist verboten, solange ein GenICam-Client verbunden ist.

Zudem bietet jedes Modul, das Laufzeitparameter bereitstellt, auch 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 des Moduls zur Hand-Auge-Kalibrierung* (Abschnitt 5.3.1.5) lassen sich beispielsweise wie folgt aufrufen:

```
curl -X GET http://<host>/api/v1/nodes/rc_hand_eye_calibration/services
```

Um einen Service eines Moduls aufzurufen, wird eine `PUT`-Anfrage mit servicespezifischen Argumenten für die jeweilige Ressource gestellt (siehe das `"args"`-Feld des *Service-Datenmodells*, Abschnitt 6.3.3). Beispielsweise lässt sich folgendermaßen eine Bildaufnahme mit dem Stereo-Matching-Modul auslösen:

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

Bemerkung: Die Services und zugehörigen Argumente sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

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

`GET /nodes`
Abruf einer Liste aller verfügbaren Module.

Musteranfrage

```
GET /api/v1/nodes HTTP/1.1
```

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"
  },
  {
    "name": "rc_hand_eye_calibration",
    "parameters": [
      "grid_width",
      "grid_height",
      "robot_mounted"
    ],
    "services": [
      "save_parameters",
      "reset_defaults",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "set_pose",
      "reset",
      "save",
      "calibrate",
      "get_calibration"
    ],
    "status": "stale"
  },
  {
    "name": "rc_stereomatching",
    "parameters": [
      "quality",
      "seg",
      "fill",
      "minconf",
      "mindepth",
      "maxdepth",
      "maxdeptherr"
    ],
    "services": [
      "save_parameters",
      "reset_defaults"
    ],
    "status": "running"
  }
]
```


Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 6.3.3)

GET /nodes/{node}

Abruf von Informationen zu einem einzelnen Modul.

Musteranfrage

```
GET /api/v1/nodes/<node> HTTP/1.1
```

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"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  ],
  "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 6.3.3)

GET /nodes/{node}/parameters

Abruf von Parametern eines Moduls.

Musteranfrage

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

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": true,
    "description": "Switching between auto and manual exposure",
    "max": true,
    "min": false,
    "name": "exp_auto",
    "type": "bool",
    "value": true
  },
  {
    "default": 0.007,
    "description": "Maximum exposure time in s if exp_auto is true",
    "max": 0.018,
    "min": 6.6e-05,
    "name": "exp_max",
    "type": "float64",
    "value": 0.007
  }
]
```

Parameter

- **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 6.3.3)

PUT /nodes/{node}/parameters

Aktualisierung mehrerer Parameter.

Musteranfrage

```
PUT /api/v1/nodes/<node>/parameters HTTP/1.1
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": true,
    "description": "Switching between auto and manual exposure",
    "max": true,
    "min": false,
    "name": "exp_auto",
    "type": "bool",
    "value": false
  },
  {
    "default": 0.005,
    "description": "Manual exposure time in s if exp_auto is false",
    "max": 0.018,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "min": 6.6e-05,
    "name": "exp_value",
    "type": "float64",
    "value": 0.005
  }
]
```


Parameter

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

JSON-Objekt-Array zur Anfrage

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

Anfrage-Header

- Accept – application/json

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter-Array)*
- 400 Bad Request – Ungültiger Parameterwert
- 403 Forbidden – Aktualisierung des Parameters verboten, z.B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist oder keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 6.3.3)
- *ParameterNameValue* (Abschnitt 6.3.3)

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

Abruf eines bestimmten Parameters eines Moduls.

Musteranfrage

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

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 6.3.3)

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

Aktualisierung eines bestimmten Parameters eines Moduls.

Musteranfrage

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

{
  "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** (*ParameterValue*) – zu aktualisierender Parameter als JSON-Objekt *(obligatorisch)*

Anfrage-Header

- Accept – application/json

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter)*
- 400 Bad Request – Ungültiger Parameterwert

- 403 Forbidden – Aktualisierung des Parameters verboten, z.B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist oder keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Parameter nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 6.3.3)
- *ParameterValue* (Abschnitt 6.3.3)

`GET /nodes/{node}/services`

Abruf von Beschreibungen aller von einem Modul angebotenen Services.

Musteranfrage

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

Musterantwort

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

[
  {
    "args": {},
    "description": "Restarts the module.",
    "name": "restart",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  },
  {
    "args": {},
    "description": "Starts the module.",
    "name": "start",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  },
  {
    "args": {},
    "description": "Stops the module.",
    "name": "stop",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  }
]
```

Parameter

- **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 6.3.3)

`GET /nodes/{node}/services/{service}`
Abruf der Beschreibung eines modulspezifischen Services.

Musteranfrage

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

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: Service)*
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.3.3)

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
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 – Serviceaufruf erledigt *(Rückgabe: Service)*
- 403 Forbidden – Service-Aufruf verboten, z.B. weil keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 6.3.3)

GET /nodes/{node}/status

Abruf des Status eines Moduls.

Musteranfrage

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

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 6.3.3)

6.3.2.2 System und Logs

Die folgenden Ressourcen und Anfragen sind für die System-Level-API des *rc_cube* 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_cube*.

GET /logs
Abruf einer Liste aller verfügbaren Logdateien.

Musteranfrage

```
GET /api/v1/logs HTTP/1.1
```

Musterantwort

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
[
  {
    "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
  }
]
```


Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *LogInfo* (Abschnitt 6.3.3)

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
```

Musterantwort


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

{
  "date": 1581609251.8168414,
  "log": [
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 registered with control access.",
      "timestamp": 1581609249.61
    },
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 deregistered.",
      "timestamp": 1581609249.739
    },
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 registered with control access.",
      "timestamp": 1581609250.94
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    },
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 deregistered.",
      "timestamp": 1581609251.819
    }
  ],
  "name": "gev.log",
  "size": 42112
}
```


Parameter

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

Anfrageparameter

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

Antwort-Headers

- Content-Type – text/plain application/json

Statuscodes

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

Referenzierte Datenmodelle

- *Log* (Abschnitt 6.3.3)

GET /system

Abruf von Systeminformationen zum Gerät.

Musteranfrage

```
GET /api/v1/system HTTP/1.1
```

Musterantwort


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

{
  "firmware": {
    "active_image": {
      "image_version": "rc_cube_v1.1.0"
    },
    "fallback_booted": true,
    "inactive_image": {
      "image_version": "rc_cube_v1.0.0"
    },
    "next_boot_image": "active_image"
  },
  "hostname": "rc-cube-02873515",
  "link_speed": 1000,
  "mac": "00:14:2D:2B:D8:AB",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  "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 6.3.3)

GET /system/backup

Abruf eines Backups der Einstellungen.

Musteranfrage

```
GET /api/v1/system/backup?nodes=<nodes>&load_carriers=<load_carriers>&regions_of_interest=
↪<regions_of_interest>&grippers=<grippers> HTTP/1.1
```

Anfrageparameter

- **nodes** (*boolean*) – Backup der Moduleinstellungen, d.h. Parameter und bevorzugte TCP-Orientierung (Standardwert: True) *(optional)*
- **load_carriers** (*boolean*) – Backup der Load Carrier (Standardwert: True) *(optional)*
- **regions_of_interest** (*boolean*) – Backup der Regions of Interest (Standardwert: True) *(optional)*
- **grippers** (*boolean*) – Backup der Greifer (Standardwert: True) *(optional)*

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

POST /system/backup

Backup einspielen.

Musteranfrage

```
POST /api/v1/system/backup HTTP/1.1
Accept: application/json

{}
```

Musterantwort

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

{
  "return_code": {
    "message": "backup restored",
    "value": 0
  },
  "warnings": []
}
```

Request JSON Object

- **backup** (*object*) – Backup-Daten als json-Objekt *(erforderlich)*

Anfrage-Header

- Accept – application/json

Antwort-Headers

- Content-Type – application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/license

Abruf von Informationen zu den auf dem Gerät installierten Lizenzen.

Musteranfrage

```
GET /api/v1/system/license HTTP/1.1
```

Musterantwort

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

{
  "components": {
    "hand_eye_calibration": true,
    "rectification": true,
    "stereo": true
  },
  "valid": true
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *LicenseInfo* (Abschnitt 6.3.3)

`POST /system/license`
Aktualisierung der auf dem Gerät installierten Lizenz mithilfe einer Lizenzdatei.

Musteranfrage

```
POST /api/v1/system/license HTTP/1.1
Accept: multipart/form-data
```

Formularparameter

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

Anfrage-Header

- Accept – Multipart/Formulardaten

Statuscodes

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

`GET /system/network`
Abruf der aktuellen Netzwerk Konfiguration.

Musteranfrage

```
GET /api/v1/system/network HTTP/1.1
```

Musterantwort

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

{
  "current_method": "DHCP",
  "default_gateway": "10.0.3.254",
  "ip_address": "10.0.1.41",
  "settings": {
    "dhcp_enabled": true,
    "persistent_default_gateway": "",
    "persistent_ip_address": "192.168.0.10",
    "persistent_ip_enabled": false,
    "persistent_subnet_mask": "255.255.255.0"
  },
  "subnet_mask": "255.255.252.0"
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *NetworkInfo* (Abschnitt 6.3.3)

`GET /system/network/settings`
Abruf der aktuellen Netzwerkeinstellungen.

Musteranfrage

```
GET /api/v1/system/network/settings HTTP/1.1
```

Musterantwort

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

{
  "dhcp_enabled": true,
  "persistent_default_gateway": "",
  "persistent_ip_address": "192.168.0.10",
  "persistent_ip_enabled": false,
  "persistent_subnet_mask": "255.255.255.0"
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *NetworkSettings* (Abschnitt 6.3.3)

PUT /system/network/settings

Setzen der aktuellen Netzwerkeinstellungen.

Musteranfrage

```
PUT /api/v1/system/network/settings HTTP/1.1
Accept: application/json

{}
```

Musterantwort

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

{
  "dhcp_enabled": true,
  "persistent_default_gateway": "",
  "persistent_ip_address": "192.168.0.10",
  "persistent_ip_enabled": false,
  "persistent_subnet_mask": "255.255.255.0"
}
```

Request JSON Object

- **settings** (*NetworkSettings*) – Anzuwendende Netzwerkeinstellungen *(obligatorisch)*

Anfrage-Header

- Accept – application/json

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

- 400 Bad Request – ungültige/fehlende Argumente
- 403 Forbidden – Das Ändern der Netzwerkeinstellungen ist nicht erlaubt, da eine laufende GigE Vision-Applikation diese sperrt.

Referenzierte Datenmodelle

- *NetworkSettings* (Abschnitt 6.3.3)

PUT /system/reboot
Neustart des Geräts.

Musteranfrage

```
PUT /api/v1/system/reboot HTTP/1.1
```

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/rollback
Abruf von Informationen zu Firmware/System-Images, die aktuell auf dem Gerät aktiv oder inaktiv sind.

Musteranfrage

```
GET /api/v1/system/rollback HTTP/1.1
```

Musterantwort

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

{
  "active_image": {
    "image_version": "rc_cube_v1.1.0"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "rc_cube_v1.0.0"
  },
  "next_boot_image": "active_image"
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 6.3.3)

PUT /system/rollback
Rollback auf vorherige Firmware-Version (inaktives System-Image).

Musteranfrage

```
PUT /api/v1/system/rollback HTTP/1.1
```

Statuscodes

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

`GET /system/update`
Abruf von Informationen zu Firmware/System-Images, die aktuell auf dem Gerät aktiv oder inaktiv sind.

Musteranfrage

```
GET /api/v1/system/update HTTP/1.1
```

Musterantwort

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

{
  "active_image": {
    "image_version": "rc_cube_v1.1.0"
  },
  "fallback_booted": false,
  "inactive_image": {
    "image_version": "rc_cube_v1.0.0"
  },
  "next_boot_image": "active_image"
}
```

Antwort-Headers

- Content-Type – application/json

Statuscodes

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

Referenzierte Datenmodelle

- *FirmwareInfo* (Abschnitt 6.3.3)

`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
Accept: multipart/form-data
```

Formularparameter

- **`file`** – Mender-Artefakt-Datei *(obligatorisch)*

Anfrage-Header

- Accept – Multipart/Formulardaten

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – Client-Fehler, z.B. kein gültiges Mender-Artefakt

6.3.3 Datentyp-Definitionen

Die REST-API definiert folgende Datenmodelle, die verwendet werden, um auf die *verfügbaren Ressourcen* (Abschnitt 6.3.2) 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.

LicenseComponentConstraint: Einschränkungen für die Modul-Version.

Ein Objekt des Typs LicenseComponentConstraint besitzt folgende Eigenschaften:

- **max_version** (string) - optionale höchste unterstützte Version (exclusive)
- **min_version** (string) - optionale minimale unterstützte Version (inclusive)

Musterobjekt

```
{
  "max_version": "string",
  "min_version": "string"
}
```

LicenseComponentConstraint-Objekte sind in *LicenseConstraints* enthalten.

LicenseComponents: Liste der Lizenzstatus-Angaben der einzelnen Softwaremodule: Der zugehörige Statusindikator ist auf TRUE gesetzt, wenn das entsprechende Modul mit einer installierten Softwarelizenz entsperrt ist.

Ein Objekt des Typs LicenseComponents besitzt folgende Eigenschaften:

- **hand_eye_calibration** (boolean): Modul zur Hand-Auge-Kalibrierung.
- **rectification** (boolean): Modul zur Bildrektifizierung.
- **stereo** (boolean): Stereo-Matching-Modul.

Musterobjekt

```
{
  "hand_eye_calibration": false,
  "rectification": false,
  "stereo": false
}
```

LicenseComponents-Objekte sind in *LicenseInfo* enthalten.

LicenseConstraints: Versionseinschränkungen für Module.

Ein Objekt des Typs LicenseConstraints besitzt folgende Eigenschaften:

- **image_version** (*LicenseComponentConstraint*) - siehe Beschreibung von *LicenseComponentConstraint*

Musterobjekt

```
{
  "image_version": {
    "max_version": "string",
    "min_version": "string"
  }
}
```

LicenseConstraints-Objekte sind in *LicenseInfo* enthalten.

LicenseInfo: Informationen zur aktuell auf dem Gerät angewandten Softwarelizenz.

Ein Objekt des Typs LicenseInfo besitzt folgende Eigenschaften:

- **components** (*LicenseComponents*): siehe Beschreibung von *LicenseComponents*.
- **components_constraints** (*LicenseConstraints*) - siehe Beschreibung von *LicenseConstraints*
- **valid** (boolean): Angabe, ob eine Lizenz gültig ist oder nicht.

Musterobjekt

```
{
  "components": {
    "hand_eye_calibration": false,
    "rectification": false,
    "stereo": false
  },
  "components_constraints": {
    "image_version": {
      "max_version": "string",
      "min_version": "string"
    }
  },
  "valid": false
}
```

LicenseInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /system/license*

Log: Inhalt einer bestimmten Logdatei im JSON-Format.

Ein Objekt des Typs Log besitzt folgende Eigenschaften:

- **date** (float): UNIX-Uhrzeit, zu der das Log zuletzt geändert wurde.
- **log** (*LogEntry*-Array): die eigentlichen Logeinträge.
- **name** (string): Name der Logdatei.
- **size** (Integer): Größe der Logdatei in Bytes.

Musterobjekt


```
{
  "date": 0,
  "log": [
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    },
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    }
  ],
  "name": "string",
  "size": 0
}
```


Log-Objekte werden in folgenden Anfragen verwendet:

- *GET /logs/{log}*

LogEntry: Darstellung eines einzelnen Logeintrags in einer Logdatei.

Ein Objekt des Typs LogEntry besitzt folgende Eigenschaften:

- **component** (string): Name des Moduls, das diesen Eintrag angelegt hat.
- **level** (string): Logstufe (mögliche Werte: `DEBUG`, `INFO`, `WARN`, `ERROR` oder `FATAL`)
- **message** (string): eigentliche Lognachricht.
- **timestamp** (float): UNIX-Uhrzeit des Logeintrags.

Musterobjekt


```
{
  "component": "string",
  "level": "string",
  "message": "string",
  "timestamp": 0
}
```


LogEntry-Objekte sind in *Log* enthalten.

LogInfo: Informationen zu einer bestimmten Logdatei.

Ein Objekt des Typs LogInfo besitzt folgende Eigenschaften:

- **date** (float): UNIX-Uhrzeit, zu der das Log zuletzt geändert wurde.

- **name** (string): Name der Logdatei.
- **size** (Integer): Größe der Logdatei in Bytes.

Musterobjekt


```
{
  "date": 0,
  "name": "string",
  "size": 0
}
```


LogInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /logs*

NetworkInfo: Aktuelle Netzwerk Konfiguration.

Ein Objekt des Typs NetworkInfo besitzt folgende Eigenschaften:

- **current_method** (string) - Methode mit der die aktuellen Einstellungen gesetzt wurden (eine von `INIT`, `LinkLocal`, `DHCP`, `PersistentIP`, `TemporaryIP`)
- **default_gateway** (string) - aktueller Default Gateway
- **ip_address** (string) - aktuelle IP-Adresse
- **settings** (*NetworkSettings*) - siehe Beschreibung von *NetworkSettings*
- **subnet_mask** (string) - aktuelle Subnetzmaske

Musterobjekt


```
{
  "current_method": "string",
  "default_gateway": "string",
  "ip_address": "string",
  "settings": {
    "dhcp_enabled": false,
    "persistent_default_gateway": "string",
    "persistent_ip_address": "string",
    "persistent_ip_enabled": false,
    "persistent_subnet_mask": "string"
  },
  "subnet_mask": "string"
}
```


NetworkInfo-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/network*

NetworkSettings: Aktuelle Netzwerk Einstellungen.

Ein Objekt des Typs NetworkSettings besitzt folgende Eigenschaften:

- **dhcp_enabled** (boolean) - DHCP eingeschaltet
- **persistent_default_gateway** (string) - Persistenter Default Gateway
- **persistent_ip_address** (string) - Persistente IP-Adresse
- **persistent_ip_enabled** (boolean) - Persistente IP aktiviert
- **persistent_subnet_mask** (string) - Persistente Subnetzmaske

Musterobjekt

```
{
  "dhcp_enabled": false,
  "persistent_default_gateway": "string",
  "persistent_ip_address": "string",
  "persistent_ip_enabled": false,
  "persistent_subnet_mask": "string"
}
```


NetworkSettings-Objekte sind in *NetworkInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/network/settings*
- *PUT /system/network/settings*

NodeInfo: Beschreibung eines auf dem Gerät 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}*

ParameterNameValue: Parametername und -wert. Der Typ des Parameterwerts (Felder ‚value‘ und ‚min‘, ‚max‘, ‚default‘) ist durch das Feld ‚type‘ angegeben und kann einer der eingebauten primitiven Datentypen sein.

Ein Objekt des Typs ParameterNameValue besitzt folgende Eigenschaften:

- **name** (string): Name des Parameters.

- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt

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

ParameterNameValue-Objekte werden in folgenden Anfragen verwendet:

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

ParameterValue: Parameterwert. Der Typ des Parameterwerts (Felder ‚value' und ‚min', ‚max', ‚default') ist durch das Feld ‚type' angegeben und kann einer der eingebauten primitiven Datentypen sein.

Ein Objekt des Typs ParameterValue besitzt folgende Eigenschaften:

- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt

```
{
  "value": {}
}
```

ParameterValue-Objekte werden in folgenden Anfragen verwendet:

- *PUT /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.

SysInfo: Systeminformationen über das Gerät.

Ein Objekt des Typs SysInfo besitzt folgende Eigenschaften:

- **firmware** (*FirmwareInfo*): siehe Beschreibung von *FirmwareInfo*.
- **hostname** (string): Host-Name.
- **link_speed** (Integer): Ethernet-Verbindungsgeschwindigkeit in Mb/Sekunde.
- **mac** (string): MAC-Adresse.
- **network** (*NetworkInfo*): siehe Beschreibung von *NetworkInfo*
- **ntp_status** (*NtpStatus*): siehe Beschreibung von *NtpStatus*.
- **ptp_status** (*PtpStatus*): siehe Beschreibung von *PtpStatus*.
- **ready** (boolean): Das System ist vollständig hochgefahren und betriebsbereit.
- **serial** (string): Seriennummer des Geräts.
- **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,
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  "mac": "string",
  "network": {
    "current_method": "string",
    "default_gateway": "string",
    "ip_address": "string",
    "settings": {
      "dhcp_enabled": false,
      "persistent_default_gateway": "string",
      "persistent_ip_address": "string",
      "persistent_ip_enabled": false,
      "persistent_subnet_mask": "string"
    },
    "subnet_mask": "string"
  },
  "ntp_status": {
    "accuracy": "string",
    "synchronized": false
  },
  "ptp_status": {
    "master_ip": "string",
    "offset": 0,
    "offset_dev": 0,
    "offset_mean": 0,
    "state": "string"
  },
  "ready": false,
  "serial": "string",
  "time": 0,
  "uptime": 0
}
```


SysInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /system*

Template: Template für die Erkennung

Ein Objekt des Typs Template besitzt folgende Eigenschaften:

- **id** (string): Eindeutiger Name des Templates

Musterobjekt


```
{
  "id": "string"
}
```


Template-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes/rc_cadmatch/templates*
- *GET /nodes/rc_cadmatch/templates/{id}*
- *PUT /nodes/rc_cadmatch/templates/{id}*
- *GET /nodes/rc_silhouettematch/templates*
- *GET /nodes/rc_silhouettematch/templates/{id}*
- *PUT /nodes/rc_silhouettematch/templates/{id}*

6.3.4 Swagger UI

Die Swagger UI des *rc_cube* ermöglicht es Entwicklern, die REST-API – beispielsweise zu Entwicklungs- und Testzwecken – leicht darzustellen und zu verwenden. Der Zugriff auf `http://<host>/api/` oder auf `http://<host>/api/swagger` (der erste Link leitet automatisch auf den zweiten Link weiter) öffnet eine Vorschau der allgemeinen API-Struktur des *rc_cube*, einschließlich aller *verfügbaren Ressourcen und Anfragen* (Abschnitt 6.3.2). Auf dieser vereinfachten Benutzeroberfläche lassen sich alle Funktionen erkunden und austesten.

Bemerkung: Der Benutzer muss bedenken, dass die Swagger UI des *rc_cube*, auch wenn sie zur Erprobung der REST-API bestimmt ist, eine voll funktionstüchtige Schnittstelle ist. Das bedeutet, dass alle ausgelösten Anfragen tatsächlich bearbeitet werden und den Zustand und/oder das Verhalten des Geräts beeinflussen. Dies gilt insbesondere für Anfragen des Typs `PUT`, `POST` und `DELETE`.

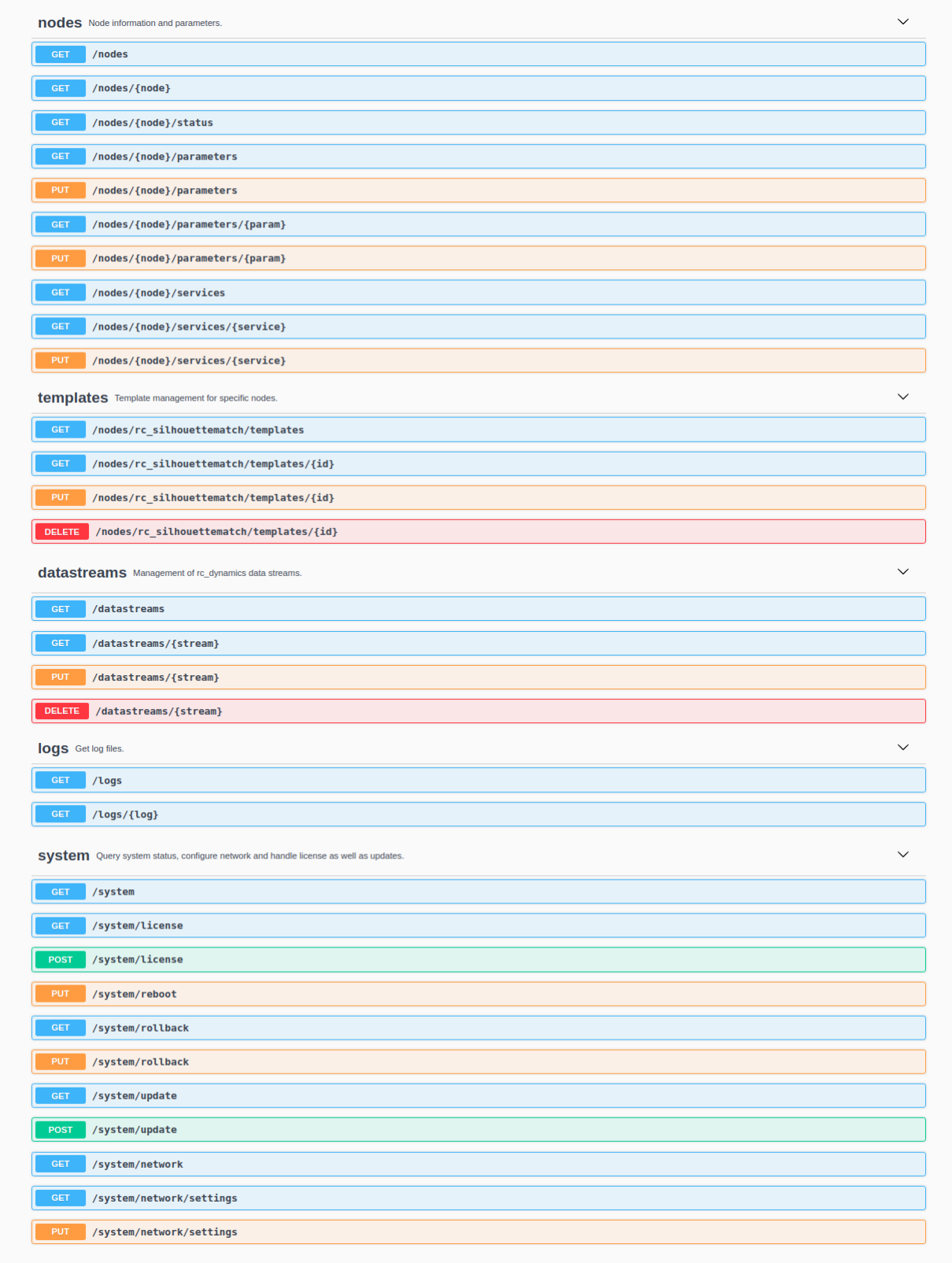


Abb. 6.2: Startansicht der Swagger UI des *rc_cube*, bei der die Ressourcen und Anfragen in `nodes`, `templates`, `datastreams`, `logs` und `system` gruppiert sind.

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 die Schaltfläche *Try it out!* betätigt, der erfor-

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

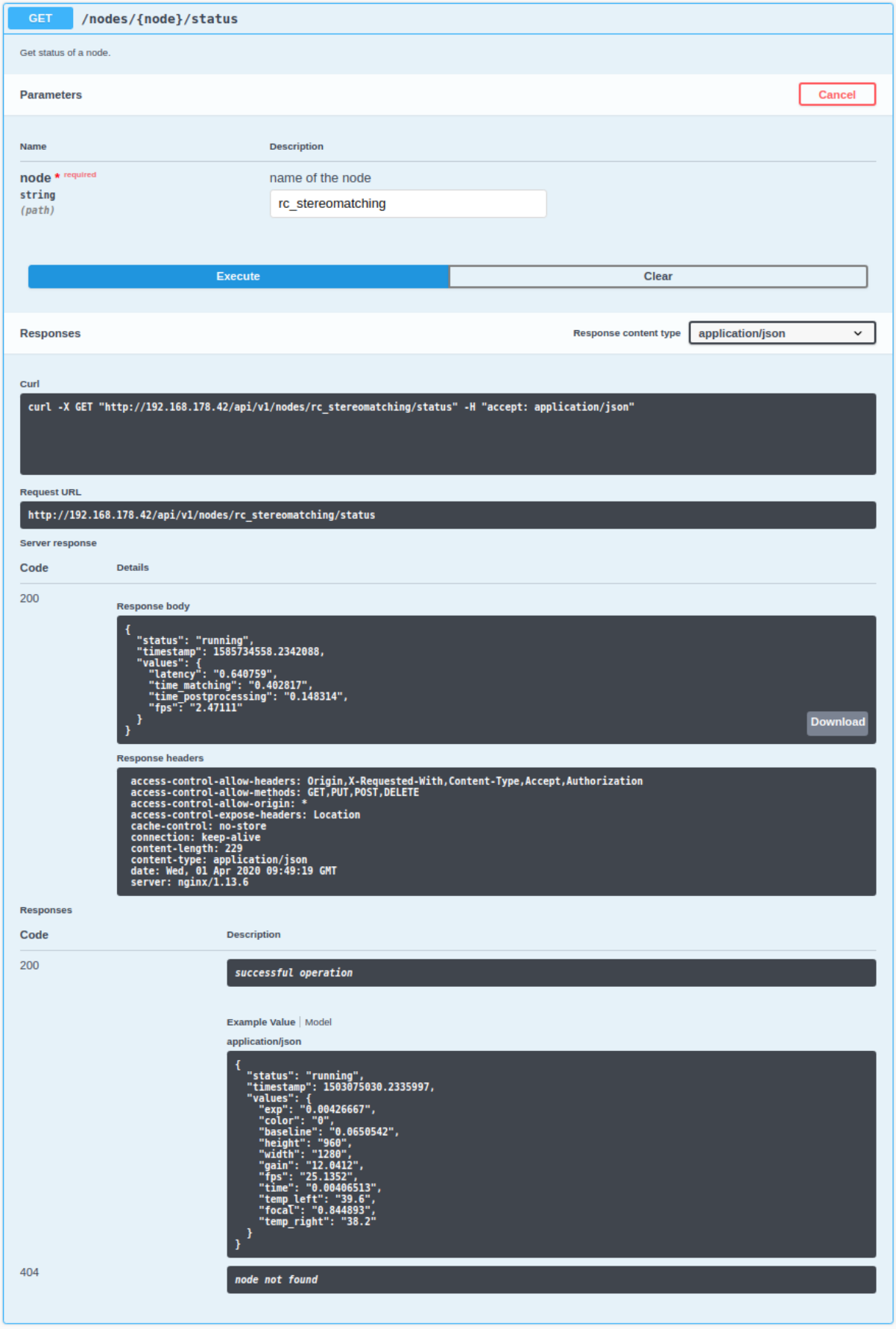


Abb. 6.3: Ergebnis nach Abfrage des Status des `rc_stereomatching`-Moduls

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

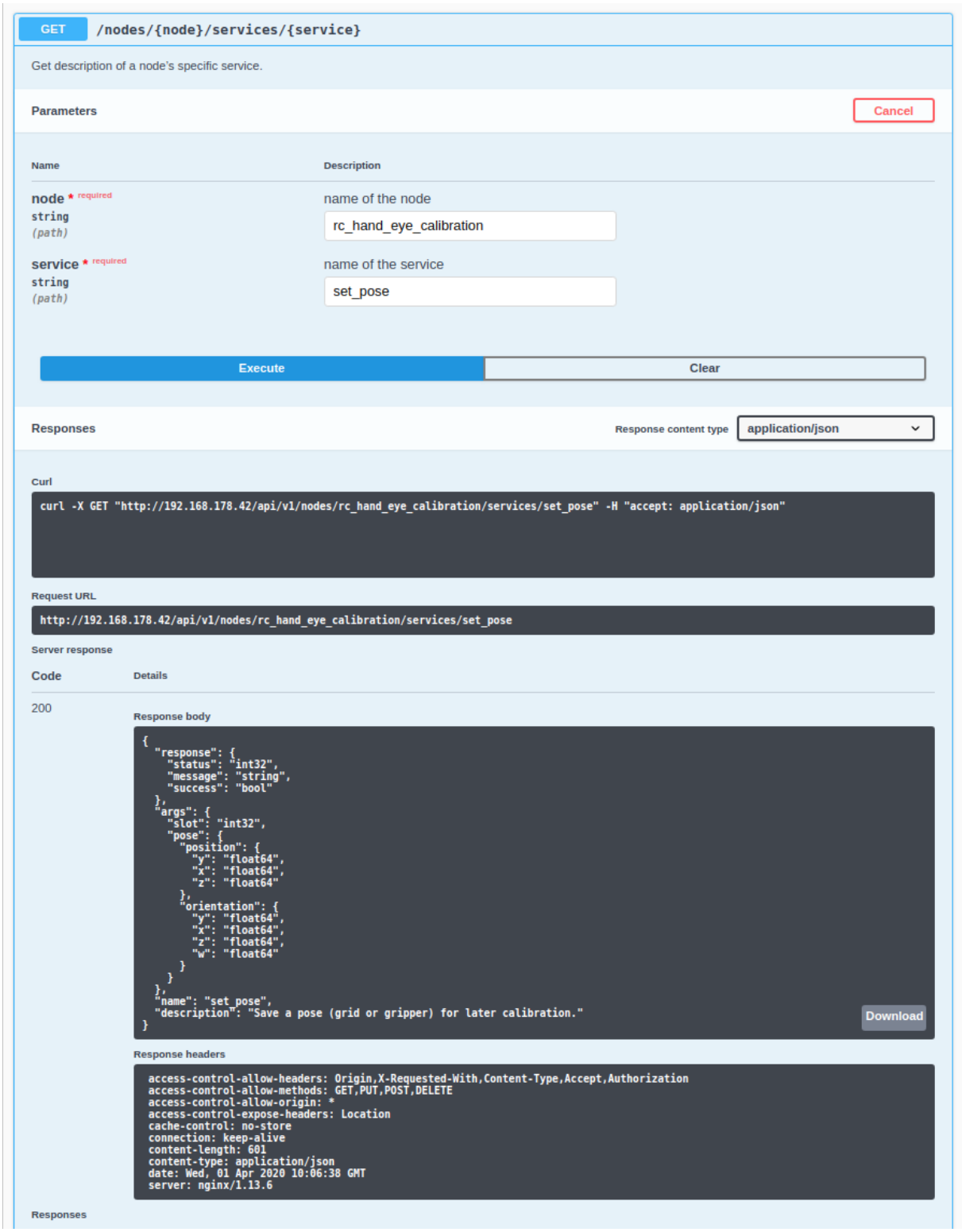


Abb. 6.4: Ergebnis der GET-Anfrage zum `set_pose`-Service zeigt die für diesen Service benötigten Argumente

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

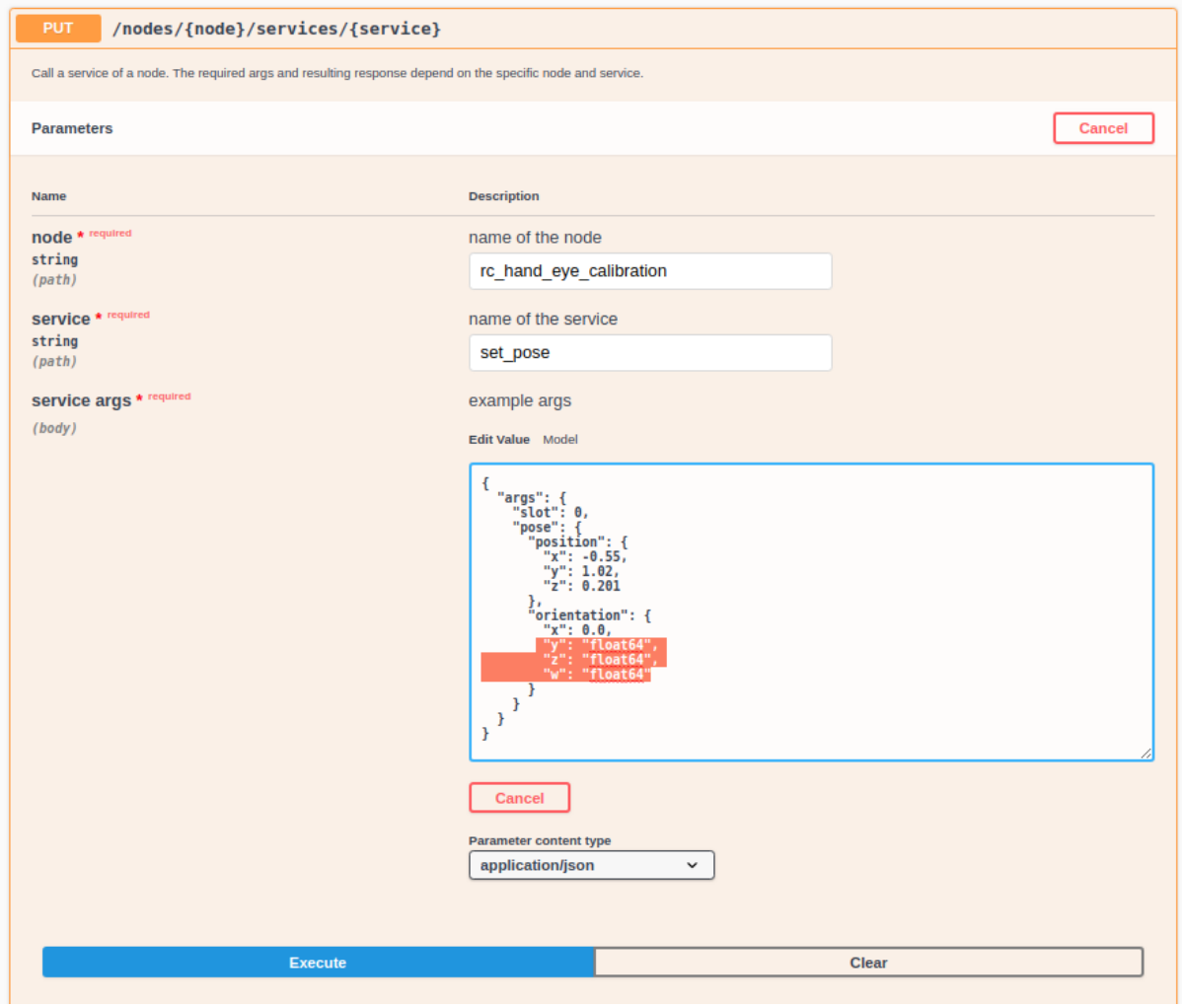


Abb. 6.5: Ausfüllen der Argumente des `set_pose`-Services

6.4 KUKA Ethernet KRL Schnittstelle

Der *rc_cube* stellt ein Ethernet KRL Interface (EKI-Bridge) zur Verfügung, welches eine Kommunikation von KUKA KRL via KUKA.EthernetKRL XML mit dem *rc_cube* erlaubt.

Bemerkung: Dieses Modul ist optional und benötigt eine gesonderte EKIBridge-*Lizenz* (Abschnitt 7.5).

Bemerkung: Das KUKA.EthernetKRL add-on Software-Paket Version 2.2 oder neuer muss auf der Robotersteuerung aktiviert sein, um dieses Modul zu benutzen.

Die EKI-Bridge kann benutzt werden, um programmatisch

- Serviceanfragen auszuführen, z.B. um individuelle Module zu starten und stoppen, oder um angebotene Services wie z.B. die Hand-Auge-Kalibrierung oder Berechnung von Greifposen zu nutzen,
- Laufzeitparameter abzufragen und zu ändern, z.B. der Kamera oder Disparitätsberechnung.

6.4.1 Konfiguration der Ethernet-Verbindung

Die EKI-Bridge hört auf Port 7000 auf EKI-XML-Nachrichten und übersetzt diese transparent zur *rc_cube REST-API* (Abschnitt 6.3). Die empfangenen EKI-Nachrichten werden in JSON umgewandelt und an die *rc_cube* REST-API weitergeleitet. Die Antwort der REST-API wird anschließend zurück in EKI-XML gewandelt.

Die EKI-Bridge erlaubt den Zugriff auf Laufzeitparameter und Services aller Module, die in *Softwaremodule* (Abschnitt 5) beschrieben sind.

Die Ethernet-Verbindung zum *rc_cube* wird auf der Robotersteuerung mit XML-Dateien konfiguriert.

Die Ethernet-Verbindung zum *rc_cube* wird auf der Robotersteuerung mit XML-Dateien konfiguriert. Die EKI-XML-Konfigurationsdateien aller Module auf dem *rc_cube* können hier heruntergeladen werden:

https://doc.rc-visard.com/latest/de/eki.html#eki-xml-configuration-files

Für jedes Softwaremodul, das Laufzeitparameter anbietet, gibt es eine XML-Konfigurationsdatei, um die Parameter abzufragen und zu setzen. Diese sind nach dem Schema `<node_name>-parameters.xml` benannt. Für jeden Service eines Softwaremoduls gibt eine eigene XML-Konfigurationsdatei. Diese ist nach dem Schema `<node_name>-<service_name>.xml` benannt.

Die XML-Konfigurationsdateien sind bereits vorausgefüllt. Lediglich die IP des *rc_cube* muss vom Benutzer ergänzt werden.

Diese Konfigurationsdateien müssen im Verzeichnis `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL` auf der Robotersteuerung abgelegt werden. Sie werden gelesen, sobald eine Verbindung initialisiert wird.

Um z.B. eine Ethernet-Verbindung mit dem Ziel aufzubauen, um die `rc_stereomatching`-Parameter zu konfigurieren, ist der folgende KRL-Code notwendig.

```
DECL EKI_Status RET
RET = EKI_INIT("rc_stereomatching-parameters")
RET = EKI_Open("rc_stereomatching-parameters")

; ----------- Desired operation -----------

RET = EKI_Close("rc_stereomatching-parameters")
```

Bemerkung: Die EKI-Bridge terminiert automatisch die Verbindung zum Client, wenn eine empfangene XML-Nachricht ungültig ist.

6.4.2 Allgemeine XML-Struktur

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

Das Wurzelelement enthält immer die folgenden Elemente.

- `<node>`: Dieses enthält ein Unterelement, über das die EKI-Bridge das Ziel-Softwaremodul identifiziert. Der Modulname ist bereits in der XML-Konfigurationsdatei vorausgefüllt.
- `<end_of_request>`: „End-of-Request“ Flag, das das Ende der Anfrage markiert und diese auslöst.

Die generische XML-Struktur sieht wie folgt aus.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Für den Datenempfang nutzt die EKI-Bridge <res> als Wurzelelement (kurz für „Response“). Das Wurzelelement enthält immer ein <return_code> Unterelement.

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Bemerkung: Standardmäßig ist in den Konfigurationsdateien 998 als Flag angegeben, über welches KRL benachrichtigt wird, sobald eine Antwortnachricht empfangen wurde. Falls dieser Wert bereits in Benutzung ist, sollte dieser in der entsprechenden Konfigurationsdatei geändert werden.

6.4.2.1 Rückgabecode

Das <return_code>-Element enthält die Attribute `value` und `message`.

Wie für alle anderen Softwaremodule gibt eine erfolgreiche Anfrage ein `res/return_code/@value` mit dem Wert 0 zurück. Negative Werte geben an, dass die Anfrage fehlgeschlagen ist. Die Fehlermeldung ist in `res/return_code/@message` enthalten. Positive Werte geben an, dass die Anfrage erfolgreich war, aber weitere Informationen in `res/return_code/@message` enthalten sind.

Die folgenden Rückgabecodes können von der EKI-Bridge zurückgegeben werden:

Tab. 6.2: Rückgabecodes der EKI-Bridge

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

Bemerkung: Die EKI-Bridge liefert auch Rückgabecodes spezifisch zu den individuellen Softwaremodulen zurück. Diese sind im jeweiligen *Softwaremodul* (Abschnitt 5) dokumentiert.

Bemerkung: Aufgrund von Limitierungen in KRL ist die maximale Länge eines Strings, der von der EKI-Bridge zurückgegeben wird, auf 512 Zeichen begrenzt. Alle längeren Strings werden gekürzt.

6.4.3 Services

Das XML-Schema für die Services der Softwaremodule wird aus den Argumenten und der Antwort in *JavaScript Object Notation (JSON)* generiert, wie in *Softwaremodule* (Abschnitt 5) beschrieben. Diese Umwandlung ist bis auf die unten beschriebenen Regeln transparent.

Konvertierung von Posen:

Eine Pose ist ein JSON-Objekt, das die Schlüssel `position` und `orientation` enthält.

```
{
  "pose": {
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Dieses JSON-Objekt wird zu einem KRL `FRAME` in der XML-Nachricht konvertiert.

```
<pose X="..." Y="..." Z="..." A="..." B="..." C="..."></pose>
```

Positionen werden von Metern in Millimetern umgerechnet und Orientierungen von Quaternionen in das KUKA-ABC-Format (in Grad).

Bemerkung: Es werden in der EKI-Bridge keine anderen Größenumrechnungen vorgenommen. Alle Abmessungen und 3D-Koordinaten, die nicht zu einer Pose gehören, werden in Metern erwartet und zurückgegeben.

Arrays:

Arrays enthalten die Unterelemente `<le>` (kurz für „List Element“). Als Beispiel wird das JSON-Objekt

```
{
  "rectangles": [
    {
      "x": "float64",
      "y": "float64"
    }
  ]
}
```

in das folgende XML-Fragment konvertiert

```
<rectangles>
  <le>
    <x>...</x>
    <y>...</y>
  </le>
</rectangles>
```

XML-Attribute:

Alle JSON-Schlüssel, deren Wert ein primitiver Datentyp ist und die nicht zu einem Array gehören, werden in XML-Attributen gespeichert. Als Beispiel wird das JSON-Objekt

```
{
  "item": {
    "uuid": "string",
    "confidence": "float64",
    "rectangle": {
      "x": "float64",
      "y": "float64"
    }
  }
}
```

in das folgende XML-Fragment konvertiert

```
<item uuid="..." confidence="...">
  <rectangle x="..." y="...">
  </rectangle>
</item>
```

6.4.3.1 Anfrage-XML-Struktur

Das <SEND>-Element in der XML-Konfigurationsdatei für einen generischen Service folgt der folgenden Spezifikation:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/service/<service_name>" Type="STRING"/>
    <ELEMENT Tag="req/args/<argX>" Type="<argX_type>"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Das <service>-Element hat ein XML-Unterelement, über das die EKI-Bridge den angefragten Service identifiziert. Es ist bereits vorausgefüllt in der Konfigurationsdatei enthalten.

Das <args> Element beinhaltet die Service-Argumente. Diese können jeweils mit der KRL-Instruktion EKI_Set<Type> gesetzt werden.

Beispielsweise sieht das <SEND>-Element des rc_itempick get_load_carriers Services (siehe *ItemPick und BoxPick*, Abschnitt 5.2.3) wie folgt aus.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/rc_itempick" Type="STRING"/>
    <ELEMENT Tag="req/service/get_load_carriers" Type="STRING"/>
    <ELEMENT Tag="req/args/load_carrier_ids/le" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Das <end_of_request>-Element erlaubt es, Anfragen mit Arrays zu übermitteln. Um ein Array zu senden, wird die Anfrage in so viele Nachrichten wie Array-Elemente aufgeteilt. Die letzte Nachricht beinhaltet alle XML-Tags inklusive dem <end_of_request>-Flag, während alle anderen Nachrichten jeweils nur ein Array-Element enthalten.

Um z.B. zwei Load-Carrier-Modelle mit dem get_load_carriers Service vom rc_itempick abzufragen, muss der Nutzer zwei XML-Nachrichten senden. Die erste XML-Nachricht lautet:

```
<req>
  <args>
    <load_carrier_ids>
      <le>load_carrier1</le>
    </load_carrier_ids>
  </args>
</req>
```

Diese Nachricht kann über KRL mit dem EKI_Send Kommando gesendet werden, indem das Listenelement als Pfad angegeben wird.

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_itempick-get_load_carriers", "req/args/load_carrier_ids/le",
↪"load_carrier1")
RET = EKI_Send("rc_itempick-get_load_carriers", "req/args/load_carrier_ids/le")
```

Die zweite Nachricht beinhaltet alle XML-Tags und löst die Anfrage beim `rc_itempick` Softwaremodul aus.

```
<req>
  <node>
    <rc_itempick></rc_itempick>
  </node>
  <service>
    <get_load_carriers></get_load_carriers>
  </service>
  <args>
    <load_carrier_ids>
      <le>load_carrier2</le>
    </load_carrier_ids>
  </args>
  <end_of_request></end_of_request>
</req>
```

Diese Nachricht kann über KRL gesendet werden, indem `req` als Pfad für `EKI_Send` angegeben wird:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_itempick-get_load_carriers", "req/args/load_carrier_ids/le",
↪"load_carrier2")
RET = EKI_Send("rc_itempick-get_load_carriers", "req")
```

6.4.3.2 Antwort-XML-Struktur

Das <SEND>-Element in der XML-Konfigurationsdatei für einen generischen Service folgt der folgenden Spezifikation:

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/<resX>" Type="<resX_type>"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Beispielsweise sieht das <RECEIVE>-Element des `rc_april_tag_detect` `detect` Services (siehe *TagDetect*, Abschnitt 5.2.2) wie folgt aus.

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose_frame" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose/@X" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Y" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Z" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@A" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@B" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@C" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/instance_id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/size" Type="REAL"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  </XML>
</RECEIVE>
```

Bei Arrays beinhaltet die Antwort mehrere Instanzen des gleichen XML-Elements. Jedes Element wird in einen separaten Puffer in EKI geschrieben und kann daraus mit KRL-Instruktionen ausgelesen werden. Die Anzahl an Instanzen (Array-Elementen) kann über `EKI_CheckBuffer` abgefragt werden und jede Instanz mit `EKI_Get<Type>` ausgelesen werden.

Beispielsweise können die Ergebnisposen aus einer Antwort des `rc_april_tag_detect detect` Services in KRL wie folgt ausgelesen werden:

```
DECL EKI_STATUS RET
DECL INT i
DECL INT num_instances
DECL FRAME poses[32]

DECL FRAME pose = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}

RET = EKI_CheckBuffer("rc_april_tag_detect-detect", "res/tags/le/pose")
num_instances = RET.Buff
for i=1 to num_instances
  RET = EKI_GetFrame("rc_april_tag_detect-detect", "res/tags/le/pose", pose)
  poses[i] = pose
endfor
RET = EKI_ClearBuffer("rc_april_tag_detect-detect", "res")
```

Bemerkung: Vor jeder Anfrage über EKI zum *rc_cube* sollten alle Puffer geleert werden, um sicherzustellen, dass nur die aktuelle Antwort in den EKI-Puffern enthalten ist.

6.4.4 Parameter

Die Parameter aller Softwaremodule können über die EKI-Bridge ausgelesen und gesetzt werden. Die XML-Konfigurationsdatei für ein generisches Softwaremodul folgt dieser Spezifikation:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="req/parameters/<parameter_y>/@value" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@default" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@min" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@max" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@value" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@default" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@min" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@max" Type="REAL"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Die Anfrage wird als Anfrage zum *Lesen* von Parametern interpretiert, wenn die `value`-Attribute aller

Parameter leer sind. Falls mindestens ein `value`-Attribut befüllt ist, wird die Anfrage als Anfrage zum *Setzen* von Parametern interpretiert und die befüllten Parameter gesetzt.

Beispielsweise können die aktuellen Werte aller Parameter von `rc_stereomatching` mit der folgenden XML-Nachricht abgefragt werden:

```
<req>
  <node>
    <rc_stereomatching></rc_stereomatching>
  </node>
  <parameters></parameters>
  <end_of_request></end_of_request>
</req>
```

Diese XML-Nachricht kann folgendermaßen über KRL gesendet werden:

```
DECL EKI_STATUS RET
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

Die Antwort der EKI-Bridge enthält alle Parameter:

```
<res>
  <parameters>
    <acquisition_mode default="Continuous" max="" min="" value="Continuous"/>
    <quality default="High" max="" min="" value="High"/>
    <static_scene default="0" max="1" min="0" value="0"/>
    <seg default="200" max="4000" min="0" value="200"/>
    <smooth default="1" max="1" min="0" value="1"/>
    <fill default="3" max="4" min="0" value="3"/>
    <minconf default="0.5" max="1.0" min="0.5" value="0.5"/>
    <mindepth default="0.1" max="100.0" min="0.1" value="0.1"/>
    <maxdepth default="100.0" max="100.0" min="0.1" value="100.0"/>
    <maxdeptherr default="100.0" max="100.0" min="0.01" value="100.0"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

Der `quality`-Parameter von `rc_stereomatching` kann mit folgender XML-Nachricht auf `Low` gesetzt werden:

```
<req>
    <node>
      <rc_stereomatching></rc_stereomatching>
    </node>
    <parameters>
      <quality value="Low"></quality>
    </parameters>
    <end_of_request></end_of_request>
</req>
```

Diese XML-Nachricht kann folgendermaßen über KRL gesendet werden:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_stereomatching-parameters", "req/parameters/quality/@value",
↪"Low")
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

In diesem Fall wird nur der gesetzte Wert von `quality` zurückgegeben:

```
<res>
  <parameters>
    <quality default="High" max="" min="" value="Low"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  </parameters>
  <return_code message="" value="0"/>
</res>
```

6.4.5 Beispielanwendungen

Ausführlichere Beispielanwendungen können unter https://github.com/roboception/eki_examples abgerufen werden.

6.5 gRPC Bilddatenschnittstelle

Die gRPC Bilddatenschnittstelle ist eine Alternative zur *GigE Vision / GenICam Schnittstelle* (Abschnitt 6.2) zum Streamen von Kamerabildern und synchronisierten Bilddaten (z.B. linkes Kamerabild und das dazugehörige Disparitätsbild). gRPC ist ein System zur Interprozesskommunikation über Rechnergrenzen hinweg, welches auch das Streamen von Daten unterstützt. Es benutzt Protocol Buffers (siehe https://developers.google.com/protocol-buffers/) als Beschreibungssprache und zur Datenserialisierung. Eine Einführung und mehr Details zu gRPC sind auf der offiziellen Webseite verfügbar (https://grpc.io/).

Die Vorteile der gRPC Schnittstelle gegenüber GigE Vision sind:

- Es ist in eigenen Programmen einfacher zu benutzen als GigE Vision.
- Es gibt gRPC Unterstützung für sehr viele Programmiersprachen (siehe https://grpc.io/).
- Die Kommunikation basiert auf TCP statt auf UDP und funktioniert deshalb besser über weniger stabile Netzwerke wie z.B. WLAN.

Die Nachteile der gRPC Schnittstelle im Vergleich zu GigE Vision sind:

- Es unterstützt nicht das Ändern von Parametern. Allerdings können alle Parameter über die *REST-API-Schnittstelle* (Abschnitt 6.3) geändert werden.
- Es unterstützt (noch) keine Farbbilder. Nur monochrome Bilder werden aktuell unterstützt.
- Es ist keine Standard-Bildverarbeitungsschnittstelle wie z.B. GigE Vision.

Der *rc_cube* bietet synchronisierte Bilddaten über gRPC Serverstreams auf Port 50051 an. Die Kommunikation wird gestartet indem eine `ImageSetRequest` Nachricht an den Server geschickt wird. Die Nachricht enthält die Information über angeforderte Bilder, d.h. linkes, rechtes, Disparitäts-, Konfidenz- oder Fehlerbild, die separat an- und abgeschaltet werden können.

Nach dem Empfangen der Anfrage sendet der Server kontinuierlich `ImageSet` Nachrichten, welche alle angeforderten Bilder mit allen Parametern enthalten, die notwendig sind, um die Bilder zu interpretieren. Die Bilder in einer `ImageSet` Nachricht sind synchronisiert, d.h. sie sind alle zum selben Zeitpunkt aufgenommen. Die einzige Ausnahme von dieser Regel besteht, wenn der *out1_mode* (Abschnitt 5.3.4.1) auf `AlternateExposureActive` gesetzt ist. In diesem Fall werden die Kamera- und Disparitätsbilder um 40 ms versetzt aufgenommen, sodass GPIO Out1 auf *aus* (LOW) steht, wenn das linke und rechte Bild aufgenommen werden, und auf *an* (HIGH) für das Disparitäts-, Konfidenz- und Fehlerbild. Dies ist sinnvoll, wenn ein Musterprojektor am *rc_visard* angeschlossen ist, da der Projektor dann bei der Aufnahme des linken und rechten Bildes aus ist und für das Disparitätsbild an, wodurch die Kamerabilder ungestört sind, aber das Disparitätsbild deutlich dichter und genauer wird.

Das Streamen von Bildern wird beendet, sobald der Client die Verbindung schließt.

6.5.1 gRPC Servicedefinition

```
syntax = "proto3";

message Time
{
  int32 sec = 1; ///< Seconds
  int32 nsec = 2; ///< Nanoseconds
}

message Gpios
{
  uint32 inputs  = 1; ///< bitmask of available inputs
  uint32 outputs = 2; ///< bitmask of available outputs
  uint32 values  = 3; ///< bitmask of GPIO values
}

message Image
{
  Time timestamp           = 1; ///< Acquisition timestamp of the image
  uint32 height            = 2; ///< image height (number of rows)
  uint32 width             = 3; ///< image width (number of columns)
  float focal_length       = 4; ///< focal length in pixels
  float principal_point_u  = 5; ///< horizontal position of the principal point
  float principal_point_v  = 6; ///< vertical position of the principal point
  string encoding          = 7; ///< Encoding of pixels ["mono8", "mono16"]
  bool is_bigendian        = 8; ///< is data bigendian, (in our case false)
  uint32 step              = 9; ///< full row length in bytes
  bytes data               = 10; ///< actual matrix data, size is (step * height)
  Gpios gpios              = 11; ///< GPIOs as of acquisition timestamp
  float exposure_time      = 12; ///< exposure time in seconds
  float gain               = 13; ///< gain factor in decibel
  float noise              = 14; ///< noise
  float out1_reduction     = 16; ///< Fraction of reduction (0.0 - 1.0) of exposure time for␣
↪images with GPIO Out1=Low in exp_auto_mode=AdaptiveOut1
  float brightness         = 17; ///< Current brightness of the image as value between 0 and 1
}

message DisparityImage
{
  Time timestamp           = 1; ///< Acquisition timestamp of the image
  float scale              = 2; ///< scale factor
  float offset             = 3; ///< offset in pixels (in our case 0)
  float invalid_data_value = 4; ///< value used to mark pixels as invalid (in our case 0)
  float baseline           = 5; ///< baseline in meters
  float delta_d            = 6; ///< Smallest allowed disparity increment. The smallest␣
↪achievable depth range resolution is delta_Z = (Z^2/image.focal_length*baseline)*delta_d.
  Image image              = 7; ///< disparity image
}

message ImageSet
{
  Time timestamp             = 1;
  Image left                 = 2;
  Image right                = 3;
  DisparityImage disparity   = 4;
  Image disparity_error      = 5;
  Image confidence           = 6;
}

message ImageSetRequest
{
  bool left_enabled            = 1;
  bool right_enabled           = 2;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  bool disparity_enabled       = 3;
  bool disparity_error_enabled = 4;
  bool confidence_enabled      = 5;
}

service ImageInterface
{
  // A server-to-client streaming RPC.
  rpc StreamImageSets(ImageSetRequest) returns (stream ImageSet) {}
}
```

6.5.2 Umwandlung von Bild-Streams

Zur Umwandlung von Disparitätsbildern in Punktwolken dient die Beschreibung im Kapitel *GigE Vision / GenICam Schnittstelle* (Abschnitt 6.2.7).

6.5.3 Einschränkungen

Aktuell werden nur monochrome Bilder unterstützt.

6.5.4 Beispielclient

Ein einfacher C++ Client kann von https://github.com/roboception/grpc_image_client_example heruntergeladen werden.

6.6 Zeitsynchronisierung

Der *rc_cube* stellt für alle Bilder und Nachrichten Zeitstempel zur Verfügung. Um diese mit der Zeit auf dem Applikations-Rechner zu vergleichen, muss die Zeit synchronisiert werden.

Die Synchronisierung der Systemzeiten auf dem *rc_cube* und dem Anwendungsrechner kann über das Network Time Protocol (NTP) erfolgen, welches standardmäßig aktiviert ist.

Die interne Synchronisierung der Systemzeiten zwischen dem *rc_cube* und dem angeschlossenen *rc_visard* wird automatisch über das Precision Time Protocol (PTP) vorgenommen.

Die aktuelle Systemzeit sowie der Status der Zeitsynchronisierung können über die *REST-API* (Abschnitt 6.3) abgerufen und darüber hinaus auch in der *Web GUI* (Abschnitt 6.1) auf der Seite *System* eingesehen werden.

Bemerkung: Abhängig von der Erreichbarkeit von NTP- oder PTP-Servern, kann es bis zu mehreren Minuten dauern, bis die Zeit synchronisiert ist.

6.6.1 NTP

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

Standardmäßig versucht der *rc_cube* den NTP-Server des NTP-Pool-Projekts zu erreichen, wozu eine Verbindung zum Internet nötig ist.

Falls die Netzwerkkonfiguration des *rc_cube* auf *DHCP* (Abschnitt 3.5.2) (entspricht der Werkseinstellung) konfiguriert ist, werden NTP-Server auch vom DHCP-Server angefordert und verwendet.

6.6.2 PTP

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

Bemerkung: Die Synchronisierung der Systemzeit zwischen dem *rc_cube* und dem Anwendungsrechner ist nur über NTP möglich.

7 Wartung

7.1 Backup der Einstellungen

Der *rc_cube* bietet die Möglichkeit, die aktuellen Einstellungen als Backup oder zum Übertragen auf einen anderen *rc_visard* oder *rc_cube* herunterzuladen.

Die aktuellen Einstellungen des *rc_cube* können über die *Web GUI* (Abschnitt 6.1) auf der Seite *System* in der Zeile *Backup rc_cube Einstellungen* heruntergeladen werden, oder über die *REST-API-Schnittstelle* (Abschnitt 6.3) des *rc_cube* mit Hilfe des Aufrufs *GET /system/backup*.

Beim Herunterladen des Backups kann der Nutzer entscheiden, welche Einstellungen das Backup enthalten soll:

- `nodes`: die Einstellungen aller Module (Parameter und bevorzugte TCP-Orientierungen)
- `load_carriers`: die erstellten Load Carrier
- `regions_of_interest`: die erstellten 2D und 3D Regions of Interest
- `grippers`: die erstellten Greifer

Das zurückgelieferte Backup sollte als .json-Datei gespeichert werden.

Ein Backup kann auf dem *rc_cube* über die *Web GUI* (Abschnitt 6.1) auf der Seite *System* in der Zeile *Backup rc_cube Einstellungen* eingespielt werden, indem die Backup .json-Datei hochgeladen wird. In der *Web GUI* werden die im Backup enthaltenen Einstellungen angezeigt und können für das Einspielen ausgewählt werden. Der zugehörige Aufruf der *REST-API-Schnittstelle* (Abschnitt 6.3) ist *POST /system/backup*.

Warnung: Wenn ein Backup von Load Carriern eingespielt wird, gehen alle bestehenden Load Carrier auf dem *rc_cube* verloren und werden durch die Load Carrier im Backup ersetzt. Das gleiche trifft auf das Einspielen von Greifern und Regions of Interest zu.

Wenn ein Backup eingespielt wird, werden nur die Einstellungen gesetzt, die für den jeweiligen *rc_cube* zutreffend sind. Parameter für Module, die nicht existieren oder keine gültige Lizenz haben, werden ignoriert. Wenn ein Backup nur teilweise eingespielt werden konnte, wird der Benutzer über Warnungen darüber informiert.

7.2 Aktualisierung der Firmware

Angaben zur aktuellen Firmware-Version sind auf der Seite *System* in der Zeile *Systeminformationen* in der *Web GUI* (Abschnitt 6.1) angegeben. Diese Informationen lassen sich mithilfe einer *GET /system*-Anfrage über die die *REST-API-Schnittstelle* (Abschnitt 6.3) des *rc_cube* abrufen. Die Aktualisierung der Firmware kann entweder über die Web GUI oder über die REST-API vorgenommen werden.

Warnung: 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 6.3) abgefragt und in der Anwendung oder auf dem Client-PC gesichert werden.

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

- die Netzwerkkonfiguration des *rc_cube*, samt der ggf. vergebenen festen IP-Adresse und des benutzerdefinierten Gerätenamens,
- das letzte Ergebnis der *Hand-Auge-Kalibrierung* (Abschnitt 5.3.1), was bedeutet, dass der *rc_cube* nicht neu zum Roboter kalibriert werden muss, es sei denn, die Montage wurde verändert, und

Schritt 1: Download der neuesten Firmware Firmware-Updates werden in Form einer Mender-Artefakt-Datei bereitgestellt, die an ihrem `.mender`-Suffix erkennbar ist.

Ist ein neues Firmware-Update für den *rc_cube* 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_cube* vorgenommen werden, kann der Benutzer auf die Anfrage `POST /system/update` zurückgreifen.

Um die Firmware über die Web GUI zu aktualisieren, muss auf der Seite *System* in der Zeile *Software-Update* die Schaltfläche *rc_cube Update hochladen* betätigt werden. Nachdem die gewünschte Update-Image-Datei (Dateierweiterung `.mender`) aus dem lokalen Dateisystem ausgewählt und geöffnet wurde, startet das Update.

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

Bemerkung: Je nach Webbrowser kann es vorkommen, dass der angezeigte Statusbalken den Abschluss des Updates zu früh angibt. Es empfiehlt sich, zu warten, bis sich ein Benachrichtigungsfenster öffnet, das das Ende des Updatevorgangs anzeigt. Insgesamt ist mit einer Update-Dauer von mindestens fünf Minuten zu rechnen.

Warnung: Die Webbrowser-Registerkarte, die die Web GUI enthält, darf weder geschlossen noch aktualisiert werden, da der Update-Vorgang anderenfalls unterbrochen wird. Ist dies der Fall, muss der Update-Vorgang neu gestartet werden.

Schritt 3: Neustart des *rc_cube* Um ein Firmware-Update auf den *rc_cube* aufzuspielen, muss nach dem Upload der neuen Image-Datei ein Neustart vorgenommen werden.

Bemerkung: Die neue Firmware-Version wird in die inaktive Partition des *rc_cube* 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_cube* inaktiv und es wird automatisch die zuvor installierte Firmware-Version von der aktiven Partition verwendet.

Über die REST-API lässt sich der Neustart mittels der Anfrage `PUT /system/reboot` vornehmen.

Nachdem die neue Firmware über die Web GUI hochgeladen wurde, öffnet sich ein Benachrichtigungsfenster, in dem der Benutzer aufgefordert wird, das Gerät sofort neu zu starten oder aber den Neustart zu verschieben. Soll der *rc_cube* zu einem späteren Zeitpunkt neu gestartet werden, kann dies über die Schaltfläche *Neustart* auf der Web GUI-Seite *System* vorgenommen werden.

Schritt 4: Bestätigung des Firmware-Updates Nach dem Neustart des *rc_cube* ist die Versionsnummer des derzeit aktiven Firmware-Images zu überprüfen, sodass sichergestellt ist, dass das ak-

tualisierte Image erfolgreich geladen wurde. Dies kann entweder über die Web GUI auf der Seite *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.

7.3 Wiederherstellung der vorherigen Firmware-Version

Nach einem erfolgreichen Firmware-Update wird das vorherige Firmware-Image auf der inaktiven Partition des *rc_cube* hinterlegt und kann von dort bei Bedarf wiederhergestellt werden. Dieses Verfahren wird auch als *Rollback* bezeichnet.

Bemerkung: Es wird dringend empfohlen, die neueste Firmware-Version zu verwenden, die von Roboception zur Verfügung gestellt wurde. Auf das Rollback sollte nur dann zurückgegriffen werden, wenn es mit der aktualisierten Firmware-Version große Probleme gibt.

Die Rollback-Funktion kann lediglich über die *REST-API-Schnittstelle* (Abschnitt 6.3) des *rc_cube* aufgerufen werden – mithilfe der Anfrage `PUT /system/rollback`. Die Anfrage kann entweder mit einem HTTP-kompatiblen Client oder, wie in *Swagger UI* (Abschnitt 6.3.4) 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.

7.4 Neustart des *rc_cube*

Nach einem Firmware-Update oder einem Software-Rollback muss der *rc_cube* neu gestartet werden. Der Neustart lässt sich entweder programmgesteuert mithilfe der Anforderung `PUT /system/reboot` über die *REST-API-Schnittstelle* (Abschnitt 6.3) des *rc_cube* oder manuell auf der Seite *System* der *Web GUI* (Abschnitt 6.1) vornehmen.

7.5 Aktualisierung der Softwarelizenz

Lizenzen, die von Roboception zur Aktivierung zusätzlicher Funktionen erworben werden, können über die Seite *System* der *Web GUI* (Abschnitt 6.1) installiert werden. Der *rc_cube* muss neu gestartet werden, um die Lizenz nutzen zu können.

Bemerkung: Für den Fall, dass ein Bildschirm sowie Maus und Tastatur an den *rc_cube* angeschlossen sind, kann die Aktualisierung der Softwarelizenz auch direkt am *rc_cube* mithilfe der Web GUI und einem separaten USB-Stick, von welchem die neue Lizenzdatei installiert werden kann, erfolgen.

7.6 Download der Logdateien

Während des Betriebs dokumentiert der *rc_cube* wichtige Informationen, Hinweise und Fehler in sogenannten Logdateien. Zeigt der *rc_cube* ein unerwartetes oder fehlerhaftes Verhalten, kann mithilfe der Logdateien nach der Fehlerursache geforscht werden. Logeinträge lassen sich über die Seite *Logs* auf der *Web GUI* (Abschnitt 6.1) ansehen und filtern. Wird der Support kontaktiert (*Kontakt*, Abschnitt 9), sind die Logdateien sehr hilfreich, um Probleme aufzuspüren. Um diese als tar.gz-Datei herunterzuladen, ist die Option *Alle Logs herunterladen* auf der Seite *Logs* der Web GUI auszuwählen.

Die Logs sind nicht nur über die Web GUI, sondern auch über die *REST-API-Schnittstelle* (Abschnitt 6.3) des *rc_cube* zugänglich. Hierfür können die Anfragen des Typs `GET /logs` und `GET /logs/{log}` verwendet werden.

Bemerkung: Für den Fall, dass ein Bildschirm sowie Maus und Tastatur an den *rc_cube* angeschlossen sind, können die Log-Dateien mithilfe der Web GUI auch direkt vom *rc_cube* auf einen separaten USB-Stick heruntergeladen werden.

8 Fehlerbehebung

8.1 Probleme mit den Kamerabildern

Kamerabild ist zu hell

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verkürzen (siehe *Parameter*, Abschnitt 5.1.1.4) oder
- schalten Sie auf automatische Belichtung um (siehe *Parameter*, Abschnitt 5.1.1.4).

Kamerabild ist zu dunkel

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verlängern (siehe *Parameter*, Abschnitt 5.1.1.4) oder
- schalten Sie auf automatische Belichtung um (siehe *Parameter*, Abschnitt 5.1.1.4).

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 5.1.1.4).

Kamerabild ist unscharf

- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt, und erhöhen Sie bei Bedarf den Abstand zwischen dem Objekt und der Linse.
- Überprüfen Sie, ob die Kameralinsen verschmutzt sind, und reinigen Sie diese bei Bedarf.
- Trifft keiner der vorstehenden Punkte zu, kann es sein, dass ein schweres Hardware-Problem vorliegt. Bitte wenden Sie sich an den *Support* (Abschnitt 9).

Kamerabild ist verschwommen

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

- verringern Sie die Bewegungsgeschwindigkeit der Kamera,
- verringern Sie die Bewegungsgeschwindigkeit von Objekten im Sichtfeld der Kamera oder
- verkürzen Sie die Belichtungszeit der Kameras (siehe *Parameter*, Abschnitt 5.1.1.4).

Bildwiederholrate ist zu niedrig

- Erhöhen Sie die Bildwiederholrate gemäß den Anweisungen in *Parameter* (Abschnitt 5.1.1.4).
- Die maximale Bildwiederholrate der Kameras beträgt 25 Hz.

8.2 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 8.1).
- Überprüfen Sie, ob die Szene genügend Textur hat (siehe *Stereo-Matching*, Abschnitt 5.1.2) und installieren Sie bei Bedarf einen Musterprojektor.
- Senken Sie den *Minimalen Abstand* (Abschnitt 5.1.2.5).
- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 5.1.2.5).
- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt, und erhöhen Sie bei Bedarf den Abstand zwischen dem Objekt und der Linse.
- Senken Sie die *Minimale Konfidenz* (Abschnitt 5.1.2.5).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 5.1.2.5).
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 5.1.2.5). Disparitätsbilder mit einer geringeren Auflösung sind in der Regel nicht so spärlich befüllt.

Bildwiederholrate der Disparitätsbilder ist zu niedrig

- Überprüfen und erhöhen Sie die Bildwiederholrate der Kamerabilder (siehe *Parameter*, Abschnitt 5.1.1.4). Die Bildwiederholrate der Disparitätsbilder kann nicht größer sein als die Bildwiederholrate der Kamerabilder.
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 5.1.2.5).
- Erhöhen Sie den *Minimalen Abstand* (Abschnitt 5.1.2.5) so viel wie für die Applikation möglich.

Disparitätsbild zeigt keine nahe liegenden Objekte

- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt, und erhöhen Sie bei Bedarf den Abstand zwischen dem Objekt und der Linse.
- Senken Sie den *Minimalen Abstand* (Abschnitt 5.1.2.5).

Disparitätsbild zeigt keine weit entfernten Objekte

- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 5.1.2.5).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 5.1.2.5).
- Senken Sie die *Minimale Konfidenz* (Abschnitt 5.1.2.5).

Disparitätsbild rauscht zu stark

- Erhöhen Sie den *Segmentierungs-Wert* (Abschnitt 5.1.2.5).
- Erhöhen Sie den *Füllen-Wert* (Abschnitt 5.1.2.5).

Disparitätswerte oder resultierende Tiefenwerte sind zu ungenau

- Verringern Sie den Abstand zwischen der Kamera und der Szene. Der Tiefenmessfehler nimmt quadratisch mit dem Abstand zu den Kameras zu.
- Überprüfen Sie, ob die Szene wiederkehrende Muster enthält und entfernen Sie diese bei Bedarf. Diese könnten falsche Disparitätsmessungen verursachen.

Disparitätsbild ist zu glatt

- Senken Sie den *Füllen-Wert* (Abschnitt 5.1.2.5).

Disparitätsbild zeigt keine feinen Strukturen

- Senken Sie den *Segmentierungs-Wert* (Abschnitt 5.1.2.5).
- Senken Sie den *Füllen-Wert* (Abschnitt 5.1.2.5).

8.3 Probleme mit GigE Vision/GenICam

Keine Bilder

- Überprüfen Sie, ob die Bildkomponenten aktiviert sind. Siehe `ComponentSelector` und `ComponentEnable` in *Wichtige Parameter der GenICam-Schnittstelle* (Abschnitt 6.2.2).

9 Kontakt

9.1 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.

9.2 Downloads

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

9.3 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

10 Anhang

10.1 Formate für Posendaten

Eine Pose besteht aus einer Translation und einer Rotation. Die Translation definiert die Verschiebung entlang der x, y und z-Achsen. Die Rotation kann auf viele verschiedene Arten definiert werden. Der *rc_cube* benutzt Quaternionen, um Rotationen zu definieren, und Translationen werden in Metern angegeben. Dies wird als XYZ+Quaternion Format bezeichnet. Dieses Kapitel erklärt Umrechnungen zwischen verschiedenen üblichen Posenformaten und dem XYZ+Quaternion Format.

Es ist weit verbreitet, Rotationen in 3D durch drei Winkel als Drehungen um die Koordinatenachsen zu definieren. Leider existieren hierfür viele verschiedene Möglichkeiten. Übliche Konventionen sind Euler- oder Kardanwinkel (letztere werden auch als Tait-Bryan Winkel bezeichnet). In beiden Konventionen können die drei Rotationen auf die bereits gedrehten Achsen (intrinsische Rotation) oder auf die Achsen des festen Koordinatensystems (extrinsische Rotation) angewendet werden.

Wir benutzen x, y und z zur Bezeichnung der drei Koordinatenachsen. x', y' und z' bezeichnen die Achsen, die einmal rotiert wurden. x'', y'' und z'' bezeichnen die Achsen nach zwei Rotationen.

In der ursprünglichen Eulerwinkelkonvention ist die erste und dritte Drehachse immer identisch. Die Rotationsreihenfolge z-x'-z'' bedeutet z.B. eine Drehung um die z-Achse, dann eine Drehung um die gedrehte x-Achse und schließlich eine Drehung um die (zweimal) gedrehte z-Achse. In der Kardanwinkelkonvention sind alle drei Drehachsen unterschiedlich, z.B. z-y'-x''. Kardanwinkel werden häufig ebenfalls als Eulerwinkel bezeichnet.

Für jede intrinsische Rotationsreihenfolge gibt es eine äquivalente extrinsische Rotationsreihenfolge, die genau umgekehrt ist. Die intrinsische Rotationsreihenfolge z-y'-x'' ist zum Beispiel äquivalent zu der extrinsischen Rotationsreihenfolge x-y-z.

Rotationen um die x, y und z-Achse können mit Quaternionen definiert werden als

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

oder durch Rotationsmatrizen als

$$r_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix},$$
$$r_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix},$$
$$r_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Die extrinsische Rotationsreihenfolge x-y-z kann durch die Multiplikation einzelner Rotationen in um-

gekehrter Reihenfolge berechnet werden, d.h. $r_z(\gamma)r_y(\beta)r_x(\alpha)$.

Basierend auf diesen Definitionen beschreiben die folgenden Abschnitte die Umrechnung zwischen üblichen Konventionen und dem XYZ+Quaternion Format.

Bemerkung: Zu beachten sind stets die Einheiten für Positionen und Orientierungen. *rc_cube* Geräte benutzen stets Meter als Längeneinheit, während die meisten Roboterhersteller Längen in Millimeter oder Inch angeben. Winkel werden üblicherweise in Grad angegeben, können aber auch im Bogenmaß angegeben sein.

10.1.1 Rotationsmatrix und Translationsvektor

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

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

Die Posentransformation für einen Punkt P ist

$$P' = RP + T.$$

10.1.1.1 Umrechnung von Rotationsmatrizen in Quaternionen

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

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

Der sign Operator gibt -1 zurück, falls sein Argument negativ ist. Sonst wird 1 zurück gegeben. Er wird zur Wiederherstellung das Vorzeichens der Wurzel benutzt. Die max Funktion stellt sicher, dass das Argument der Wurzel nicht negativ ist, was in der Praxis durch Rundungsfehler passieren kann.

10.1.1.2 Umrechnung von Quaternionen in Rotationsmatrizen

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}^T$ mit $||q|| = 1$ in eine Rotationsmatrix kann wie folgt durchgeführt werden.

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

10.1.2 ABB Posenformat

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

10.1.3 FANUC XYZ-WPR Format

Das Posenformat, welches von FANUC Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung WPR, welche durch drei Winkel in Grad gegeben ist. W rotiert um die x-Achse, P rotiert um die y-Achse und R rotiert um die z-Achse. Die Rotationsreihenfolge ist x-y-z und wird berechnet durch $r_z(R)r_y(P)r_x(W)$.

10.1.3.1 Umrechnung von FANUC-WPR in Quaternionen

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

$$\begin{aligned} W_r &= W\frac{\pi}{180}, \\ P_r &= P\frac{\pi}{180}, \\ R_r &= R\frac{\pi}{180}, \end{aligned}$$

und damit wird die Quaternion berechnet als

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

10.1.3.2 Umrechnung von Quaternionen in FANUC-WPR

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

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

10.1.4 Kawasaki XYZ-OAT Format

Das Posenformat, welches von Kawasaki Robotern benutzt wird, besteht aus einer Position XYZ und einer Orientierung OAT, welche durch drei Winkel in Grad angegeben wird. O rotiert um die z-Achse, A rotiert um die gedrehte y-Achse und T rotiert um die gedrehte z-Achse. Die Rotationsreihenfolge ist z-y'-z'' (d.h. z-y-z) und wird berechnet durch $r_z(O)r_y(A)r_z(T)$.

10.1.4.1 Umrechnung von Kawasaki-OAT in Quaternionen

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

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

und damit wird die Quaternion berechnet durch

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

10.1.4.2 Umrechnung von Quaternionen in Kawasaki-OAT

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

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

10.1.5 KUKA XYZ-ABC Format

KUKA Roboter nutzen das sogenannte XYZ-ABC Format. XYZ ist die Position in Millimetern. ABC sind Winkel in Grad, wobei A um die z-Achse rotiert, B rotiert um die y-Achse und C rotiert um die x-Achse. Die Rotationsreihenfolge ist z-y'-x'' (i.e. x-y-z) und wird berechnet durch $r_z(A)r_y(B)r_x(C)$.

10.1.5.1 Umrechnung von KUKA-ABC in Quaternionen

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

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

und damit die Quaternion berechnet durch

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

10.1.5.2 Umrechnung von Quaternionen in KUKA-ABC

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

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

10.1.6 Mitsubishi XYZ-ABC Format

Das Posenformat, welches von Mitsubishi Robotern benutzt wird, ist das gleiche wie für KUKA Roboter (siehe *KUKA XYZ-ABC Format*, Abschnitt 10.1.5), außer, dass der Winkel A um die x-Achse rotiert und C eine Rotation um die z-Achse ist. Damit wird die Rotation berechnet durch $r_z(C)r_y(B)r_x(A)$.

10.1.6.1 Umrechnung von Mitsubishi-ABC in Quaternionen

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

$$A_r = A\frac{\pi}{180},$$
$$B_r = B\frac{\pi}{180},$$
$$C_r = C\frac{\pi}{180},$$

und damit die Quaternion berechnet durch

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

10.1.6.2 Umrechnung von Quaternionen in Mitsubishi-ABC

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

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

10.1.7 Universal Robots Posenformat

Das Posenformat, welches von Universal Robots verwendet wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung im Angle-Axis Format mit dem Winkel θ im Bogenmaß als Länge der Rotationsachse U.

$$V = \begin{pmatrix} \theta u_x \\ \theta u_y \\ \theta u_z \end{pmatrix}$$

Dies wird als Rotationsvektor bezeichnet.

10.1.7.1 Umrechnung vom Angle-Axis Format in Quaternionen

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

Zunächst wird der Winkel θ im Bogenmaß aus dem Rotationsvektor V gewonnen durch

$$\theta = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

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

$$\begin{aligned} x &= v_x \frac{\sin(\theta/2)}{\theta}, \\ y &= v_y \frac{\sin(\theta/2)}{\theta}, \\ z &= v_z \frac{\sin(\theta/2)}{\theta}, \\ w &= \cos(\theta/2). \end{aligned}$$

10.1.7.2 Umrechnung von Quaternionen ins Angle-Axis Format

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

Zunächst wird der Winkel θ im Bogenmaß aus dem Quaternion gewonnen durch

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

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

$$\begin{aligned} v_x &= \theta \frac{x}{\sqrt{1 - w^2}}, \\ v_y &= \theta \frac{y}{\sqrt{1 - w^2}}, \\ v_z &= \theta \frac{z}{\sqrt{1 - w^2}}. \end{aligned}$$

HTTP Routing Table

/logs

/nodes

/system

Stichwortverzeichnis

Sonderzeichen

A

B

C

D

E

F

G

H

I

V

W

X

Z

roboception

rc_cube Beschleuniger für den rc_visard

MONTAGE- UND BETRIEBSANLEITUNG

Roboception GmbH

Kaflerstraße 2
81241 München
Deutschland

info@roboception.de
www.roboception.de

Tutorials: http://tutorials.roboception.de
GitHub: https://github.com/roboception
Dokumentation: http://doc.rc-visard.com
Shop: https://roboception.com/shop

Für Kundensupport kontaktieren Sie

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

support@roboception.de