# IB-RM-OP-2022-23

## Entwicklung eines Autonomen Ladevorgangs für das Robotersystem Rollin‘ Justin Bachelorarbeit

Adrian Michael Müller

Deutsches Zentrum
DLR für Luft- und Raumfahrt


Deutsches Zentrum für Luft- und Raumfahrt (DLR)
Institut für Robotik und Mechatronik

Technische Hochschule Ulm
Fakultät Elektrotechnik und Informationstechnik

# Entwicklung eines Autonomen Ladevorgangs für das Robotersystem Rollin'Justin

## Bachelorarbeit

eingereicht am
**25.Februar 2022**

| | |
|---|---|
| Verfasser: | Adrian Michael Müller |
| Matrikelnummer: | 3128916 |
| Erstgutachter: | Frau Prof. Dr. Marianne von Schwerin |
| Zweitgutachter: | Herr Prof. Dr.-Ing. Silko-Matthias Kruse |
| Ausbildungsbetrieb: | Deutsches Zentrum für Luft- und Raumfahrt |
| Betreuer: | Herr Peter Schmaus, M.Sc. |

# Eidesstattliche Erklärung

Diese Abschlussarbeit wurde von mir selbständig verfasst. Es wurden nur die angegebenen Quellen und Hilfsmittel verwendet. Alle wörtlichen und sinngemäßen Zitate sind in dieser Arbeit als solche kenntlich gemacht.

Ulm, 25.2.2022
Ort, Datum

Unterzeichner

# Abstract

Die vorliegende Arbeit dokumentiert die Entwicklung und die Umsetzung eines autonomen Ladeprozesses für das Robotersystem Rollin' Justin des Deutschen Zentrums für Luft- und Raumfahrt (DLR) in Oberpfaffenhofen, um dem Robotersystem Langzeitautonomie zu ermöglichen. Sie beinhaltet die einzelnen Entwicklungsschritte von der Fertigstellung der Ladestation über die Erstellung und die Umsetzung eines Ladekonzepts, sowie die Integration dessen in die autonomen Fähigkeiten des Robotersystems. Dies geschieht anhand der Lokalisation der Ladestation via AprilTags, der Routenplanung mit Hilfe eines hybriden Planungsalgorithmus, der daraufhin folgenden Anfahrt der Ladestation, der genauen Berechnung des Abstands zwischen Roboter und Ladestation und schlussendlich des präzisen Andockens des Roboters an diese. Zum Schluss wird das erstellte Ladekonzept evaluiert und auf mögliche Erweiterungen sowie auf den Nutzen des Systems eingegangen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

**DLR** Deutsches Zentrum für Luft- und Raumfahrt

**DFS** depth-first search

**DLT** Direkte Lineare Transformation

**SPUs** Smart Payload Units

**CPU** Central Processing Unit

**GPU** Graphics Processing Unit

**KI** Künstliche Intelligenz

**LN** Links and Nodes

**PDDL** Planning Domain Definition Language

**XML** Extensible Markup Language

**odb** Objektdatenbank

**ROS** Robot Operating System

**LN** Links and Nodes

**DHCP** Dynamic Host Configuration Protocol

**Wifi** Wireless Fidelity

**GUI** Graphical User Interface

# 1. Einführung

Ob in der Logistik, der Industrie oder im Servicebereich: Robotersysteme sind weltweit auf dem Vormarsch. Sogenannte Serviceroboter sind darauf ausgelegt Dienstleistungen für den Menschen zu erbringen, darunter auch alltägliche Manipulationsaufgaben, wie z.B. das Putzen von Fenstern. Der humanoide Roboter Rollin' Justin ist eine Plattform für die Forschung in diesem Bereich. Er ist insbesondere für den Einsatz im Haushalt und in der Assistenz von Astronauten im Weltraum konzipiert. Um einen reibungslosen autonomen Betrieb zu garantieren, benötigt der Roboter eine lange Akkulaufzeit und die Möglichkeit zum Nachladen.

Idealerweise sollte der Ladeprozess vom Roboter autonom durchgeführt werden können, damit Langzeitautonomie möglich wird und der Roboter in seinen autarken Fähigkeiten nicht durch die Kapazität seines Akkus beschränkt wird. Im Moment ist für den Ladeprozess und den Transfer großer Daten beim Robotersystem Rollin‘ Justin menschliches Eingreifen notwendig, in Form eines manuellen Einsteckens des Lade- bzw. Datenkabels. Die vorliegende Bachelorarbeit befasst sich mit der Automatisierung des Ladeprozesses. Angestrebt ist ein sensorgesteuertes Koppeln von Rollin‘ Justin an die Ladestation und ein selbständiges Entkoppeln nach vollendetem Ladevorgang.
Für die fehlerfreie Durchführung des Koppelprozesses muss der Roboter in der Lage sein, die Ladestation im Gelände zu erkennen, seine Route dorthin festzulegen und sich in eine geeignete Andockposition zu navigieren, um daraufhin exakt anzudocken.

## 1.1. Aufgabenstellung

Dem Robotersystem Rollin‘ Justin soll durch diese Bachelorarbeit ein autonomes Laden, durch selbstständiges Anfahren/Andocken/Abdocken an eine bereits konstruierte Ladestation, ermöglicht werden (siehe Abbildung 1.1). Dies soll über einen Magnetstecker bewerkstelligt werden, der am hinteren Teil der Plattform des Roboters befestigt ist. Hierzu muss der Roboter rückwärts die Ladestation präzise anfahren, um sich mit dem dort frei hängend montierten Gegenstück zu verbinden.
Schwierigkeiten sind bei der Fahrgenauigkeit des Roboters zu erwarten, da der Roboter nur über ein Führungsrad gesteuert wird. Die restlichen drei Räder des Roboters setzen Winkelveränderungen daher mit einer gewissen Latenz um. Auch das Sichtfeld der hinteren Plattformkameras ist problematisch, da diese einen toten Winkel/Teilbereich hinter dem Roboter besitzen und sich so der Stecker beim Ansteckvorgang außerhalb des Kamerasichtfeldes befindet.

Abbildung 1.1.: Abbildung des an die Ladestation angedockten Robotersystem

## 1.2. Aufbau der Arbeit

Die vorliegende Bachelorarbeit ist in sechs Kapitel unterteilt und befasst sich zunächst mit den Grundlagen der Posen-Bestimmung durch Referenzmarken. Es folgt eine nähere Beschreibung der Hard- und Software des Robotersystems Rollin' Justin und der vorhandenen Ladestation. Im Hauptteil ist das Konzept zur Anfahrt an die Ladestation in fünf Schritten dargestellt. Dieses beinhaltet die Lokalisation der Ladestation, die Routenplanung und die Anfahrt, die Berechnung des Steckerabstandes zwischen Ladestation und Robotersystem, den Ansteckvorgang und den Absteckvorgang. Im weiteren wird genauer auf die vorbereitenden Tätigkeiten zur Umsetzung des Konzepts durch Pythonskripte eingegangen. Danach wird die Evaluation der Ausführung der Pythonskripte und die Integration in die autonomen Systeme des Roboters geschildert. Mit einem Fazit und einem Ausblick auf weiterführende Arbeiten wird die Arbeit abgeschlossen.

# 2. Grundlagen

Zum Verständnis dieser Arbeit ist es notwendig sich mit der Lokalisation von Objekten im dreidimensionalen Raum mit Hilfe von Referenzmarken auseinanderzusetzen. Zunächst wird erläutert, wie die Darstellung und die Berechnung von Position und Orientierung eines Objektes in Bezug auf ein anderes funktioniert. Im weiteren wird die Funktionalität der Referenzmarke AprilTag erklärt, welche zur Lokalisierung der Ladestation genutzt wird.

## 2.1. State of the Art

Um die Arbeit des Menschen zu erleichtern sind heutzutage schon viele Arbeitsprozesse teil- oder vollautomatisiert. Der nächste Entwicklungsschritt geht in Richtung ganz autonom arbeitender Systeme. Diese Systeme benötigen dann kaum bis kein menschliches Eingreifen mehr. Schon real und für alle erfahrbar ist dies bei bereits autonomen arbeitenden Systemen, die im Alltag genutzt werden wie z.B. Mäh- und Saugroboter oder in der Automobilbranche. Weitere befinden sich in der Entwicklung. Für die Umsetzung eines langzeitautonomen Systems wird ein autonomer Ladeprozess benötigt. Dieser beinhaltet die Navigation zur Ladestation sowie den Andockprozess.

Die Lösungen des autonomen Andockens und Ladens der genannten Systeme wurden genauer analysiert. Der Mähbereich eines Mähroboters wird durch ein im Boden verlegtes Begrenzungskabel festgelegt. Die Ladestation eines solchen Roboters ist oft zusätzlich mit einem Nahsignalsender ausgestattet. Dieser wird für die genaue Navigation und den Andockprozess genutzt. Um die Ladestation zu finden, haben sich drei Konzepte durchgesetzt: Suchen per Zufallsprinzip, Suchen per Begrenzungskabel und Suchen per Suchkabel. Bei der Zufallssuche fährt der Roboter, solange im Garten herum bis er das Nahsignal der Ladestation empfängt. Bei der zweiten Version fährt der Roboter solange am Begrenzungskabel entlang bis er die Ladestation findet. Ähnlich funktioniert es mit Suchkabel, dieses separate Kabel muss jedoch möglichst zentral durch die Mähfläche verlaufen. In allen drei Fällen orientiert sich der Roboter ausschließlich am Nahsignal, sobald er dieses empfängt. Mit dessen Hilfe wird auch der Andockprozess umgesetzt.

Ein Saugroboter orientiert sich oft an einer Umgebungskarte, welche er mithilfe von LIDAR oder anderen Sensoren erstellt. Zusätzlich senden viele Ladestationen ein Infrarot-Leitsignal aus. Dieses dient zur genauen Navigation in die Ladestation. Mäh- und Saugroboter werden beim Andockprozess in fast allen Fällen durch Führungsschienen und Begrenzungen beim Einfahren in die Ladestation zentriert. Geladen werden beide Roboterarten über Schleifkontakte.

Für Automobile gibt es von Volkswagen ein Konzept zum autonomen Laden von Elektro-

autos namens *V-Charge*: Hierfür wurde fahrerloses Parken und ein Parkplatz mit induktiver Ladefunktion kombiniert. Die Navigation des Automobils wird mithilfe von zwei Stereo-Kameras, vier mono-Kameras und zwölf Ultraschallsensoren sowie durch Kommunikation mit einem intelligenten Parkhaus, welches einen Parkhausplan und weitere Informationen bereitstellt, bewerkstelligt. Als Orientierungspunkte bei der Navigation in eine Parklücke dienen die Parkplatzbegrenzungen. Geladen wird das Auto induktiv.[8]

Durch diese Analyse wurden folgende Möglichkeiten zur Navigation und Orientierung aufgefunden: Führungsdraht, Infrarot, LIDAR und über Kameras mittels Orientierungspunkten. Für das Robotersystem Rollin' Justin ist die Navigation über einen Führungsdraht nicht möglich, da das Verlegen eines Führungsdrahts im Labor nicht umsetzbar ist und dies auch den Aktionsbereich des Robotersystems enorm einschränken würde. Eine Navigation über Infrarot oder LIDAR ist generell denkbar, allerdings verfügt das Robotersystem momentan nicht über entsprechende Hardware. Da der Roboter jedoch mit verschiedene Kamerasysteme ausgestattet ist, werden diese genutzt um über Orientierungspunkte zu navigieren. Hierfür stehen prinzipiell zwei Möglichkeiten zur Verfügung, um Objekte zu erkennen und deren Position zu bestimmen: Objekterkennung oder Referenzmarken. Bei der Objekterkennung werden durch eine Künstliche Intelligenz (KI) Objekte auf Bildern klassifiziert und deren Position durch Größe und Position auf der Bildebene ermittelt. Bei der Orientierung durch Referenzmarken kann aus einem Kamerabild die Position und die Orientierung der Marke zur Kamera bestimmt werden. Ist die Position und Orientierung der Marke auf dem Objekt bekannt, kann damit die Position und Orientierung des Objekts zur Kamera festgestellt werden. Da für das Robotersystem bereits Referenzmarken genutzt werden, bietet sich deren Verwendung an. Deshalb wurde im Rahmen dieser Arbeit die Erkennung von Objekten über das Referenzmarkensystem AprilTag realisiert. Es konnte hier bereits auf Erfahrungen mit dem Roboter zurückgegriffen werden, zusätzlich waren die Grundfunktionen zur Erkennung von Objekten durch diese Referenzmarken bereits in den Funktionen des Robotersystems implementiert.

Generell werden als Lademöglichkeiten induktives Laden oder Laden über Schleifkontakte für autonome Systeme genutzt. Der Aufbau des Roboters lässt kein induktives Laden zu und auch nicht die Nachrüstung eines solchen Systems. Laden über Schleifkontakte wäre möglich jedoch müsste der Roboter hierfür millimetergenau navigiert werden. Das ist mit der momentanen Navigationsgenauigkeit des Roboter nicht realistisch. Um eine möglichst reibungslose Ladeverbindung zu gewährleisten und die Navigationsungenauigkeiten auszugleichen, wurde deshalb ein Magnetsteckersystem gewählt. Da autonomes Laden über eine Steckverbindung nicht üblich ist, wurden einige Versuche mit verschiedenen Konzepten durchgeführt. Hierbei wurden unterschiedliche Lagerungen des Steckers miteinander verglichen. Das Augenmerk hierbei lag auf der möglichst guten Kompensation von Anfahrtswinkeln und Offsets. Ein an Schnüren hängender Magnetstecker konnte den größten Offset in Translation und Rotation ausgleichen. Ergebnis aller Vorversuche und Untersuchungen ist eine Ladestation, welche über einen von einem Galgen herabhängenden Stecker verfügt sowie über einen Apriltag als Navigationshilfe.

## 2.2. Posen

Im Folgenden wird der Grundbegriff der Pose anhand der Quellen [7], [22] und [23] erklärt.

Eine Pose beschreibt die Position und Orientierung zweier Objekte und deren körperfesten Koordinatensysteme zueinander und ermöglicht z.B. die Umrechnung eines Punktes von einem in ein anderes Koordinatensystem. In der vorliegenden Arbeit werden ausschließlich Posen im dreidimensionalen Raum genutzt, um die Position und die Orientierung von Objekten und deren Koordinatensystemen im Roboterkoordinatensystem darzustellen. Um also die Lage eines Objektes B aus Sicht eines Objektes A zu bestimmen, werden Objekte durch körperfeste Objektkoordinatensysteme repräsentiert. Hierbei ist der Ursprung und die Orientierung frei wählbar. Die Position und Orientierung des Objektes B bezüglich eines Objektes A wird durch eine Transformation beschrieben, welche das Koordinatensystem A in das Koordinatensystem B überführt. Für die Anordnung eines Koordinatensystems innerhalb eines anderen gibt es sechs Freiheitsgrade: drei translatorische entlang und drei rotatorische um die Koordinatenachsen des Koordinatensystems. Eine Pose wird mathematisch durch eine Transformationsmatrix beschrieben; diese besteht aus einer Rotationsmatrix und einem Translationsvektor.

Die Position des Ursprungs von Objekt B innerhalb des Koordinatensystems von A kann durch einen dreidimensionalen Vektors ${}_At^B$ dargestellt werden (siehe Abbildung 2.1). Dieser beinhaltet die Translation entlang der drei Achsen des Weltkoordinatensystems in der Reihenfolge xyz.

$$ {}_At^B = \begin{pmatrix} x \\ y \\ z \end{pmatrix} $$

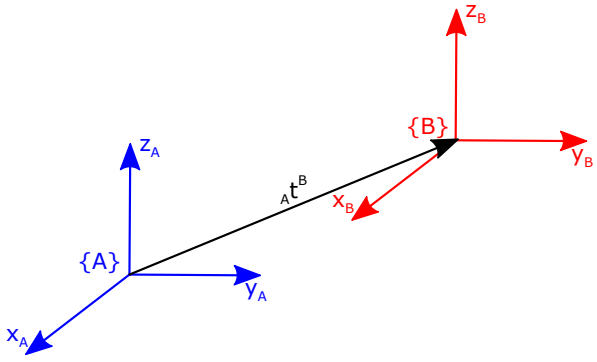


Abbildung 2.1.: Darstellung der Translation des Koordinatensystems B in Bezug auf das Koordinatensystem A

Die Rotationen um die Achsen des Koordinatensystems des Objekts A werden mit Hilfe von Rotationsmatrizen beschrieben. Das Produkt der einzelnen Rotationen (siehe Abbildung 2.2) bestimmt die Gesamtrotation ${}_AR^B$ beziehungsweise die Orientierung des Objekt B in Bezug auf das Objekt A.

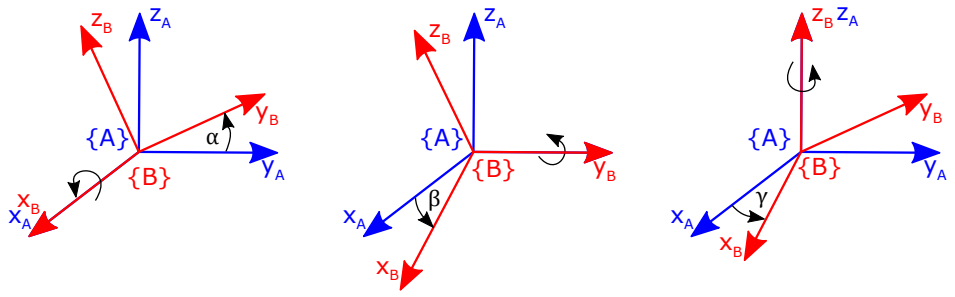


Abbildung 2.2.: Darstellung der einzelnen Rotationen eines Koordinatensystems um die x-Achse (rechts), y-Achse (Mitte) und z-Achse (links)

Die Matrix $R_x(\alpha)$ stellt die Rotation um die x-Achse eines Koordinatensystems mit dem Winkel $\alpha$ dar, dies wird auch Rollen genannt.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix}$$

Die Rotation um die y-Achse mit dem Winkel $\beta$, auch Neigen genannt, wird mithilfe der Matrix $R_y(\beta)$ beschrieben.

$$R_y(\beta) = \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix}$$

Als Gieren wird die Rotation um die z-Achse mit dem Winkel $\gamma$ bezeichnet; dies wird durch die Matrix $R_z(\gamma)$ abgebildet.

$$R_z(\gamma) = \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Zu beachten ist, dass die Drehrichtung der Winkel in positiver Drehsinn beziehungsweise gegen den Uhrzeigersinn definiert ist (siehe Abbildung 2.3).

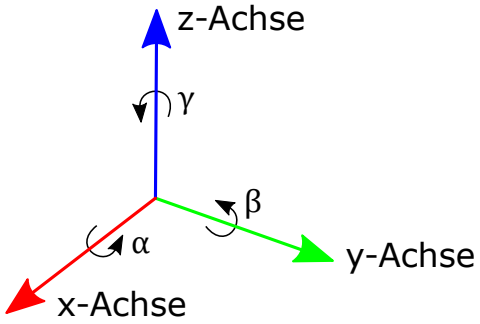


Abbildung 2.3.: Darstellung der Koordinatenachsen und Definition der Rotationsrichtungen

Die Multiplikationsreihenfolge der Rotationsmatrizen ist von essenzieller Bedeutung und wird durch verschiedene Konventionen festgelegt; unterschiedliche Reihenfolgen der Faktoren ergeben ein unterschiedliches Produkt. Man spricht auch von einer Multiplikation, bei der keine Kommutativität besteht. In dieser Arbeit findet die Euler-Konvention, bei der die Reihenfolge der Rotationen auf ZYX festgelegt ist, Anwendung. Dies bedeutet, dass bei der ersten Rotation mit dem Winkel $\gamma$ um die z-Achse rotiert wird; daraufhin um die y-Achse mit dem Winkel $\beta$ und zuletzt um die x-Achse mit dem Winkel $\alpha$. Somit lässt sich die Gesamtrotation in Euler-Konvention in folgender Rotationsmatrix darstellen.

$$ {}_AR^B_{zyx}(\gamma,\beta,\alpha) = R_z(\gamma) * R_y(\beta) * R_x(\alpha) $$

$$ = \begin{pmatrix} cos(\beta)cos(\gamma) & sin(\alpha)sin(\beta)cos(\gamma) - cos(\alpha)sin(\gamma) & cos(\alpha)sin(\beta)cos(\gamma) + sin(\alpha)sin(\gamma) \\ cos(\beta)sin(\gamma) & sin(\alpha)sin(\beta)sin(\gamma) + cos(\alpha)cos(\gamma) & cos(\alpha)sin(\beta)sin(\gamma) - sin(\alpha)cos(\gamma) \\ -sin(\beta) & sin(\alpha)cos(\beta) & cos(\alpha)cos(\beta) \end{pmatrix} $$

Die Transformationsmatrix ${}_AT^B$ beschreibt die Beziehung der Objekte A und B durch die Kombination von Rotation und Translation in einer 4x4-Matrix. Die Rotationsmatrix r11 bis r33 und der Translationsvektor t1 bis t3 werden in der Matrix wie folgt angeordnet:

$$ {}_AT^B = \begin{pmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Die vierte Zeile gewährleistet die richtige Berechnung einer Transformationskette. Um eine Transformationsmatrix ${}_AT^C$ über das Koordinatensystem B zu berechnen (siehe Abbildung 2.4) werden die beiden Transformationsmatrizen ${}_AT^B$ und ${}_BT^C$ multipliziert. Die gewählte Besetzung der vierten Matrizenzeile (0 0 0 1) führt bei der Multiplikation der Transformationsmatrizen zu einer Multiplikation deren Rotationsmatrizen und einer Addition der Translationsvektoren.

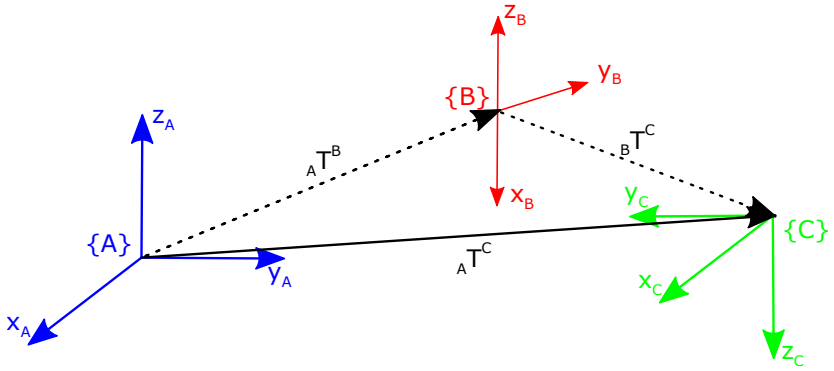


Abbildung 2.4.: Darstellung einer Kette von Transformationen

$$ {}_AT^C =_A T^B *_B T^C $$

Bei einer gegebenen Transformationsmatrix können mit Hilfe des Rotationsteils der Matrix die einzelnen Winkel $\alpha$, $\beta$, $\gamma$ nach den folgenden Formeln berechnet werden. Dies ist möglich unter Verwendung des atan2 und des trigonometrischen Pythagoras unter der Voraussetzung, dass $\beta$ nicht gleich $\pm 90°$ ist.

$$ \beta = atan2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}) $$
$$ \gamma = atan2(r_{21}/cos(\beta), r_{11}/cos(\beta)) $$
$$ \alpha = atan2(r_{32}/cos(\beta), r_{33}/cos(\beta)) $$

Es ist also möglich den Bezug zwischen zwei Objekten sowohl mit Hilfe der Winkel in einer Rotationsmatrix zu beschreiben als auch von der Matrix auf die rotierten Winkel zurückzurechnen.

## 2.3. AprilTags

Dieses Kapitel erklärt die Grundlagen des Referenzmarkensystems AprilTag. Die folgenden Erklärungen orientieren sich an der Quelle [6].

AprilTags sind ein von dem April Robotik Labor der Universität Michigan entwickeltes Referenzmarkensystem. Die Referenzmarken wurden für Agumented Reality Anwendungen, Anwendungen in der Robotik und zur Kamerakalibrierung entwickelt.

### 2.3.1. Referenzmarke

Bei AprilTags handelt es sich um einen zweidimensionalen Matrixcode, welcher je nach Darstellungsart bzw. Familie, eine unterschiedliche Datenmenge codiert. Zudem sind sie auf eine hohe Lokalisationsgenauigkeit ausgelegt. Jede Marke besteht aus einem schwarzen Rahmen, der von einem weißen eingefasst ist (siehe Abbildung 2.5). (Um in der Abbildung den weißen Rahmen des AprilTags sichtbar zu machen, wurde ein dünner schwarzer

Rahmen eingefügt.) Innerhalb des schwarzen Rahmens sind die Informationen mithilfe von schwarzen und weißen Quadraten codiert. Die Breite des schwarzen Rahmens entspricht der Kantenlänge der einzelnen Quadrate. Durch das Muster der schwarzen und weißen Quadrate werden von 0 aufsteigende Werte, sogenannte IDs, codiert. Die Anzahl der IDs wird durch die Familie der AprilTags bestimmt. Diese legt fest, aus wie vielen Quadraten der Tag besteht und welche Hamming-Distanz zwischen zwei IDs liegt.

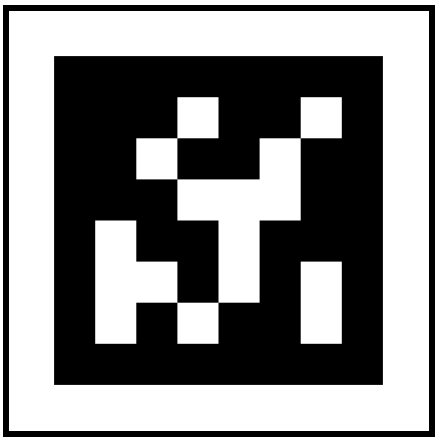

Abbildung 2.5.: Abbildung eines AprilTag der Familie 36h11 mit der ID 525

Die Hamming-Distanz beschreibt die Anzahl der unterschiedlich besetzten Stellen eines Codes auf einen Bezugswert. Je höher die Hamming-Distanz, desto leichter können Fehler im Code erkannt und korrigiert werden. Ein hoher Hamming-Abstand minimiert die Fehlinterpretationswahrscheinlichkeit eines Wertes. Am Beispiel der in Binärcode umgewandelten Dezimalwerte 0 bis 7 wird die Hamming-Distanz in der Tabelle 2.1 erklärt. Hierbei bezieht sich die Hemming-Distanz auf den Binärwert der Dezimalzahl 0.

| Dezimaler Wert | Binärer Wert | Hamming-Distanz |
|---|---|---|
| 0 | 000 | (0) |
| 1 | 00<u>1</u> | 1 |
| 2 | 0<u>1</u>0 | 1 |
| 3 | 0<u>11</u> | 2 |
| 4 | <u>1</u>00 | 1 |
| 5 | <u>1</u>0<u>1</u> | 2 |
| 6 | <u>11</u>0 | 2 |
| 7 | <u>111</u> | 3 |

Tabelle 2.1.: Darstellung der Hamming-Distanz anhand des Binärcodes der Dezimalzahlen 0-7 mit Bezug auf den Binärwert 000

Oft genutzte AprilTag-Familien sind 25h9 und 36h11. Die Zahl vor dem "h" bestimmt die Anzahl der Quadrate, aus der ein einzelner Tag aufgebaut ist. Die Zahl hinter dem "h" bestimmt die Hamming-Distanz. So umfasst die Tag-Familie 36h11 insgesamt 587 IDs und ist mit einer sehr hohen Hamming-Distanz von 11 besonders sicher gegen Verwechslung. Dies war das entscheidende Kriterium zur Nutzung dieser Familie für diese Arbeit.

### 2.3.2. Detektor

Bei dem AprilTag-Detektor handelt es sich um ein Programm, mit welchem aus Kamerabildern die Position und die Orientierung sowie die ID aller darauf abgebildeter AprilTags ermittelt werden kann. Durch den weißen und schwarzen Rahmen der Tags hebt sich der Tag aus der natürlichen Umgebung ab; dies erleichtert das Auffinden des Tags für das Programm. Der Erkennungsprozess beinhaltet mehrere Schritte: die Suche nach linearen Segmenten, die Erkennung von Quadraten, die Berechnung der Position und der Ausrichtung des Tags und die Dekodierung des Barcodes.

#### Erkennung von Liniensegmenten

Um Apriltags auf einem Bild zu detektieren wird zunächst nach Linien gesucht. Dies wird durch Berechnung der Richtung und des Betrags der Gradienten an jedem Pixel des Bildes bewerkstelligt. Daraufhin werden die Pixel mit ähnlicher Gradientenrichtung und -stärke zu Kanten zusammengefasst und erhalten eine Gewichtung. In einem weiteren Schritt werden diese nach ihrem Gewicht sortiert.

#### Erkennung von Quadraten

Als nächstes wird nach einer Reihe von Liniensegmenten gesucht, die eine 4-seitige Form, d.h. ein Parallelogramm, bilden. Hierfür wird eine rekursive Tiefensuche (depth-first search (DFS)) mit einer Tiefe von vier genutzt. Jede Ebene des Suchbaums fügt eine Kante des Quadrates hinzu. Auf der ersten Ebene werden alle Liniensegmente berücksichtigt. Auf den Ebenen zwei bis vier werden alle Liniensegmente berücksichtigt, die "nahe genug" an der Stelle beginnen, an der das vorherige Liniensegment endet. Sobald vier Kanten gefunden wurden, welche ein Viereck bilden, werden die Ecken des Vierecks durch Pixelanpassung vervollständigt.

#### Berechnung der Position und Ausrichtung des Tags

Mit der gegebenen physikalischen Größe des Apriltags und der Kamerabrennweite kann durch Direkte Lineare Transformation (DLT) die Pose des Apriltags in Bezug auf die Kamerakoordinaten berechnet werden. Die DLT kann auch genutzt werden, um die Pose einer kalibrierten Kamera zum AprilTag durch mindestens drei Punkte oder einer nicht kalibrierten durch sechs Punkte zu berechnen. Zu diesen auf der Bildebene liegenden Punkten muss die reale Lage in Form von XYZ-Koordinaten bekannt sein. Nun kann durch die folgende Gleichung die Matrix P berechnet werden, welche die Projektion der 3D-Koordinaten auf die 2D-Bildkoordinaten beschreibt.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = P * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Die (3x4) Matrix P beschreibt die intrinsischen Parameter der Kamera und die Pose zum Apriltag. Da die intrinsischen Informationen im Fall des Roboters durch die Kalibrierung der Kamera bekannt sind, können die restlichen sechs Parameter (Pose) durch die Lösung eines

homogenen linearen Gleichungssystems mit den Werten der drei bekannten Punkte berechnet werden. Drei Punkte mit jeweils zwei Dimensionen ergeben sechs Gleichungssysteme, durch die die sechs Parameter der Pose bestimmt werden. Bei der Erkennung von Apriltags werden Punkte auf dem schon detektierten Quadrat genutzt. Die äquivalenten dreidimensionalen Werte können über die Größe des Tags erschlossen werden. Die aus der direkten linearen Transformation resultierende Pose beschreibt die Orientierung und die Position des Tag-Mittelpunktes in Kamerakoordinaten.

Zu beachten ist die Orientierung des Koordinatensystems des AprilTags und der Kamera; dies ist besonders wichtig bei der weiteren Verarbeitung. Der Ursprungspunkt des Kamerakoordinatensystems liegt in der Mitte der Kameralinse. Die z-Achse zeigt von der Kameramitte gerade aus dem Kameraobjektiv. Die x-Achse zeigt aus Kameraperspektive nach rechts und die y-Achse nach unten. Der Ursprungspunkt des Tags befindet sich zentriert in dessen Mitte. Wobei die x-Achse nach rechts, die y-Achse nach unten und die z-Achse in den AprilTag hinein zeigt.

#### Decodierung der Nutzlast

Bei der Decodierung der Nutzlast besteht die Aufgabe darin, die Bits aus dem Nutzdatenfeld zu lesen. Hierzu werden die Koordinaten jedes Feldes in Apriltag-Koordinaten anhand des erkannten Rahmens berechnet. Als Referenzwerte für Weiß- und Schwarzwerte der Quadrate wird der helle und der dunkle Rahmen des Tags genutzt.

#### Library

Für die Detektion von AprilTags kann eine C-Library in Python, C, Matlab und OpenCV eingebunden werden. Diese stellt die Funktion des Apriltag Detektors zur Einbindung in den Programmcode zur Verfügung. Für die Nutzung wird lediglich die Tagfamilie, die Taggröße in Meter, die Brennweite der Kamera sowie deren Fokuszentrum benötigt. Bezogen werden kann die Library aus dem folgenden github repository [20].

## 2.4. Kamera Kalibrierung

Für die Detektion von AprilTags und zur weiteren Nutzung dieser Daten werden die intrinsischen und extrinsischen Parameter der aufnehmenden Kamera benötigt. Bei der intrinsischen Kalibrierung werden Parameter, wie z.B. die Brennweite und Pixelgröße der Kamera, bestimmt. Diese weichen durch Ungenauigkeiten in der Produktion von Kamera zu Kamera des gleichen Models voneinander ab. Theoretisch müssten diese nur ein einziges Mal bestimmt werden, da sich diese Werte nicht verändern. Da die Kalibrierung jedoch nur eine "genaue Schätzung" der Parameter darstellt, werden diese im DLR bei jeder Kalibrierung erneut mitbestimmt. Die extrinsische Kalibrierung ermittelt die Position und Orientierung einer Kamera in Bezug auf einen Fixpunkt; im Falle des Roboters Justin zum Mittelpunkt der Plattform.

Zur Durchführung einer Kalibrierung wird eine Kalibrierplatte genutzt und Bilder dieser Aus

verschieden Ansichten benötigt. Eine solche Platte besteht aus einem Schachbrettmuster und dient als Orientierung. Da die Größe der schwarzen und weißen Quadrate bekannt ist, kann durch eine Software die Intrinsik und Extrinsik berechnet werden. Im Falle der Rollin' Justin Plattformkameras wird zur Kalibrierung dieser jeweils die Sicht der Plattformkamera und die Sicht der Kopfkamera genutzt. Bei der Kalibrierung der Kopfkamera können unterschiedliche Ansichten der Kalibrierplatte durch die Bewegung des Kopfes generiert werden. Dies ist bei den Plattformkameras nicht möglich da die genaue Position der zwei Sichten zueinander bekannt sein muss und nur der Kopf mit einer ausreichenden Genauigkeit bewegt werden kann. Die erstellten Bilder der Kalibrierplatte werden mit der eigens dafür entwickelten Software ausgewertet. Mithilfe des Programmes *CalDe* werden zunächst die Ecken des Schachbrettmusters auf dem Bild bestimmt. Anschließend werden mit der Software *CalLab* die Parameter der Kamera, aus der von *CalDe* erstellten Punktewolke, errechnet. Aus diesen Parametern wird eine Kalibrierungsdatei erstellt; für die hinteren Plattformkameras ist diese im Anhang A.2 zu finden.[18]

# 3. Umfeld der Arbeit

Im vorherigen Kapitel sind die Grundlagen der Lokalisation mittels des Refenzmarkensystems AprilTag erläutert worden. Nun wird näher auf die Laborumgebung, die Hard- und Software des Robotersystems Rollin' Justin und den Aufbau und die Funktion der zu lokalisierenden Ladestation eingegangen.

## 3.1. Marslabor

Für das Robotersystem Rollin' Justin gibt es mehrere Labore / Umgebungen. Die in dieser Arbeit genutzte Umgebung repräsentiert eine Marslandschaft (siehe Abbildung 3.1). In dieser stehen ein Marslander und drei Smart Payload Units (SPUs). Die SPUs sind abstrahierte Module von Containern an welchen verschiedene Missionsaufgaben ausgeführt werden können. So verfügen zwei der SPUs über ein Solarpaneel, welches durch den Roboter gereinigt werden kann sowie Stecker und Kippschalter zur Simulation der Bedienung dieser. Zusätzlich können Module in die SPUs durch das Robotersystem eingebaut oder gewechselt werden.

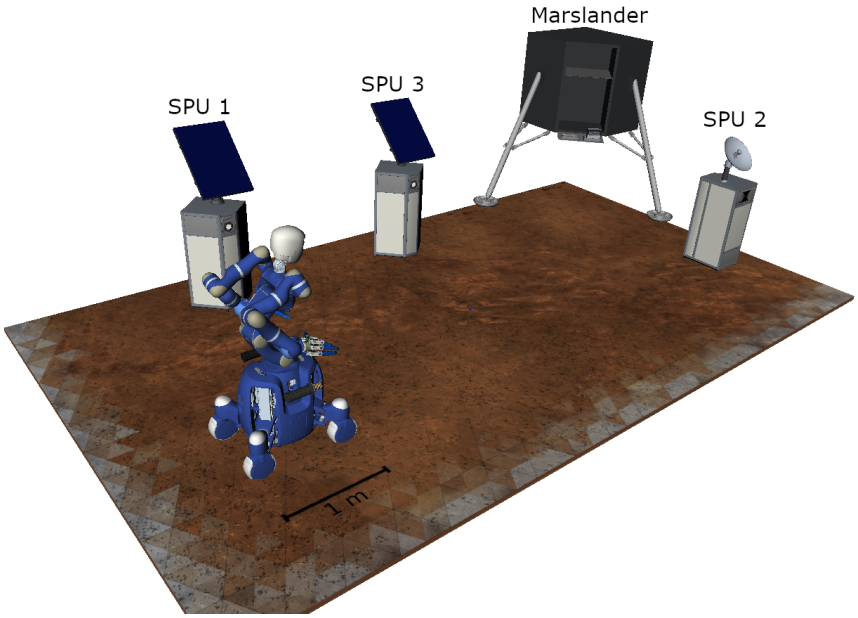


Abbildung 3.1.: Abbildung des Marslabors des Robotersystems Rollin' Justin (entnommen aus der digitalen Darstellung des Labors)

## 3.2. Robotersystem

Bei dem Robotersystem Rollin'Justin handelt es sich um ein humanoides System (siehe Abbildung 3.2), welches vom Deutschen Zentrum für Luft- und Raumfahrt für die Forschung im Bereich der Servicerobotik entwickelt wurde. Der Schwerpunkt des Forschungsgebietes liegt insbesondere auf der zweihändigen Manipulation und der Unterstützung des Menschen in alltäglichen Umgebungen und dem Weltraum. Der Roboter ist 2008 das erste Mal der Öffentlichkeit präsentiert. Da es sich bei dem Roboter um ein komplexes System aus eigenentwickelter Hard- und Software handelt, wird im Folgenden nur auf die relevanten Komponenten des Systems eingegangen.

### 3.2.1. Hardware

Die Hardware des Roboters kann in den Oberkörper und die Mobile Plattform unterteilt werden. Zusätzlich wird auf die Komponenten im Inneren der Plattform eingegangen.


Abbildung 3.2.: Das Robotersystem Rollin'Justin

### Oberkörper

Die mechanische Konstruktion des Oberkörpers ist so ausgelegt, dass das System in der Lage ist sowohl Objekte auf dem Boden als auch Objekte auf einem Regal bis zu einer Höhe von etwa 2m zu erreichen. Umgesetzt ist dies durch die Montage von zwei DLR Light Weight Robot III (DLR-LWRIII) und dem Hand-IIb System als Hand-Arm-System an einen Torso mit drei Freiheitsgraden. Bei dem DLR-LWRIII handelt es sich um die dritte Generation eines Leichtbauroboterarms, welcher die Funktionen eines menschlichen Arms durch sieben Freiheitsgrade nachstellt. Die DLR-Hand IIb ist die zweite Generation eines multisensoriellen Handmodells mit vier Fingern; ebenfalls entwickelt durch das Deutsches Zentrum für Luft- und Raumfahrt (DLR). Für weitere Informationen über DLR-LWRIII und DLR-HandIIb empfehlen sich folgende Quellen [17], [10]. Zusammengenommen verfügt der Oberkörper über 43 Freiheitsgrade: zwölf an jeder Hand, sieben an jedem Arm, drei durch den Torso und zwei im Hals. [3], [1]

### Plattform

Die fahrbare Plattform ermöglicht dem System die mobile Interaktion mit dem Menschen und bringt es einen großen Schritt näher an die Funktionen eines universell einsetzbaren Serviceroboters. Um die Stabilität des Systems bei Manipulationsaufgaben durch den Oberkörper zu gewährleisten, besitzt die Plattform vier einzeln ausfahrbare Beine. An jedem Bein ist ein drehbares omnidirektionales Rad montiert. Zusätzlich ist jedes Bein mit einem passiven Feder-Dämpfer-System ausgestattet, welches dem System ermöglicht Unebenheiten des Bodens auszugleichen und über kleine Hindernisse zu fahren. Im Inneren der Plattform befindet sich neben den Rechnern auch der Akku des Roboters. [3], [1]

### Stromversorgung

Das Robotersystem wird von einem eigens konzipierten Akku mit Strom versorgt. Dieser ist aus 48 Bleizellen des Modells Cyclon Type E zusammengesetzt, welche nach 2P24S, d.h. 24 Zellenpaare seriell, verschaltet sind. Geladen wird der gesamte Akku mit 30A bei 53V, in Summe also mit ca. 1,6kW von dem Labornetzteil EA-PS 9080-120 der Firma EA ELEKTRO-AUTOMATIK GMBH & CO. KG.

### Hannibal

Neben dem Akku befindet sich der Computer mit der Bezeichnung *hannibal* in der Plattform. Hierbei handelt es sich um ein Kontron mITX-KBL-H-CM238 Board mit dem Linux Betriebssystem debian9; dieses bildet das Gehirn des Robotersystems Rollin' Justin.
Das Kontron mITX-KBL-H-CM238 Board ist ein Mini-ITX Motherboard ausgestattet mit der 7. Generation an Intel-14NM QuadCore Prozessoren und vielen Anschlüssen (siehe Tabelle 3.1). Über einen Switch sind drei Jetson TX2 mit *hannibal* verbunden. [13]

| | |
|---|---|
| Prozessor | Intel® Xeon® E3, Core™ i7/i5/i3 |
| Arbeitsspeicher | 2x DDR4 SODIMM 2133 MHz (up to 32 GByte) |
| Lan-Anschlüsse | 4x Ethernet on front<br>IO Gigabit-LAN (10/100/1000MBit ), 2x optional |
| SATA-Anschlüsse | 4x SATA Gen 3.0 |
| USB-Anschlüsse | 2x USB 3.0 (Internal)+ 1x USB3.0 (Client)<br>+ 2x USB 3.0 (Rear)+ 2x USB 2.0 (Rear)<br>+ 2x USB 2.0 (Front) + 1x USB 2.0 (mPCIe) |
| Serielle-Anschlüsse | 1xRS232; 1xRS485 Half duplex /2 pairs RS42 |
| weiter Anschlüsse | 3x Display Port |
| Arbeitstemperatur | 0 °C to 60 °C |
| Stromversorgung | 12-24 V DC |

Tabelle 3.1.: Technische Spezifikationen Kontron mITX-KBL-H-CM238 [13]

### Jetsons

Der Jetson TX2 ist ein Einplatinencomputer von NVIDIA mit hoher Rechenleistung (siehe Tabelle 3.2). Im Vergleich zum NVIDIA Jetson Nano bietet dieser bis zu 2.5-mal so viel Leistung. Dies wird durch den NVIDIA Pascal™-Graphikprozessor mit 256 Kernen ermöglicht. Unterstützt wird die Graphics Processing Unit (GPU) von einem Dual-Core NVIDIA Denver 2 und einem Quad-Core Arm® Cortex®-A57 Prozessor. Im Robotersystem Rollin' Justin sind drei dieser Mini-Computer für die Übertragung und Verarbeitung von Livebildern der Kameras verantwortlich. [21]

| | |
|---|---|
| Grafikprozessor | NVIDIA Pascal™-Architektur mit 256 NVIDIA CUDA-Recheneinheiten |
| CPU | Dual-Core NVIDIA Denver 2 64-Bit CPU<br>ARM® Cortex®-A57 MPCore-Prozessorkomplex mit vier Kernen |
| Arbeitsspeicher | 8GB 128-bit LPDDR4 Memory<br>59.7 GB/s |
| Datenspeicher | 32GB eMMC 5.1 |
| Leistung | 7.5W/15W |

Tabelle 3.2.: Technische Spezifikationen NVIDIA Jetson TX2 [21]

Diese Einplatinencomputer sind in Trägerplatinen von AUVIDEA eingefasst. Die *Jetsons1* und *2*, die in der Plattform des Robotersystems verbaut sind, sind auf jeweils einer J120 Trägerplatine und der *Jetson3* ist im Kopf des Roboters auf der Nachfolgerplatine J140 montiert. Die Trägerplatine ermöglicht den Anschluss von Peripheriegeräten und die Anbindung an ein Netzwerk über USB, RJ45, HDMI und weitere Anschlüsse. In der nachfolgenden Tabelle 3.3 werden die Eigenschaften der Trägerplatinen J120 und J140 gegenübergestellt.

| Merkmal | J120 | J140 |
|---|---|---|
| HDMI out | mini | mini |
| USB 3.0 Typ A | 2 | 1 + USB2 |
| micro USB 2.0 | 1 | 1 |
| Wifi-Antennen | Ja | Ja |
| 10/100/1000 Ethernet (RJ45) | 1 | 2 |
| micro SD card | Ja | Ja |
| Lüfteranschluss | Ja | Ja |
| Schalter: power, reset | Ja | Ja |

Tabelle 3.3.: Vergleich der Trägerplatinen J120 und J140 von AUVIDEA

### Kameras

An die Jetsons sind die Kameras des Roboters angeschlossen. Das Robotersystem verfügt über fünf Kamerasysteme mit Tiefenfunktion und zwei Kameras in den Augen für menschenähnliche Stereosicht. Für die vorliegende Arbeit wird ausschließlich die Bilder der Tiefenkameras, ein Kamerasystem der Firma Intel RealSense mit der Bezeichnung Depth Camera D435, genutzt. Über jedem Bein der Roboterplattform ist eine dieser Kameras angebracht; die fünfte befindet sich am Kopf oberhalb der "Augen". Das System D435 verfügt über zwei Imager, einen Infrarot Projektor und ein RGB Modul. Das breite Sichtfeld der Kamera eignet sich perfekt für Anwendungen in der Robotik. [11]

### Netzwerkstruktur

Das Netzwerk des Roboters besteht aus drei Rechnern, drei Nvidia Jetson Modulen und zwei Switches (siehe Abbildung 3.3). Hierbei fungiert der Linux Hauptrechner *hannibal* als Dynamic Host Configuration Protocol (DHCP) Server; dieser ist über LAN mit dem Echtzeit Linux Rechner *Face* verbunden. Zur Fehlersuche sind beide Computer zusätzlich über eine serielle Verbindung vernetzt. Der QNX Rechner Ammit dient zur Steuerung der Hände. Zur Verarbeitung der Videostreams der fünf Kameras sind die Jetsons zuständig. Die Kameras der linken Roboterseite sind über USB-Schnittstellen an *Jetson1* angeschlossen. Die Trägerplatine J120 des Nvidia Jetson TX2 Moduls kann nur eine Kamera über eine USB3-Schnittstelle betreiben, deshalb ist der Videostream der hinteren Plattformkamera, welche über USB2 verbunden ist, auf ein Farb- oder ein Tiefenbild beschränkt. Von der vorderen Plattformkamera hingegen kann das gesamte Bildmaterial gleichzeitig abgerufen werden. Analog zu *Jetson1* sind die rechten Plattformkameras des Roboters mit *Jetson2* verbunden. Die am Kopf angebrachte Tiefenkamera sowie das Stereokamerasystem in den Augen ist an den, im Kopf angebrachten, *Jetson3* angebunden. Zentraler Knotenpunkt des Netzes ist Switch 1; dieser dient auch als Wireless Fidelity (Wifi) Accesspoint von außen. Der zweite Switch verbindet die drei Jetsons mit *hannibal*.

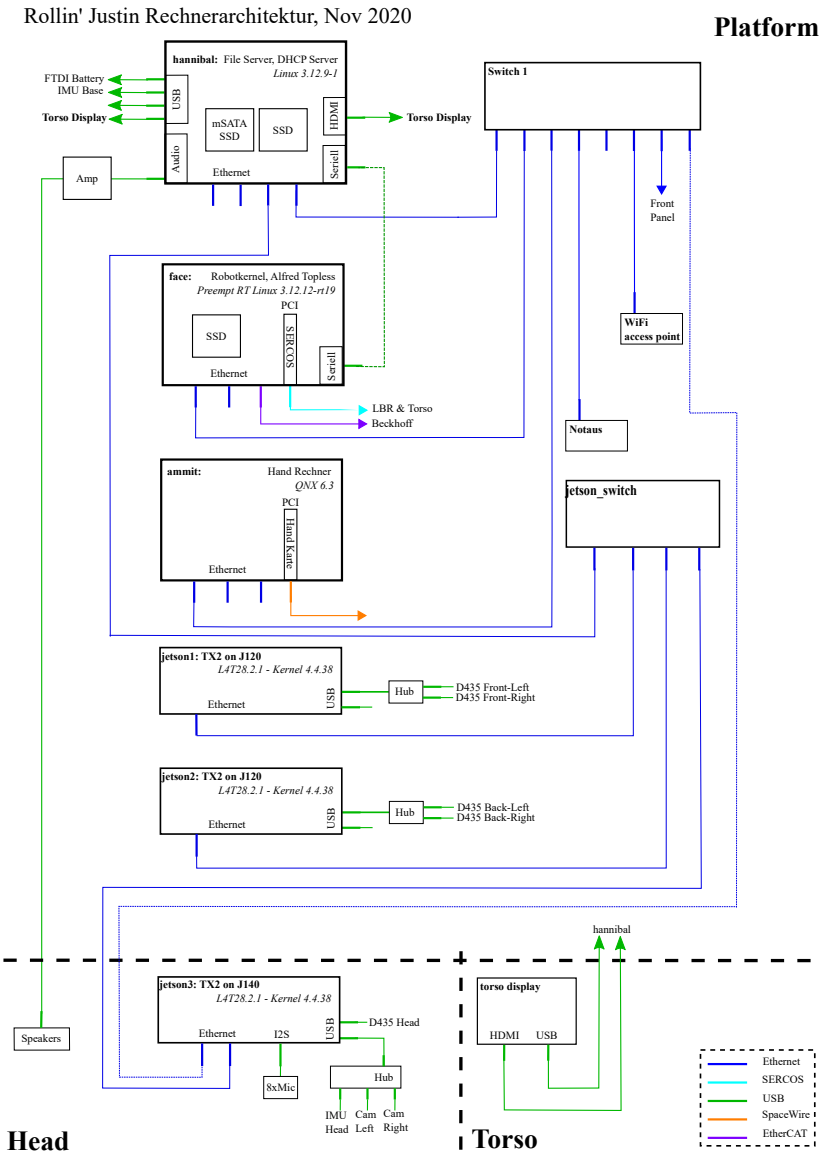


Abbildung 3.3.: Netzwerkarchitektur des Robotersystems Rollin'Justin [2]

### 3.2.2. Software

Die wichtigsten Softwarekomponenten des Rollin‘ Justin sind die Middleware *SensorNet* sowie *Links and Nodes*, die Ablaufsteuerung *Verbose2*, der Wissensspeicher *Objektdatenbank*, die Weltrepräsentation und der Ausführungsplaner *Hybrid Planer*. Da es sich bei all diesen Komponenten um Eigenentwicklungen handelt, welche nicht für den kommerziellen Gebrauch gedacht sind, beschränkt sich die Quelle dieser Informationen auf das interne Wiki des Instituts für Robotik und Mechatronik [5].

#### Middleware: SensorNet

Bei SensorNet handelt es sich um eine Bibliothek, welche einen Mechanismus zur Verteilung von großen Sensordatenströmen (z.B. Kamerabildern) bereitstellt. Die Besonderheit

besteht in der simultanen Verteilung von Sensordaten an unterschiedliche Anwendungen mit geringer Latenz und der Möglichkeit der Fernkonfiguration dieser Sensoren. Die Streamingdaten werden über eine Shared-Memory-Schnittstelle von einem Server den Anwendungen bereitgestellt. Dieser Shared-Memory befindet sich auf dem Arbeitsspeicher des jeweiligen Servers. Zudem können Konfigurationen an den Sensoren über den ConfigChannel mittels einer tcp/ip Remote-Schnittstelle vorgenommen werden.

### Middleware: Links and Nodes

Links and Nodes (LN) ist eine Middleware, welche es unterschiedlichen Prozessen erlaubt, miteinander zu kommunizieren; ähnlich wie Robot Operating System (ROS). Sie soll für einen organisierten Austausch von Daten sorgen, sowie eine klare Übersicht über alle laufende Systemmodule schaffen. Der Austausch der Daten wird mittels des Publish and Subscribe Models durchgeführt. Hierbei werden Informationen von sogenannten Publishern auf Links and Nodes bereitgestellt, welche wiederum von den Subscribern abgerufen werden können. Der LN-Manager dient hierbei als Vermittler zwischen Publisher und Subscriber und organisiert den Verbindungsaufbau um Daten zwischen diesen Parteien zu versenden.

### Ablaufsteuerung: Verbose2

Bei *Verbose2* handelt es sich um eine Ablaufsteuerung, mit welcher Python Skripte erstellt und miteinander verknüpft werden können. Neue Ideen können so zunächst in dieser Umgebung am Robotersystem getestet werden. Das Graphical User Interface (GUI) der Ablaufsteuerung besteht aus verschiedenen Seiten. Diese repräsentieren Projektordner oder Ordner von Mitarbeitern (siehe Abbildung 3.4). Dort werden die Skripte und Notebooks durch Fenster mit Button repräsentiert. Dies ermöglicht eine schnelle und einfache Ausführung und Verknüpfung von Skripten.

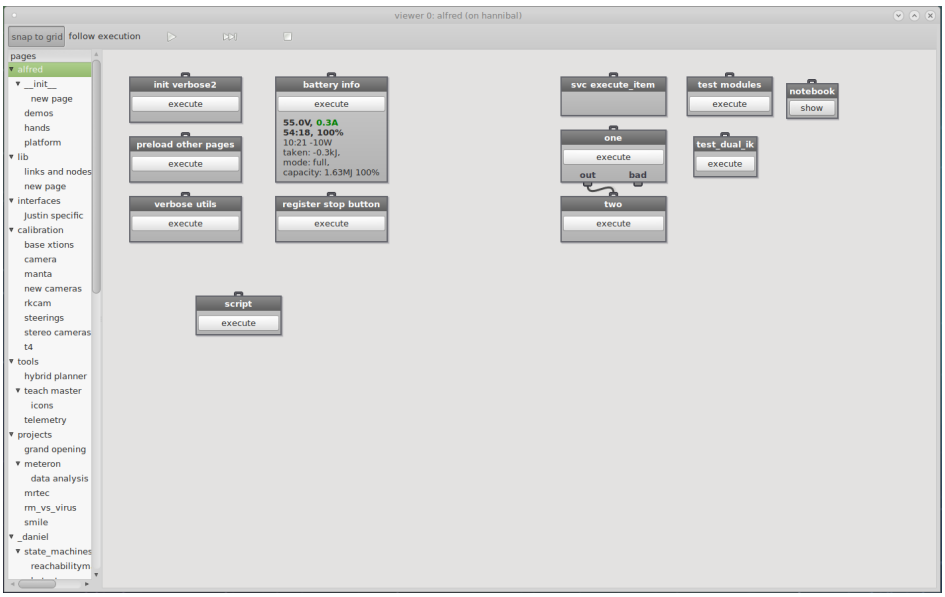


Abbildung 3.4.: Abbildung der Benutzeroberfläche der Ablaufsteuerung Verbose2

Um das Testen von Ideen noch einfacher zu gestalten, sind sogenannte Interfaces in *Ver-*

*bose2* implementiert. Dabei handelt es sich um Python Skripte, die eine Klasse beinhalten. Diese stellt je nach Thema unterschiedliche Funktionen zur Verfügung, welche die Nutzung verschiedener Hardware erleichtert. So wird z.B in *platform_interface.py* die Funktion *cartesian_move( x, y, rot)* bereitgestellt (unter Angabe von x und y in Meter und des Rotationswinkels rot in rad), welche es ermöglicht verschiedene Vektoren sowie Rotationen mit der Plattform anzufahren. Im Hintergrund wird hierfür die Geschwindigkeit, die Beschleunigung und vieles mehr gesetzt und die benötigte Ausrichtung der Räder bestimmt. Für das Andockmanöver wird die Funktion cartesian_move_slow(x, y, rot) in das Plattforminterface hinzugefügt (siehe Listing 3.1); diese ermöglicht ein langsameres und damit genaueres Navigieren der Plattform.

Listing 3.1: cartesian_move_slow

```
def cartesian_move_slow(self, x=0., y=0., rot=0.):
       app.glob.qsv_ipol.ipol_platform([x,y,rot],
       platformTransMaxVel=0.1, # [m/s]
       platformTransMaxAcc=0.25, # [m/s**2]
       platformTransMaxJerk=2.50, # [m/s**3]
       platformTransMaxDev=0.01) # [m/s]
```

### Wissensspeicher: Objektdatenbank

Bei der Objektdatenbank (odb) handelt es sich nicht um eine klassische Datenbank (wie z.B. MySQL usw.), da Objektinformationen mit einem strukturierten Dateisystem organisiert werden. Der Gedanke dahinter ist die Organisation und Verknüpfung von Objekten anhand deren Funktionalität. Das Dateisystem befindet sich auf dem Roboter und beinhaltet alle Informationen über die Objekte und deren Manipulationsmöglichkeiten, mit denen der Roboter interagieren kann. Objekte der gleichen Klasse teilen sich die gleiche Funktionalität und können deshalb unter Berücksichtigung ihrer jeweils spezifischen Eigenschaften wie z.B. Abmessungen in gleicher Art manipuliert werden.
Für jedes Objekt wird im Ordner odb des Roboters eine feste Ordnerstruktur angelegt. Die Abbildung 3.6 zeigt diese Ordnerstruktur für die Ladestation.
Der Ordner *geometry* enthält unterschiedliche 3D-Modelle des angelegten Objektes für verschiedene Anwendungen. Für die Ladestation ist ein Modell für die Visualisierung Abbildung 3.5 in der virtuellen Welt des Roboters und ein vereinfachtes Modell zur Kollisionsvermeidung Abbildung 3.5 als .obj-Datei abgelegt.

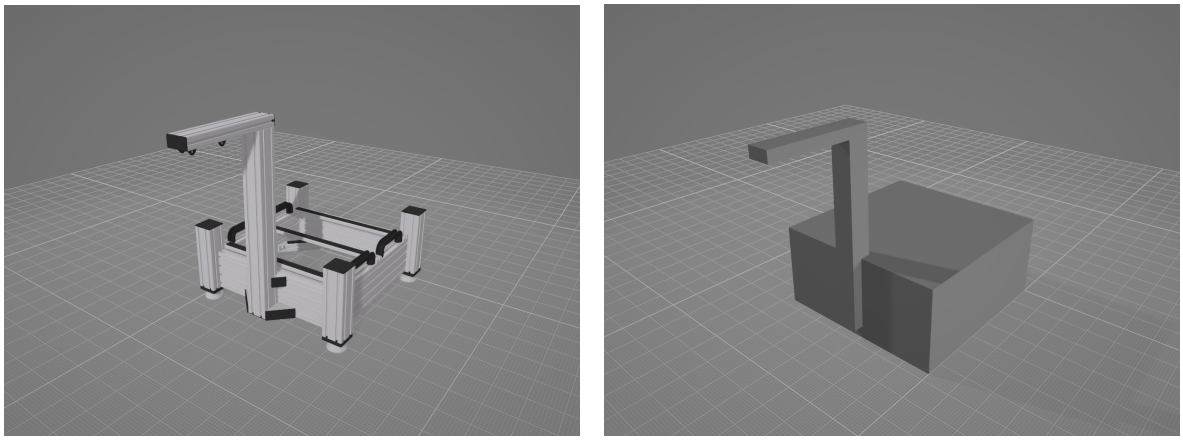

Abbildung 3.5.: Abbildung des CAD-Modells der Ladestation zur Visualisierung (links) und Kollisionsvermeidung (rechts)

Im Ordner *logic* sind Skripte hinterlegt, welche Informationen für den hybriden Planungsalgorithmus zu Interaktionsmöglichkeiten des Roboters mit dem Objekt bereitstellen.
Im Ordner *vision* sind die Referenzmarken des Objekts definiert, die zur Lokalisation des Objekts genutzt werden (Die Dateien der Referenzmarken der Ladestation befinden sich im Anhang A.3.1 und A.3.2). In einem Textdokument wird die einzigartige ID, die Familie, die Größe und die Position auf dem Objekt des Apriltags angegeben.
Im *Manifest*, einer Datei beschrieben in der Extensible Markup Language (XML), wird das Objekt, in diesem Fall die Ladestation, definiert und die Dateien der Ordner geometry, logic und vision hierarchisch strukturiert.(siehe Anhang A.3.8). Bei XML handelt es sich um eine Sprache, welche die Möglichkeit bietet hierarchische Strukturen von Dateien in einem Textdokument darzustellen. Die so beschriebenen Objekte werden z.B. in der Weltrepräsentation genutzt, um ein digitales Abbild der Laborumgebung zu schaffen.

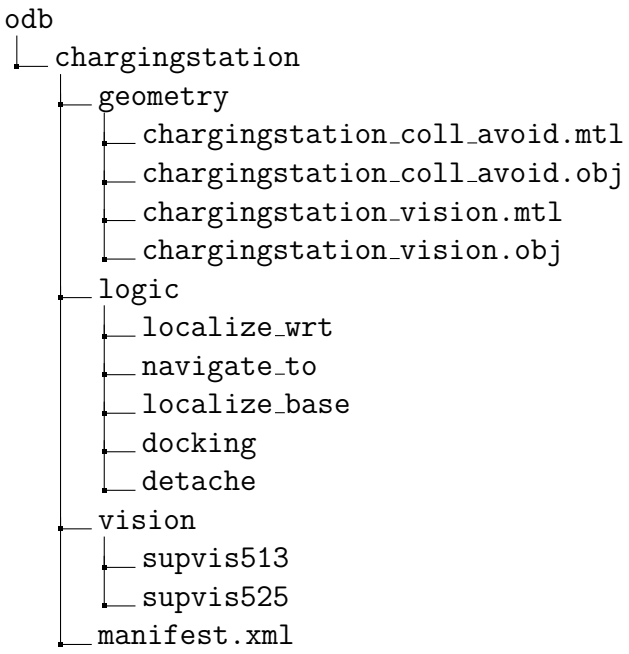


Abbildung 3.6.: Ordnerstruktur der Ladestation in der Objektdatenbank odb

### Weltrepräsentation

Die Weltrepräsentation ist ein Tool, das verschiedene digitale Zwillinge der Laborumgebungen, sogenannte Welten, beinhaltet. In diesen Welten werden Objekte der Objektdatenbank instantiiert; hier wird zwischen Objekten mit festem Platz im Raum, sogenannten Landmarks und beweglichen Objekte unterschieden. Landmarks sind in der virtuellen sowie in der realen Laborumgebung ortsfest und können deshalb zur Bestimmung der Position und Orientierung des Roboters genutzt werden. Für jedes Objekt wird eine Pose, also die zu erwartende Position und die Orientierung des Gegenstandes bezogen auf das Labor, festgelegt (siehe Abbildung 3.7). Die Pose von beweglichen Objekten wird bei deren Lokalisation in Bezug zum Labor aktualisiert. Verwendung findet die Weltrepräsentation unter anderem in der hybriden Planung von Aufgaben.

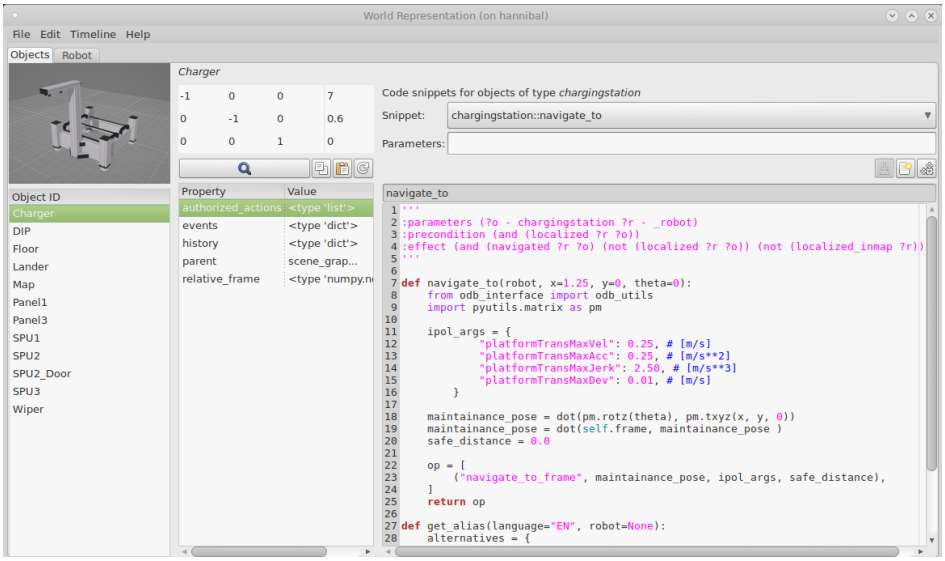


Abbildung 3.7.: Abbildung der Benutzeroberfläche der Weltrepräsentation

### Ausführungsplaner: hybrider Planer

Der hybride Planer ist das Herzstück der autonomen Funktionen des Roboters. Hierbei handelt es sich um einen Algorithmus zur Nachbildung der menschlichen Problemlösung von Manipulationsaufgaben.
Dieser ermöglicht dem Roboter autonom einen vordefinierten Zielzustand, welcher z.B in einem *Verbose2*-Skript gesetzt werden kann, einzunehmen.
Hierfür wird ein symbolischer und ein geometrischer Plan anhand des aktuellen Zustands der Objekte in der Weltrepräsentation (siehe Abbildung 3.8(unten links)) erstellt, simuliert und darauf umgesetzt. Der Weltrepräsentation werden Informationen über alle in der Welt instantiierten Objekte aus der Objektdatenbank (siehe Abbildung 3.8 oben links) übergeben. Die Informationen beinhalten z.B. geometrische Maße und Positionen der Referenzmarken und auch Aktionsvorlagen für jede mögliche Interaktion mit diesen Objekten, sogenannte Action Templates (siehe Abbildung 3.8(blau markiert)). Die Action Templates nutzt der hybride Planer zunächst um einen symbolischen Plan zum vorgegeben Zielzustand zu erstellen (siehe Abbildung 3.8 oben rechts) und daraufhin diesen unter anderem durch Bahnplanung in einen geometrischen Plan zu übersetzen (siehe Abbildung 3.8 unten rechts). Ist die Planung erfolgreich, wird der erstellte Plan ausgeführt und der Zustand der Objekte in der Weltrepräsentation aktualisiert. Hierbei überprüft der Roboter im Moment nicht die Ausführung des Plans auf Erfolg und Richtigkeit, jedoch wird derzeit eine Erweiterung entwickelt, welche dies ermöglichen soll.

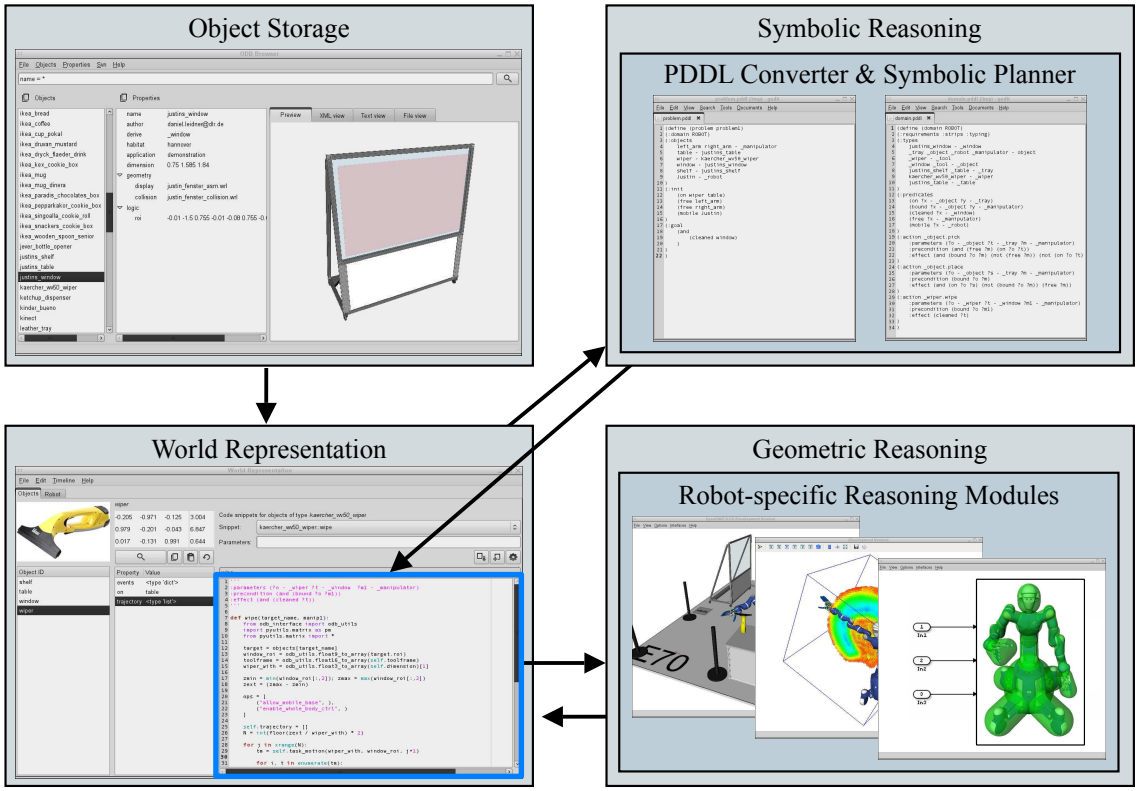


Abbildung 3.8.: Darstellung der Architektur des hybriden Planers [14]

Die in der odb angelegten Action Templates bestehen aus einem Kopf für die symbolische Planung und einer in Python verfassten Aktionsvorlage im Hauptteil für die geometrische Planung. Der symbolische Kopf ist in einer für den Einsatz mit KI entwickelte Planungssprache Planning Domain Definition Language (PDDL) verfasst und beschreibt die Parameter, die Vorbedingungen und den zu erwartenden Effekt der Interaktion. Als Parameter sind die an der Aktion beteiligten Objekte definiert. Als Vorbedingungen sind die Zustände, die zur Umsetzung der Aktion notwendig sind, mit sogenannten Prädikaten beschrieben. Die Prädikate beschreiben den Zielzustand vorheriger Aktionen. Über Prädikate wird auch der zu erwartende Effekt der Aktion ausgedrückt. Im Hauptteil des Action Templates wird in einer Funktion eine roboterunabhängige Operation parametrisiert, wie z.B. navigiere zu dieser Position oder lokalisiere dich. Diese Operationen sind in Interfaces definiert und sprechen wiederum die Module des tatsächlich planenden Roboters wie z.B. Bahnplaner, Dynamikplaner, Kameras und den AprilTag-Detektor an. Dieser modulare Aufbau ermöglicht es dem Programmierer, Code für verschiedene Aktionen wiederzuverwenden. Zusätzlich befindet sich die Funktion *get_alias* im Hauptteil jedes Action Templates, welche Phrasen in Englisch und Deutsch vorgibt, um mit der Sprachausgabe korrekte Sätze in Bezug auf die Aktion bilden zu können.

Mithilfe der Informationen aus den passenden Action Templates ist der hybride Planer in der Lage einen symbolischen Plan, bestehend aus einer Abfolge von Aktionen, welche zum gewünschten Zielzustand führen, zu erzeugen. Die symbolische Planung wird mittels des fast downward planning Systems durchgeführt (vgl. [9]). Im zweiten Schritt wird

versucht, die symbolische Planung in ein geometrisches Äquivalent zu übersetzen; dazu wird der Hauptteil der Action Templates verwendet. Für die Simulation der geometrischen Durchführbarkeit des symbolischen Planes können unter anderem Bahnplanungsmethoden genutzt werden. Diese sollen einen für das Robotersystem umsetzbaren, kollisionsfreien Weg zur Abarbeitung der Aktionen des symbolischen Plans finden. Beim Robotersystem Rollin' Justin übernimmt diese Aufgabe der OpenRAVE Motionplaner [4] entwickelt von Rosen Diankov an der Cernegie Mellon Universität in Pennsylvania. Falls eine Teilaktion nicht in der symbolisch geplanten Weise geometrisch simuliert werden kann, wird diese mit angepassten Parametern wiederholt (geometrisches Backtracking). Mögliche Fehlerquellen sind hierbei Kollisionen mit Hindernissen, fehlende Aufgabeninformationen oder unzureichender Zugang zu einem Objekt. Je nach Aktion sind unterschiedlich viele alternative Parameter in dessen Action Template definiert. So sind beispielsweise für das Greifen einer Flasche verschiedene Griffmuster und Winkel hinterlegt. Schlägt auch die Simulation mit veränderten Parameter fehl, kommt das symbolische Backtracking zum Einsatz; d.h. eine Überarbeitung des symbolischen Plans. Die schon angewandten Strategien werden für das Backtracking in der Objekthistorie gespeichert. Im Erfolgsfall wird dieser Plan auf dem realen Roboter ausgeführt und die geometrischen sowie symbolischen Effekte auf die Weltrepräsentation übertragen. [14],[16],[15]

Im folgenden wird der Ablauf eines hybriden Planungsvorgangs noch einmal anhand des in Abbildung 3.8 zu sehenden Beispiels erläutert. Die Aufgabe des Roboter besteht in der Reinigung eines Whiteboards mithilfe eines Fensterputzers, welcher auf einem Tisch liegt. Die Objektdatenbank (oben links) stellt Informationen für alle verfügbaren Objekte bereit. In diesem Beispiel sind dies der Fensterputzer, der Tisch und das Whiteboard. Die Weltdarstellung (unten links) übergibt den aktuellen Zustand der Umgebung des Roboters und die symbolischen und geometrischen Eigenschaften der darin befindlichen Objekte an den hybriden Planer. Übertragen in das Beispiel sind dies die Position, die Orientierung und die Interaktionsmöglichkeiten der Objekte, sowie die Position und die Orientierung des Roboters im Labor. Das Action Template (blau markiert) beinhaltet die symbolischen Aktionsdefinition in PDDL-Sprache (oben rechts) und die geometrischen Anweisungen in Form von ausführbarem Code. In der Aktionsdefinition sind die zur Erfüllung der Aufgabe benötigten Objekte und deren Funktion unter *:parameter* aufgelistet; in diesem Beispiel sind dies die Arme des Roboters als Manipulatoren, Justin als ausführender Roboter und die Objekte Tisch, Fensterputzer, Whiteboard (window) und ein Regal. Unter *:precondition* sind die initialen Begebenheiten aufgeführt. Für die Reinigung des Whiteboards liegt der Fensterputzer auf dem Tisch und die Hände des Roboters sind leer. Der erwünschte Zielzustand (geputztes Whiteboard) ist unter dem Punkt *:effect* aufgeführt. Aus diesen Informationen wird mittels des symbolischen Planers eine Ereigniskette erstellt, um den Zielzustand zu erreichen; es wird ausgehend vom gewünschten Zielzustand geplant. Übertragen in das Beispiel muss zum Putzen des Whiteboards der Fensterputzer über dessen Fläche bewegt werden; dafür muss zuvor zu diesem navigiert werden. Um den Putzvorgang ausführen zu können, sollte sich der Fensterputzer in einer der Hände des Roboters befinden. Hierfür muss der Fensterputzer vorher vom Tisch aufgenommen werden. Dies ist wiederum nur möglich, wenn zuvor zum Tisch navigiert worden ist. Hierdurch entsteht die folgende Aktionskette: Zum Tisch fahren, Fensterputzer in eine Hand nehmen, zum Whiteboard navigieren und putzen. Im nächsten Schritt werden mithilfe des Geometric Reasoning Moduls (OpenRAVE) die Na-

vigation oder die Bewegungsplanung für die symbolisch geplanten Aufgaben geometrisch simuliert. Hierbei werden einzelne Komponenten des Roboters adressiert, um die gestellten Aufgaben auf geometrischer Ebene zu lösen (unten rechts). In dem verwendeten Beispiel werden die Trajektorien zum Navigieren an den Tisch und zurück, die Bewegungen des Arms zum Aufnehmen des Fensterputzers unter Berücksichtigung der vorgegebenen Griffmuster und die Bewegungen des Fensterputzers über das Whiteboard mit vorgegebenen Wischmustern geplant. Bei erfolgreicher Simulation wird der geometrisch erstellte Plan in die Realität umgesetzt.[14]

#### Framework: AprilTagWrapper

Das Framework *AprilTagWrapper* ist vom DLR entwickelt; es stellt die Apriltag-Erkennung in den Umgebungen Objektdatenbank, Links and Nodes und SensorNet bereit. Das Framework umfasst mehrere Funktionen zur Detektierung von AprilTags aus einem Bild oder aus einem SensorNet Stream. Die unten aufgeführten Funktionen erstellen eine Liste aller erkannten AprilTags aus den zuvor genannten Quellen.
get_apriltag_detections_from_image
get_apriltag_detections_from_sn

Der Rückgabewert der folgenden Funktionen besteht aus einer Liste aller über einen Apriltag erkannten Objekte der Objektdatenbank und deren Position und Orientierung.
get_detected_objects_from_image
get_detected_objects_from_sn
[5]

## 3.3. Ladestation

Die Ladestation (siehe Abbildung 3.9) wurde als Machbarkeitsstudie für ein autonomes Laden des Robotersystems Rollin'Justin entwickelt. Der Aufbau und die Integration in das System ist im Rahmen der vorliegenden Bachelorarbeit durchgeführt worden.

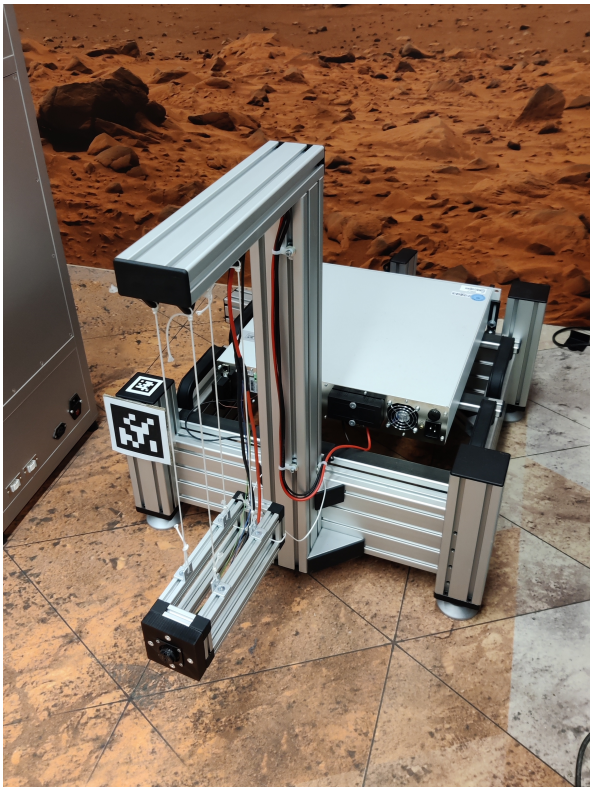

Abbildung 3.9.: Ladestation mit montiertem Labornetzgerät

### 3.3.1. Konzept

Ziel der Ladestation ist es einen unkomplizierten autonomen Ladevorgang zu gewährleisten und es der Roboterplattform zu ermöglichen völlig autonom heranzufahren, anzudocken und sich aufzuladen. Umgesetzt ist dies mithilfe eines freihängenden Steckerhalters und einem magnetischen Steckverbindungssystem. Der hängend gelagerte Stecker gleicht Ungenauigkeiten während der Anfahrt des Roboters in Rotation und Translation aus.

### 3.3.2. Aufbau der Ladestation

Die Ladestation verfügt über einen modularen Aufbau; dieser macht eine konstante Weiterentwicklung und die unkomplizierte Erweiterung der Funktionen möglich. Grundlegende Bausteine der Ladestation sind die Basis, der Galgen und die davon herabhängende Steckerhalterung.

Als Basis dient ein viereckiges Gestell; es ist zusammengesetzt aus Item Profilen. Es handelt sich um ein Baukastensystem der Firma Item Industriesysteme für konstruktive Aufgaben aus dem Maschinen- und Betriebsmittelbau. Die Form der Basis orientiert sich an den Abmessungen des Labornetzgerätes (EA-PS 9080-120), welches auf der Station angebracht ist. An der Basisfront ist der Galgen befestigt, von welchem die Steckerhalterung herabhängt; diese besteht ebenfalls aus Item Profilen und fasst den magnetischen Steckverbinder ein.

### 3.3.3. Steckverbindung

Die hängende Steckerhalterung kann beim Ansteckvorgang verschiedene Winkel und Offsets von bis zu 3 cm ausgleichen. Um den Stecker an der herabhängenden Steckerhalterung zu befestigen, ist dieser in ein 3D-gedrucktes Verbindungsstück eingelassen. So ist ein einfacher Wechsel der Steckverbindung gewährleistet. Ein Steckerwechsel ist vorgesehen sobald ein neuer Stecker mit weiteren Funktionen verfügbar ist.

Für die Montage der 3D gedruckten Halterung am Rahmen des Roboters (siehe Abbildung 3.10), musste die hintere Verkleidung der Roboterplattform entfernt werden. Da der Stecker für eine bessere Erreichbarkeit 5cm übersteht, kann diese im Moment nicht angebracht werden, da keine Öffnung für den Stecker existiert. Sobald der neue Stecker entwickelt und getestet ist, wird die Verkleidung des Roboters so angepasst, dass sie wieder montiert werden kann.

Abbildung 3.10.: Montierte Steckerhalterung am Robotersystem Rollin'Justin

Bei den an den Halterungen befestigten Steckern handelt es sich um einen Power-Data-Steckverbinder der Firma Rosenberger, welcher über eine magnetische Selbstfindung verfügt. Bei Annäherung des Steckers an sein Gegenstück wird dieser durch die magnetische Kraft angezogen und gleitet verpolungssicher in dieses. Dadurch ist eine korrekte Verbindung der sechs Pins des Steckers gewährleistet; vier Pins werden für die Datenkommunikation zwischen Akku und Ladegerät, die anderen zwei Pins zur Übertragung des Ladestroms verwendet. Die Datenkommunikation wird zur Regelung des Labornetzgerätes genutzt.

Bei der Entwicklung des neuen Steckers werden zusätzlich acht Pins für eine Ethernet-Übertragung eingeplant, um große Datenmengen nicht mehr kabellos übertragen zu müssen.

Auf der Roboterseite ist der Steckverbinder so angeschlossen, dass das Laden des Akkus sowohl manuell als auch über die Ladestation möglich ist.

### 3.3.4. AprilTags der Ladestation

Die Position und Orientierung der Ladestation soll mit AprilTags ermittelt werden. Hierfür ist ein großer und ein kleiner AprilTag an dieser montiert. Der Große Apriltag soll es ermöglichen die Ladestation aus weiter Entfernung zu erkennen und wird bei der Posenermittlung durch die Plattformkameras genutzt. Für eine optimale optische Erreichbarkeit ist er an der Sichtseite der Ladestation befestigt. Die genauere Bestimmung der Position dieses Tags folgt im Kapitel 4.2. Da sich der große Apriltag bei zu geringem Abstand des Roboters zur Ladestation nicht mehr im Sichtfeld der Kopfkamera befindet, ist der kleine AprilTag nach oben schauend auf dem vorderen rechten Pfosten der Ladestation angebracht. In der folgenden Tabelle sind weitere Informationen der AprilTags enthalten.

| Merkmal | kleiner Tag | großer Tag |
|---|---|---|
| ID | 513 | 525 |
| Familie | 36h11 | 36h11 |
| Größe in Meter | 0.036 | 0.1 |

Tabelle 3.4.: Spezifikationen der Apriltags der Ladestation

# 4. Entwicklung des autonomen Ladevorgangs

Das folgende Kapitel befasst sich mit der Ausarbeitung des Ladekonzepts, den nötigen Vorarbeiten sowie der Implementierung und der Integration dieses Konzeptes in die vorhandene Robotereinheit.

## 4.1. Konzept des Ladevorgangs

Die in den vorangegangenen Kapiteln angeführten Grundlagen werden nun zu einem Konzept zur Umsetzung der gestellten Aufgabe zusammengeführt. Dieses ist in fünf Schritte gegliedert: Erkennung der Ladestation, Routenplanung und Anfahren an die Ladestation, Berechnung des Steckerabstands, Andocken und Abdocken.

### 4.1.1. Erkennung der Ladestation

Der erste Schritt, um einen Ladevorgang starten zu können, ist die Lokalisation der Ladestation durch das Robotersystem. Dies wird mittels optischer Verfahren unter Zuhilfenahme von AprilTags und der Objektdatenbank realisiert. Hierfür werden verschiedene CAD-Modelle der Ladestation, sowie die genaue Position der AprilTags an der Ladestation benötigt. Zusätzlich kann dem Roboter eine Position hinterlegt werden, wo dieser die Ladestation im Labor vermuten soll.

### 4.1.2. Routenplanung und Anfahren an die Ladestation

Nach der Lokalisation der Ladestation durch das Robotersystem muss eine Route zur Zielposition der Roboterplattform vor der Ladestation geplant und abgefahren werden. Die angestrebte Zielstellung des Roboters ist rücklings parallel zur Ladestation, mit einer Entfernung von ca. 40 cm zwischen den Steckverbindern (vergleiche Abbildung 4.1). Steht der Roboter in dieser Zielstellung, kann das eigentliche Andockmanöver geplant und durchgeführt werden.

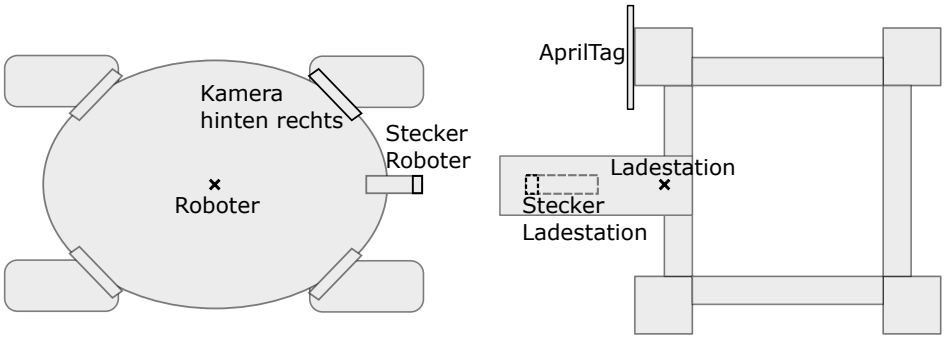


Abbildung 4.1.: Gewünschter Endzustand nach der Navigation an die Ladestation

### 4.1.3. Berechnung des Steckerabstands

Zur Planung des Andockmanövers wird zunächst der Abstand und der Rotationswinkel zwischen Roboter und Ladestation benötigt. Dies kann mithilfe der Transformationsmatrizen zwischen Stecker der Ladestation und Ladestation, zwischen Ladestation und AprilTag, zwischen Apriltag und Kamera des Roboters, zwischen Kamera des Roborters und Roboter und zwischen Roboter und Stecker des Roboters berechnet werden (siehe Abbildung 4.2). Über den AprilTag-Detektor wird die Pose der Ladestation in Kamerakoordinaten ermittelt. Die restlichen Transformationsmatrizen werden am realen Objekt vermessen.

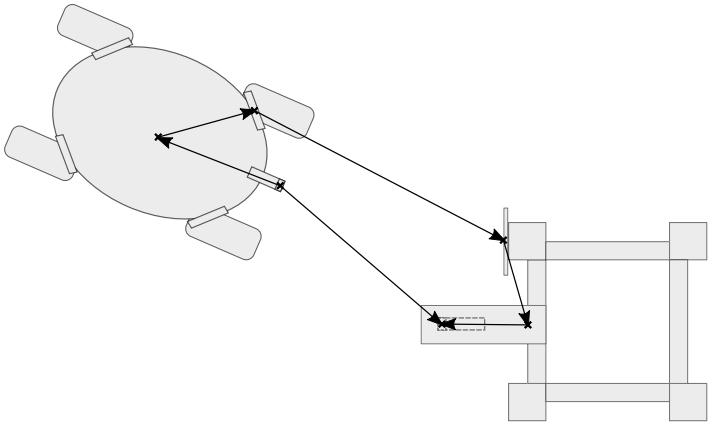

Abbildung 4.2.: Skizze der zu berechnenden Posen von Roboter zur Ladestation

### 4.1.4. Andockmanöver

Für einen erfolgreichen Ansteckvorgang muss der Roboter aus wenigen Zentimetern Entfernung präzise manövrieren. Ein erfolgreiches Andockmanöver ist definiert durch das Erreichen eines stabilen Ladevorgangs des Roboterakkus.

### 4.1.5. Abdocken

Der letzte Schritt eines erfolgreichen Ladevorgangs ist das automatische Lösen der Steckverbindung zwischen Ladestation und Roboter und die Anfahrt einer bestimmten, vorgegebenen Endposition.

## 4.2. Vorbereitungen

Vorbereitende Arbeiten zur Implementierung des Ladevorgang sind die Platzierung der Apriltags durch Bestimmung des Robotersichtfeldes und die Ermittlung der Posen der beim Andockprozess beteiligten Komponenten.

### 4.2.1. Sichtfeld des Roboters

Um eine erste Vorstellung vom Sichtbereich der Roboterplattformkameras zu bekommen, wurde eine Berechnung mit dem Programm *MATLAB R2021b* des Herstellers MathWorks durchgeführt; dazu wurden die extrinsischen und intrinsischen Parameter der Kameras genutzt. Besonderes Augenmerk lag dabei auf der Ermittlung der Schnittgeraden (siehe Abbildung 4.3 grüne Linien) von Sichtfeld und Bodenebene; diese Linien trennen den sichtbaren vom nichtsichtbaren Bereich auf Bodenhöhe. Bei der Berechnung stellte sich heraus, dass die Kamerasichtfelder bis zu diesem Punkt noch keine Überschneidung aufweisen.

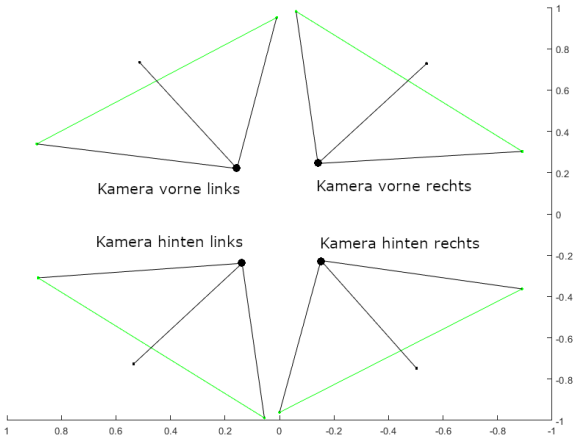


Abbildung 4.3.: Mit MATLAB berechnetes Sichtfeld der Plattformkameras des Robotersystems Rollin' Justin

Um eine genauere Abschätzung des Sichtfeldes zu erlangen, wurde der sichtbare Bereich des Bodens hinter dem Roboter experimentell ermittelt. Hierzu wurde der auf den Livebildern der hinteren Kameras zu sehende Laborboden mit einer Schnur markiert (siehe Abbildung 4.4).

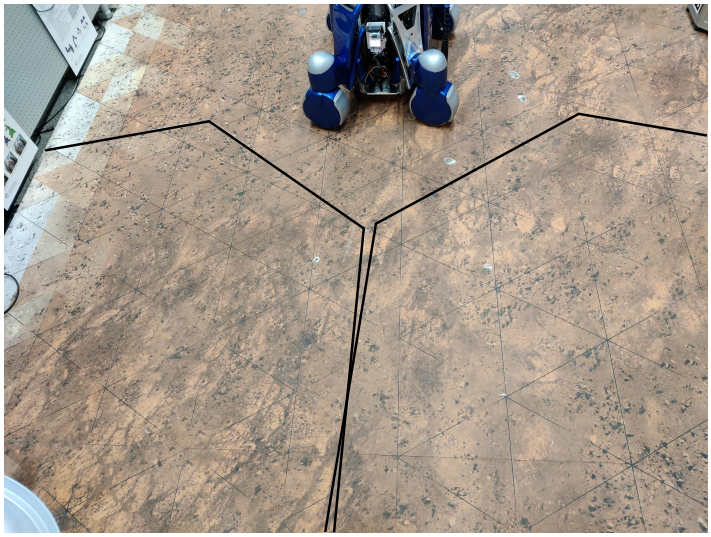

Abbildung 4.4.: Experimentell bestimmtes Sichtfeld der hinteren beiden Plattformkameras (Markierungen Verstärkt)

Die entstandene Fläche wurde ausgemessen und in folgende Grafik übertragen.

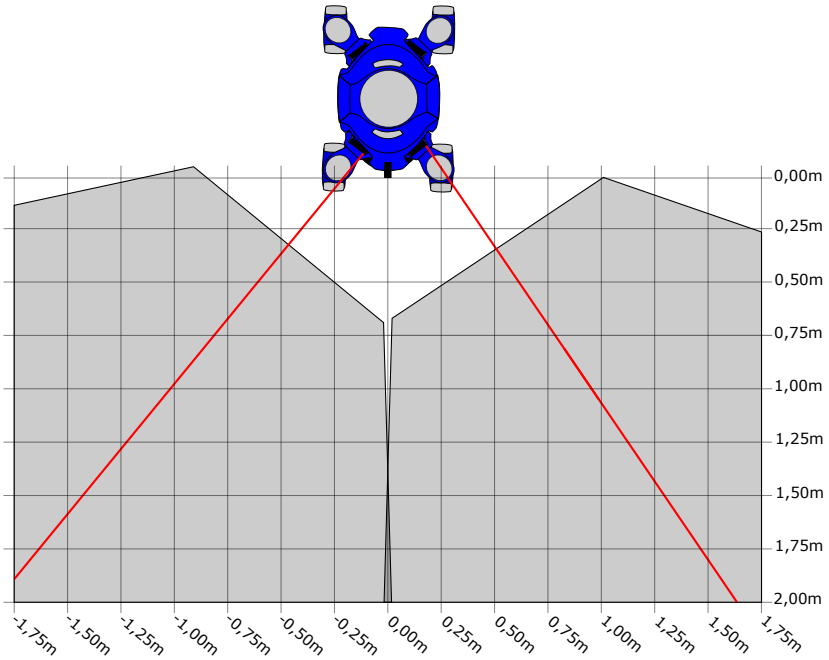


Abbildung 4.5.: Darstellung des Sichtfeldes der hinteren Plattformkameras durch experimentelle Bestimmung

Hierbei fiel auf, dass sich das Sichtfeld der beiden Kameras auf Bodenhöhe erst nach über einem Meter hinter dem Roboter schneidet. Dies bedeutet auch, dass der mittlere Teil der Ladestation beim Ansteckvorgang von den Kameras nicht erfasst wird und so der Galgen und die Steckerhalterung für eine AprilTag-Anbringung nicht in Frage kommen. Durch die Kameralivebilder der hinteren Plattformkameras (siehe Abbildung 4.6) im angedockten Zustand wurde als optimale AprilTag-Position die obere Kante der rechten vorderen Säule der Ladestation bestimmt.


Abbildung 4.6.: Sichtfeld der linken (rechts) und rechten (links) hinteren Plattformkamera in angedocktem Zustand

### 4.2.2. Objektkoordinatensysteme

Für die Ermittlung der Pose zwischen den zwei Teilen des Steckersystems wird eine Transformationskette der beim Andockprozess beteiligten Komponenten erstellt. Alle in der Transformationskette involvierten Objekte werden durch ein körperfestes Koordinatensystem repräsentiert. Es setzt sich aus der Lage des Nullpunkts und seiner Orientierung zusammen. Bei den verwendeten Koordinatensystemen handelt es sich um Systeme der rechten-Hand-Regel(siehe Abbildung 2.3). Im nachfolgenden Absatz werden die verschiedenen Koordinatensysteme erläutert (siehe Abbildung 4.7).

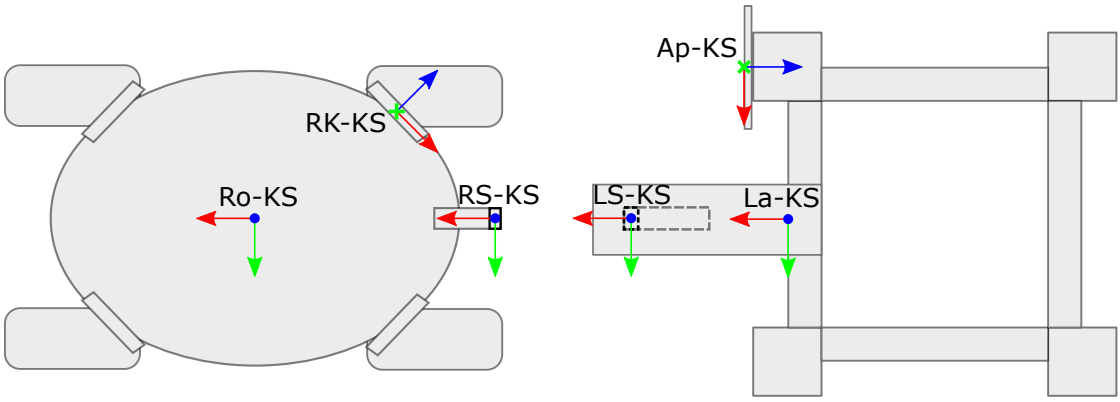


Abbildung 4.7.: Position und Orientierung der Koordinatensysteme von Roboter und Ladestation

#### AprilTag

Die Ausrichtung und Lage eines AprilTag-Koordinatensystems (Ap-KS) wurde von der Universität von Michigan festgelegt. Sein Nullpunkt befindet sich im Zentrum der Marke, die x-Achse verläuft nach rechts, die y-Achse nach unten und die z-Achse in den Tag hinein.[7]

#### Ladestation

Da die Ladestation keinen festgelegten Nullpunkt besitzt konnte dieser frei und zweckmäßig gesetzt werden. Für eine möglichst einfache Ermittlung der Pose zum AprilTag und zur Ladestationsteckverbindung, wurde der Nullpunkt vorne mittig am unteren Ende der Galgensäule positioniert. Die Orientierung des Koordinatensystems (La-KS) ist wie folgt: die x-Achse zeigt aus der Ladestation heraus, die y-Achse nach rechts und die z-Achse nach oben.

#### Ladestationstecker

Das Koordinatensystems des Steckers (LS-KS) konnte ebenfalls frei gewählt werden. Der Nullpunkt wurde mittig auf die Steckfläche gesetzt; dies ermöglicht die optimale Berechnung des Abstands der beiden Steckverbinder. Die Achsen des Koordinatensystems zeigen in folgende Richtungen: die x-Achse aus dem Stecker heraus, die y-Achse nach rechts und die z-Achse nach oben.

#### Roboterkamera

Das Koordinatensystem der Kamera (RK-KS) hat seinen Ursprung in der Kameramitte. Die z-Achse zeigt vom Nullpunkt aus gesehen aus dem Kameraobjektiv heraus. Die x-Achse zeigt aus Sicht der Kamera nach rechts, und die y-Achse ist nach unten gerichtet.

#### Roboter

Das Roboterkoordinatensystem (Ro-KS) sitzt im Mittelpunkt der Roboterplattform ca. 50 cm vom Boden entfernt. Die x-Achse schaut nach vorne aus dem Roboter heraus, die y-Achse zeigt aus Robotersicht nach links und die z-Achse nach oben.

#### Roboterstecker

Das Koordinatensystem der Steckverbindung des Roboters (RS-KS) besitzt die gleiche Orientierung wie das System des Roboters. Der Ursprung ist auf die Mitte des Steckverbinders gesetzt worden, so dass im angesteckten Zustand die Koordinatensysteme beider Steckverbinder aufeinander liegen.

### 4.2.3. Statische Transformationen

Fest miteinander verbundene Objekte besitzen statische Posen zueinander; dies bedeutet einmal erstellt, ändern sich die Transformationsmatrizen nicht mehr. Die Ladestation hat eine statische Pose zu ihrer Steckverbindung und eine zum AprilTag; der Roboter eine zu jeder Plattformkamera und zu seiner Steckverbindung. Nachfolgend werden diese statischen Transformationsmatrizen erläutert (siehe auch Abbildung 4.8).

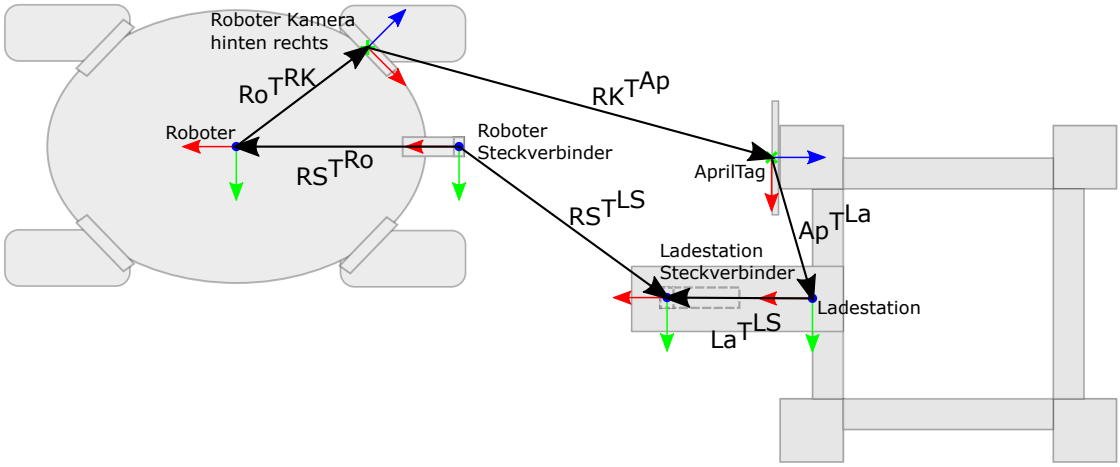


Abbildung 4.8.: Transformationsmatrizen, der beim Andockprozess beteiligten Objekte

### Apriltag-Ladestation

Die Position der Ladestation (La) aus Sicht des AprilTags (Ap) wurde am realen Objekt vermessen. Hierbei ergibt sich folgender Vektor:

$$ {}_{Ap}t^{La} = \begin{pmatrix} -0.30 \\ 0.35 \\ -0.04 \end{pmatrix} $$

Die Orientierung der beiden Koordinatensysteme zueinander wurde aus der Ausrichtung der einzelnen Systeme erschlossen. Dies ergibt folgende Transformationsmatrix:

$$ {}_{Ap}T^{La} = \begin{pmatrix} 0 & 0 & -1 & -0.08 \\ 1 & 0 & 0 & -0.19 \\ 0 & -1 & 0 & 0.08 \\ 0 & 0 & 0 & 1.00 \end{pmatrix} $$

### Ladestation-Ladestationstecker

Die Position des Steckverbinders (LS) in Ladestationkoordinaten (La) wurde ebenfalls gemessen. Hierbei ergibt sich folgender Vektor:

$$ {}_{La}t^{LS} = \begin{pmatrix} 0.38 \\ 0.00 \\ 0.17 \end{pmatrix} $$

Als Transformationsmatrix ergibt sich folgende Matrix:

$$ {}_{La}T^{LS} = \begin{pmatrix} 1 & 0 & 0 & 0.38 \\ 0 & 1 & 0 & 0.00 \\ 0 & 0 & 1 & 0.17 \\ 0 & 0 & 0 & 1.00 \end{pmatrix} $$

### Roboter-Kamera

Die Pose des Kamerakoordinatensystems in Roboterkoordinaten wird auch extrinsische Kalibration genannt. Um verlässliche Kamerawerte zu erhalten wird die Kalibration in regelmäßigen Abständen wiederholt. Für die Ermittlung der extrinsischen Parameter sind verschiedene Methoden möglich. Das DLR nutzt die selbst entwickelten Programme *CalLab/-CalDe*, wie im Kapitel 4.3.1 erläutert.
Aus der Kalibration geht folgende Pose für die hintere rechte Plattformkamera hervor:

$$\begin{pmatrix} 0.829900775429784 & 0.141991192610443 & -0.53954145738108 & -0.151851899351267 \\ -0.557912093151982 & 0.210055381009486 & -0.802875312919903 & -0.225514829783030 \\ -0.000668574437348919 & 0.96732314938599 & 0.253543212621263 & -0.001955486749382 \end{pmatrix}$$

Um die Pose nutzen zu können, muss das Koordinatensystem, auf welches kalibriert wurde, an das Roboterkoordinatensystem angepasst werden. Der Nullpunkt der beiden Systeme befindet sich an derselben Stelle. Für eine Anpassung wurde folgende Transformationsmatrix erstellt:

$$T = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Sie bewirkt einen Tausch der x- und y-Achse und eine Invertierung der z-Achse. Dies ist gleichzusetzen mit einer 90°-Rotation um die z-Achse und einer 180°-Rotation um die y-Achse. Ein weiterer durch die Kalibration bestimmter Parameter ist die Brennweite, die für die Nutzung des AprilTagdetektor benötigt wird.

$$fx : 383.743$$

### Roboter-Roboterstecker

Die Position des Roboters (Ro) aus Sicht dessen Steckers (RS) wird durch folgenden Vektor definiert:

$${}_{RS}t^{Ro} = \begin{pmatrix} -0.35 \\ 0 \\ -0.2 \end{pmatrix}$$

Da die Orientierung der beiden Koordinatensysteme identisch ist, ergibt sich folgende Transformationsmatrix:

$${}_{RS}T^{Ro} = \begin{pmatrix} 1 & 0 & 0 & -0.35 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## 4.3. Einbettung in das Robotersystem

Wie in Kapitel 4.1 näher erläutert, wurde das Konzept des autonomen Ladevorgangs des Robotersystems Rollin'Justin in einzelne Schritte aufgeteilt. Im nachfolgenden Kapitel werden diese nun detaillierter betrachtet. Zur Umsetzung der ersten zwei Schritte Lokalisation und Routenplanung und Anfahrt an die Ladestation konnte auf bereits bestehende Action Templates zurückgegriffen werden, so dass diese mit kleinen Anpassungen übernommen werden konnten. Für die weiteren Schritte wurde zunächst ein gemeinsames *Verbose2*-Skrpit entwickelt, welches nach Evaluation in Action Templates überführt wurde. Abbildung 4.9 zeigt das Skript *charging*, welches die einzelnen Schritte durch einen Klick auf den passenden Button ausführt. Zusätzlich werden die x- und y-Werte des berechneten Steckverbinderabstands und der Modus des Akkus angezeigt.

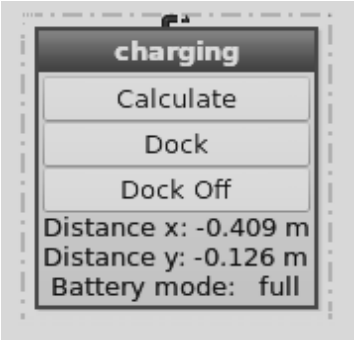


Abbildung 4.9.: Darstellung des Verbose2-Skript-Fensters für die Funktionen: Steckerabstand berechnen, Andocken und Abdocken

### 4.3.1. Erkennung der Ladestation

Grundsätzlich muss die Ladestation zuerst erkannt werden, bevor sie angefahren und eine Steckverbindung hergestellt werden kann; dies erfolgt über einen festgelegten Ablauf (siehe Abbildung 4.10). Davor muss zunächst noch die Position des Roboters in der Welt / Marslabor bestimmt werden. Hierfür wird dem Roboter ein virtueller Nachbau seiner Umgebung übergeben, die Weltrepräsentation; diese wird durch das *OpenRAVE* visualisiert (siehe Abbildung 4.11). Für das Ladeszenario wurde eine neue Welt (siehe Abbildung 4.11) in der Weltrepräsentation angelegt; in welcher die Ladestation sowie der Marslander und die SPUs 1-3 instanziiert sind. Der Marslander und die SPUs dienen in dieser Welt als Landmarks. In Zukunft wäre es auch denkbar, die Ladestation selbst als Landmark zu deklarieren, sobald diese einen festen Platz zugewiesen bekommen hat.

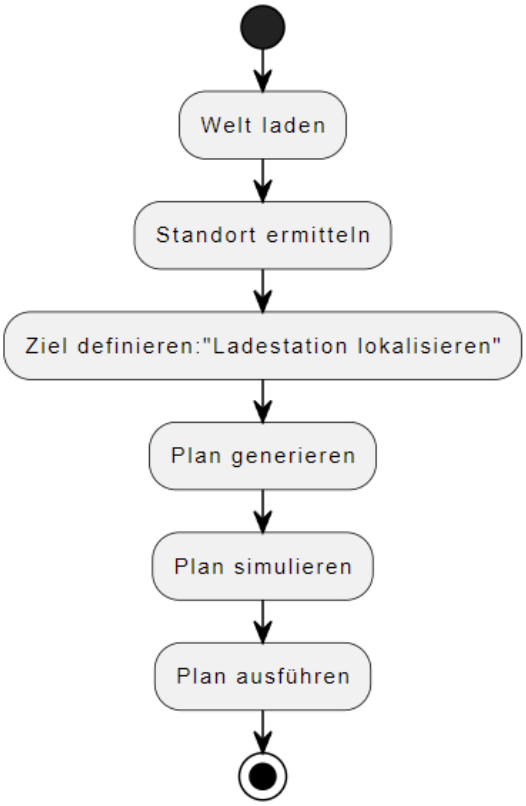


Abbildung 4.10.: Ablaufdiagramm: Lokalisation der Ladestation [erstellt mit dem PlantUML Web Servers anhand [12]]

Zur Bestimmung der Roboterposition wird die Pose zu einem Landmark über die Kopfkamera ermittelt. Mit den Informationen der virtuellen Welt und der Pose des Landmarks zum Roboter wird durch Multiplikation der Transformationsmatrizen der aktuelle Standort des Roboters im realen Raum bestimmt und in die Weltrepräsentation übernommen.

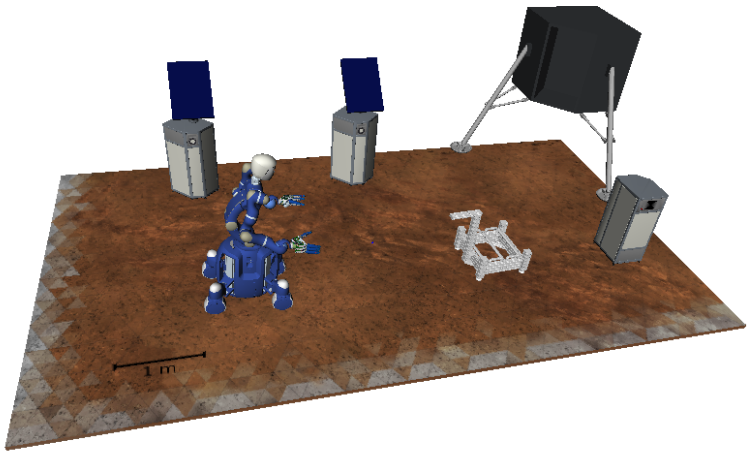


Abbildung 4.11.: Darstellung der detektierten Ladestation im OpenRAVE

Die Ladestation wird mithilfe des hybriden Planers lokalisiert. Hierfür wurde das Action Template *localize_wrt* (siehe Anhang A.3.3) angelegt, das durch den Befehl *object_localized Charger* im *Verbose2*-Skript des hybriden Planers ausgeführt werden kann. Voraussetzung dafür ist eine gute Sicht der Kopfkamera auf die AprilTags der Ladestation. Für die Bestimmung der Position der Ladestation wird aus dem Kamerabild der Kopfkamera mithilfe des AprilTagDetectors die Pose des erkannten AprilTags zur Kamera bestimmt. Diese Pose wird mit der aktuellen Pose des Kopfes, des Torsos und der in der Objektdatenbank gespeicherten Pose des AprilTags zum Ladestationsnullpunkt verrechnet. Daraus wird die Position der Ladestation zum Roboternullpunkt und die Position und die Orientierung der Ladestation innerhalb der Laborumgebung bestimmt. Dieses Vorgehen ist für den hybriden Planer im Action Template *localize_wrt* beschrieben. Im Anschluss wird die Ladestation, wie in Abbildung 4.11 zu sehen, im *OpenRAVE* in der erkannten Position und Orientierung dargestellt.

### 4.3.2. Routenplanung und Anfahrt an die Ladestation

Mithilfe der Informationen aus Kapitel 4.3.1 kann nun der hybride Planer einen symbolischen und geometrischen Plan für die Navigation zur Ladestation entwerfen (siehe Abbildung 4.12). Hierfür wurde das Action Template *navigate_to* (siehe Anhang A.3.4) angelegt, welches durch den Befehl *navigated Justin Charger* im *Verbose2*-Skript des hybriden Planers ausgeführt werden kann.

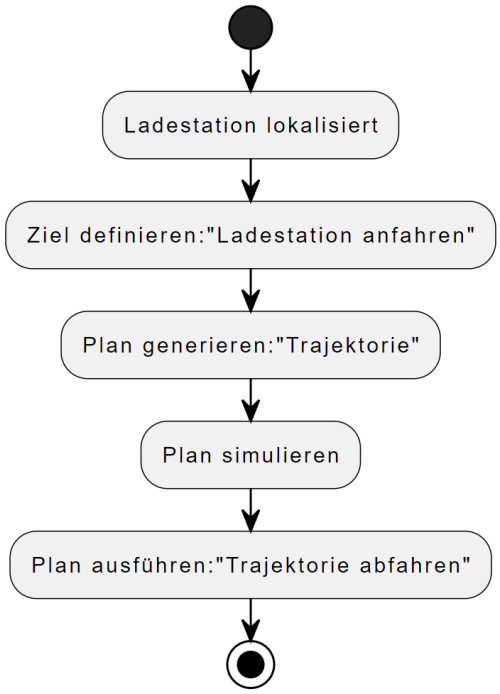


Abbildung 4.12.: Ablaufdiagramm: Navigation des Roboters zur Ladestation [erstellt mit dem PlantUML Web Servers anhand [12]]

Zunächst wird die Aufgabe (Navigation zur Ladestation) auf Durchführbarkeit geprüft. Gelingt dies, wird eine geometrische Trajektorie geplant und in der virtuellen Welt simu-

liert. Im Gegensatz zu einer Route, welche eine geometrische Bahn von Start- zu Zielpunkt beschreibt, behandelt eine Trajektorie die Position und Orientierung des Roboters in Abhängigkeit der Zeit. So wird jedem Punkt auf der Route ein Zeitpunkt zugewiesen. [19] Das Ergebnis der Simulation wird im *OpenRAVE* mithilfe von Punkten und einer kurzen Bewegungsvorschau visualisiert (siehe Abbildung 4.13). Kann die Simulation erfolgreich beendet werden, wird nun die geplante Trajektorie abgefahren. Der Zielpunkt und weitere Parameter wie Anfahrtsgeschwindigkeiten und Beschleunigungen des Roboters können bei Bedarf im Action Template *navigate_to* (siehe Anhang A.3.4) angepasst werden.

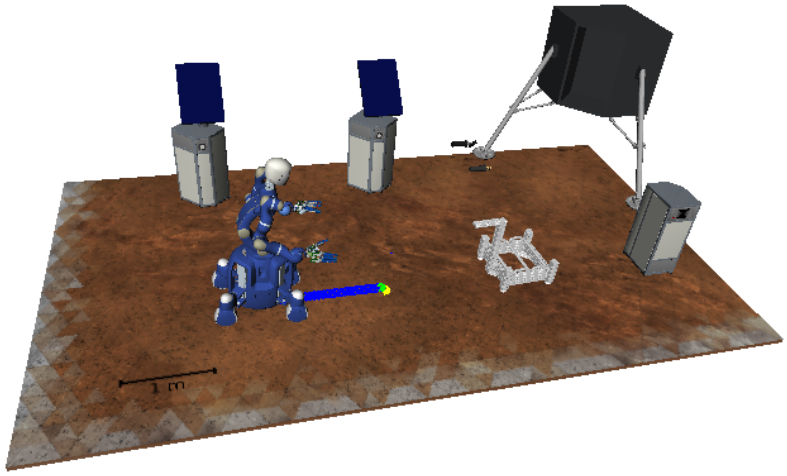


Abbildung 4.13.: Darstellung einer geplanten Trajektorie des Roboters zur Ladestation im OpenRAVE

### 4.3.3. Berechnung des Steckerabstands

Durch Ungenauigkeiten bei der Anfahrt zur Zielposition vor die Ladestation muss nun zunächst der genaue Abstand und die Ausrichtung des Roboters in Bezug auf die Ladestation ermittelt werden. Hieraus wird für das Andockmanöver der Abstand zwischen den Steckverbindern des Roboters und der Ladestation berechnet.
Als erstes wird mithilfe eines Links and Nodes Services (LN-Service) das aktuelle Bild der hinteren rechten Plattformkamera aus dem *sharedmemory* geladen. Im nächsten Schritt wird anhand des Bildes die Transformationsmatrix zwischen Kamera und Ladestation ermittelt. Hierfür wird ein weiterer LN-Service (*apriltag.get_detected_objects_from_image*) genutzt. Dieser bestimmt die Position und die Orientierung aller auf dem Bild erkannten Objekte, die in der Objektdatenbank angelegt sind und gibt diese Informationen, sowie den Name des Objekts und die ID des darauf angebrachten AprilTags in Form einer Liste von Arrays zurück.
Hierfür muss die Größe des AprilTags und die Brennweite der Kamera an den LN-Service übergeben werden. Wird die Ladestation nicht erkannt, gibt das Programm einen Fehler aus und bittet um menschliches Eingreifen. Dies wurde nur für das *Verbose2*-Skript implementiert, um Fehler ausfindig zu machen.
Im Weiteren wird aus den erkannten Objekten die Ladestation herausgefiltert und deren

Pose durch Neuanordnung des 12-teiligen Arrays in eine 4×4-Matrix (${}_{RK}T^{La}$) überführt. Die letzte Zeile der Matrix wird, wie bereits im Kapitel 2.2 erläutert, mit 0001 aufgefüllt. Um die Pose zwischen Roboter und Ladestation (repräsentiert durch ${}_{Ro}T^{La}$) zu berechnen, wird die Transformationsmatrix der Kamera zur Ladestation (${}_{RK}T^{La}$) mit der extrinsischen Kalibrationsmatrix der Kamera (${}_{RK}T^{RK}$) multipliziert.

$$ {}_{Ro}T^{La} = {}_{Ro}\,T^{RK} * {}_{RK}\,T^{La} $$

Die Pose zwischen den Steckern (repräsentiert durch ${}_{RS}T^{LS}$) kann durch Multiplikation der Transformationsmatrizen von Ladestation und Steckverbinder der Ladestation (${}_{La}T^{LS}$), Kamera und Ladestation (${}_{RK}T^{La}$), Roboter und Kamera (${}_{Ro}T^{RK}$), Steckverbinder des Roboters und Roboter (${}_{RS}T^{Ro}$) ermittelt werden.

$$ {}_{RS}T^{LS} = {}_{RS}\,T^{Ro} * {}_{Ro}\,T^{RK} * {}_{RK}\,T^{La} * {}_{La}\,T^{LS} $$

Im Weiteren kann die Distanz in x- und y-Richtung der beiden Steckerverbinder aus Robotersicht aus dem Translationsvektor der Transformationsmatrix (${}_{RS}T^{LS}$) entnommen werden. Der Abstand in x-Richtung kann aus t1 in Metern und der Abstand in y-Richtung aus t2 in Metern abgelesen werden.

$$ {}_{RS}T^{LS} = \begin{pmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Zudem wird der Winkel $\gamma$ berechnet, um den das Koordinatensystem der Ladestation in Bezug zum Koordinatensystem des Roboters um die z-Achse des Roboters gedreht ist. Dies geht aus der Rotationsmatrix hervor, die Teil der Transformationsmatrix ist.

$$ \beta = atan2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}) $$
$$ \gamma = atan2(r_{21}/cos(\beta), r_{11}/cos(\beta)) $$

### 4.3.4. Andockmanöver

Die Ausgangsposition für das Andockmanöver wird mithilfe des hybriden Planers angefahren und befindet sich 40 cm vor der Ladestation. Die Ausrichtung des Roboters ist rücklings und parallel zur Ladestation, sodass sich die Stecker genau gegenüberliegen.
Die Ladestation wird, mithilfe der im vorangegangenen Abschnitt aus der Transformationsmatrix entnommenen Abstände in x- und y-Richtung, sowie der berechneten Rotation über den Befehl cartesian_move_slow( x, y, rot), direkt angefahren.
Um dieses Manöver auf Erfolg zu überprüfen, wird die Information benötigt, ob Strom in den Akku fließt; hierfür wird das Akkuinterface genutzt. Es beinhaltet eine Funktion, welche den Modus des Akkus zurückgibt. Die verschiedenen Modi sind: wird geladen, wird entladen oder ist voll. Diese Funktion wird mithilfe einer IF-Abfrage zur Erfolgsüberprüfung verwendet. Ist das Andockmanöver erfolgreich durchgeführt, verharrt der Roboter im Ladezustand. Ist dies nicht der Fall, wird ein erneuter Andockversuch gestartet, indem der Roboter 0,2 m

zurücksetzt, sich neu ausrichtet und das Manöver von neuem startet (vgl Abbildung 4.14). Dieser Mechanismus wird nach dem dritten Durchlauf abgebrochen und der Roboter bittet akustisch um menschliches Eingreifen.

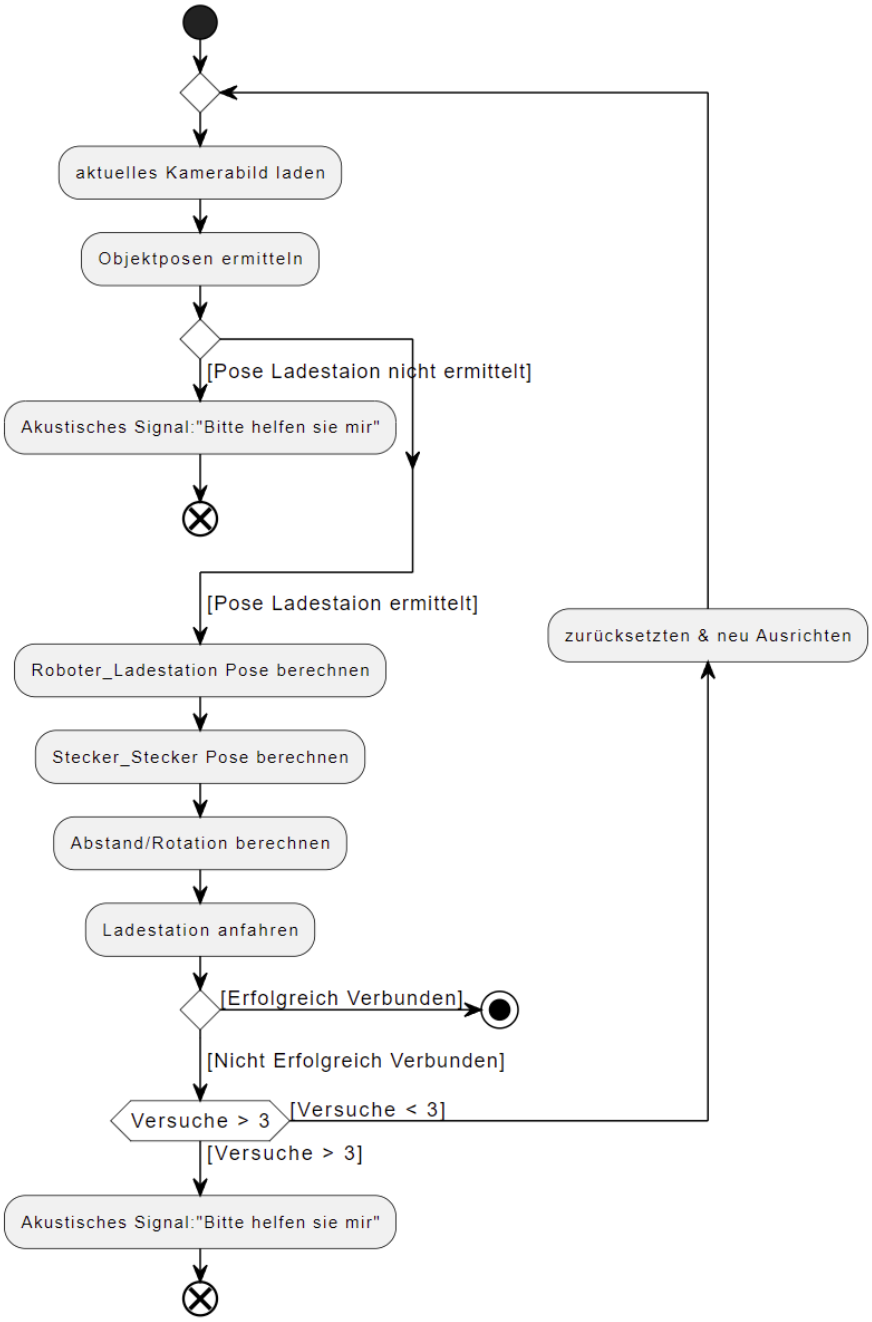


Abbildung 4.14.: Aktivitätsdiagramm: Andockmanöver des Robotersystems an die Ladestation [erstellt mit dem PlantUML Web Servers anhand [12]]

### 4.3.5. Abdocken

Durch das Wegfahren des Roboters mit der Funktion cartesian_move_slow von der Ladestation, löst sich bei einer Entfernung von ca. 0,15 m die Magnetsteckverbindung. Dies

wird unterstützt durch die Rückhalteschnur, welche die frei hängende Steckerhalterung mit dem Galgen verbindet und so die Bewegungsmöglichkeit des Ladestationsteckverbinders in x-Richtung (aus Sicht der Ladestation) begrenzt.

## 4.4. Evaluation

Im folgenden werden die zuvor implementierten Aufgaben Anfahrt, Andocken und Abdocken an der Ladestation auf Funktionalität und Erfolgsgenauigkeit geprüft.

### 4.4.1. Anfahrt

Zur Bewertung des Anfahrtsmanövers wurde dieses 15-mal durchgeführt und die Abweichung zur angestrebten Zielposition des Roboters in x- und y-Richtung sowie der zwischen Roboter und Ladestation liegende Rotationswinkel ausgewertet. Die Abweichungen wurden aus den gemessenen Werten der Positionierung (siehe Tabelle A.1) und dem gewünschten Zielabstand der Beiden Steckverbinder von 0,4 m in x-Richtung 0,0 m in y-Richtung und einem Winkel von 0 rad berechnet. Der Zielabstand wurde so gewählt das ein Sicherheitsabstand von mindestens 0,2 m zwischen dem Roboter und der Ladestation liegen und genügten Platz zum Ausgleich von Positionierungsfehlern durch des Andockmanöver besteht. Dies gibt Auskunft darüber wie konstant die Ausgangsposition für das Andockmanöver angefahren wird. Wie der Tabelle Tabelle 4.1 zu entnehmen ist, weicht die Positionierung des Roboters durchschnittlich um 0,017 m in x-Richtung und 0,056 m in y-Richtung (links aus Sicht der Ladestation) von angestrebten Position ab. Zudem beträgt der durchschnittliche Rotationswinkel zwischen Roboter und Ladestation 0,015 rad (0,86°). Hierdurch ist zu erkennen das die Positionierung in x-Richtung sowie der Winkel gute Wert erzielt, jedoch eine durchschnittliche Abweichung von ca. 5 cm zur erwünschten Position in y-Richtung besteht. Dies lässt sich auf Ungenauigkeiten bei der Rotation zurückführen, beeinträchtigt jedoch nicht den Erfolg des Andockmanövers und kann durch Anpassung der Zielposition ausgleichen werden.

| | Abweichung in x-Richtung [m] | Abweichung in y-Richtung [m] | Rotationswinkel [rad] |
|---|---|---|---|
| Minimum | -0,08 | -0,003 | 0,00 |
| unters Quartil | 0,0085 | -0,043 | 0,008 |
| Median | 0,017 | -0,056 | 0,015 |
| oberes Quartil | 0,0185 | -0,064 | 0,0325 |
| Maximum | 0,114 | -0,087 | 0,126 |

Tabelle 4.1.: Auswertung der Anfahrtsposition

### 4.4.2. Andockmanöver

Der Erfolg des Andockmanövers wird durch den Modus des Akkus überprüft. Im Anschluss der 15 Anfahrmanöver wurde immer auch ein Dockingmanöver durchgeführt; diese hatten eine Erfolgsquote von 100% beim ersten Andockversuch. Zusätzlich wurde das Andockmanöver aus 25 weiteren, nicht durch das Anfahrtsmanöver angefahren Positionen mit unterschiedlichen Abständen und Winkeln zur Ladestation durchgeführt. Hierbei wurden die Grenzen des Manövers ausgetestet. Bei diesen Tests wurde jeweils nur der Erfolg des ersten Andockversuchs berücksichtigt; die Ergebnisse wurden in Erfolg und Misserfolg unterteilt, zudem wurden die Entfernungen und Rotationswinkel beim Start und nach Durchführung des Andockvorgangs festgehalten (siehe Tabelle A.2 und Tabelle A.3).
Die Messungen geben Aufschluss darüber, aus welchen Entfernungen-Winkeln-Kombinationen mit einem erfolgreichen Vorgang zu rechnen ist. In Abbildung 4.15 ist erkennbar, dass der erfolgversprechendste Startbereich zwischen 0,2 m bis 0,42 m in x-Richtung und -0,25 bis 0,1 m in y-Richtung liegt. Der Misserfolg bei der Position (0,35/-0,17) ist durch den hohen Rotationswinkel von 0,27 rad (ca. 15,5°) zu erklären, siehe hierzu Tabelle A.3. Bei den Rotationswinkeln hat sich gezeigt, dass bei den oben genannten Entfernungen ein Wert bis zu 0,23 rad (ca. 13°) zu einem positiven Ergebnis führt. Anhand dieser Ergebnisse lässt sich schlussfolgern das eine Anfahrt der Position mit einem Abstand von 0,3 m in x-Richtung der Steckverbinder zu einem noch stabileren Andockvorgangs führen kann.

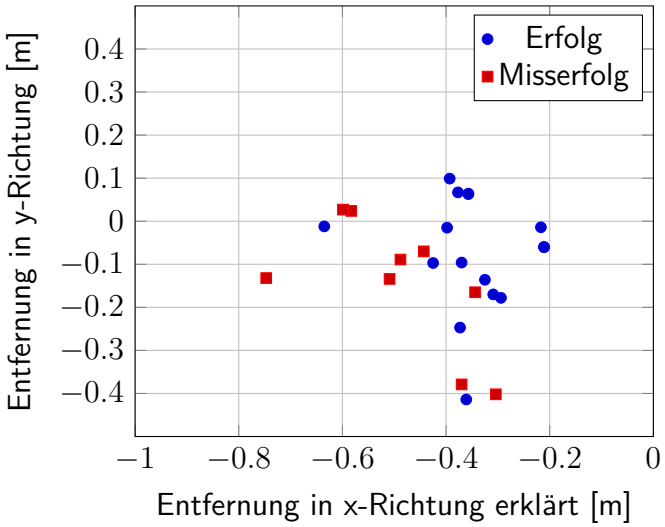


Abbildung 4.15.: Diagramm über Erfolg und Misserfolg des Andockmanövers aus unterschiedlichen Entfernungen in Meter

### 4.4.3. Abdockmanöver

Für die Evaluation des Abdockmanövers ist dieses 10-mal durchgeführt worden. Hierbei wurde auf den Erfolg der Lösung der Magnetsteckverbindung und die Nachschwingzeit des Steckers der Ladestation geachtet. Der Erfolg des Ansteckens wurde optisch und die Nachschwingzeit mit einer Stoppuhr ermittelt. Bei allen Manövern wurde die Steckverbindung

| Versuch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Zeit [s] | 13,3 | 12,5 | 13,9 | 11,7 | 12,4 | 14,1 | 13,4 | 13,4 | 13,3 | 12,3 |
| Erfolg | Ja | Ja | Ja | Ja | Ja | Ja | Ja | Ja | Ja | Ja |

Tabelle 4.2.: Auswertung des Abdockmanövers in Erfolg und Ausschwingzeit des Steckers

erfolgreich durch Anfahren einer Position 30cm vor der Ladestation gelöst. Die durchschnittliche Ausschwingdauer beträgt ca. 13 Sekunden. Jedoch ist es möglich bereits unter 13s eine neue Steckverbindung aufzubauen. Das Schwingen vereinfacht den Vorgang sogar.

## 4.5. Integration in die autonomen Fähigkeiten des Robotersystems

Zur Übertragung des Konzepts in die autonomen Funktionen des Robotersystems wurden zu den bereits vorhandenen Action Templates drei weitere hinzugefügt. Im folgenden wird genauer auf den Aufbau und die Funktionsweise der neuen Action Templates eingegangen.

### 4.5.1. Action Template: localize_base

Dieses Action Template ermittelt die Position des Roboters vor der Ladestation (siehe A.3.5). Durch die Deklarierung der Ladestation als Landmark wird bei deren Lokalisation nicht ihre eigene Position berechnet, sondern die Position des Roboters in Bezug auf die festen Koordinaten der Ladestation im Labor.

Im Kopf des Action Templates werden die Parameter, die Voraussetzungen und die zu erwartenden Effekte für die symbolische Planung definiert. Unter Parameter sind die an der Aktion beteiligten Objekte Ladestation (?o) und Roboter (?r) aufgeführt. Die Voraussetzungen zur Ausführung der Aktion sind, dass sie nicht bereits ausgeführt wurde und dass bereits vor die Ladestation navigiert wurde. Der erwartete Zustand durch das Ausführen dieser Aktion ist zum einen die erfolgreiche Lokalisation der Ladestation durch die Plattformkameras (*base_localized ?o ?r*) und zum anderen der nicht gedockte Zustand des Roboters (*not (docked ?o ?r)*).
Im Hauptteil des Templates ist die Funktion *localize_base()* aufgeführt, welche in einer Operation (op) wiederum die Funktion *localize_base()* mit den Übergabewerten Name der Ladestation (self.name) und der Position der zu verwendenden Kamera (rear right) des AprilTag Interfaces aufruft. Die Funktion *localize_base()* ruft innerhalb des Apriltag Interfaces die Funktion *localize_to_with_base(self,object_name)* auf. Diese ermittelt den Standort des Roboter im Labor mithilfe der Pose zwischen Ladestation und Roboter und überträgt ihn in die Weltrepräsentation. Zusätzlich zur Funktion *localize_base()* verfügt der Hauptteil des Action Templates über die Funktion *get_alias*, welche Teilsätze in Englisch und Deutsch für die Sprachausgabe des Roboters vorgibt um den Erfolg oder Misserfolg der Planung einer Aufgabe mitzuteilen. Diese Funktion ist in jedem Action Template zu finden und wird deswegen im weiteren nicht mehr erwähnt.

### 4.5.2. Action Template: docking

Der Kopf des Action Templates *docking* (siehe A.3.6) definiert die gleichen beteiligten Objekte (Ladestation und Roboter) wie im vorherigen Template. Die Voraussetzung für das Docking ist die vorherige Lokalisation der Ladestation mithilfe der Plattformkamera und der erwartete Effekt ist, dass der Roboter angedockt und nicht abgedockt ist.

Im Hauptteil ist die Funktion *docking()* aufgeführt; diese berechnet die Position im Labor, die der Roboter einnehmen soll, um an der Ladestation angedockt zu sein. Zudem wird die Geschwindigkeit und die Beschleunigung definiert (*ipol_args*), mit welcher der Roboter diese Position anfahren soll und welchen Sicherheitsabstand der Roboter zur Ladestation einhalten soll. All diese Informationen werden dem Plattform Interface in der Funktion *navigate_to_frame* übergeben, welche die Trajektorie zum gewünschten Punkt erstellt und abfährt.

### 4.5.3. Action Template: detach

Das Action Template *detach* ist identisch zum Template *docking* (siehe A.3.7) aufgebaut; jedoch wird eine Position vor der Ladestation ermittelt und der Funktion *navigate_to_frame* übergeben, die Voraussetzung ist ein zuvor erfolgtes Docking und der erwünschte Effekt der Zustand abgedockt und nicht mehr angedockt.

# 5. Fazit und Ausblick

In dieser Arbeit wurde ein Ladekonzept für den Roboter Rollin'Justin entwickelt, getestet und in die autonomen Funktionen des Robotersystems integriert. Dieses befasst sich mit der Lokalisation der Ladestation durch AprilTags und der darauffolgenden Navigation vor die Ladestation. Über die rechte hintere Plattformkamera wird daraufhin die Ladestation erneut lokalisiert und das Robotersystem nahe genug an die Ladestation navigiert, sodass eine erfolgreiche Steckverbindung der beiden Magnetsteckverbinder zustande kommt. Zudem wurde das Lösen der Steckverbindung durch Navigieren in eine Position 30cm vor der Ladestation umgesetzt. All dies wurde mithilfe von Aktionsvorlagen (Action Templates) in die autonomen Funktionen des Robotersystems integriert. Hierfür wurde das schon vorhandene AprilTag Interface angepasst. Dadurch wird die Lokalisation von Objekten über eine der vier Plattformkameras des Roboters möglich. Einsatz kann dies in der genaueren Bestimmung der Roboterposition finden. Über die unterschiedlichen Kameras können parallel mehrere Landmarks erfasst werden, die jeweils zur Standortbestimmung des Robotersystem verwendet werden können. Mittels Vergleich und Verrechnung dieser Daten kann die Genauigkeit der ermittelten Position erhöht werden. Zudem ermöglicht es eine Alternative zur Ermittlung der Roboterposition, falls sich kein Landmark im Sichtfeld der Kopfkamera des Roboters befindet. Die Integration des Ladekonzepts durch Action Templates ermöglicht das Kommandieren einzelner Schritte sowie die Bildung und Ausführung einer Aktionskette durch den hybriden Planer. Das Ziel der Arbeit war die Bereitstellung von Funktionen, welche es ermöglichen die Ladestation im Laboralltag sowie in Missionen zu nutzen. Dies konnte erfolgreich durch diese Aktionsvorlagen umgesetzt werden und ermöglicht die Nutzung dieses Dienstes in kommenden Projekten und Missionen, wie der Surface Avatar Mission. Bei dieser sollen ab Mai 2022 verschiedene Roboter, darunter auch Rollin' Justin, durch die ESA Astronautin Samantha Cristoforetti von der Internationalen Raumstation kommandiert werden. Durch Telepräsenz soll hierbei der Aufbau von Infrastruktur und deren Wartung in extraterrestrischen Umgebungen koordiniert werden.
In weiteren Arbeiten ist es denkbar Optimierungen an der Hardware der Ladestation sowie der Software wie folgt vorzunehmen. Softwareseitig soll die Abfrage der Kameraparameter optimiert werden, indem man diese über einen LN-Service bereitstellt. Zudem soll eine Abfrage integriert werden ob der gewünschte Effekt der Aktion wirklich eingetreten ist. An der Hardware der Ladestation wird in Zukunft die in Kapitel 3.3.3 neue Steckverbindung montiert und getestet werden. Sobald die Ladestation aktiv in Betrieb genommen wird, soll das Design noch ansprechender und mehr in das Marssetting passend gestaltet werden und ihr ein fester, gut erreichbarer Platz im Labor geschaffen werden. Hierbei muss die Position der Ladestation genauestens in Bezug auf den Labornullpunkt vermessen werden und die dementsprechenden Werte der Weltrepräsentation angepasst werden. Jedoch ist auch eine Übertragung des Ladekonzeptes auf andere Robotersysteme des DLRs wie den Agile Justin, den Autonomen Industriellen Mobilen Manipulator (AIMM) oder das mobile Assistenzrobotiksystem EDAN denkbar um auch diesen Systemen eine Langzeitautonomie zu ermöglichen.

# Literatur


[1] Alin Albu-Schäffer u. a. "Anthropomorphic Soft Robotics – From Torque Control to Variable Intrinsic Compliance". In: *Robotics Research*. Hrsg. von Cédric Pradalier, Roland Siegwart und Gerhard Hirzinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 185–207. ISBN: 978-3-642-19457-3.

[2] Tim Bodenmüller. *justin_wiring. Rollin' Justin Rechnerarchitektur*. 2020. URL: https://rmc-github.robotic.dlr.de/rollin-justin/ (besucht am 10. 01. 2022).

[3] Borst Christoph u. a. "Rollin' Justin - Mobile platform with variable base". In: *2009 IEEE International Conference on Robotics and Automation*. 2009, S. 1597–1598. DOI: 10.1109/ROBOT.2009.5152586.

[4] Rosen Diankov und James Kuffner. *OpenRAVE: A Planning Architecture for Autonomous Robotics*. Techn. Ber. CMU-RI-TR-08-34. Pittsburgh, PA: Carnegie Mellon University, Juli 2008.

[5] DLR. *internes Wiki RM*. 2010. URL: https://wiki.robotic.dlr.de/ (besucht am 27. 12. 2021).

[6] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May. IEEE, 2011, S. 3400–3407.

[7] Kristian Ehlers. *Echtzeitfähige 3D Posenbestimmung des Menschen in der Robotik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019. ISBN: 978-3-658-24821-5. DOI: 10.1007/978-3-658-24822-2.

[8] Paul Furgale u. a. "Toward automated driving in cities using close-to-market sensors: An overview of the V-Charge Project". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. 2013, S. 809–816. DOI: 10.1109/IVS.2013.6629566.

[9] M. Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006). Journal Of Artificial Intelligence Research, Volume 26, pages 191-246, 2006, S. 191–246. ISSN: 1076-9757. DOI: 10.1613/jair.1705. URL: https://arxiv.org/pdf/1109.6051.

[10] Gerhard Hirzinger u. a. "DLR's torque-controlled light weight robot III - are we reaching the technological limits now?" In: *icra*. 2002, S. 1710–1716.

[11] Intel RealSense Depth and Tracking Cameras. *Depth Camera D435*. 2021. URL: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf (besucht am 27. 12. 2021).

[12] Stephan Kleuker. *Grundkurs Software-Engineering mit UML. Der pragmatische Weg zu erfolgreichen Softwareprojekten*. ger. 2., korrigierte und erw. Aufl. Studium. Wiesbaden: Vieweg + Teubner, 2011. 371 S. ISBN: 978-3-8348-1417-3. DOI: 10.1007/978-3-8348-9843-2.

[13] kontron. *Datasheet mITX-KBL-H-CM238*. 2017. URL: https://www.iesy.com/media/Default/02-Produkte/02-02-Boards/02-02-04-mini-ITX/kontron-mitx-kbl-h-cm238-datasheet.pdf (besucht am 11. 02. 2022).

[14] Daniel Leidner. *Cognitive Reasoning for Compliant Robot Manipulation*. en. 2017. URL: https://elib.dlr.de/115679/.

[15] Daniel Leidner, Christoph Borst und Gerd Hirzinger. "Things Are Made for What They Are: Solving Manipulation Tasks by Using Functional Object Classes". In: *International Conference on Humanoid Robots (HUMANOIDS)*. Juni 2012. URL: https://elib.dlr.de/80508/.

[16] Daniel Leidner u. a. "Object-Centered Hybrid Reasoning for Whole-Body Mobile Manipulation". In: *2014 IEEE International Conference on Robotics and Automation*. Juni 2014, S. 1828–1835. URL: https://elib.dlr.de/88960/.

[17] H. Liu u. a. "DLR's Multisensory Articulated Hand, Part II: The Parallel Torque, Position Control System". In: *ICRA'98, Leuven, Belgien, 1998*. Hrsg. von IEEE. Bd. 3. LIDO-Berichtsjahr=2000, Omnipress, 1998, S. 2087–2093. URL: https://elib.dlr.de/11612/.

[18] Xiaozhou Luo. *Extrinsic-Camera-Calibration. Rollin' Justin Plattform Kamera Kalibrierung*. 2021. URL: https://rmc-github.robotic.dlr.de/luo-xi/ (besucht am 07. 02. 2022).

[19] Jörg Mareczek. *Grundlagen der Roboter-Manipulatoren – Band 2*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020. ISBN: 978-3-662-59560-2. DOI: 10.1007/978-3-662-59561-9.

[20] Max Krogius und Simone Gasparini. *AprilTag 3 github*. 2018. URL: https://github.com/AprilRobotics/apriltag (besucht am 20. 02. 2022).

[21] NVIDIA. *Jetson TX Developer Kit - Technical Specifications.* 2020. URL: https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-tx2/ (besucht am 08. 02. 2022).

[22] Georg Rill und Thomas Schaeffer. *Grundlagen und Methodik der Mehrkörpersimulation*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017. ISBN: 978-3-658-16008-1. DOI: 10.1007/978-3-658-16009-8.

[23] Bruno Siciliano und Oussama Khatib. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-23957-4. DOI: 10.1007/978-3-540-30301-5.

# A. Anhang

## A.1. Tabellen der Evaluierung

| Versuch | x | y | rot |
|---|---|---|---|
| 1 | -0,514 | -0,017 | 0,096 |
| 2 | -0,418 | -0,061 | 0,015 |
| 3 | -0,363 | -0,061 | 0,019 |
| 4 | -0,417 | -0,003 | 0,126 |
| 5 | -0,321 | -0,064 | 0,002 |
| 6 | -0,419 | -0,053 | 0,011 |
| 7 | -0,431 | -0,043 | 0,032 |
| 8 | -0,414 | -0,043 | 0,006 |
| 9 | -0,427 | -0,075 | 0,033 |
| 10 | -0,417 | -0,064 | 0,005 |
| 11 | -0,410 | -0,087 | 0,094 |
| 12 | -0,407 | -0,085 | 0,010 |
| 13 | -0,417 | -0,056 | 0,025 |
| 14 | -0,385 | -0,013 | 0,015 |
| 15 | -0,411 | -0,055 | 0,005 |

Tabelle A.1.: Auswertung Anfahrt

| Erfolg | Start | | | Ziel | | |
|---|---|---|---|---|---|---|
| Versuch | x | y | rot | x | y | rot |
| 1 | -0,398 | -0,015 | 0,118 | 0,003 | -0,013 | 0,106 |
| 2 | -0,357 | 0,063 | 0,177 | 0,006 | -0,012 | 0,167 |
| 3 | -0,211 | -0,06 | 0,001 | 0,007 | -0,012 | 0,009 |
| 4 | -0,294 | -0,178 | 0,235 | 0,003 | -0,018 | 0,005 |
| 5 | -0,211 | -0,06 | 0,001 | 0,005 | -0,011 | 0,009 |
| 6 | -0,309 | -0,17 | 0,014 | 0,019 | -0,02 | 0,014 |
| 7 | -0,393 | 0,099 | 0,004 | 0,004 | -0,003 | 0,018 |
| 8 | -0,635 | -0,012 | 0,036 | 0,011 | -0,045 | 0,055 |
| 9 | -0,425 | -0,097 | 0,149 | 0,003 | -0,022 | 0,286 |
| 10 | -0,357 | 0,064 | 0,102 | 0,008 | -0,026 | 0,065 |
| 11 | -0,37 | -0,096 | 0,105 | 0,011 | -0,022 | 0,098 |
| 12 | -0,373 | -0,247 | 0,281 | 0,006 | -0,025 | 0,008 |
| 13 | -0,217 | -0,014 | 0,004 | 0,005 | -0,013 | 0,001 |
| 14 | -0,325 | -0,136 | 0,015 | 0,013 | -0,013 | 0,006 |
| 15 | -0,377 | 0,067 | 0,014 | 0,008 | -0,026 | 0,011 |
| 16 | -0,361 | -0,414 | 0,308 | 0,02 | -0,032 | 0,009 |

Tabelle A.2.: Auswertung erfolgreiche Andockversuche

| Kein Erfolg | Start | | | Ziel | | |
|---|---|---|---|---|---|---|
| Versuch | x | y | rot | x | y | rot |
| 1 | -0,583 | 0,024 | 0,001 | 0,02 | -0,044 | 0,008 |
| 2 | -0,509 | -0,134 | 0,006 | 0,024 | -0,034 | 0,017 |
| 3 | -0,599 | 0,027 | 0,024 | 0,014 | -0,043 | 0,016 |
| 4 | -0,488 | -0,089 | 0,073 | 0,011 | -0,038 | 0,11 |
| 5 | -0,344 | -0,165 | 0,097 | 0,011 | -0,031 | 0,166 |
| 6 | -0,304 | -0,402 | 0,275 | 0,016 | -0,027 | 0,008 |
| 7 | -0,37 | -0,379 | 0,405 | 0,011 | -0,027 | 0,007 |
| 8 | -0,443 | -0,07 | 0,007 | 0,014 | -0,027 | 0,004 |

Tabelle A.3.: Auswertung nicht erfolgreiche Andockversuche

## A.2. Kamera Kalibrationsdateien

Die nachfolgenden .txt Dateien wurden dem DLR internen GitHub entnommen.[18]

### A.2.1. camera_calibration_base_rear_left_rgbd.txt

```
# DLR-RMC camera calibration file by DLR CalLab.
# Parameters for describing the geometry of pinhole images.
# Created: Thu Dec 5 17:55:26 2019

# Version number of the parameter file.
version=2

# Image coordinate origin at the center of the upper-left pixel (center
   ↪ )
# or at the upper-left corner of the upper-left pixel (corner).

origin=center

# The extrinsic parameters describe the transformation of a point Pc in
   ↪  the
# camera coordinate system into a point Pw in the world coordinate
   ↪ system,
# Pw = R*Pc + T
# Default: [1 0 0; 0 1 0; 0 0 1] [0 0 0]

camera.0.R=[ 0.774340245296575 -0.159466547146753 0.612345875852283;
              0.632765296286369 0.191846959496256 -0.750201568018136;
              0.00215501184586327 0.968382484588021 0.249460578450093]
camera.0.T=[ -817.017350616366 -906.01883789361 -610.072723256141]

# The point Pc in the camera coordinate system is projected and
   ↪ transformed
# into distored coordinates [u v]. All parameters are used in the same
   ↪ way as
# in OpenCV.

camera.0.k1= 0.000465142
camera.0.k2= 0.00178791
camera.0.A=[ 388.388 -0.00381795 322.181; 0.00000 388.824 235.917;
   ↪ 0.00000 0.00000 1.00000]
camera.0.width= 640
camera.0.height= 480
camera.0.rmsInt= 0.208287
# End of file.
```

### A.2.2. camera_calibration_base_rear_right_rgbd.txt

```
# DLR-RMC camera calibration file by DLR CalLab.
# Parameters for describing the geometry of pinhole images.
# Created: Thu Feb 20 08:55:26 2020

# Version number of the parameter file.
version=2

# Image coordinate origin at the center of the upper-left pixel (center
   ↪ )
# or at the upper-left corner of the upper-left pixel (corner).

origin=center

# The extrinsic parameters describe the transformation of a point Pc in
   ↪  the
# camera coordinate system into a point Pw in the world coordinate
   ↪ system,
# Pw = R*Pc + T
# Default: [1 0 0; 0 1 0; 0 0 1] [0 0 0]

camera.0.R=[ 0.829900775429784 0.141991192610443 -0.53954145738108;
              -0.557912093151982 0.210055381009486 -0.802875312919903;
              -0.000668574437348919 0.96732314938599 0.253543212621263]
camera.0.T=[ -1105.93824584571 -892.595902846958 -608.081283066317]

# The point Pc in the camera coordinate system is projected and
   ↪ transformed
# into distored coordinates [u v]. All parameters are used in the same
   ↪ way as
# in OpenCV.

camera.0.k1= 0.00260679
camera.0.k2= -0.00184699
camera.0.A=[ 383.743 0.259209 321.048; 0.00000 384.154 235.508; 0.00000
   ↪  0.00000 1.00000]
camera.0.width= 640
camera.0.height= 480
camera.0.rmsInt= 0.271760

# End of file.
```

## A.3. Objektdatenbank-Skripte

### A.3.1. supvis513

```
---
id: 513
descriptor: "supvis"
family: "36h11"
size: 0.036
frame:
- [ 1, 0, 0, -0.040]
- [ 0, 1, 0, -0.300]
- [ 0, 0, 1, 0.245]
```

### A.3.2. supvis525

```
---
id: 525
descriptor: "supvis"
family: "36h11"
size: 0.1
frame:
- [ 0, 0, -1, -0.080]
- [ 1, 0, 0, -0.190]
- [ 0, -1, 0, 0.080]
```

### A.3.3. localize_wrt

```
'''
:parameters (?o - chargingstation ?r - _robot)
:precondition (and (not (localized ?r ?o)))
:effect (and (localized ?r ?o) (not (navigated ?r ?o)) )
'''

def localize_wrt(robot):
op = [
("localize_wrt", self.name),]
return op

def get_alias(language="EN", robot=None):
alternatives = {
       "EN": "Locate",
```

```
       "DE": "Lokalisiere"
}
ret = alternatives.get(language, lambda: "")
return ret+" "+self.name if ret else ""
```

### A.3.4. navigate_to

```
'''
:parameters (?o - chargingstation ?r - _robot)
:precondition (and (localized ?r ?o))
:effect (and (navigated ?r ?o) (not (localized ?r ?o)) (not (
   ↪ localized_inmap ?r)) )
'''

def navigate_to(robot, x=1.25, y=0, theta=0):
from odb_interface import odb_utils
import pyutils.matrix as pm

ipol_args = {
       "platformTransMaxVel": 0.25, # [m/s]
       "platformTransMaxAcc": 0.25, # [m/s**2]
       "platformTransMaxJerk": 2.50, # [m/s**3]
       "platformTransMaxDev": 0.01, # [m/s]
}

maintainance_pose = dot(pm.rotz(theta), pm.txyz(x, y, 0))
maintainance_pose = dot(self.frame, maintainance_pose )
safe_distance = 0.0

op = [
("navigate_to_frame", maintainance_pose, ipol_args, safe_distance),]
return op

def get_alias(language="EN", robot=None):
alternatives = {
       "EN": "Navigate to",
       "DE": "Navigiere zu"
}
ret = alternatives.get(language, lambda: "")
return ret+" "+self.name if ret else ""
```

### A.3.5. localize_base

```
'''
:parameters (?o - chargingstation ?r - _robot)
:precondition (and (not (base_localized ?o ?r)) (navigated ?r ?o) )
:effect (and (base_localized ?o ?r) (not (docked ?o ?r)) )
'''

def localize_base(robot):
op = [
("localize_base", self.name, 'rear','right'),]
return op

def get_alias(language="EN", robot=None):
alternatives = {
       "EN": "Locate over base",
       "DE": "Lokalisiere ueber Basis"
}
ret = alternatives.get(language, lambda: "")
return ret+" "+self.name if ret else ""
```

### A.3.6. docking

```
'''
:parameters (?o - chargingstation ?r - _robot)
:precondition (and (base_localized ?o ?r))
:effect (and (docked ?o ?r) (not (undocked ?o ?r)) )
'''

def docking(robot):
from odb_interface import odb_utils
import pyutils.matrix as pm
import numpy as np

pose_charger_cplug = array([[1, 0, 0, 0.38],[0, 1, 0, 0],[0, 0, 1,
   ↪ 0.17],[0, 0, 0, 1]])
pose_robot_jplug = array([[1,0,0,-0.35],[0,1,0,0],[0,0,1,0],[0,0,0,1]])

ipol_args = {
       "platformTransMaxVel": 0.1, # [m/s]
       "platformTransMaxAcc": 0.15, # [m/s**2]
       "platformTransMaxJerk": 1.50, # [m/s**3]
       "platformTransMaxDev": 0.01, # [m/s]
}
```

```
maintainance_pose = np.dot(self.frame,np.dot(pose_charger_cplug,pm.inv(
    ↪ pose_robot_jplug)))
safe_distance = 0.0

op = [
("navigate_to_frame", maintainance_pose, ipol_args, safe_distance, [
    ↪ self.name]),]
return op

def get_alias(language="EN", robot=None):
alternatives = {
        "EN": "Dock on",
        "DE": "Andocke "
}
ret = alternatives.get(language, lambda: "")
return ret+" "+self.name if ret else ""
```

### A.3.7. detach

```
'''
:parameters (?o - chargingstation ?r - _robot)
:precondition (and (docked ?o ?r))
:effect (and (not (docked ?o ?r) ) (undocked ?o ?r) )
'''

def detach():
from numpy import np
ipol_args = {
        "platformTransMaxVel": 0.1, # [m/s]
        "platformTransMaxAcc": 0.15, # [m/s**2]
        "platformTransMaxJerk": 1.50, # [m/s**3]
        "platformTransMaxDev": 0.01, # [m/s]
}

maintainance_pose = np.dot(np.dot(self.frame,np.dot(pose_charger_cplug,
    ↪ pose_robot_jplug)), np.array
    ↪ ([[1,0,0,0.3],[0,1,0,0],[0,0,1,0],[0,0,0,1]]))
safe_distance = 0.0

op = [
("navigate_to_frame", maintainance_pose, ipol_args, safe_distance),]
return op

def get_alias(language="EN", robot=None):
alternatives = {
```

```
        "EN": "Dock off",
        "DE": "Abdocken"
}
ret = alternatives.get(language, lambda: "")
return ret+" "+self.name if ret else ""
```

### A.3.8. manifest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<object>
<name>charging_station</name>
<author>a.mueller@dlr.de</author>
<derive>_object</derive>
<dimension>0.8 1.1 0.8</dimension>
<label>_landmark</label>
<geometry>
<display>chargingstation_vision.obj</display>
<collision>chargingstation_coll_avoid.obj</collision>
</geometry>
<vision>
<april_tag>supvis513</april_tag>
<april_tag>supvis525</april_tag>
</vision>
<logic>
<code_snippet>localize_wrt</code_snippet>
<code_snippet>navigate_to</code_snippet>
<code_snippet>localize_base</code_snippet>
<code_snippet>docking</code_snippet>
<code_snippet>detache</code_snippet>
<predicate>base_localized ?o - charging_station ?r - _robot </predicate
   ↪ >
<predicate>docked ?o - charging_station ?r - _robot </predicate>
<predicate>undocked ?o - charging_station ?r - _robot </predicate>
</logic>
</object>
```