
Abschluss Bericht für das Praktikum Mobile Roboter im WS 17/18

Florian Dreschner, Yassine El Himer, Daniel Klitzke, Robin Weitemeyer

9. Februar 2018

Inhaltsverzeichnis

1	Einleitung	4
1.1	Aufgabenstellung	4
1.2	Ausgangssituation	4
1.3	Systemarchitektur	4
1.3.1	Hardware	4
1.3.2	Software	4
1.4	Arbeitspakete	6
2	State of the Art	8
2.1	Bilderkennung	8
2.2	Bahnplanung & Greifen	9
3	Arbeitsbericht	9
3.1	Allgemein	9
3.2	Bilderkennung	10
3.2.1	Tassenerkennung	10
3.2.2	Turtleboterkennung	11
3.2.3	Automatische Kamerakalibrierung	12
3.3	Bahnplanung & Greifen	13
3.3.1	Bahnplanung	13
3.3.2	Greifen	14
3.4	High-level Steuerung & Kommunikation	15
3.5	Sonderaufgaben	16
3.5.1	Hardwareaufbau	16
3.5.2	High-level Steuerung	16
3.5.3	Kommunikation	16
3.5.4	Dokumentation	16
4	Beschreibung des Gesamtsystems	18
4.1	Bilderkennung	18
4.1.1	Tassenerkennung	18
4.1.2	Automatische Kamerakalibrierung	19
4.2	Bahnplanung & Greifen	20
4.2.1	Bahnplanung	20
4.2.2	Greifen	22
4.3	High-level Steuerung & Kommunikation	22
5	Evaluation & Ausblick	23
5.1	Bilderkennung	23
5.1.1	Tassenerkennung	23
5.1.2	Turtleboterkennung	23
5.1.3	Automatische Kamerakalibrierung	23

5.2	Bahnplanung & Greifen	23
5.2.1	Bahnplanung	23
5.2.2	Greifen	24
5.3	High-level Steuerung & Kommunikation	25

1 Einleitung

1.1 Aufgabenstellung

Ziel des diesjährigen Praktikums Mobile Roboter mit dem Thema „Coffee to go“ war es, eine Tasse Kaffee, welche auf einem Tisch platziert wurde mittels eines Roboterarms zu greifen, auf einem mobilen Roboter zu platzieren und diese, nach Transport durch selbigen, dann mittels eines zweiten Roboterarms wieder auf einen zweiten Tisch zu stellen. Die Aufgabe unserer Gruppe war hierbei, die Tasse auf dem ersten Tisch zu lokalisieren, sie zu greifen und auf dem mobilen Roboter zu platzieren.

1.2 Ausgangssituation

Als Ausgangspunkt für die Realisierung der Aufgabe wurde uns sowohl diverse Hardware als auch Software zu Verfügung gestellt. So stand uns für das Greifen der Tasse ein UR5 Roboter von Universal Robots, welcher bereits auf einem Tisch befestigt wurde. Zur Lokalisierung der Tasse waren eine Kinect bzw. alternativ eine Intel Realsense Kamera verfügbar. Außerdem standen uns für die Entwicklung der Software sowohl vorkonfigurierte Rechner in Poolräumen, als auch ein an den UR5 Roboter angeschlossener Shuttle PC zur Verfügung. Als Grundlage für die zu entwickelnde Software dienten ROS Indigo bzw. Kinetic.

1.3 Systemarchitektur

1.3.1 Hardware

Zur Erfüllung unserer Aufgabe wurden uns ein UR5 Roboterarm, ein Schunk PG70 und eine Microsoft Kinect zur Verfügung gestellt. Der UR5 wurde für uns in der linken Ecke am Tisch montiert und der PG70 über ein Verbindungsstück mit dem Roboterarm verbunden. Zum Greifen der Tasse mussten wir selbst modellierte Greifer anfertigen lassen, welche passgenau am PG70 angebracht wurden. Mit Hilfe von Bosch Elementen haben wir die Kinect Kamera in der unteren Ecke des Tisches in einer Höhe von einem Metern über der Tischplatte platziert. Diese Höhe war notwendig, um die linke, obere und rechte Ecke des Tisches, sowie den Bereich direkt links unterhalb des UR5 einfangen zu können. Die drei Ecken wurden als Orientierungspunkte der Kamerakalibrierung benutzt. Den Aufbau der Hardware kann man in Abbildung 1 sehen.

Weiterhin wurde festgelegt, dass sich der Turtlebot im Bereich direkt links unterhalb des Roboterarms zur Tassenübergabe platzieren soll.

1.3.2 Software

Der Softwareteil besteht grundsätzlich aus vier Komponenten. Die Lokalisierungskomponente ermöglicht die Detektion der Tasse und des Turtlebots. Sie stellt zudem diverse Werkzeuge zur automatischen Kamerakalibrierung, Poseschätzung der Tasse, Manipulation der Punktwolken und Annotation der Datensätze zur Verfügung. Die Bahnpla-

Abbildung 1: Aufbau der Hardware: Kinect Kamera (links oben), UR5 (rechts), PG70 (am Ende des UR5).

nungskomponente führt die Bahnplanung des Roboterarms zwischen der erkannten Tasse und dem erkannten Turtlebot aus. Die Greifkomponente steuert den Fingerabstand, um die erkannte Tasse zu greifen bzw. die Tasse auf den erkannten Turtlebot zu legen. Letztendlich dient die High-Level-Steuerung zur Koordination der verschiedenen Softwarekomponenten und bietet die Schnittstelle zur Kommunikation mit dem zentralen State Manager der drei Gruppen an.

1.4 Arbeitspakete

Die Lösung der gestellten Aufgabe haben wir in folgende Arbeitspakete unterteilt:

Hauptaufgabe	Arbeitspaket	Zuständigkeit
Bildererkennung		
	Erstellung der Softwarearchitektur für die Bildererkennung	Daniel Klitzke
	Tassendetektion mit FCN	Daniel Klitzke, Yassine El Himer
	Tassendetektion Segmentierung + Neuronale Netze	Daniel Klitzke
	Tassendetektion SVM + HOG Features	Daniel Klitzke
	Segmentierung der Tasse aus der Punktwolke	Daniel Klitzke
	Erkennung der Tassenorientierung	Daniel Klitzke
	Evaluation von YOLO als Alternative für die Tassendetektion	Yassine El Himer
	Turtlebotdetektion SVM + HOG Features	Yassine El Himer
	Segmentierung des Turtlebot aus der Punktwolke	Yassine El Himer
	Automatische Kamerakalibrierung	Daniel Klitzke
Bahnplanung & Greifen		
	Aufsetzen der Simulation	Robin Weitemeyer
	Modellierung der Roboterumgebung	Robin Weitemeyer
	Konfiguration von MoveIt	Robin Weitemeyer
	Bahnplanung mit MoveIt	Robin Weitemeyer
	Modellierung eines Greifers	Florian Dreschner
	Ansteuerung und Verwendung des UR5	Robin Weitemeyer
	Ansteuerung und Verwendung des Greifers	Florian Dreschner
	Implementierung der Planungslogik	Robin Weitemeyer
High-level Steuerung & Kommunikationsschnittstellen		
	Bestimmung und Evaluierung der Schnittstellen	Yassine El Himer, Florian Dreschner
	Implementierung des Managers	Florian Dreschner
Sonderaufgaben		
	Hardwareaufbau	Robin Weitemeyer
	High-level Steuerung	Yassine El Himer
	Kommunikation	Florian Dreschner
	Dokumentation	Daniel Klitzke

2 State of the Art

2.1 Bilderkennung

In diesem Abschnitt sollen die für unsere Aufgabenstellung relevanten Themen aus der aktuellen Forschung vorgestellt werden.

Die Lokalisierung der Tasse lässt sich auf vielfältige Art und Weise realisieren. Eine Strategie, die wir auch in unserem Ansatz verfolgen, ist es die Tasse in den Bilddaten zu detektieren und die exakten Abmessungen anschließend aus der Punktwolke zu berechnen. Aus diesem Grund ist für uns vor allem das große Feld der Objekterkennung interessant. Hier zeichnet sich in den letzten Jahren deutlich der Trend hin zu Verfahren des maschinellen Lernens, insbesondere des Deep Learnings, ab. Vorangetrieben wird dies insbesondere durch die Entwicklung leistungsfähiger Hardware, insbesondere von GPUs, sowie der Verfügbarkeit von großen Trainingsdatensätzen. Deep-Learning basierte Ansätze wurden in folgenden für uns relevanten Bereichen bereits eingesetzt und übertrafen bisherige Ansätze deutlich.

Bild Klassifikation Ein für uns relevanter Bereich ist die Bild Klassifikation. Das Ziel hierbei ist es in Bildern präsente Objekte korrekt zu klassifizieren. Aktuelle Ansätze wie setzen hierbei auf der Verwendung von Convolutional Neural Networks (CNNs). Wichtige Publikationen sind u.A. ?? und ??

Objekt Detektion Das Ziel bei der Objekt Detektion ist es, Objekte einer bestimmten Klasse in einem Bild zu detektieren. Für die Klassifikation von Bildern entworfene Neuronale Netze werden häufig auch für die Objekt Detektion eingesetzt, indem sie mit anderen Verfahren kombiniert werden. Für konkrete Beispiele zur Objekt Detektion mittels Deep Learning siehe auch ?? und ??.

Semantische Segmentierung Insbesondere im Bereich des autonomen Fahrens gewonnen Ansätze zur Semantischen Segmentierung in den letzten Jahren stark an Popularität. Das Ziel einer semantischen Segmentierung ist es, jedem Pixel eines Eingabebildes eine Objektklasse zuzuordnen. Realisiert wird dies zumeist über sogenannte Fully Convolutional Neural Networks. Wichtige Publikationen im Bereich der semantischen Segmentierung sind u.A. ??, ?? und ??.

Neben den vorgestellten Deep Learning Ansätzen kann es auch durchaus angebrachter sein andere Verfahren des Maschinellen Lernens zu verwenden. Gründe hierfür könnten unter anderem die Knappheit von Trainingsdaten, Limitierte Hardware Ressourcen als auch aufgabenspezifische Eigenheiten sein. Um insbesondere im Bezug auf limitierte Hardware Ressourcen und die Knappheit von Trainingsdaten Abhilfe zu schaffen kann es insbesondere bei überschaubaren Problemstellungen gerechtfertigt sein auch flache Klassifikatoren einzusetzen. Ein populärer Vertreter der flachen Klassifikationsverfahren sind die Support Vector Machines. Diese wurden in ?? erstmals in Kombination mit HOG Features für die Personen Detektion in Bilddaten eingesetzt.

Eine Alternative wäre es gewesen, die Tasse nicht mithilfe der Bilddaten, sondern mithilfe der Punktwolke zu detektieren. Auch hier existieren Deep Learning basierte Ansätze wie z.B. ?? oder ??.

2.2 Bahnplanung & Greifen

Im Bereich der Bahnplanung genießen auf Abtastung basierende Algorithmen wie *probabilistic roadmap* (PRM) oder *rapidly-exploring random tree* (RRT) große Beliebtheit. Was nicht zuletzt an ihrer asymptotischen Optimalität ? liegt.

Die PRM-Planungsmethode ? besteht aus zwei Phasen:

1. Learning-Phase
2. Query-Phase

In der Learning-Phase wird ein kollisionsfreier Graph im Konfigurationsraums des Roboters erstellt, indem zufällige Punkte im Raum auf Kollision untersucht wird und anschließend mit Hilfe eines lokalen Pfadplaners mit dem bestehenden Graph, ebenfalls kollisionsfrei, verbunden wird. In der Query-Phase wird durch einen Graphsuchalgorithmus ein Pfad vom Anfangs- zum Zielpunkt bestimmt.

Die RRT-Methode ? benötigt keine Learning-Phase. Zwar werden auch hier zufällig Punkte im Konfigurationsraum ausgewählt und auf Kollision mit der Umgebung geprüft, der Aufbau des Graphes, welcher die Form eines Suchbaums besitzt, geschieht jedoch für jede Pfadanfrage von Neuem. Somit benötigt eine Suchanfrage bei der RRT-Methode potentiell länger im Vergleich zu PRM, der Graph berücksichtigt jedoch auch dynamische Objekte, deren Position sich in der Umgebung verändern kann.

RRT-Connect ? ist eine Erweiterung von der RRT-Planungsmethode. Hier werden zwei Suchbäume gleichzeitig aufgebaut. Einer beginnt vom Startzustand, der andere vom Zielzustand. Dabei wird eine greedy-Heuristik verwendet, um die Wahrscheinlichkeit zu erhöhen, dass sich die beiden Bäume aufeinander zubewegen. Nach einer vorgegebenen Anzahl von Iterationen wird versucht, die beiden Bäume direkt zu verbinden.

3 Arbeitsbericht

3.1 Allgemein

Simulation Zum einfacheren Testen der Tassenerkennungs- und Bahnplanungskomponenten wurde eine Simulation in Gazebo und RViz aufgesetzt. Die Simulation orientiert sich an der Vorgehensweise im ROS-Paket „Motion_Interface“. Sie beinhaltet das urdf-Modell der statischen Roboterszene, siehe Abbildung 6, sowie eine virtuelle Kinect Kamera. Die UI-Konfiguration der Simulation wird in unserem eigenen MoveIt-Paket definiert.

Da die Simulation auf einer älteren Gazebo-Version aufgesetzt wurde, ist sie im momentanen Zustand nicht voll funktionsfähig.

3.2 Bilderkennung

3.2.1 Tassenerkennung

Allgemein Für die Tassenerkennung wurden keinerlei ROS Pakete verwendet. Stattdessen wurden die einzelnen Komponenten in Python selbst implementiert. Die am häufigsten zur Hilfe genommenen Bibliotheken sind unter anderem:

- NumPy (Effizienter Umgang mit Matrizen und Vektoren)
- scikit-image (Bildverarbeitung)
- scikit-learn (Algorithmen für Maschinelles Lernen)
- Keras (High-level API für die Erstellung von Neuronalen Netzen)
- Matplotlib (Visualisierung)

Detektion der Tasse in den Bilddaten Für die Detektion der Tasse in den Bilddaten wurden eine Reihe von Ansätzen getestet, bevor die finale Lösung feststand.

Zunächst war der Plan die Tasse mittels eines Fully Convolutional Neural Network aus den Bilddaten zu segmentieren. Als Basisstruktur wurde hierfür die Architektur des Resnet50 Netzes verwendet. Die Eingabe für das Netz stellt ein RGB Bild dar und es liefert als Ausgabe zwei Feature Maps, welche die Wahrscheinlichkeiten für die Zugehörigkeiten der Pixel des Eingabebildes zu den Klassen „Tasse“ bzw. „Keine Tasse“ enthalten. Trainiert wurde auf den Microsoft COCO Datensatz unter Nutzung von GPU Kapazitäten bei Amazon Webservices. Der Ansatz wurde letztendlich verworfen, da es sich herausstellte dass das Netz zu überdimensioniert für die Segmentierung einer Klasse aus einem RGB Datensatz ist und nicht genügend Trainingsdaten zur Verfügung standen.

Der zweite getestete Ansatz basierte auf der Segmentierung des Bildes in eine Menge von Segmenten sowie einer Klassifizierung durch ein Neuronales Netz. Das Bild wird hierfür zunächst mittels des SLIC Algorithmus in eine Menge von Segmenten zerlegt. Diese Segmente werden dann unter Einbeziehung eines gewissen Kontextfensters dem Neuronalen Netz zur Klassifikation übergeben. Jedem Segment wird entweder als „Tasse“ oder „Keine Tasse“ klassifiziert. Das Ergebnis einer Vorhersage mit dem beschriebenen Algorithmus ist in Abbildung 3 abgebildet.

Der dritte gewählte Ansatz wurde letztendlich auch in das fertige System übernommen und basiert auf einer SVM welche Bildsegmente, die mithilfe eines Sliding Window Ansatzes gewonnen werden klassifiziert. Details zu diesem Ansatz werden im Abschnitt 4.1.1 näher erläutert.

Segmentierung der Tasse in der Punktwolke Nachdem die Tasse in den Bilddaten detektiert wurde wird sie mittels eines Clustering Algorithmus aus den Bilddaten segmentiert. Hierbei traten keinerlei Komplikationen auf. Der Ansatz wird ebenfalls in 4.1.1 beschrieben.



Abbildung 2: Eingabe für das Neuronale Netz zur Segment Klassifikation

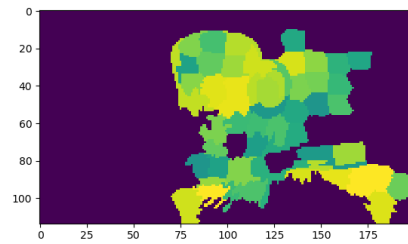


Abbildung 3: Vom Netz ausgegebene Wahrscheinlichkeiten

Erkennung der Tassenorientierung Hier wurden zwei verschiedene Ansätze implementiert wovon letztendlich einer für das finale System ausgewählt wurde. Der erste Ansatz basiert auf dem RANSAC Algorithmus. Zunächst wird die Punktwolke der Tasse auf die x-y-Ebene projiziert und deren Zentrum bestimmt. Anschließend wird in jeder Iteration des Ransac Algorithmus ein zufälliger Punkt ausgewählt und eine Linie durch selbigen und das Zentrum der Tasse gelegt. Es wird die Konfiguration mit den meisten Inliern als Richtung des Tassengriffs ausgewählt. Es stellte sich hierbei heraus, dass die zum Tassengriff gehörigen Punkte im Vergleich zu den restlichen Punkten so wenig ins Gewicht fallen dass so keine zuverlässige Schätzung möglich war. Der zweite Ansatz basiert auf dem Local Outlier Factor Schätzer und erwies sich als deutlich robuster als der zuvor beschriebene Ansatz. Er wird in 4.1.1 näher beschrieben.

3.2.2 Turtleboterkennung

Allgemein Für die Turtleboterkennung wurden keine ROS Pakete verwendet. Die verwendete Bibliotheken sind unter anderem NumPy, openCV, scikit-image, scikit-learn, Matplotlib.

Detektion des Turtlbots in den Bilddaten Die erste Überlegung war, das Turtlebot anhand eines Convolutional neural Networks zu erkennen. Mangels an Datensätze, die annotierten Bilder von der Klasse *Turtlebot* in unterschiedlichen Kontexten beinhalten, wurde dieser überwachte Ansatz verworfen.

Eine günstigere aber robuste Alternative war ein Model basiert auf einer SVM, die im Vergleich zu CNNs weniger annotierte Daten benötigen, um letztendlich die Stützvektoren zu finden. Nachdem einen Datensatz von verschiedenen annotierten Posen des Turtlebots in einer Szene mithilfe der Kinect erstellt und segmentiert wurde, wurde die SVM auf unterschiedlichen negativen und positiven Patches trainiert. Mehr zu diesem Ansatz wird im Abschnitt ?? detailliert.

Der zweite getestete Ansatz basierte sich auf ein Template-Matching. Hier wurde ein Bild von der obigen Seite des Turtlebots aufgenommen und als Schablone verwendet.

Dabei wird eine Szene von der Kinect in mehreren Segmenten gespaltet. [noch in bearbeitung]

wurde nicht in das fertige System übernommen

Segmentierung des Turtlebots in der Punktwolke Aus der Detektion des Turtlebots in einem Bild werden die x und y Koordinate im Weltkoordinatensystem gewonnen. Die z Koordinate ist durch die Punktwolke zu bestimmen. Ein naïves Verfahren war, die Punkte aus der Wolke zu nehmen, die zwischen einer Tiefe von der Höhe des Turtlebots und der maximalen von der Kinect gelieferten Tiefe. Werden die x,y und z Koordinate kombiniert, wird das 3D-Bounding-Box des Turtlebots bestimmt.

3.2.3 Automatische Kamerakalibrierung

So wie auch die Tassenerkennung wurde auch die Kalibrierung der Kamera ohne die Verwendung von fertig verfügbaren ROS Paketen realisiert. Es wurden zwei Ansätze realisiert wobei sich der als zweites implementierte Ansatz als deutlich robuster und genauer erwiesen hat und deshalb in das Gesamtsystem aufgenommen wurde. Der zunächst implementierte Algorithmus basiert auf der Segmentierung der Tischplatte aus der Punktwolke. So wird als erster Schritt die Tischebene mittels des RANSAC Algorithmus aus der Punktwolke segmentiert. Der berechnete Normalenvektor der Eben wird anschließend dazu verwendet um die Tischplatte so zu rotieren, dass sie parallel zur x-y-Ebene ist. Daraufhin wird die minimale Bounding Box der Tisch Punktwolke im 2 Dimensionalen Raum berechnet und als Tischfläche angenommen. Es stellte sich beim testen des Algorithmus heraus, dass die Methode zwar schnell ist, jedoch zu ungenaue Ergebnisse liefert. Die Ungenauigkeiten waren wohl vor allem darauf zurückzuführen, dass die Punktwolke an den Ecken der Tischplatte einige Löcher aufwies und die Bounding Box so nicht den eigentlichen Umriss des Tisches wiedergab. Neben den Ungenauigkeiten bei der Tischdetektion ergab sich ein weiterer Nachteil aus den zur Ausführung des RANSAC Algorithmus verwendeten Python Bindings für PCL. So wurden diese, obwohl von der PCL Webseite verlinkt, augenscheinlich nicht mehr sehr aktiv gewartet und waren doch sehr unvollständig und teilweise fehlerhaft. Dies weckte zusätzlich den Wunsch die entsprechende Bibliothek aus dem Projekt zu entfernen. So wurde schlussendlich ein zweite Ansatz implementiert. Dieser basiert auf der Detektion der Tischkanten aus den Bild-daten sowie einer anschließenden Segmentierung der die Tischkanten repräsentierenden Geraden in der Punktwolke. Beim testen des Ansatzes erwies sich dieser als deutlich genauer als die vorher implementierte Methode und auch die RANSAC Implementierung konnte mit geringem Mehraufwand, der durch die Implementierung einer geeigneten BaseEstimator Klasse entstand, durch die Implementierung von scikit-learn ersetzt werden. Die Details des schlussendlich implementierten Algorithmus sind in 4.1.2 beschrieben.

3.3 Bahnplanung & Greifen

3.3.1 Bahnplanung

Die Aufgabe der Bahnplanung war es, den PG70 unter Verwendung des UR5 zur Position der Kaffeetasse zu manövrieren, um diese zu greifen und anschließend sicher und ohne das Verschütten jeglicher Flüssigkeit auf den Turtlebot abzustellen. Dies führte jedoch zu mehreren Problemen, die es zu bewältigen gab.

Zum einen musste dem Roboterarm gesagt werden, wo sich die Tasse bzw. der Turtlebot befindet und wie er die beiden Ziele ansteuern soll. Dabei musste sowohl eine Selbstkollision, als auch eine Kollision mit der Umgebung vermieden werden. Des Weiteren war es auf der Strecke von der Kaffeetasse zum Turtlebot notwendig, dass die Tasse ihre aufrechte Orientierung beibehält, damit der Kaffee bei dem Transport nicht verschüttet wird.

Um diese Aufgabe zu erfüllen und die dabei auftretenden Probleme zu lösen, hatten wir uns zwei Optionen herausgesucht, zwischen denen wir uns entscheiden mussten:

1. Die Bahnplanung mit dem Motion Planning Framework MoveIt ? realisieren.
2. Die Bahnplanung mit der FZI-hauseigenen Motion Pipeline ? realisieren.

MoveIt ist eine State of the Art Software zur Steuerung von Robotern. Sie beinhaltet verschiedene Lösungen der inversen Kinematik (IK), unterschiedliche Planer zur Trajektorienerstellung und Optimierungsmöglichkeiten der gefundenen Bahnen. MoveIt ist mit all seinen Möglichkeiten und Anwendungsgebieten sehr umfangreich und komplex. Dafür existieren jedoch ausführliche Dokumentationen und Tutorials, sowie eine aktive Community, die sich bei Fragen und Problemen untereinander zu Rat steht.

Die FZI-eigene Motion Pipeline ist ein Framework zur einfachen Ausführung von Trajektorien auf bekannten Kinematiken. Dank einer rqt-GUI ist es besonders benutzerfreundlich, zum Beispiel Bahnen für einen Roboterarm zu lernen. Diese Trajektorien können dann gespeichert werden und müssen nur noch vor der Ausführung geladen werden. Auch die FZI Motion Pipeline besitzt eine Dokumentation, welche im Vergleich zu MoveIt allerdings weniger umfangreich ist.

Letzten Endes haben wir uns für den Einsatz von MoveIt entschieden. Denn obwohl dieses Framework für unsere Anwendungszwecke übermächtig ist und aufgrund der Komplexität eine länger Einarbeitsphase benötigt, wollten wir uns trotzdem im Bereich der Bahnplanung keine Grenzen setzen. MoveIt schien uns daher als das Mittel der Wahl.

Der erste Schritt bei der Integration von MoveIt in unser Projekt war es daher, das Framework für unseren genauen Roboteraufbau zu konfigurieren. Unter Verwendung des MoveIt Setup Assistant ? erzeugten wir uns, mit Hilfe unseres selbst erstellten URDF-Modells, unser eigenes ROS-Paket mit allen Konfigurations- und Launch-Dateien, die wir für die Benutzung von MoveIt auf unserer Hardware benötigten.

Zur Bahnplanung in MoveIt verwendeten wir verschiedene Planer aus der Open Motion Planning Library (OMPL), wie zum Beispiel PRM-Varianten, wobei wir uns am Ende für den RRTConnectDefault entschieden. Dieser implementiert den RRT-Connect Algorithmus und lieferte uns die zuverlässigsten Ergebnisse. Bei den IK-Implementierungen gab es keine beobachtbaren Unterschiede.

Für die eigentliche Bestimmung der Trajektorie des UR5 stellt MoveIt zwei Methoden zur Verfügung.

1. Bahnplanung vom momentanen Zustand zur Zielpose.
2. Bahnplanung vom momentanen Zustand über eine Liste von Zielposen.

Bei der ersten Variante wird nur der momentane Zustand des Roboters gesetzt. Anschließend wird mit der Methode *set_target_pose* die Zielpose gesetzt und mit *plan* die Bewegung geplant. Die zweite Methodik basiert auf der Bahnplanung über mehrere Wegpunkte. Die Methode *compute_cartesian_path* bestimmt dabei aus mehreren Posen eine Trajektorie. Wir entschieden uns schließlich für die zweite Variante, da diese ohne Pfad-Constraints und weitere Optimierungsschritte für unsere Aufgabe optimale Bahnen bestimmte und auch die Ausrichtung der Tasse beibehielt.

3.3.2 Greifen

Ein weiterer Teil der Aufgabe befasste sich mit dem Greifen der Tasse. Dafür steht der Greifer Schunk PG70 zur Verfügung, sowie ein 3D-Drucker zur Erstellung von Fingern. Zu Beginn wurden zwei Griffvarianten in Betracht gezogen: 1) Seitliches Greifen und 2) Greifen von oben. Während seitliches Greifen natürlicher erscheint, erwies sich die Planung für einen Griff der Tasse von oben als einfacher. Um keine Variante vorzeitig auszuschließen, wurde ein Fingermodell mit zwei möglichen Griffflächen erstellt, eine Fläche am Ende des Fingers, sowie eine Ausbuchtung am Schaft des Fingers. Die Fläche am Endes des Fingers führte zu genau zwei Kontaktpunkten zwischen der Tasse und dem Endeffektor. Die Tasse wurde sehr instabil gehalten und neigte zu Schaukeln bei schnelleren Bewegungen. Desweiteren reichte die Reibung zwischen Kunststoff und Ton nicht für einen festen Griff aus.

Aus diesem Grund wurde die Anzahl der Kontaktpunkte durch eine eckige Ausbuchtung von zwei auf vier verdoppelt. Bei ersten Tests mit der eigentlichen Hardware wurde ein weiteres Problem erkannt: Der Gesamthub des Greifers ist geringer als der durchschnittliche Durchmesser der Tasse. Daher konnte zuerst nur eine von zwei Tassen gegriffen werden, außerdem beschränkte sich die Pufferzone auf weniger als 1cm pro Fingerbacke. Die ersten Ergebnisse aus der Bilderkennung legte eine deutliche Verbreitung der Griffbreite nahe, um Ungenauigkeiten bei der Approximation der Tasse auszugleichen.

Die folgende Generation an Fingern vergrößerte die Griffbreite um 4cm durch „Nachhinten“-Lagerung jeder Backe. Zusätzlich wurde der Schaft verlängert und erhielt eine ähnliche kantige Ausbuchtung, wie bereits das Ende des Fingers. Diese Änderung vergrößerte den Arbeitsbereich des Roboters, den Sicherheitsabstand zum Tisch und die Kontaktfläche der Tasse am Schaft des Fingers.

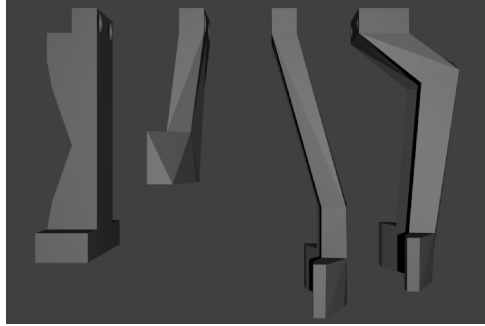


Abbildung 4: Evolution des Fingers

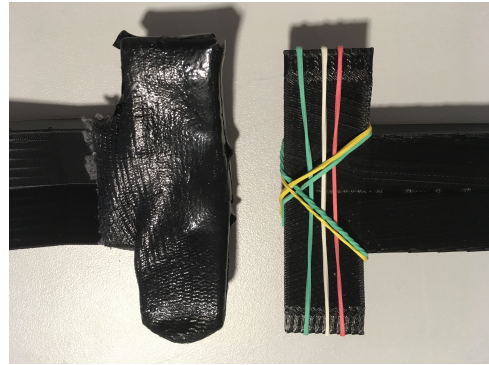


Abbildung 5: Erhöhter Grip

Bei einem weiteren Versuch wurde durch einen Konfigurationsfehler zu tief unter die Kante der Tasse eingetaucht, was in letzter Konsequenz zum Brechen eines Fingers führte. Um dieses Problem bei einer weiteren Iteration kategorisch auszuschließen, wurde die Verschiebung des Schaftes an den Montagebereich gelegt, eine abfallende Kante in Richtung Ende des Fingers konstruiert und die Gesamtlänge von 12cm auf 11cm reduziert. Außerdem wurde die Druckrichtung von Horizontal auf Vertikal verändert, um die Stabilität zu erhöhen.

Zusätzlich wurden zwei Varianten für eine bessere Reibung getestet: 1) Arbeitshandschuh und 2) Haushaltsgummi. Aus dem Handschuh wurden zwei Finger entfernt und mit doppelseitigem Klebeband auf dem gedruckten Finger befestigt. Der Grip zwischen Finger und Tasse erhöht sich, jedoch ist die hinzugefügte Schicht dünn, weshalb der Finger sich bei erhöhtem Druck verbiegt. Zur Vermeidung wird eine dickere Schicht zwischen Finger und Tasse benötigt. Haushaltsgummis in einem X-Muster erwiesen sich als besser geeignet, da sie mehr Reibung und eine dickere Schicht bieten.

Abhängig von der Tasse wurden verschiedene Konstanten für die Breite der Finger empirisch getestet. Zur Vereinfachung gehen wir nun von genau einem Tassenmodell mit einer festen Breite aus. Anfangs wurde mit einem Fingerabstand von 9mm gegriffen, jedoch lässt sich dieser Wert zugunsten der Griffestigkeit deutlich verkleinern. Es zeigte sich, dass bereits ein Abstand von 8mm ausreicht.

Nebenläufig zur Evolution der Finger wurde ein ROS Paket mit dem Namen „cup_gripper“ erstellt. Die erste Variante beinhaltet zwei Services: GrabCup und ReleaseCup. GrabCup akzeptierte als Parameter cup_diameter in Meter und gab, wie auch ReleaseCup, ein boolean success zurück. Es stellte sich heraus, dass Actions besser geeignet sind, da sie längere Operationen abbilden und Feedback geben können. Aus diesem Grund wurden die Services in Actions umgeschrieben.

3.4 High-level Steuerung & Kommunikation

Zur Koordination der verschiedenen Softwarekomponenten der Gruppe wird ein Managementknoten benötigt, der gleichzeitig die Kommunikation mit anderen Gruppen leitet. Schnittstellen zwischen den Gruppen wurden in mehreren gemeinsamen Treffen

einstimmig beschlossen und selbstständig vom jeweiligen Team implementiert. Für eine strukturierte Kommunikation erhielt die vorgestellte Komponente den Namen „Cup Acceptor“. Die Entwicklung findet zuerst unter dem Namen „Highlevelcontrol“ statt, jedoch wird das Paket in „cup_acceptor_manager“ umbenannt. Der Namen des Repositories änderte sich nicht, um Migrationsaufwand zu vermeiden. Lokalisierung, Bahn- und Griffplanung werden durch das Paket gestartet. Zu Beginn stand eine Implementierung unter Verwendung von Services in Frage, jedoch war der Mehrwert von Actions schnell ersichtlich. Bevor der Integration mit anderen Teams, wurde diese Komponente mit dem axclient der actionlib aufgerufen.

3.5 Sonderaufgaben

3.5.1 Hardwareaufbau

Zur Erfüllung der Sonderaufgabe des Hardware-Aufbaus wurden folgende Teilleistungen erbracht:

Die Anbringung des UR5, PG70 und der Kinect Kamera wurden, wie in ?? beschrieben, erarbeitet und durchgeführt. In Zusammenarbeit mit der zweiten Projektgruppe wurde außerdem festgelegt, welche Seite des Tisches vom Turtlebot angesteuert werden soll. Zusammen mit der dritten Gruppe wurde weiterhin entschieden, welche Tassen für die Live-Demo greifbar sein sollen. Die Modellierung der Szenerie als URDF-Modell, siehe Abbildung 6, sowie das Aufsetzen der Simulation wurden ebenfalls im Zuge dieser Sonderaufgabe durchgeführt.

3.5.2 High-level Steuerung

3.5.3 Kommunikation

Als Teil der Sonderaufgabe „Kommunikation“ wurden zwei Treffen mit den Zuständigen der jeweiligen Gruppen abgehalten. Während der Treffen wurden Schnittstellen definiert, Anknüpfungspunkte bei ähnlichen Problemen gesucht und ein generelles Verständnis der Gesamtaufgabe geprägt.

3.5.4 Dokumentation

Im Rahmen der Sonderaufgabe Dokumentation wurden die folgenden Aufgaben übernommen. Für die Sicherstellung der Koordination und Dokumentation der Arbeitsaufgaben wurde das Projektmanagement Werkzeug Trello eingerichtet. Zudem wurde ein Wiki für die projektinterne Dokumentation angelegt. Desweiteren wurde die Erstellung der Abschlusspräsentation, der Abschlussausarbeitung, einer Dokumentation im FZI Wiki, sowie das Zusammenschneiden des gemeinsamen Videos aller Gruppen koordiniert.

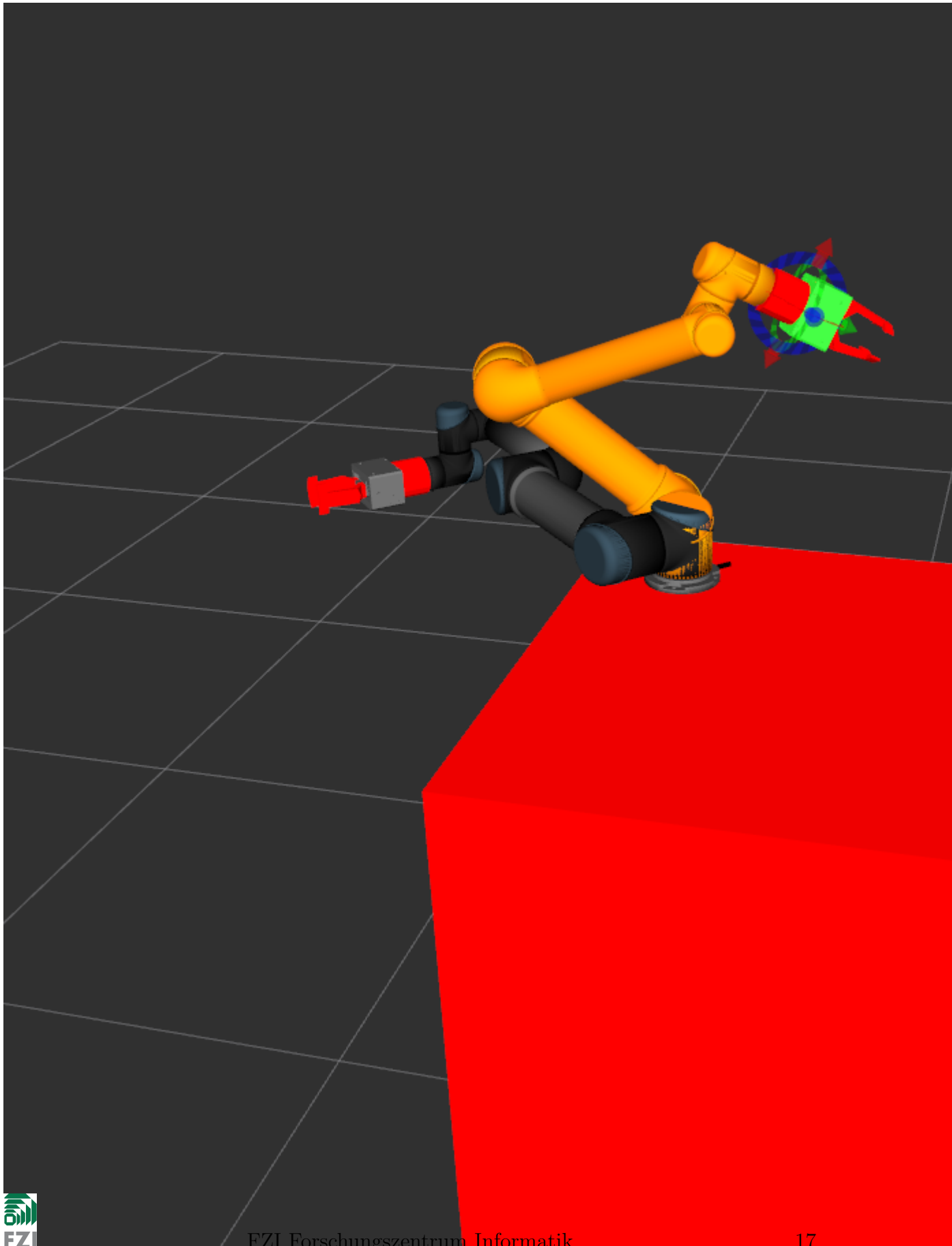


Abbildung 6: Das für die Simulation und MoveIt-Bahnplanung verwendete URDF-Modell der statischen Szene.

4 Beschreibung des Gesamtsystems

4.1 Bilderkennung

4.1.1 Tassenerkennung

Das Ziel des Algorithmus zur Tassenerkennung ist es eine möglichst genaue quaderförmige Bounding Box für die Tasse zu errechnen. Der Algorithmus gliedert sich in folgende grobe Schritte:

- Detektion der Tasse in den Bilddaten
- Segmentierung der Tasse aus der Punktwolke
- Berechnung der Tassenorientierung.

Die einzelnen Schritte sollen in den nächsten Abschnitten detailliert beschrieben werden.

Detektion der Tasse in den Bilddaten Die Detektion der Tasse basiert auf der Klassifikation von Bildsegmenten mittels einer Support Vector Machine (SVM). Hierfür wird im Rahmen eines Sliding Window Ansatzes das ganze Bild durchlaufen und an jeder Stelle ein quadratisches Bildsegment extrahiert. Jedes dieser Bildsegmente soll im nächsten Schritt nun klassifiziert werden, wobei es sich hier um ein binäres Klassifikationsproblem („Tasse“/„Keine Tasse“) handelt. Die Klassifikation jedes Segments erfolgt hierbei anhand zuvor extrahierter Histogram of Oriented Gradient (HOG) Features. Als Resultat gibt liefert die SVM eine Liste an Tupeln, welche die Wahrscheinlichkeiten der Klassenzugehörigkeiten eines jeden Segments enthalten. Die zuvor beschriebene Klassifikation wird hierbei in 8 Threads ausgeführt, was die benötigte Zeit in etwa halbiert. Für die Weiterverarbeitung werden nun die Bounding Boxen ausgewählt, welche gemäß der Klassifikation mit einer Wahrscheinlichkeit von mehr als 0.9 zur Klasse Tasse gehören. Zudem werden in diesem Schritt auch noch nah zusammenliegende Boundingboxen, welche vermutlich zum selben Objekt gehören, durch das Clustering ihrer Zentren mittels DBSCAN zusammengefasst.

Segmentierung der Tasse aus der Punktwolke Nachdem die Tasse in den Bilddaten erkannt wurde sollen nun die zugehörigen Punkte aus der Punktwolke segmentiert werden. Bevor die eigentlich Segmentierung durchgeführt wird, werden aber noch die zuvor berechneten Bounding Boxen mithilfe von Daten aus der Punktwolke auf Plausibilität überprüft. Hierbei wird überprüft, ob der Mittelwert der z-Koordinate der zur Bounding Box gehörenden Punkte weniger als 10cm von der Tischhöhe abweicht. So können beispielsweise zum Roboterarm gehörige Fehlerkennungen frühzeitig ausgeschlossen werden. Anschließend beginnt die eigentliche Segmentierung. Hierfür werden die Mittelwerte aller Koordinatenachsen des die Tasse enthaltenden Segments berechnet. Anschließend wird ein Bereich um die errechneten Koordinaten-Mittelwerte aus der Punktwolke herausgeschnitten und die Tischplatte aus der Punktwolke entfernt. Das herausgeschnittene

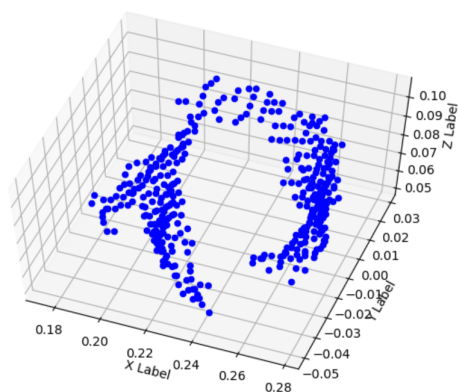


Abbildung 7: Segmentierung der Tasse aus der Punktwolke

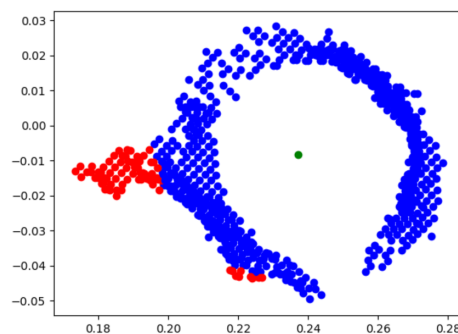


Abbildung 8: Orientierungserkennung mittels LOF

Segment der Punktwolke enthält nun zwar schon die Tasse, ggf. sind aber noch einzelne andere Punkte enthalten. Um diese von der Tasse zu trennen wird wiederum das DCSCAN Clustering Verfahren angewandt und der größte gefundene Cluster als Tasse ausgewählt (siehe Abbildung 7).

Berechnung der Tassenorientierung Aus dem vorhergehenden Schritt erhalten wir die Punkte der von der Kinect gelieferten Punktwolke, die zur Tasse gehören. Aus diesen soll nun die Orientierung des Tassenhenkels bestimmt werden. Unser Algorithmus stützt sich hier auf die Annahme, dass die zum Henkel gehörenden Punkte in einem deutlich weniger dicht besetzten Gebiet der Punktwolke liegen als die restlichen zur Tasse gehörigen Punkte. Um dies auszunutzen wenden wir den sogenannten Local Outlier Factor (LOF) Algorithmus an, welcher häufig zur Ausreißer Detektion verwendet wird indem er die Dichte eines Punktes mit der seiner Nachbarn vergleicht. Die vom Algorithmus zurückgelieferten Ausreißer, welche eine geringere Dichte aufweisen als ihre Nachbarn betrachten wir im folgenden als zum Henkel gehörende Punkte (siehe Abbildung 8). Um aus ihnen die Orientierung des Henkels zu bestimmen genügt es, das Zentrum der Tassenpunktwolke mit dem Durchschnittswert der zum Henkel gehörenden Punkte zu verbinden und den Winkel zwischen der y-Achse und dem so berechneten Vektor zu bestimmen.

4.1.2 Automatische Kamerakalibrierung

In diesem Abschnitt soll der durch den in 3.2.1 beschriebene Arbeitsprozess entstandene Algorithmus für die automatische Kamerakalibrierung beschrieben werden. Das Ziel des Algorithmus ist es, die Kanten der rechten Seite des Tisches (siehe ??) zu detektieren um eine Transformation vom Kamera- ins Weltkoordinatensystem zu berechnen. Der Ablauf des Algorithmus ist in ?? dargestellt. Zunächst wird der Canny-Algorithmus zur Kantendetektion auf das Bild angewendet. Anschließend werden die Kantenpixel mittels Hough-Transformation in einen dualen Parameterraum transformiert, in dem durch die Suche nach Häufungen zu den Tischkanten korrespondierende Geraden extrahiert

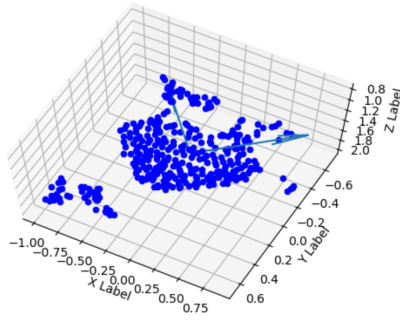


Abbildung 9: Ergebnis der automatischen Kamerakalibrierung

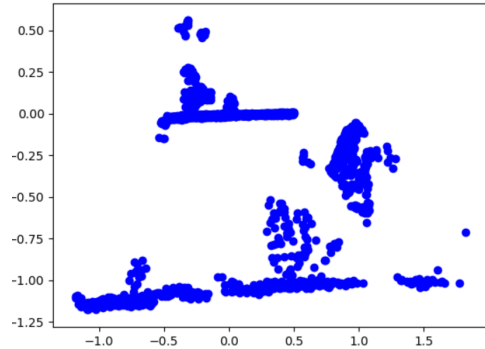


Abbildung 10: Oberfläche bei der manuellen Kalibrierung

werden können. Anschließend werden die zu den Tischkanten gehörigen Punkte aus der Punktwolke extrahiert. Aus diesen Punkten werden mittels des RANSAC Algorithmus Geraden aus der Punktwolke segmentiert. Zudem wird auch die Tischplatte mittels des RANSAC Algorithmus aus der Punktwolke segmentiert. Die gefundenen Geraden werden unter Zuhilfenahme des ermittelten Normalenvektors der Tischplatte so rotiert, dass sie parallel zur x-y-Ebene verlaufen. Die zum Roboterarm zeigende Kante wird als y-Achse verwendet, während die zweite Kante als x-Achse fungiert. Die z-Achse ergibt sich aus dem Normalenvektor. Den Ursprung des Weltkoordinatensystems bildet das Zentrum des Tisches, welches aus x- und y-Achse sowie den bekannten Maßen des Tisches berechnet werden kann. Eine Darstellung der automatisch detektierten Achsen ist in Abbildung 9 dargestellt. Sollte das Ergebnis nicht zufriedenstellend sein, so kann die Kalibrierung auch manuell über eine simple Benutzeroberfläche nachkorrigiert werden wie in Abbildung 10 dargestellt.

4.2 Bahnplanung & Greifen

4.2.1 Bahnplanung

Die gesamte Bahnplanungskomponente besteht aus zwei Paketen:

1. *cup_acceptor_planning*:
Erhält von der Highlevel-Steuerung die Posen der Tasse und des Turtlebots. Setzt Zwischenwegpunkt für die Trajektorienbestimmung. Regelt den Ablauf zwischen der Steuerung des UR5 und des PG70.
2. *cup_acceptor_pathing*:
Initialisiert MoveIt. Erhält von *cup_acceptor_planning* eine Liste von Wegpunkten. Bestimmt mit auszuführende Trajektorie für die gegebenen Wegpunkt und führt sie auf dem UR5 aus.

cup_acceptor_planning stellt die Logik hinter der Bahnplanung dar. Sie startet Actions von *cup_acceptor_pathing*, sowie *cup_gripper* und regelt das Zusammenspiel dieser beiden ROS-Pakete. Der eigentliche Bahnplanungsablauf ist in fünf Etappen unterteilt:

1. Sicherstellung, dass sich der UR5 samt PG70 in Home-Konfiguration befindet.
2. Führung des PG70 mit dem UR5 zur Tasse mit einem Zwischenwegpunkt einige Zentimeter über dem eigentlichen Ziel. Schließen des PG70 zum Greifen der Tasse.
3. Führung des PG70, welcher die Tasse greift, mit dem UR5 zum Turtlebot. Dabei werden zwei Zwischenwegpunkte gesetzt: einer über der Tasse und einer über dem Turtlebot einige Zentimeter über der Tischplatte. Öffnen des PG70 zum Loslassen der Tasse.
4. Führung des PG70 mit dem UR5 von der Turtlebot-Position horizontal in die Höhe bis auf einige Zentimeter über der Tischplatte.
5. Sicherstellung, dass sich der UR5 samt PG70 in Home-Konfiguration befindet.

cup_acceptor_planning stellt eine einzige Action bereit, welche die oben beschriebene Logik ausführt. Der erste und letzte Schritt scheint redundant, ist jedoch wichtig, damit der Roboterarm während der Lokalisierung der Tasse nicht unnötigerweise in das Kinect-Bild hineinragt. Weiterhin sind die Zwischenwegpunkte notwendig, um die Bahnplanung zu optimieren, zu beschleunigen und eine aufrechte Haltung der Tasse auch ohne Pfad-Constraints zu gewährleisten. Außerdem wurde der Tisch in Zonen unterteilt und überprüft, ob sich die Arm-Basis im direkten Pfad zwischen der Tasse und dem Turtlebot befindet. Ist dies nämlich der Fall oder steht die Tasse in bestimmten Zonen, so müssen manuell Umgehungspunkte gesetzt werden. Diese verhindern, dass der EEF des UR5 direkt über die Arm-Basis gefahren wird.

cup_acceptor_pathing ist für die eigentliche Bahnplanung und Ansteuerung der Hardware mit MoveIt zuständig. Da wir die Bahnplanung in Python implementierten, mussten wir das ROS-Paket „Moveit_Commander“ zu unserem Projekt hinzufügen. *cup_acceptor_pathing* erhält also eine Liste von EEF-Posen, über die es eine Trajektorie berechnen soll, welche vom UR5 ohne Kollisionen abgefahren werden kann. Dazu wird MoveIt verwendet. MoveIt stellt dabei einen kollisionsfreien Pfad sicher, da es auf unserem Modell plant, welches neben dem UR5 auch noch den PG70 samt Greiferfinger und Zwischenstück sowie den Tisch beinhaltet. Auch die IK wird von MoveIt gelöst. Es wird also online eine Abfolge von Gelenkpositionen für den UR5 bestimmt und abgefahren. *cup_acceptor_pathing* beinhaltet dabei zwei Actions. Die *plan_and_execute* berechnet mit *compute_cartesian_path* über die Wegpunkt eine Trajektorie und führt sie aus, wohingegen *go_home* den Roboterarm in die Home-Konfiguration führt. Dabei wird *set_target_pose* und *plan* verwendet.

Die gesamte Bahnplanungskomponente besitzt dabei außerdem eine vollständige Fehlererkennung, sowie Fallback-Protokolle für die einzelnen Etappen im Ablauf. Die beiden Actions von *cup_acceptor_pathing* können neben dem Ergebnis-Code 200 für Erfolg

auch noch 500 für Misserfolg zurück liefern. In diesem Fall fängt *cup_acceptor_planning* dies ab und weiß, dass die Bahnplanung fehlschlug. Je nachdem, in welchem Abschnitt sich der UR5 mit der Tasse befindet, wird ein dementsprechendes Protokoll abgespielt, nachdem sich der Roboterarm wieder in Home-Konfiguration und die Tasse wieder in initialer Pose auf dem Tisch befindet. Zusätzlich stellt die Planungskomponente am Anfang der Durchführung sicher, dass sowohl die Tasse als auch der Turtlebot erreichbar ist. Dies wird einfachheitshalber über die kartesische Distanz überprüft. Sollte sich eins von beidem zu weit entfernt befinden, so wird der Ergebnis-Code 301 bzw. 302 an die Highlevel-Steuerung zurückgeliefert.

4.2.2 Greifen

Die Hardware besteht aus dem Greifer Schunk PG70 mit 3D-gedruckten Fingern, welche durch ein Muster aus Haushaltsgummis erweitert wurden. Die Kommunikation mit dem Greifer findet über eine CAN Schnittstelle statt, die per USB am Shuttle Computer angeschlossen ist. Ein externes Netzteil versorgt den Greifer mit Strom.

Der Greifer wird durch den „*schunk_canopen_driver*“ des FZIs angesteuert, welches eine Action zur Konfiguration der Backenabstände anbietet. Das Paket wird von „*cup_gripper*“ verwendet, ein ROS Paket mit den beiden Actionen *GrabCup* und *ReleaseCup*. *GrabCup* akzeptiert ein *cup_diameter* als Parameter, um den Abstand der Finger auf die jeweilige Tasse anzupassen. Innerhalb des Packets wird der Greifer durch „*Gripper*“ abstrahiert, was einen vereinfachten Tausch der Hardware ermöglichen würde.

4.3 High-level Steuerung & Kommunikation

Die Highlevel Steuerung besteht aus einer Komponenten „*cup_acceptor_manager*“, die sowohl die Koordination der Teile innerhalb der Gruppe, als auch die Kommunikation mit dem Super Manager koordiniert. Dazu stellte diese Komponente eine Action mit dem Namen *AcceptCup* zur Verfügung, die nach einer erfolgreichen Bilderkennung die Tasse greift und auf den Turtlebot stellt. Der Aktivitätenfluss gestaltet sich wie folgt:

1. Abonnement auf Ergebnisse der Localization Komponente
2. *AcceptCup* Action wird aufgerufen
3. Überprüfe Bounding Boxes für Tasse und Turtlebot. Ohne Ergebnisse wird gewartet und Fortschritt gemeldet
4. Aufrufen der *Gripper Arm* Action
5. Result zurück geben

Der besondere Augenmerk lag auf einer korrekten Implementierung der Schnittstellen zu den anderen Gruppen.

5 Evaluation & Ausblick

5.1 Bilderkennung

5.1.1 Tassenerkennung

Während die Tassenerkennung insgesamt recht robust funktioniert (Hier ggf. noch quantitative oder qualitative Analyse) gibt es durchaus noch Potential zur Verbesserung.

So könnte der für die Klassifikation der Tasse verwendete SVM Klassifikator mit variablen Trainingsdaten trainiert werden um die Generalisierung des selbigen zu verbessern und die Erkennung auch unter schwierigen bzw. varriierenden Lichtverhältnisse und mit verschiedenen Tassen robuster zu machen. Außerdem könnte der Sliding Window Ansatz, der im Moment als Standardverfahren für die Gewinnung der zu klassifizierenden Bildsegmente aktiv ist mit einem Interest Point Detektor ersetzt bzw. kombiniert werden um die Anzahl der Segmente zu reduzieren und die Erkennung damit zu beschleunigen.

Weiterer Spielraum zur Verbesserung besteht bei der Erkennung der Orientierung der Tasse. Wie beschrieben versucht der Algorithmus die Orientierung des Henkels anhand von dessen Position in der Punktwolke zu erkennen. Es stellte sich jedoch bei Tests heraus, dass der Henkel, insbesondere wenn sich die Tasse an Randpositionen des Tisches befindet, in der Punktwolke kaum auszumachen ist, weshalb die Erkennung hier häufiger versagt. Ggf. könnte man die Orientierung stattdessen mit Hilfe der Bilddaten z.B. über das Training eines Neuronalen Netzes zu realisieren.

5.1.2 Turtleboterkennung

5.1.3 Automatische Kamerakalibrierung

Generell funktioniert der beschriebene Ansatz recht gut, jedoch kommt es gelegentlich zu Fehlerkennungen bei der Detektion der Kanten des Tisches, welche zur Berechnung einer falschen Transformation vom Kamera- ins Weltkoordinaten System führen. Hier könnten weitere Überprüfungen auf die Plausibilität des Detektionsergebnisses Abhilfe schaffen. Ein weiterer Nachteil des Ansatzes besteht darin, dass er auf den Labortisch zugeschnitten wurde. Für den Einsatz in einer anderen umgebung müsste er entsprechend angepasst werden. Ggf. wäre Alternativ eine Kalibrierung mittels eines entsprechenden Kalibrierungsobjekts bzw. Musters denkbar um die Kalibrierung flexibler zu gestalten.

5.2 Bahnplanung & Greifen

5.2.1 Bahnplanung

Die erzielten Trajektorien waren für unseren Anwendungszweck annähernd optimal. Zwar haben wir zwei Zwischenpunkte über den Zielen hinzugefügt, zwischen den einzelnen vorgegebenen Wegpunkte wurden allerdings die direkten Pfade gefunden und auch der Kaffee wurde dabei nicht verschüttet.

Obwohl unsere Bahnplanung vollständig zur Laufzeit stattfand, wurde die Trajektorien

ohne merkbare Verzögerung berechnet und ausgeführt. Die Bahnplanung eines einzelnen Abschnittes hat dabei nie länger als 500ms für die Berechnung benötigt. Längere Verzögerungen in den Videoaufnahmen lassen sich mit der zusätzlichen Verzögerung von 500ms erklären, welche manuell eingefügt wurde, damit MoveIt genügend Zeit hat, um den momentanen Zustand des Roboterarms zu aktualisieren.

Bei der bisher verwendeten *compute_cartesian_path*-Methodik gibt es jedoch einen entscheidenden Nachteil. Die Bahnplanung umfährt den Fuß des UR5 nicht eigenständig, sondern fährt gegebenenfalls den EEF über die Arm-Basis und beendet dann die Durchführung, da für die restliche Strecke keine Trajektorie mehr berechnet werden konnte. Bisher mussten wir also in manchen Situationen noch vordefinierte Umgehungspunkte hinzufügen. *compute_cartesian_path* scheint nämlich den Arbeitsraum des UR5, welchen man ebenfalls für MoveIt anpassen kann, nicht bei der Planung zu berücksichtigen. Den Bereich über dem Roboterfuß als unzulässig zu deklarieren lieferte daher nicht den gewünschten Effekt.

Mit mehr Zeit würden wir als nächstes versuchen, die Bahnplanung gänzlich ohne Zwischenpunkte durchzuführen. Um dabei trotzdem die Tasse aufrecht zu halten, müsste man Pfad-Constraints hinzufügen und die Bahnplanung nur über die *set_target_pose*-Methode durchführen.

Die so berechneten Trajektorien müssten, wie wir feststellen konnten, weiter optimiert werden, um zumindest annähernd die direkteste Bahn abzufahren. Mit der bisherigen Implementierung der *set_target_pose*-Variante sind die Trajektorien nämlich oft noch zu ausschweifend. Da bei der Optimierung jedoch die Kinematik des Armes berücksichtigt werden muss, um die Machbarkeit zu gewährleisten und Kollisionen zu vermeiden, ist dies kein trivialer Nachbearbeitungsschritt.

Im bisherigen Stand der Bahnplanung ist außerdem keine Kollisionserkennung mit zusätzlichen Objekten der Umgebung implementiert. Diese Erweiterung wäre jedoch recht einfach, da das Planning Interface von MoveIt den Zustand der Szene überwacht. Könnte die Lokalisierung also für jedes Objekt auf dem Tisch und um den Tisch eine annähernd exakte Bounding Box bereitstellen, so müsste man diese nur über eine Methode zur Szene hinzufügen und schon würden sie bei der Bahnplanung mitberücksichtigt werden. Des Weiteren kann man genauso Bounding Boxen an den Roboterarm anbringen. Man könnte also auch eine Kollision der gegriffenen Tasse mit der Umgebung vermeiden.

5.2.2 Greifen

Das beschriebene System wurde auf Grifffestigkeit geprüft. Dazu wurden drei unterschiedliche Ladungen einer Tasse getestet: Leer, Schrauben und Wasser (Füllstand: circa 2cm unter Rand). Die Bewegung der Tasse durch den Arm führte weder zu Verrutschen, noch zu Brechen eines Fingers. Somit kann die Tasse sicher transportiert werden, ohne Verlust des Tasseninhalts.

Jedoch bietet das implementierte System einen Nachteil: Es können nur bekannte

Tassen gegriffen werden. Leider war es uns nicht möglich die Ergebnisse der Objekterkennung für eine zuverlässige Aussage über die Tassenbreite zu verwenden. Besser wäre ein Drucksensor an den Fingern oder ein anderer Greifer, der den Druck messen kann. Dieses Feedback lässt den Griff an beliebige Tassen anpassen.

5.3 High-level Steuerung & Kommunikation

Zur Evaluation der Highlevel Steuerung wurden die Schnittstellen auf Funktionalität manuell getestet. Desweiteren wurden einzelne Komponenten deaktiviert, um die korrekte Rückgabe von Fehlern und Fortschritt zu prüfen. Die Verwendung eines zentralen Paketes, welches die einzelnen Schritte startet, trennt die verschiedenen Zuständigkeiten und sorgt für einen klaren Ausführungsfluss.

Literatur

- (2018). Moveit! setup assistant tutorial. http://docs.ros.org/hydro/api/moveit_setup_assistant/html/doc/tutorial.html. Accessed: 2018-02-08.
- Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.
- Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.
- Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2.
- Lavalle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical report.
- Sucan, I. (2018). moveit_commander. http://wiki.ros.org/moveit_commander. Accessed: 2018-02-08.
- Sucan, I. A. and Chitta, S. (2018). Moveit! motion planning framework. <http://moveit.ros.org/>. Accessed: 2018-02-08.