Hochschule Karlsruhe
University of
Applied Sciences

Mobile Robotics

Robogistics – Robots in Logistics

(RKIM221)

Fakultät Maschinenbau und Mechatronik - MMT

Mathias Fuhrer

Robin Wolf

Maurice Droll

Andreas Schmitt

Leo Schäfer

Dozent: Prof. Dr.-Ing. Christian Wurll

SS 2024

26.07.2024

I. Eigenständigkeitserklärung

Wir versichern hiermit wahrheitsgemäß, die vorliegende Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles einzeln kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

II. Inhaltsverzeichnis

III. Abbildungsverzeichnis

IV. Tabellenverzeichnis

1. Übersicht der Projektarbeit

1.1 Einführung

Die Projektarbeit ist Teil der Vorlesung „Robogistics“ der Hochschule Karlsruhe im Studiengang „Robotik und Künstliche Intelligenz in der Produktion“. Die Vorlesung beinhaltet Themen wie beispielsweise Einsatzgebiete von Robotern in Warenhäuser oder die Planung und Simulation von Roboter-Applikationen in der Logistik.

1.2 Aufgabenstellung

Die Aufgabenstellung dieser Projektarbeit beschreibt die vollumfängliche Planung und Umsetzung einer Roboter-Applikation, die in der Logistik bzw. der Servicerobotik zum Einsatz kommen könnte. Um diese Aufgabenstellung ins studentische Umfeld zu transferieren, wurde entschieden, autonom ein Bierpong-Spiel auf einem Tisch aufzubauen. Während der Bearbeitung der Aufgabe soll eine Kamera ins Robotersystem integriert werden.

Im Rahmen der Planung dieser Projektarbeit wurde sich für die Umsetzung einer Applikation mit dem Neobotix MMO-500 Roboter entschieden. Der Neobotix ist eine autonome, mobile Plattform mit montiertem Manipulator von Universal Robots.

1.3 Konzept

Zu Beginn der Projektarbeit wurde ein Konzept ausgearbeitet, um den Rahmen und die Methodik festzulegen. Die Idee des Projekts ist es, den Neobotix-Roboter als Serviceroboter auszulegen. Dieser soll Gegenstände transportieren und an bestimmten Positionen positionieren. Der Gegenstand soll ein handelsüblicher Plastikbecher sein. Die Navigation der Plattform und die Bewegungsbefehle des Manipulators sollen durch einen übergeordneten Behavior Tree gesteuert werden. Die Referenzierung des Manipulators an den Navigationszielen (Lagebestimmung) soll durch Aruco-Codes erfolgen. Die Bildverarbeitung erfolgt mit einer Tiefenbildkamera.

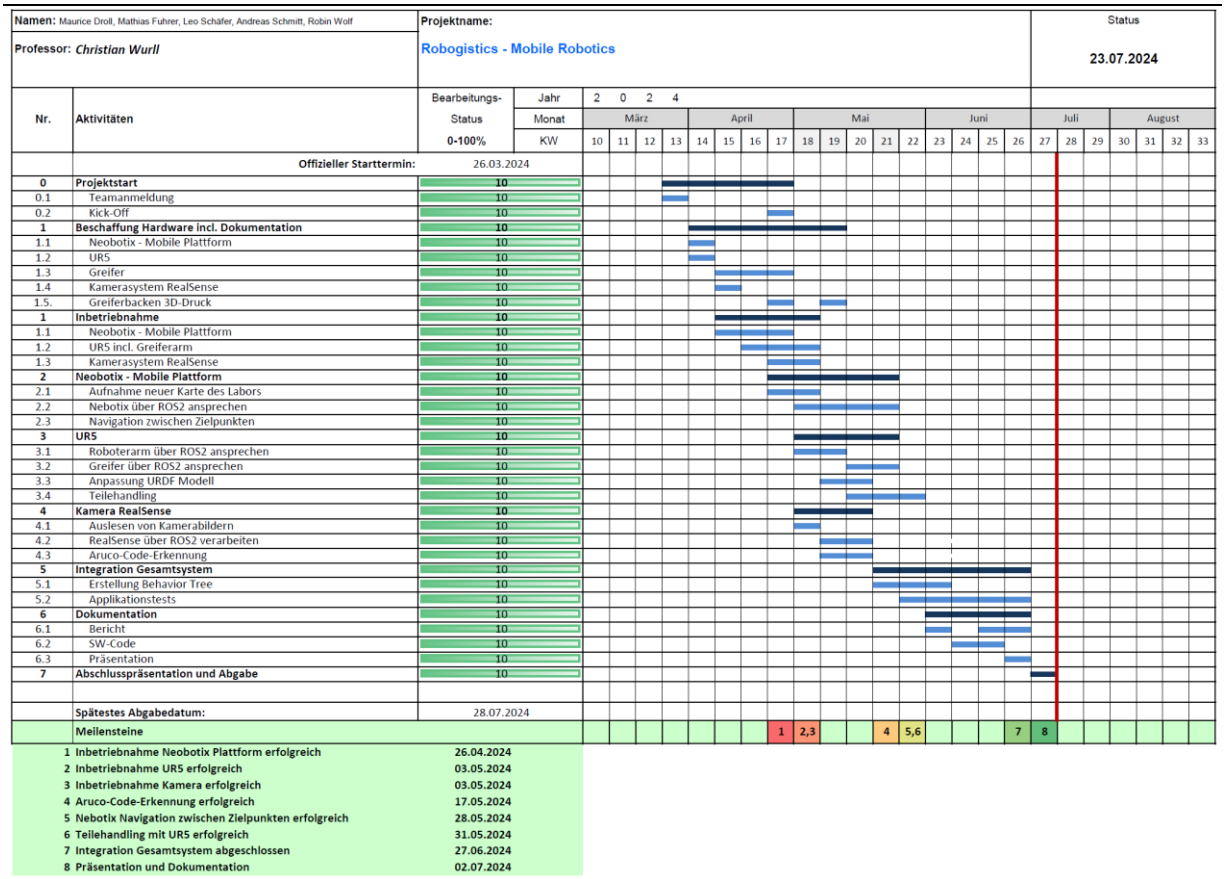
1.4 Projektdurchführung

Um eine erfolgreiche Durchführung des Projekts zu erreichen, wurden für die Bearbeitung des Projekts folgende Meilensteine definiert (siehe Tabelle 1). Die Dokumentation des Projekts erfolgt fortlaufend.

Nr.	Meilenstein	Datum
1	Inbetriebnahme Neobotix Plattform (Lokalisierung, Ansteuerung usw.) erfolgreich	26.04.2024
2	Inbetriebnahme UR5 inkl. Greifer erfolgreich	03.05.2024
3	Inbetriebnahme Kamera erfolgreich	03.05.2024
4	Aruco-Code-Erkennung erfolgreich	17.05.2024
5	Neobotix Plattform navigiert zwischen Zielpunkten	28.05.2024
6	Teilehandling mit UR5 erfolgreich	31.05.2024
7	Integration Gesamtsystem abgeschlossen	27.06.2024
8	Präsentation	02.07.2024

Tabelle 1-1: Meilensteine des Projekts

Zur Erreichung der definierten Meilensteine wurde ein strukturierter Terminplan in Form eines Gantt-Diagramms ausgearbeitet und fortlaufend überwacht:



Namen: Maurice Droll, Mathias Führer, Leo Schäfer, Andreas Schmitt, Robin Wolf		Projektname:		Status
Professor: *Christian Wurll*		Robogistics - Mobile Robotics		23.07.2024

Nr.	Aktivitäten	Bearbeitungs-Status 0-100%	Jahr / Monat / KW	2024	März				April				Mai					Juni				Juli				August				
				10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	Offizieller Starttermin:	26.03.2024																											
0	**Projektstart**	10																											
0.1	Teamanmeldung	10																											
0.2	Kick-Off	10																											
1	**Beschaffung Hardware incl. Dokumentation**	10																											
1.1	Neobotix - Mobile Plattform	10																											
1.2	UR5	10																											
1.3	Greifer	10																											
1.4	Kamerasystem RealSense	10																											
1.5.	Greiferbacken 3D-Druck	10																											
1	**Inbetriebnahme**	10																											
1.1	Neobotix - Mobile Plattform	10																											
1.2	UR5 incl. Greiferarm	10																											
1.3	Kamerasystem RealSense	10																											
2	**Neobotix - Mobile Plattform**	10																											
2.1	Aufnahme neuer Karte des Labors	10																											
2.2	Nebotix über ROS2 ansprechen	10																											
2.3	Navigation zwischen Zielpunkten	10																											
3	**UR5**	10																											
3.1	Roboterarm über ROS2 ansprechen	10																											
3.2	Greifer über ROS2 ansprechen	10																											
3.3	Anpassung URDF Modell	10																											
3.4	Teilehandling	10																											
4	**Kamera RealSense**	10																											
4.1	Auslesen von Kamerabildern	10																											
4.2	RealSense über ROS2 verarbeiten	10																											
4.3	Aruco-Code-Erkennung	10																											
5	**Integration Gesamtsystem**	10																											
5.1	Erstellung Behavior Tree	10																											
5.2	Applikationstests	10																											
6	**Dokumentation**	10																											
6.1	Bericht	10																											
6.2	SW-Code	10																											
6.3	Präsentation	10																											
7	**Abschlusspräsentation und Abgabe**	10																											
	Spätestes Abgabedatum:	28.07.2024																											
	Meilensteine										1	2,3			4	5,6				7	8								

1	Inbetriebnahme Neobotix Plattform erfolgreich	26.04.2024
2	Inbetriebnahme UR5 erfolgreich	03.05.2024
3	Inbetriebnahme Kamera erfolgreich	03.05.2024
4	Aruco-Code-Erkennung erfolgreich	17.05.2024
5	Nebotix Navigation zwischen Zielpunkten erfolgreich	28.05.2024
6	Teilehandling mit UR5 erfolgreich	31.05.2024
7	Integration Gesamtsystem abgeschlossen	27.06.2024
8	Präsentation und Dokumentation	02.07.2024

Abbildung 1-1: Zeitplan des Projekts

2. Übersicht der Systemkomponenten

2.1 Neobotix Plattform MMO-500

Die Neobotix MMO-500 ist eine modulare mobile Roboterplattform, die speziell für den industriellen Einsatz entwickelt wurde. Sie bietet eine robuste Bauweise und ist mit leistungsstarken Antriebsmotoren ausgestattet. Präzise Bewegungen in alle Richtungen sind übergangslos aufgrund der Mecanum-Räder möglich. Zudem verfügt die Plattform über fortschrittliche Sensorik und Steuerungssysteme (Lidarscanner mit Datenschnittstelle), die eine autonome Navigation und sichere Interaktion mit der Umgebung gewährleisten. Die offene Architektur der MMO-500 erlaubt es, unterschiedliche Aufbauten und Erweiterungen zu integrieren, was sie zu einer vielseitigen Lösung für diverse Einsatzgebiete macht [1].

2.2 UR 5

Der UR5 ist ein kollaborativer Roboterarm von Universal Robots, der sich ideal für den Einsatz auf mobilen Roboterplattformen eignet. Mit seiner Traglast von 5 kg und einer Reichweite von 850 mm ist er vielseitig einsetzbar und kann Aufgaben wie Materialhandling, Montage und Qualitätskontrolle übernehmen.

Die einfache Programmierung und schnelle Einrichtung machen ihn besonders attraktiv für den Einsatz in verschiedenen industriellen Anwendungen. Darüber hinaus sorgen seine Sicherheitsfunktionen dafür, dass er sicher mit Menschen und anderen Maschinen zusammenarbeiten kann, was die Produktivität und Effizienz in der Produktion steigert [2].

Der UR5 ist auf der Neobotix-Plattform installiert und verdrahtet. Hierdurch fungiert er als mobiler kollaborativer Manipulator und kann flexibel eingesetzt werden.

2.3 Tiefenkamera Realsense Intel D415

Die Intel RealSense D415 ist eine kompakte und vielseitige Tiefenkamera, die speziell für Entwickler und Wissenschaftler entworfen wurde. Mit ihrem präzisen Tiefensensor ermöglicht sie die Erfassung von detaillierten 3D-Bildern und -Videos. Die D415 verwendet eine Kombination aus einer RGB-Stereokamera und einem Infrarot-Musterprojektor, um die Tiefe der Umgebung zu messen.

Durch die hohe Auflösung von 1280 x 720 Pixeln für die Tiefenerfassung und 1920 x 1080 Pixeln für die Farberfassung eignet sie sich ideal für die Anwendungen in der Robotik. Aufgrund einer einfachen Integration kann die Kamera mittels den verfügbaren Software Development Kits und der API-Unterstützung unkompliziert in eigene Projekte eingebunden werden. Insbesondere der RealSense Treiber lässt sich ohne Weiteres auf Linux und damit in ROS2 einbinden.

Aufgrund der kleinen Abmessungen und des geringen Gewichtes ist diese Tiefenbildkamera ideal für mobile Anwendungen geeignet [3].

2.4 Greifer Zimmer GEP5010IO-00-A

Der Zimmer GEP5010IO-00-A ist ein leistungsstarker elektrischer Parallelgreifer, der sich durch seine Präzision und Zuverlässigkeit auszeichnet. Er bietet eine Greifkraft von bis zu 600 N und eine maximale Hubweite von 10 mm pro Backe, was ihn ideal für anspruchsvolle Industrieanwendungen macht. Der Greifer ist kompakt und robust gebaut, wodurch er auch in engen und herausfordernden Umgebungen eingesetzt werden kann. Seine hohe Wiederholgenauigkeit und die flexible Steuerbarkeit machen ihn zu einer optimalen Lösung für präzise und schnelle Greifprozesse in der modernen Produktion. Durch seine Masse von 1.6 kg ist er leicht genug, um am vorhandenen Roboterarm eingesetzt werden zu können [4].

Der Greifer wurde über die die reguläre In-/Output Schnittstelle mit dem UR verbunden. Hierdurch wird der Greifer mit seiner Betriebsspannung von 24 Volt versorgt. Ebenfalls wurde die Signalleitung über diese Schnittstelle verdrahtet, was das Steuern der Greiferbacken ermöglicht. Der gewünschte Greifhub wurde direkt am Greifer eingestellt, Zur mechanischen Befestigung wurde eine Adapterplatte im CAD konstruiert und im 3D Druck Labor gedruckt, um eine sichere Verbindung zwischen dem Roboterflansch, der Kamerahalterung und dem Greifer zu gewährleisten.

2.5 Greiferbacken

Aufgrund der geringen Oberflächenhaftung an den Plastikbechern und deren komplex zu handhabenden Geometrieeigenschaften wurden verschiedene Konzepte für Prototypen der Greiferbacken eruiert. Diese wurden eigenständig im CAD konstruiert und im 3D Druck Labor am LTC gedruckt.

Der erste Ansatz, war die Becher von innen zu greifen. Dafür wurden Greiferbacken mit der Innenform der Becher konstruiert (siehe Abbildung 2-1 und Abbildung 2-4). Durch das Greifen von innen ist es möglich die Becher nebeneinander zu positionieren ohne Kollisionen des Greifers mit nebenstehenden Bechern herbeizuführen. Nach

ersten Tests wurde dieses Konzept verworfen, da es sich als problematisch herausstellte eine ausreichende Haltekraft zu erzeugen, ohne die Becher zu verformen.

Beim zweiten Konzept wurde der überstehende Rand der Becher zur Aufnahme genutzt. Die zwei Greiferbacken sind am Flansch jeweils über eine Schraube fixiert. Ein Becher wird von außen von zwei Seiten am oberen Rand gegriffen. Dabei entspricht die Kontur des Greifers der negativen Halbform des Becherrandes (siehe Abbildung 2-2 und Abbildung 2-3). Zum Greifen wird der gesamte Hub pro Backe des Greifers verwendet.

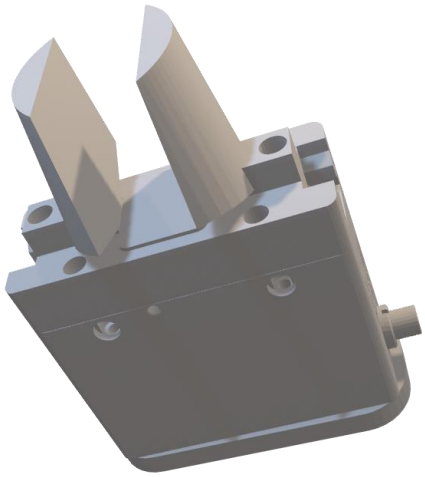


Abbildung 2-1: Greifer - von innen greifen

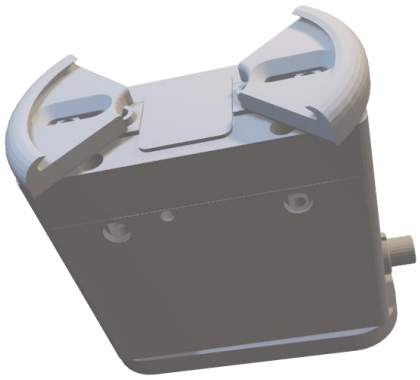


Abbildung 2-2: Greifer - von außen greifen

Abbildung 2-4: Greiferbacken - von innen greifen

Abbildung 2-3: Greiferbacken - von außen greifen

2.6 Bechervereinzelung

Um die Becher einzeln zu greifen, musste eine Vorrichtung konstruiert werden, die die restlichen Becher des Stapels zurückzuhalten. Dafür wurden zwei Konzepte getestet.

Abbildung 2-5: Bechervereinzeler Konzept 1

Das erste Konzept ist in Abbildung 2-5 dargestellt. Die Idee hierbei ist es die Becher durch Zahnräder so weit auseinander zu drücken, dass der vorderste Becher entnommen werden kann, ohne den nächsten Becher mitzuziehen. Nach ersten Tests wurde dieser Ansatz verworfen.

Die Iterationen des zweiten Konzeptes sind in Abbildung 2-7 dargestellt. Hier werden die nachfolgenden Becher durch vorgespannte „Haken“ zurückgehalten. Dieses Konzept erwies sich nach ersten Tests als praxistauglich. Um die Becher von oben aus einer Vorrichtung zu greifen, wird eine Vorspannung benötigt, die die Becher, entgegen der Schwerkraft nach oben gegen die Haken drückt. Hierfür wurde wie in Abbildung 2-6 dargestellt ein Ansatz mit handelsüblichen Gummis getestet. Da die Vorspannung je nach Becheranzahl jedoch stark variiert, wurde dieser Ansatz verworfen.

Abbildung 2-7: Iterationen Bechervereinzeler Konzept 2

Abbildung 2-6: Bechervereinzeler Vorspannung mit Gummi

Stattdessen wurde ein Becherspender konstruiert, bei dem die Becher unten aus der Vorrichtung gegriffen werden können. Der fertige Becherspender ist in Abbildung 2-8 zu sehen. Hierbei wird die Schwerkraft für den Nachschub der Becher genutzt. Die Greiferbacken können den Becher dabei in den der Aussparung des Vereinzelers greifen. Dieses Konzept ermöglicht zudem ein einfaches Nachfüllen der Becher von oben. Der Becherspender ist mit einem Aluprofil an die Neobotix Plattform geschraubt (siehe Abbildung 2-9).

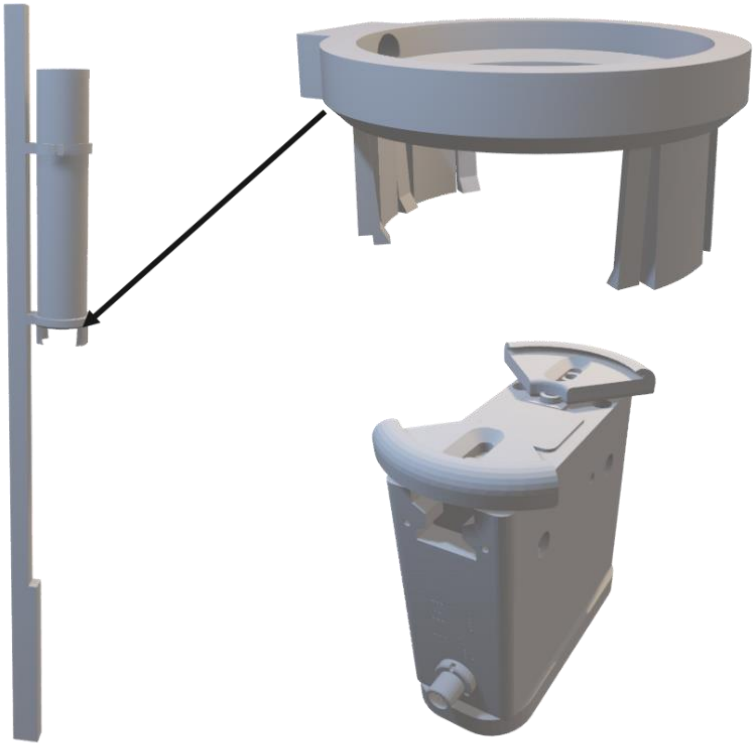


Abbildung 2-8: Konzept Becherspender

Abbildung 2-9: Neobotix mit Becherspender

3. Softwarearchitektur

In diesem Kapitel wird die Softwarearchitektur inklusive ihrer einzelnen Komponenten erläutert. Ausgehend von der übergeordneten Systemübersicht werden die einzelnen Bestandteile in den nachfolgenden Unterkapiteln erläutert.

3.1 Übersicht der Docker Container

Für das Gesamtsystem werden insgesamt fünf Docker Container benötigt, die miteinander kommunizieren können. Aufgrund der erforderlichen Rechenkapazität wurden diese auf den Neobotix-PC, einen mitfahrenden Laptop, sowie zwei externe Laptops, die sich im gleichen Netzwerk befinden, aufgeteilt.

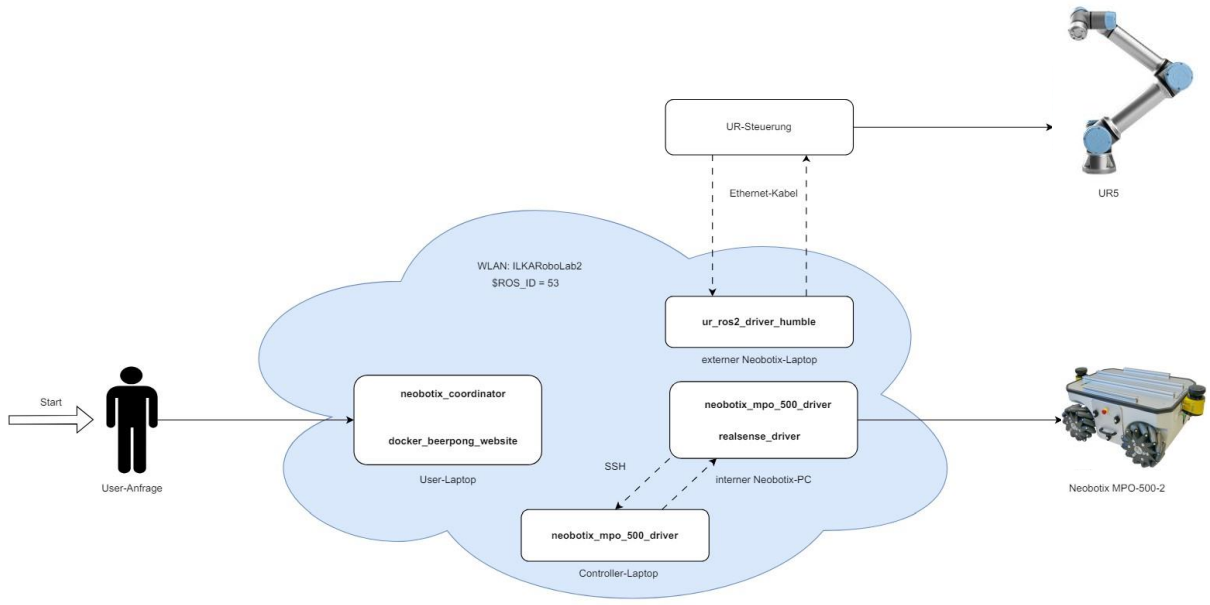


Abbildung 3-1: Systemübersicht Bierpong- Robotersystem

Damit auf Nodes, Topics, Services und Actions in allen Containern gleichermaßen zugegriffen werden kann, müssen sich alle Container mit der gleichen ROS_ID im Netzwerk registrieren (hier ROS_ID=53).

3.1.1 neobotix_coordinator

Der enthaltene Behavior Tree steuert übergeordnet die obigen Komponenten und damit den Gesamtablauf der Applikation. Die darin enthaltenen Elemente fungieren als Clients, die Gegenseite agiert als Server in den ROS2- Services. Die hier beschriebene Gegenseite wird, durch die von allen anderen Containern zur Verfügung gestellten Nodes repräsentiert. Weitere Erläuterungen zum Behavior Tree befinden sich in Kapitel 3.4.

3.1.2 neobotix_mpo500_2_driver

Dieser Docker Container stellt alle Nodes zur Verfügung, die zum Bewegen des Neobotix benötigt werden. Hier ist zum einen die komplette Hardwarekommunikation mit den Motoren und Lidarscannern implementiert, zum anderen kann innerhalb dieses Containers auch das Mapping oder die Navigation gestartet werden.

3.1.3 ur_ros2_driver_humble

Dieser Docker Container stellt alle Nodes zur Verfügung, die zur Steuerung des UR5 benötigt werden und die die komplette Bahnplanungs-Pipeline zur Verfügung stellen.

Neben den Hardware-Treibern, die sich über das UR-Cap „External-Control“ direkt mit der Hardware verbinden, wurde MoveIt als Bahnplanungsframework bereitgestellt.

Damit das MoveIt Framework (über das C++ Move-Group Interface) vom Behavior Tree angesprochen werden kann, wurden spezielle Funktionalitäten wie ptp, lin, joint_space_ptp und weitere Eingriffe in die Bahnplanungspipeline als Service-Server in einem Wrapper-Package implementiert (moveit_wrapper).

3.1.4 realsense_driver

Dieser Docker Container stellt die Verbindung zwischen ROS2 und der RealSense Kamera her. Außerdem sind alle Server, die Funktionen im Umgang mit dem Aruco-

Marker darstellen hier implementiert. Dieser Container stellt also alle Nodes bezüglich der Perzeption des Systems bereit.

3.1.5 docker_website_beerpong

Dieser Docker Container dient zum Hosten der Webseite. Über die auf der Website hinterlegten Karte und die eingezeichneten Tischmöglichkeiten kann der User den gewünschten Tisch selektieren. Die Webseite wurde mit Flask, einem Python Webframework implementiert.

Die Eingabe kann hierbei ebenfalls durch einen ROS2- Service abgefragt werden.

3.2 Systemübersicht

Die Architektur und übergeordneten Zusammenhänge der einzelnen Arbeitsschritte des mobilen Manipulators in der vorgegebenen Aufgabenstellung können wie folgt visualisiert werden:

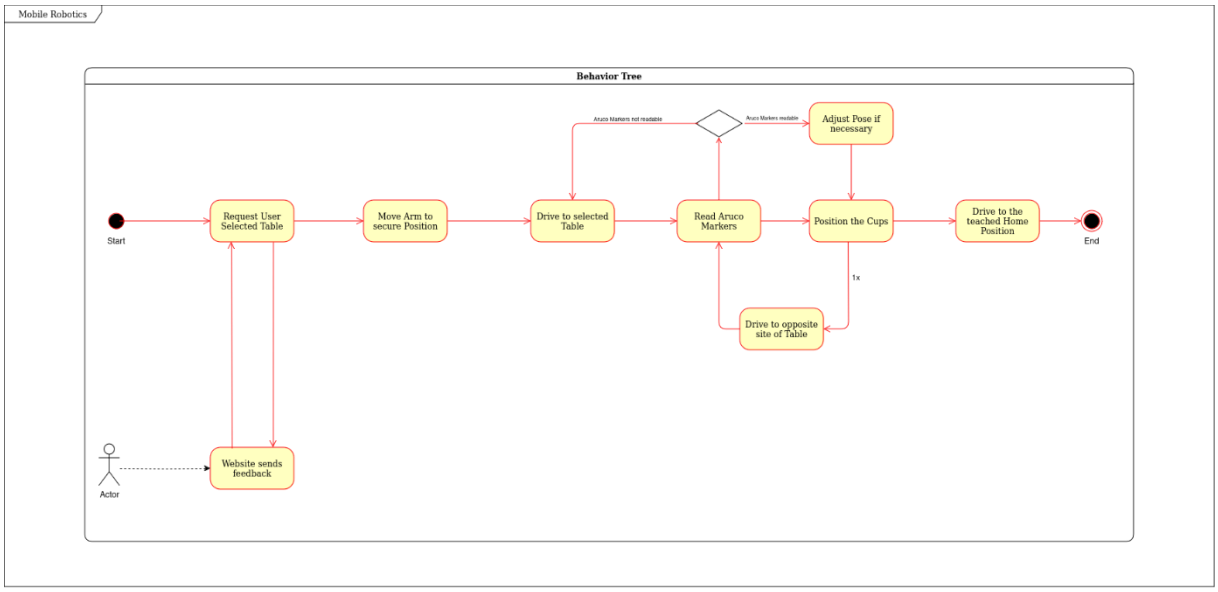


Abbildung 3-2: Zustandsdiagramm, Ablauf der gesamten Applikationssequenz

3.3 Schnittstellen

Die nachfolgende Abbildung zeigt die Schnittstellen, welche vom System für den Behavior Tree bereitgestellt werden, inklusive der Datentypen, Übergabewerten und den auf dem Blackboard gespeicherten Werten. Das Blackboard stellt eine Art Zwischenspeicher für Variablen des Behavior Trees dar (vgl. Kapitel 3.4).

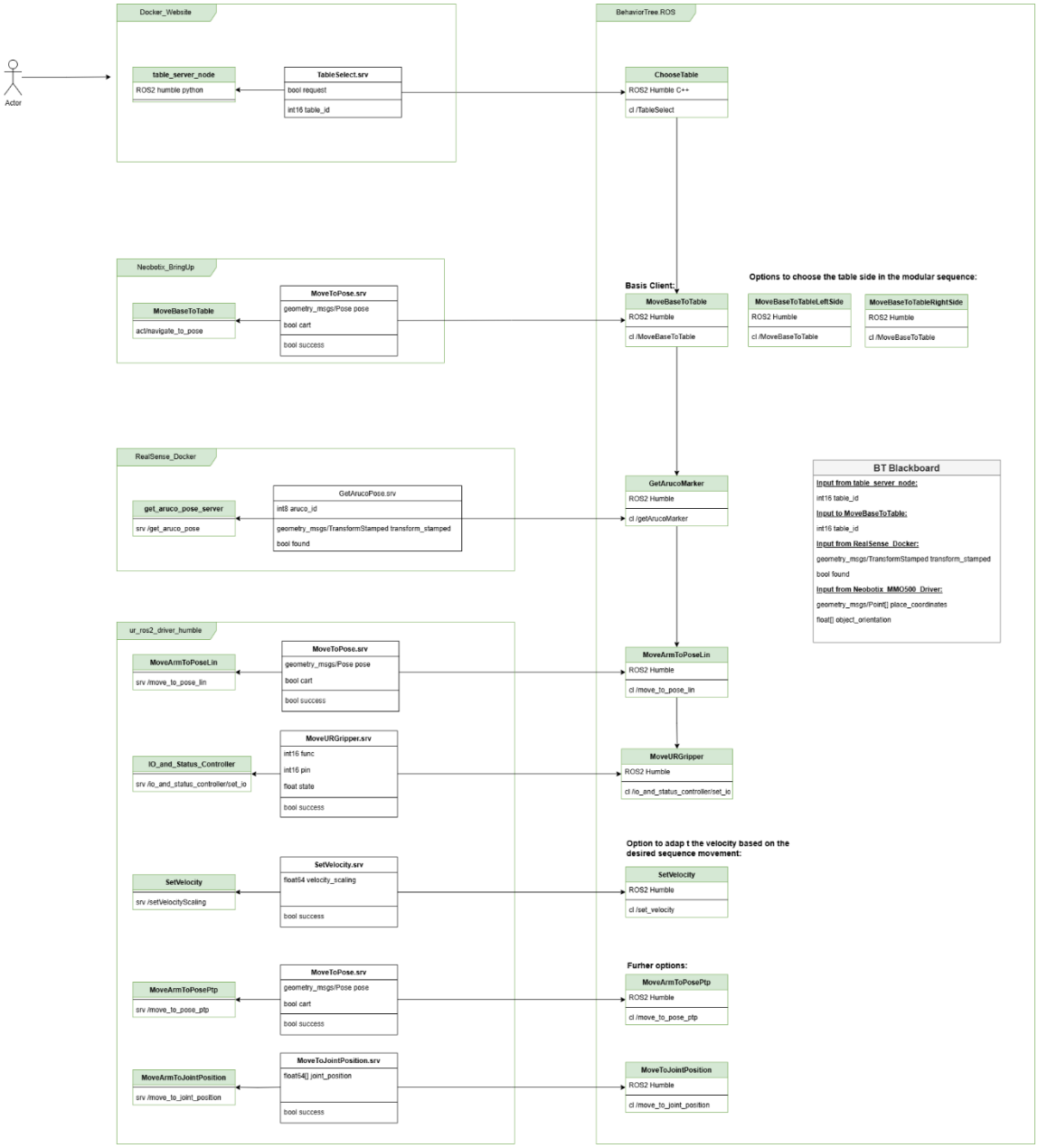


Abbildung 3-3: Übersicht der Schnittstellen

3.4 Behavior Tree

Zur Umsetzung der High-Level Task-Control wird in diesem Projekt das Konzept der Behavior Trees umgesetzt.

Die Behavior Tree Bibliothek beinhaltet ein C++ Framework zur Ablaufsteuerung. Ein Verhaltensbaum (Behavior Tree, BT) dient zur Strukturierung des Wechsels zwischen verschiedenen Aktionen bzw. Aufgaben von einem autonomen Agenten wie beispielweise eines Roboters. Es ermöglicht es komplexe Systeme zu steuern, die modular und reaktiv sind. Hierzu werden einzelne Knoten und Teilbäume mit verschiedenen Hierarchieebenen erstellt, um auf verschiedenen Steuerungsebenen Logiken zu verknüpfen.

Eine Nutzung von Behavior Trees bietet folgende Vorteile:

- Modularität und Wiederverwendbarkeit der Komponenten
- Asynchrone Actions
- Ablaufbäume (Trees) werden zur Laufzeit erstellt, indem eine Interpretersprache verwendet wird (basierend auf XML)
- Logging/Profiling-Infrastruktur, um Zustandsübergänge zu visualisieren, aufzuzeichnen, wiederzugeben und zu analysieren
- Nutzerfreundliche, einfache grafische Zusammenstellung neuer Trees im visuellen Groot Editor [5]

In ROS2 stellt Nav2 den grundlegenden Navigation Stack für autonome mobile Systeme dar. Dessen Implementierung basiert ebenfalls intern auf verschiedenen modularen Behavior Trees.

3.4.1 Neobotix Coordinator

In den vorhergehenden Projekten wurde weder die Sequenzplanung der Neobotix Plattform noch der Manipulator über einen High Level Behavior Tree angesteuert. Stattdessen wurden nativ Skripte mit URCap über eine Ethernet-IP Schnittstelle

nacheinander auf die Steuerung geladen. Die mobile Plattform wurde über eine Simple Commander API (Python) von Nav2 angesprochen.

Um eine Task Control im Neobotix Coordinator via Behavior Trees zu ermöglichen, müssen die einzelnen Server und Clients für die Actions bzw. Services aufgesetzt werden.

Für die High Level Control werden im Behavior Tree Package entsprechende Clients erstellt, mit denen die Server der zugrunde liegenden Actions bzw. Services getriggert werden. Es wurden die folgenden Clients erstellt:

Name des Clients	**Zweck**
ChooseTable	Abfragen des UserInputs der Website
GetArucoPosition	RealSense Kamera
MoveBase	Grundlegende Bewegung der Neobotix Plattform
MoveBaseToTable	Bewegung der Neobotix Plattform inklusive Übersetzung der erhaltenen Table ID in die korrekten Posen und Rotationen in der Map des Labors
MoveArmToPosLin	Lineare Bewegungen der UR-Manipulators (kartesisch)
MoveArmToPosePtp	PTP-Bewegungen des Roboterarms
MoveArmToJoints	Bewegung des Arms im Gelenkwinkelraum
PlaceCup	Positionen der Becher in definierter Anordnung auf Tisch basierend auf Aruco-Marker-Lokalisierung
SetVelocity	Festsetzen der Bewegungsgeschwindigkeit des Arms
MoveURGripper	Öffnen/Schließen des Zimmer Greifers am UR

Tabelle 3-1: Übersicht der Clients im Behavior Tree

Um die Kommunikationskanäle korrekt anzusprechen, müssen die Schnittstellen der Server und Clients abgestimmt, korrekt integriert und über Package-Grenzen hinweg übereinstimmen. Dabei sind vordefinierte Standard-Messages zu präferieren, insofern

die konkrete Applikation keine Ergänzungen oder Änderungen erfordert. Eine Übersicht zu den Schnittstellen kann in Kapitel 3.3 entnommen werden.

Zusätzlich müssen die Nodes in der Behavior Tree Factory (node.cpp) registriert werden, um die Groot Auswahlpalette an verfügbaren Nodes zu ergänzen. Nach einem neuen Launch des Groots können die neuen Nodes in der "graphischen" Modellierung/ Programmierung des Behavior Trees eingesetzt werden.

3.4.2 Ablauf des Behavior Trees

Die Behavior Trees lassen sich visualisieren und editieren mit der graphischen Oberfläche des Groots. Die nachfolgende Abbildung zeigt den MainTree der Neobotix Applikation, der neben regulären Clients für Nodes auch gesamte Subtrees beinhaltet.

MainTree:

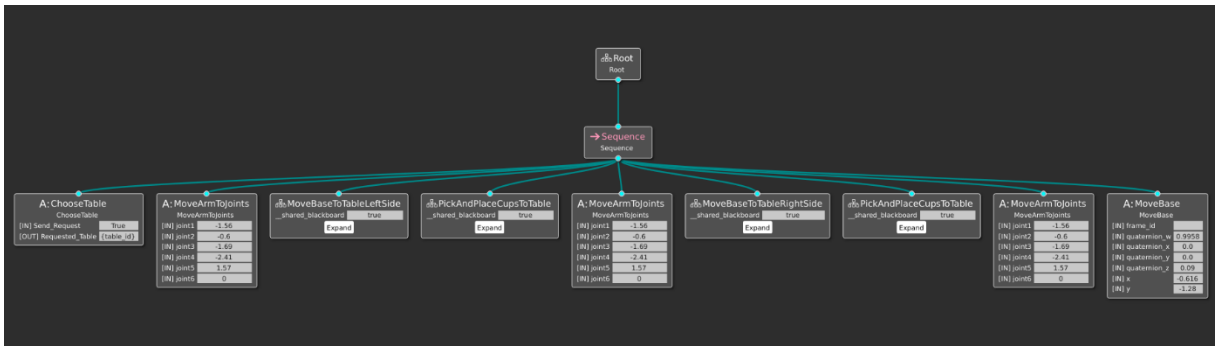


Abbildung 3-4: Main Tree des Behavior Trees

Der Main Tree sieht folgenden Ablauf vor:

1. User wählt den Tisch, an dem der Serviceroboter die Becher platzieren soll (links und rechts)
2. Bewegen des UR-Roboterarms zu einer sicheren Position
3. Bewegen der Neobotix-Plattform zur linken Seite des Tisches
4. Platzierung des Becher-Musters auf dem Tisch (inkl. Referenzierung und Bildverarbeitung des Aruco-Markers)
5. Bewegung des Armes in eine sichere Position
6. Bewegung der Neobotix-Plattform zur rechten Seite des Tisches

7. Platzierung des Becher-Musters auf dem Tisch (inkl. Referenzierung und Bildverarbeitung des Aruco-Markers)
8. Bewegung des Armes in eine sichere Position
9. Bewegung der Neobotix-Plattform zurück zur Home-Position im Labor

In den obenstehenden Abbildungen wurden bereits die einzelnen Subtrees (über mehrere Hierarchiestufen ineinander geschachtelt) aufgezeigt, die zur Modularisierung und schnelleren Applikationsengineering genutzt wurden.

Diese einzelnen Subtrees werden nachfolgend expliziert visualisiert.

Subtree – PickAndPlaceCupsToTable:

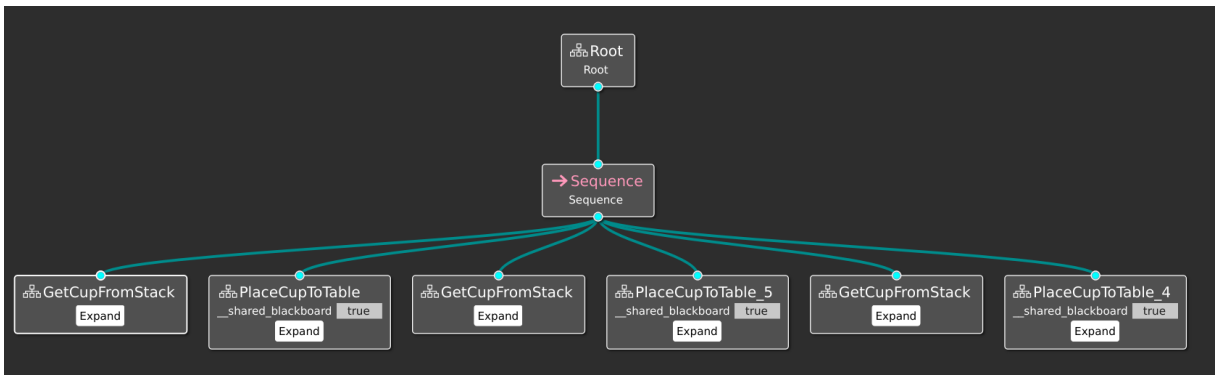


Abbildung 3-5: Subtree – PickAndPlaceCupsToTable

Dieser Subtree beinhaltet das Greifen eines Bechers aus der Halterung sowie das Platzieren des Bechers auf dem Tisch inklusive der Höhenermittlung über die Bildverarbeitung des Aruco-Tags. Der SubTree wird direkt in den MainTree integriert.

Die Becher werden nach der folgenden Struktur aufgebaut:

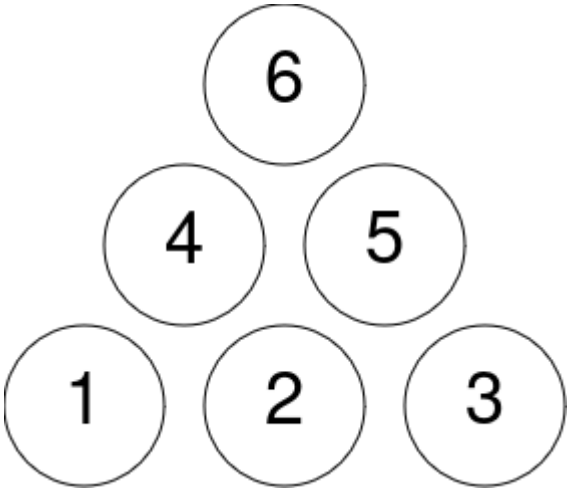


Abbildung 3-6: Zielformation der Becher

Die Live-Demonstration umfasst aus Zeitgründen lediglich das Platzieren von drei Bechern (4,5,6).

Subtree – GetCupFromStack:

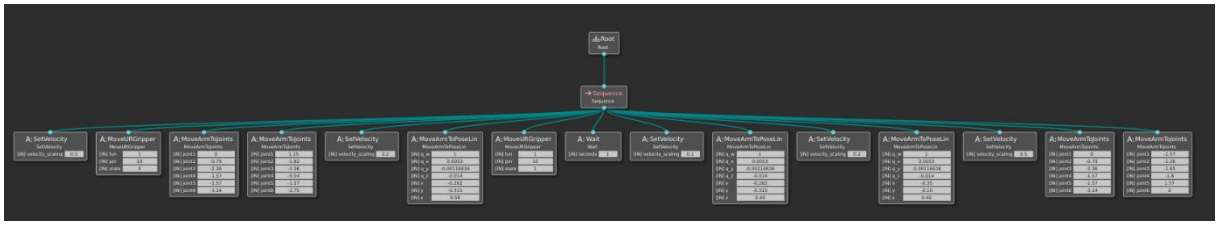


Abbildung 3-7: Subtree – GetCupFromStack

Dieser Subtree bildet das Greifen eines Bechers aus dem Becherstapel ab und nutzt für die Bewegungsplanung einzelne Zwischenposen und verschiedene Bewegungsgeschwindigkeiten des Arms. Abschließend befindet sich der Arm mit dem Becher in einer Position leicht vor dem Neobotix in Fahrtrichtung. Dieser Subtree wird in den Subtree „PickAndPlaceCupsToTable“ integriert.

Subtree – MoveBaseToTable:

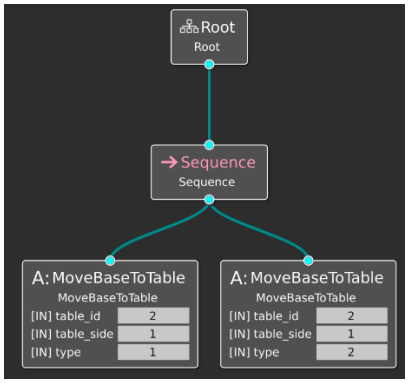


Abbildung 3-8: Subtree – MoveBaseToTable

Dieser Subtree wird genutzt, um die Anfahrt des Neobotix zum Tisch zu steuern. Hierzu wird die Tischnummer sowie die Seite des Tisches spezifiziert. Zusätzlich wurde über den Typ eine Approach Pose (1) definiert, die der Neobotix zuerst anfährt. Damit wird ermöglicht, dass der Neobotix die finale Pose (2) am Tisch zuverlässiger erreicht. Der Subtree wird direkt in den MainTree integriert.

Subtree – PlaceCupToTable:

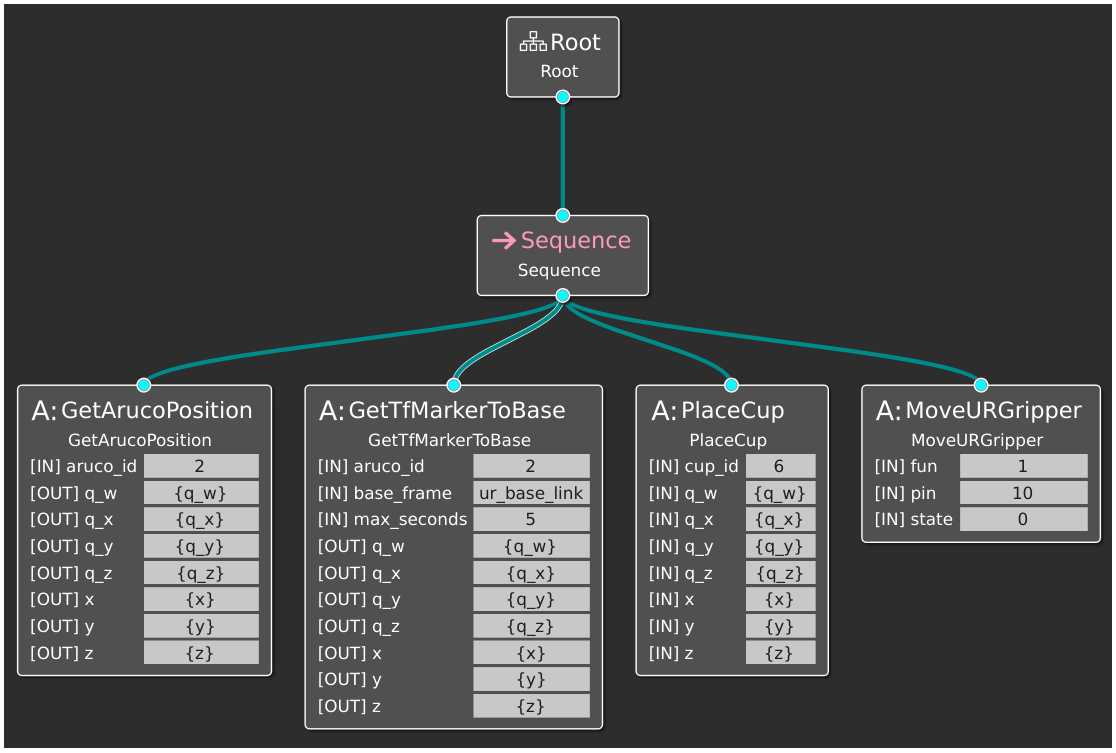


Abbildung 3-9: Subtree – PlaceCupToTable

Der Subtree steuert die Bildverarbeitung sowie Referenzierung anhand des Aruco-Markers ab. Hierzu wird zunächst über „GetArucoPosition“ die korrekte Position der richtigen (Aruco-ID) ausgelesen sowie die Transformation von Marker zu Basis berechnet und übergeben. Diese Transformation stellt den Input für die korrekte Platzierung der Becher auf dem Tisch dar. Abschließend wird der Greifer geöffnet, um den Becher abzusetzen. Dieser Subtree wird in den Subtree PickAndPlaceCupsToTable integriert.

Anmerkung: Die Referenzierung zum Aruco Marker erfolgt lediglich einmal auf jeder Tischseite, um die z-Höhe des Tisches über die Positioniersequenz konstant zu halten. Dies hilft Ungenauigkeiten der Tiefenbildkamera über die Sequenz zu vermeiden.

4. Steuerung UR

Bisher wurde der UR5 auf dem Neobotix lediglich nativ über UR-Skripte angesprochen. Folglich musste der UR grundlegend neu in ROS2 integriert und in Betrieb genommen werden.

4.1 MoveIt – Wrapper

Zur Bewegungsplanung wird ein eigener Python Wrapper eingesetzt, der die Funktionalitäten des MoveIt Frameworks (C++ move group interface) in der gewünschten Form bereitstellt. MoveIt ist eine Open Source Software zur Bahnplanung von Robotertrajektorien, mit der kollisionsfreie Trajektorien (ptp, lin und joint_ptp) geplant werden können.

Der eigene Service Server im wrapper Package bedient dabei die Client Requests aus dem Behavior Tree zur Planung und Ausführung der Robotertrajektorien auf dem UR.

Um Kollisionen zwischen Roboterkomponenten und dessen Greifer mit umliegenden Konturen im Arbeitsraum des Roboters zu verhindern, wird ein URDF und SRDF-Modell aufgesetzt. (siehe: full_robot.urdf.xacro)

Das URDF-Modell (Unified Robot Description Format) besitzt die folgenden Bestandteile:

- Bezug zur World (Planning Reference Frame)
- Modell der Neobotix MMO500 Plattform
- UR5-Manipulator
- Halterung der Kamera
- Adapterplatte für den Greifer
- Greifer von Zimmer
- Greiferbacken
- Halterung für Bechervereinzelung
- Bechervereinzelung

Das SRDF-File (**S**emantic **R**obot **D**escription **F**ormat) ergänzt semantische Informationen zum URDF-Modell. Es beschreibt die physische Struktur der Gelenke, Glieder und kollisionsrelevante Beziehung des Roboters und seiner Umgebung (beerpong_macro.srdf.xacro).

Ein weiteres Problem für die Bahnplanung des Roboters ist, dass keine 3D Informationen der aktuellen Umwelt zur Verfügung stehen. Somit können Arbeitsraumhindernisse, die an die Position des Neobotix auf der Map gekoppelt sind, nicht berücksichtigt werden. Um dieses Problem möglichst effizient zu beseitigen, wurde eine Kollisionsebene auf Höhe der UR-Base mir einem Radius von einem Meter eingeführt. Damit schränkt sich der Suchraum für Robotertrajektorien auf den Raum oberhalb der UR-Base bzw. Neobotix Cabin ein. So konnten speziell Kollisionen mit Tischen in der Front des Neobotix vermieden werden. Allerdings ist damit auch die minimale Höhe des Tisches, auf dem die Becher abgestellt werden können auf ca. 750mm über dem Boden begrenzt.

Entsprechende Screenshots aus Rviz sind im folgenden Kapitel zu finden.

4.2 RViz

Über RViz kann der Roboter wie in Abbildung 4-1 zu sehen visualisiert werden. Hier können verschiedene Eigenschaften angezeigt werden. Über TF können die Transformationen zwischen den einzelnen Links angezeigt werden (siehe Abbildung 4-2) Über *RobotModel* kann der statische Roboter in der aktuellen Stellung visualisiert werden. Darüber hinaus können separate Kollisionsgeometrien definiert werden. Diese sind etwas größer und einfacher aufgebaut, um die Berechnung der Kollisionskontrolle zu vereinfachen. Mit *MotionPlanning* kann der Roboter zudem über die grafische Oberfläche bewegt werden. Hierzu kann die Zielstellung entweder durch Einstellen der einzelnen Achsen oder durch ziehen der Kugel am TCP eingestellt werden. Über die Buttons „plan“, „execute“ oder „plan and execute“ kann der Roboter dann in die

Zielstellung verfahren werden. Der hiermit steuerbare Teil des Robotersystems, die sogenannte MoveGroup ist in Abbildung 4-3 orange dargestellt.

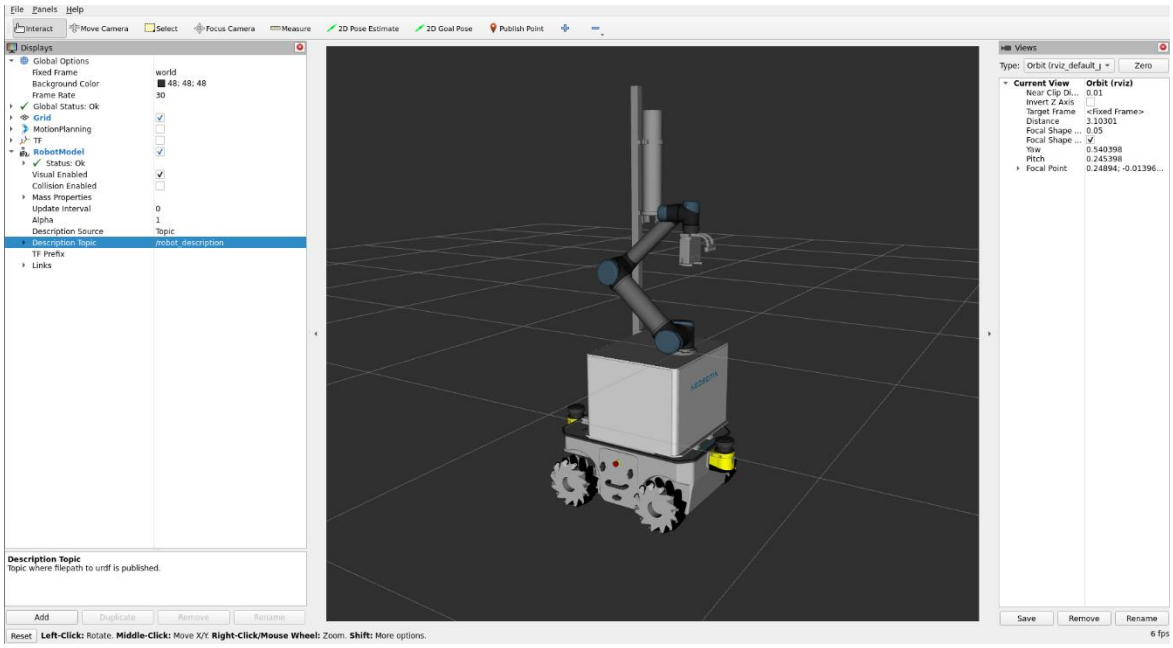


Abbildung 4-1: RViz RobotModel

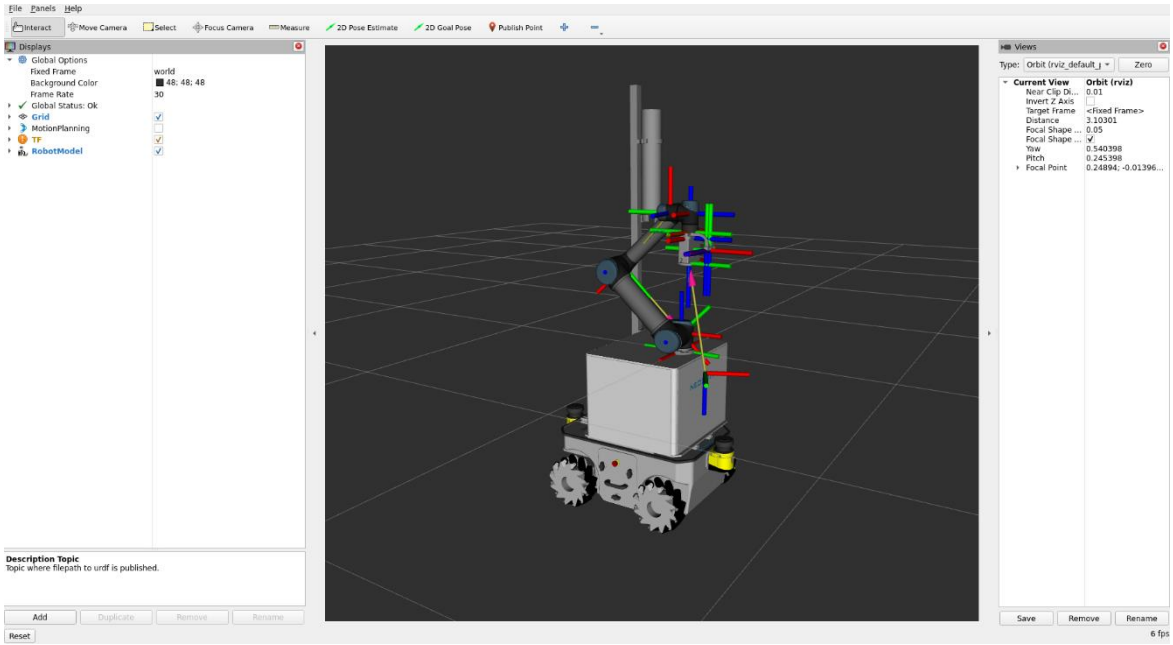


Abbildung 4-2: RViz TF

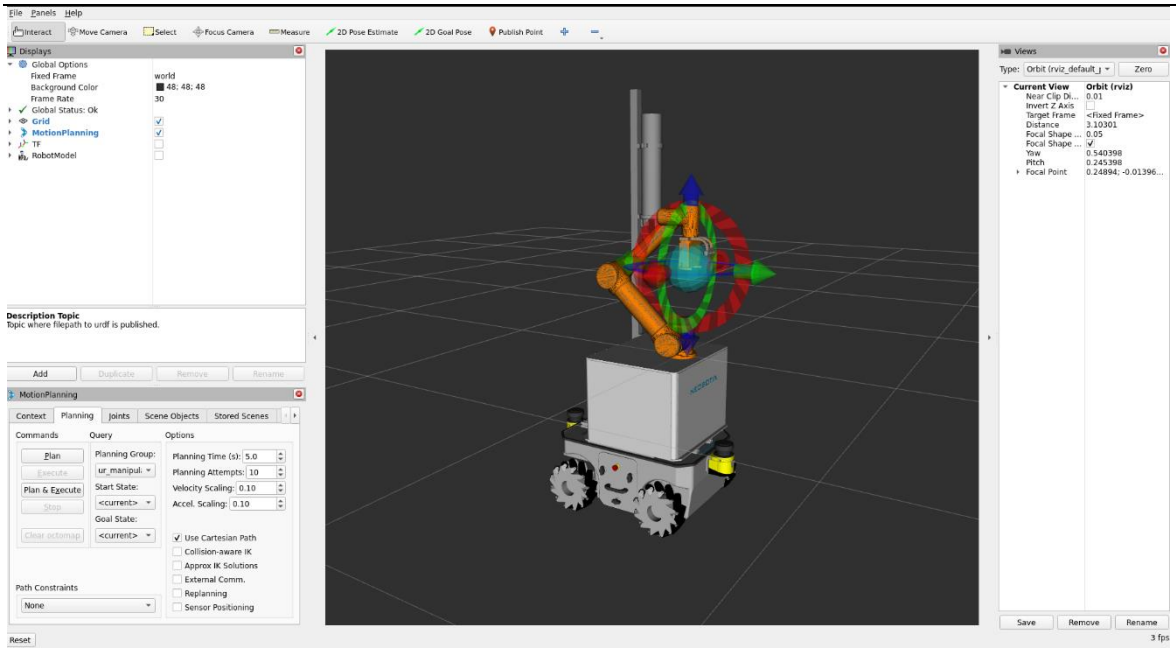


Abbildung 4-3: RViz MotionPlanning

4.3 Ansteuerung des Greifers

Der Greifer ist an den IO-Ports des UR5 angeschlossen und über den Hardwaretreiber von UR in ROS2 integriert. Zur Ansteuerung des Greifers wird der Service *io_and_status_controller/set_io* verwendet. Die zwei möglichen Zustände des Greifers können über die folgenden Parameter in der Message *ur_msgs/srv/SetIO* angesprochen werden:

Gewünschter Zustand	**Fun**	**Pin**	**State**
Öffnen	1	10	0.0
Schließen	1	1	1.0

Tabelle 4-1: Ansteuerung des Greifers

5. Steuerung Neobotix

Die Navigation des Neobotix nutzt das Navigationsframework „Nav2 Stack“. Es ist ein Teil des ROS2 Ökosystems. Der Nav2 Stack bietet zahlreiche Funktionen für autonome Navigation, einschließlich Pfadplanung, Lokalisierung und Vermeidung von Hindernissen.

Der Nav2 Navigation Stack verbessert und erweitert die Fähigkeiten des ursprünglichen Navigation Stack aus ROS 1 und ist speziell für die erhöhten Anforderungen und die verbesserte Hardware-Unterstützung in modernen Roboterplattformen optimiert. Zu den Kernkomponenten gehören der Planner, Controller und der Recovery Manager, die zusammenarbeiten, um eine sichere und effiziente Navigation in komplexen Umgebungen zu ermöglichen [6].

5.1 Aufnahme einer neuen Karte

Um eine schnelle, präzise Lokalisierung und Pfadplanung der Neobotix-Plattform zu ermöglichen, ist eine aktuelle Karte der Applikationsumgebung aufzunehmen. Hierzu ist die Plattform mit Mapping spezifischen Nodes (SLAM-Toolbox) durch das dazugehörigen Launch File (mapping.launch.py) im neobotix_mmo500_bringup Docker zu starten. Es ist zu beachten, dass lediglich ein LIDAR-Scanner (ungefiltert) zur Aufnahme der Karte verwendet werden sollte, um Überlagerungen der erhaltenen Abstandsinformationen zu vermeiden, da die Neobotix-Plattform zum Zeitpunkt des Mapping noch keine zuverlässige automatische Lokalisierung bietet. Der Input von zwei Lidar-Scans der Umgebung lässt den Mapping Algorithmus instabil werden. Nach einer umfassenden Rundfahrt in der Betriebsumgebung (Labor im HKA 2030+ Campus) kann die neue aufgenommene Karte über Funktionen des map_server Pakages gespeichert werden.

Die nachfolgende Grafik visualisiert die aufgenommene Karte, die als Grundlage für die Bewegungsplanung der Neobotix-Plattorm dient:

Abbildung 5-1: Aufgenommene Karte des Labors

5.2 Navigation mit Nav2 in RVIZ

Zur Navigation in RViz über das Nav2-Stack benötigt der Neobotix zunächst eine manuelle Referenzierung in der Karte, um die fortlaufend eingehenden Sensorsignale korrekt in die zu Grunde liegende Karte zu projizieren. Dies ermöglicht der mobilen Plattform eine korrekte Lokalisierung im Raum.

Anschließend können in der grafischen Oberfläche von RViz verschiedene Zielpunkte als Nav2Goal (unter Angabe der Plattform-Ausrichtung) vorgegeben werden. Die Zielpose kann außerdem mit Hilfe des MoveToTable-Clients des Neobotix Coordinators von der Webseite aus an den Nav2 Stack weitergeleitet werden. Daraufhin plant der Neobotix selbstständig den kürzesten Pfad unter Berücksichtigung der Global and Local Cost Map. Diese Trajektorie wird in RVIZ rot visualisiert. Zur Bahnplanung wird aktuell der Neo_Local_Planer verwendet, welcher auf einem A*-Algorithmus basiert. Die Bahnplanung wird während der Fahrt des Neobotix ständig anhand der neu erhaltenen Umgebungsinformationen (via Lidar) überprüft. Insofern die Local Costmap vorhersehbare Kollisionen vorhersagt, die nicht in der Karte enthalten sind, kann dies zu einer neuen Ziel-Trajektorie führen, welche das lokale Hindernis umfährt.

Die nachfolgende Grafik visualisiert die Bahnplanung des Neobotix zu einem beispielhaften Zielpunkt im Labor des HKA 2030+ Campus.

Abbildung 5-2: Navigation in RViz

Zusätzlicher zur Navigation zu einem Zielpunkt, können mehrere aufeinanderfolgende Zielposen vorgegeben werden. Unter Verwendung der NavigateThroughPoses Funktion werden die einzelnen Wegpunkte nacheinander geplant von der mobilen Plattform angefahren. Diese Funktionalität wird allerdings in diesem Projekt nicht weiterverwendet.

6. Bildverarbeitung

Zur Perzeption wird die Intel RealSense D415 Stereokamera verwendet. Hierzu wurde ein eigener Docker Container (*realsense_driver*) aufgesetzt. Dieser enthält alle nötigen Softwarepakete, um die Kamera über ROS2 anzusteuern. Zusätzlich enthält es ein eigenständig implementiertes ROS2-Python Package (neobotix_realsense415), welches die Logik zur Bildverarbeitung enthält und Service Server für den übergeordneten Behavior-Tree bereitstellt.

6.1 Aufgabe der Bildverarbeitung

Der Service Roboter navigiert über den Behavior Tree und das ROS Navigation2 Framework zu einer gegebenen Zielposition vor dem gewählten Tisch.

Technisch bedingt muss bei einer mobilen Plattform (dieser Ausführung) mit Positionsabweichungen von ca. ±10 cm von der vorgegebenen Zielposition gerechnet werden. Aufgrund anschließender Manipulationsaufgaben mit dem UR muss die Positionierunsicherheit der mobilen Plattform vorher rechnerisch ausgeglichen werden (Verschiebung in x, y und Rotation um die z-Achse der Roboterbasis).

Eine weitere Herausforderung für die Manipulationssequenz des Serviceroboters ist es, unterschiedliche Tischhöhen beim Platzieren der Becher handhaben zu können. Hierzu ist es vor Allem notwendig, die z-Koordinate der Soll-Platzierungs-Positionen der Becher an jedem Tisch neu zu ermitteln und anzupassen.

Folglich muss die eindeutige Pose des Tisches, bzw. seines Referenz-koordinatensystems unabhängig von der genauen Parkposition der mobilen Plattform in Relation zur Roboterbasis bestimmt werden. In dem Referenzkoordinatensystem werden dann die Manipulationsaufgaben (Becherposen) definiert. Als technische Lösung werden hierzu Aruco-Marker eingesetzt.

6.2 Aruco Marker

Aruco-Marker werden in der Robotik und Computer Vision verwendet, um reale Referenzpositionen (6DoF) durch Standardbildverarbeitung aus RGB-Bildern zu bestimmen. Zusätzlich wird durch das schwarz-weiße Muster eine Zahl (ID) codiert.

Die Open-CV Bibliothek beinhaltet eine eigene Klasse zum Handling von Aruco Markern, welche auch in dem in diesem Projekt eingesetzt wird.

Zur Aufnahme der RealSense Kamerabilder wird der Roboter in einer geeignete Position oberhalb der Marker verfahren. Die folgende Abbildung zeigt eine entsprechende Kameraaufnahme mit einem Aruco Marker. Es ist zu beachten, dass das gezeigte Koordinatensystem (Open-CV-Format) nicht der in ROS2 implementierten Transformation entspricht [7]. Dies wird in Kapitel 6.4 weiter erläutert.

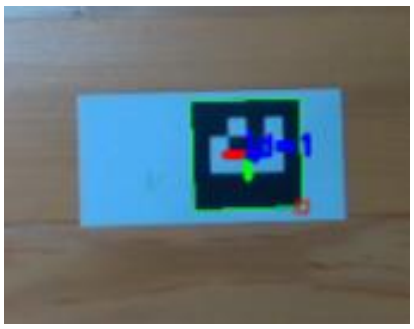


Abbildung 6-1: Erkannter Aruco Marker in OpenCV

6.3 Kalibrierung der Kamera

Zur Überführung der detektierten Aruco-Code Posen vom Sensorkoordinatensystem (Bildkoordinaten in Pixeln) in das Kamerakoordinatensystem muss zuerst eine intrinsische Kalibrierung der Kamera erfolgen.

Dazu wurden ebenfalls gängige Methoden aus Open-CV genutzt, welche die Kalibrierparameter (Kameramatrix und Verzerrungskoeffizienten) durch einen Best-Fit Ansatz (Perception-n-Point Algorithmus) aus mehreren verschiedenen Bildern eines Kalibriermusters berechnen. Das Kalibriermuster ist für gewöhnlich ein Schachbrettmuster.

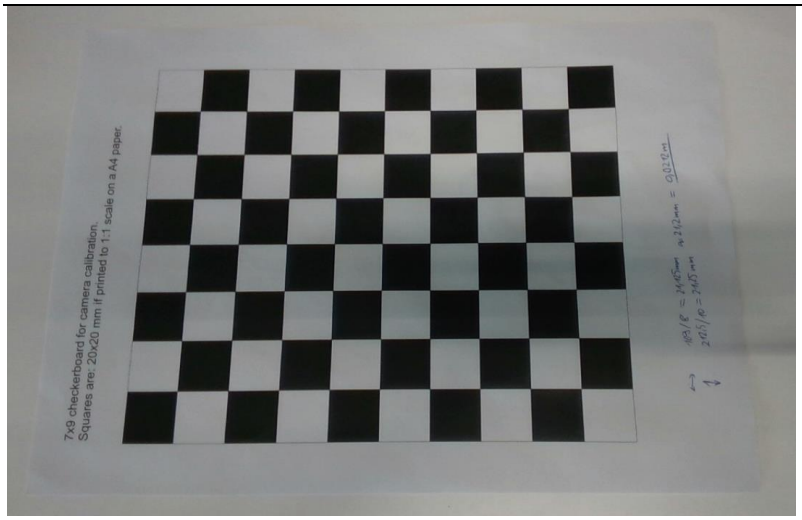


Abbildung 6-2: Verwendetes Kalibrierpattern zur intrinsischen Kalibrierung

Im Anschluss an die intrinsische Kalibrierung (Beziehung zwischen Bildebene zur Kamerakoordinatensystem) folgt die extrinsische Kalibrierung.

Durch die extrinsische Kalibrierung werden die Posen aus dem Kamerakoordinatensystem in ein repräsentativeres Koordinatensystem für den Robotertask (z.B. TCP-Koordinaten) überführt.

Dies wurde zuerst durch statische Offsets (nur translatorische Verschiebung) ausgeglichen. Hierzu wurden die Offsets mit einem Maßband vermessen.

Weitere Tests ergaben, dass die Pose des auf dem Tisch platzierten Markers bei unterschiedlichen Abständen zwischen Kamera und Tisch unterschiedlich präzise bestimmt wurde. Genauer bedeutet dies, dass die Pose des Markers bei gleicher Bildaufnahmepose des Arms und variierender Tischhöhe nicht zuverlässig detektiert werden kann. Die Applikation dieses Projektes erfordert allerdings vor allem eine hohe Präzision und Wiederholgenauigkeit der Posen Bestimmung in z-Richtung. Da eine Wiederholbarkeit der Abweichungen gegeben war, wird die herkömmliche Kamerakalibrierung mit Hilfe von Lookup-Tabellen verfeinert, sodass die präzise und wiederholgenau detektierte Aruco-Pose über alle möglichen Tischhöhen in anderen Programmteilen weiterverarbeitet werden kann.

6.4 6D Posen Bestimmung aus RGB-Bild

Um aus einem 2D RGB-Bild eine Höheninformation zu gewinnen, muss die Dimensionsanzahl vergrößert werden. Dies ist ohne die Angabe weiterer Informationen mathematisch nicht möglich. Aus diesem Grund muss zur Berechnung der Aruco-Pose neben den Kalibrierparametern auch die Kantenlänge des Markers in Meter angegeben werden. Neben der Tiefenreferenzierung dient dies als Referenzwert für alle weiteren Dimensionen der Transformation von der Bildebene in Kamerakoordinaten.

Nach einem Client Service Call durch den Behavior-Tree wird der folgende Server aus dem neobotix_realsense415 Package mit folgender Pipeline aufgerufen:

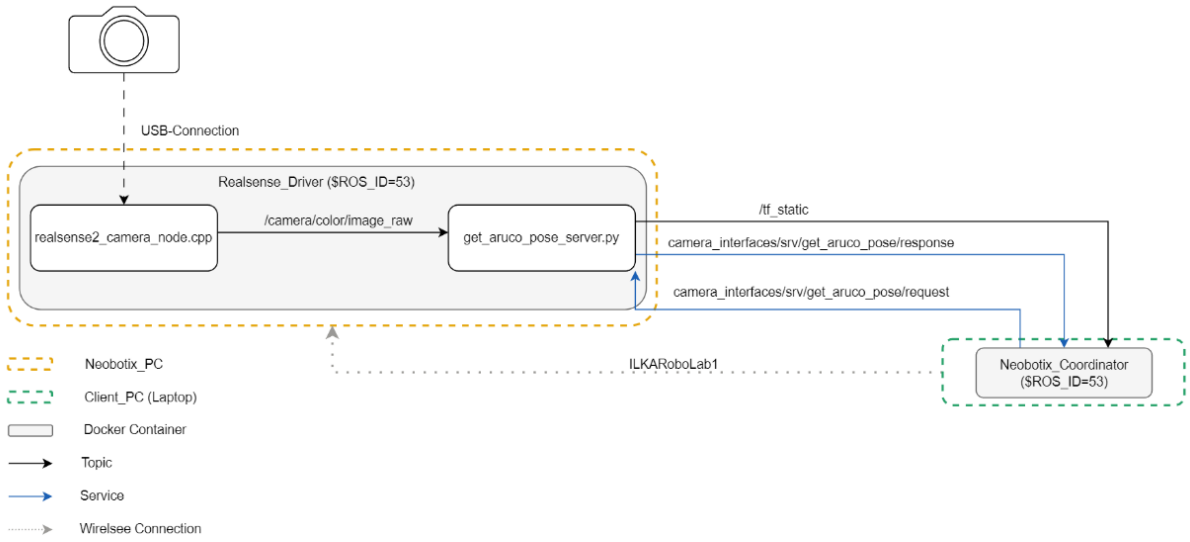


Abbildung 6-3: Aruco Erkennungs Pipeline

Zuerst abonniert der get_aruco_pose Server Node den Kameraaufnahme-Live-Stream, welcher durch die Realsense Kamera auf ein Topic publisht wird (*/camera/color/image_raw)*. Immer wenn ein neues Bild auf dem genannten Topic zur Verfügung steht, wird es in eine Klassenvariable gespeichert.

Nach dem Aufruf des Servers, wird nur das letzte gespeicherte Bild weiterverarbeitet.

In diesem Bild wird überprüft, ob die ID des detektierten Markers mit der ID des Client Requests übereinstimmt. Falls gegeben, folgt eine Erkennung der Pose und Transformation in das TCP-Koordinatensystem. Zusätzlich wird die erkannte Pose relativ zum

TCP-Koordinatensystem in den TF-Tree als tf2.TransformStamped auf das Topic /tf_static publisht. Damit kann diese Pose als relative Transformationsmatrix zum lokalen Koordinatensystem der in der Hierarchie aller Transformationen der Szene ganz einfach in jedes andere Koordinatensystem transformiert werden.

Es gilt zu beachten:

- Das Koordinatensystem des Markers, welches auf tf_static publisht wird, besitzt eine andere Orientierung als das, welches von Open-CV in den Marker gelegt wird. Die gewählte Repräsentation in ROS2 vereinfacht das Handling in der Manipulationsaufgabe.
- Die Roboterpose zur Aufnahme des Kamerabildes (bzw. Markers) muss zwangsläufig parallel zur erwarteten Pose des Markers sein, damit die nachträgliche Kalibrierung ihre volle Genauigkeit bieten kann. In diesem Use-Case entspricht dies einer horizontalen Ausrichtung.
- Der Marker Frame kann durch den tf_buffer aus der tf2 Klasse von ROS2 in das Roboter-Basissystem (welches dem Welt-Koordinatensystem entspricht) transformiert werden. Allerdings muss dies vom Behavior-Tree in einem separaten Task gemanaged werden und ist nicht Teil der Bildverarbeitung.

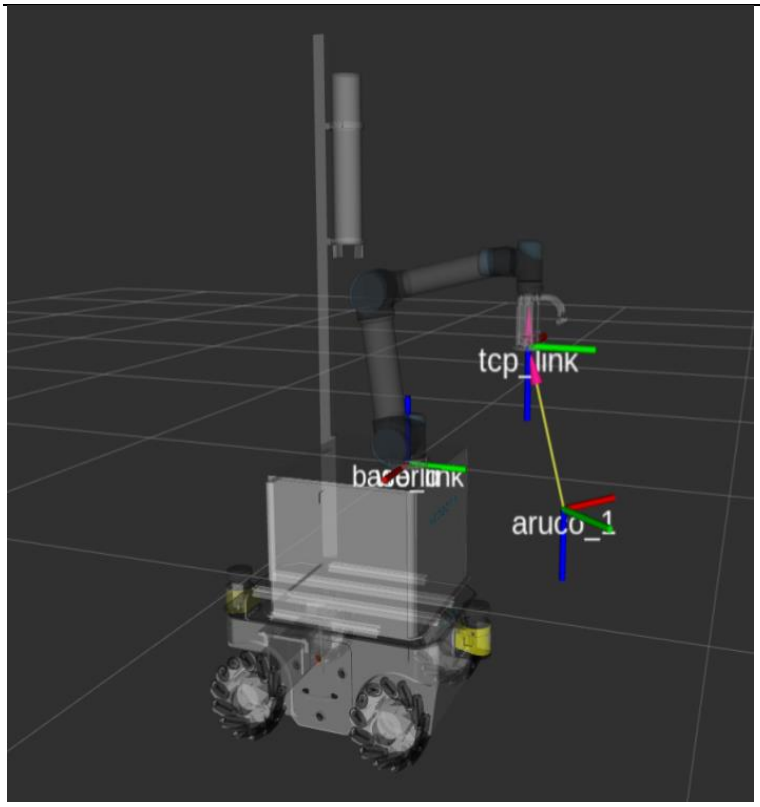


Abbildung 6-4: Koordinatensystem des Aruco-Tags im Bezug zum TCP

7. Benutzerinteraktion

Für eine benutzerfreundliche Bedienung des Systems durch den Benutzer wurde eine Website programmiert, die eine Karte des Labors anzeigt und Auswahlmöglichkeiten für die Tische bietet, zu denen die Neobotix-Plattform navigieren kann.

Die Webseite ist über die lokale Adresse 127.0.0.1 sowie dem Standard HTTP Port 8080 auf dem Docker ausführenden PC erreichbar. Der User kann über drei Buttons den entsprechenden Tisch auswählen. Der Hintergrund der Webseite stellt dabei die aufgenommene Karte der Neobotix-Plattform des Roboterlabors am LTC dar.

Damit die Tischauswahl von der koordinierenden Einheit (Neobotix_Coordinator) verarbeitet werden kann, wurde ein ROS2-Service implementiert. Die koordinierende Einheit stellt den Request an den Servicenamen „/table_server“, woraufhin die Webseite die vom User gewählte Tischnummer an den Neobotix_Coordinator übermittelt. Solange der User keine Tischauswahl getroffen hat, ist der Ablauf des Behavior Trees an dieser Stelle unterbrochen.

Zur technischen Umsetzung der Website wurde das Python-Framework Flask verwendet. Dieses Framework ist besonders gut geeignet, da die Website in einem Docker-Container ausgeführt wird und sich Python optimal für die Integration mit ROS2 anbietet.

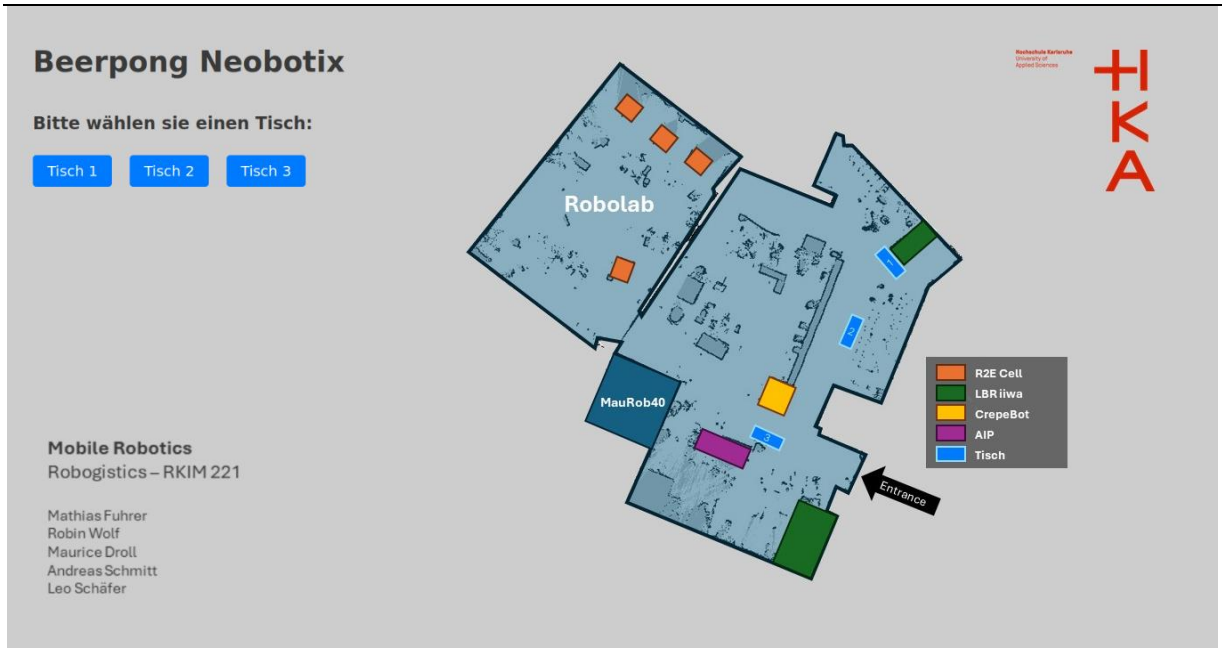


Abbildung 7-1: Design der Website

8. Herausforderungen

8.1 Wireless-LAN / Netzwerk

Die Neobotix-Plattform war mit einem WLAN-Netzwerk (ILKA_RoboLab1) verbunden, welches über keinen Internetzugang verfügte.

Wie in der Softwareentwicklung üblich, werden Container regelmäßig neu gebaut und erstellt. Dafür wird allerdings eine aktive Internetverbindung benötigt, um das Ubuntu 22.04 ROS2 Base Image von Docker Hub oder diverse Repos von GitHub zu klonen. Auch müssen des Öfteren Softwarepakete mit apt oder pip installiert werden.

Es musste oft zwischen den Netzwerken ILKA_RoboLab1 und KA-Wlan gewechselt werden. Daher war es nicht möglich, die Docker-Container zu bauen, ohne die Verbindung zum Roboter zu verlieren und anschließend neu aufbauen zu müssen.

Zusammenfassend lässt sich feststellen, dass diese Netzwerkwechsel einen erheblichen Zeitaufwand durch abbrechende Verbindungen und deren Wiederherstellungen verursachten. Optimal wäre es, wenn das IRAS interne WLAN ILKA_RoboLab1 einen Internetzugang hätte, sodass sich die Docker-Container ohne ständige Netzwerkwechsel bauen und starten ließen.

8.2 Migration von ROS2 Foxy zu Humble

Ein Teil der Docker-Container (Neobotix) war zu Projektbeginn vorhanden, jedoch basierten diese noch auf der veralteten ROS2 Foxy Version. Um diese zusammen mit den neueren ROS2-Containern nutzen zu können, musste die Version Humble auf alle Container aufgespielt werden. Jedoch beeinflusste dies die Funktionalität von mehreren Bibliotheken, die daraufhin ausgetauscht bzw. aktualisiert werden mussten.

8.3 Kabelführung

Da an der Neobotix-Plattform zwei Lidar-Sensoren angebracht sind, muss bei der Kabelführung besonders darauf geachtet werden, dass die verlegten Kabel nicht dauerhaft einen Sensorstopp auslösen. Insbesondere das USB-Kabel zur RealSense-Kamera musste durch den Sensorbereich geführt werden.

8.4 Kartenaufnahme

Zur Lokalisierung der Neobotix-Plattform im Raum wurde eine Karte des Roboter-Labors erstellt. Dabei wurde das Roboterlabor im Handmodus der Neobotix-Plattform abgefahren, sodass die Objekte im Raum bekannt sind.

Dieser Prozess erfolgt mit den Funktionalitäten des Navigation Stacks. Schwierigkeiten ergaben sich bei einigen Objekten, wie beispielsweise die Glasfassade der Werkstatt, die vom Lidar-Sensor nur unzureichend erfasst werden oder Stuhlbeinen.

Zudem darf bei der Kartierung lediglich ein Lidar-Sensor genutzt werden, da es sonst zu Überlagerungen und Verschiebungen in der aufgezeichneten Karte kommt, sodass die Karte nicht genutzt werden kann.

8.5 Kamera-Kalibrierung

Die innerhalb dieses Projektes verwendete Tiefenbildkamera Intel Realsense 415D wurde zuvor in anderen Projekten verwendet und dort entsprechende Kalibrierparameter überschreiben. Somit waren die Sensordaten, welche die Kamera geliefert hat innerhalb dieses Projektes nicht zu gebrauchen. Da die Aruco-Detektion bereits in Open-CV implementiert war, wurde auch die komplette intrinsische und extrinsischen Kamerakalibrierung mit Open-CV Methoden durchgeführt. Die detektierte Aruco-Pose kann nun relativ zum TCP bestimmt werden.

Die Kamera wurde in dem Use-Case innerhalb der vorgesehenen Anwendung vom Roboter geführt. Es entstand das Problem, dass eine nicht zu akzeptierende

Messabweichung in allen Raumrichtungen, speziell in Abhängigkeit des Z-Abstandes zwischen Kamera und erkanntem Marker entstand. Nach einigen Versuchen wurde festgestellt, dass die bestimmten Aruco-Posen zwar wiederholgenau, aber nicht präzise waren. Um die Präzision zu steigern, damit eine annehmbare Messungenauigkeit erreicht werden kann, wurde das Ergebnis der Posen Bestimmung mit einer Lookup-Tabelle nachkalibriert. So konnte die Messungenauigkeit des Kamerasystems von ca. +/- 20mm auf ca. +/- 3mm in unterschiedlichen Messhöhen bei gleichbleibender Belichtung des Markers reduziert werden.

8.6 Greifer Beschaffung bei der Firma Zimmer

Zunächst war kein geeigneter und verfügbarer Greifer vorhanden. In Zusammenarbeit mit der Firma Zimmer konnte jedoch ein passender elektrischer Zweibackengreifer organisiert werden. Allerdings führte dies zu einer Verzögerung im Projektablauf.

8.7 Systemintegration

Dieser Abschnitt befasst sich mit der Systemintegration bzw. der kombinierten Inbetriebnahme der Neobotix-Plattform und des UR-Manipulatorarms.

8.7.1 Tausch des Planning Reference Frames

Per Default ist das Referenz-Koordinatensystem, in dem alle Roboterposen angegeben werden können auf das Koordinatensystem des ersten Joints in der kinematischen Kette festgelegt. Im Fall des kombinierten Robotermodells (UR + Neobotix) war dies auf den „Neobotix Base Link“ festgelegt, welcher sich zentral unter der mobilen Plattform auf Höhe des Unterbodens befindet. Da diese Referenz für die konkrete Arbeitsaufgabe nicht sehr intuitiv war, wurde der sogenannte „Planning Reference Frame“ durch entsprechende Befehle beim Initialisieren der Move Group auf das „UR Base Link“ Koordinatensystem bzw. das World-Koordinatensystem umgeändert. Diese

beiden Koordinatensysteme unterscheiden sich lediglich im Namen. Ein separates World-Koordinatensystem wurde nur zu zur besseren Übersicht eingeführt.

8.7.2 Tausch des Inverskinematik-Solvers

Sobald der UR Zielposen in kartesischen Koordinaten erhalten hat, war die Berechnung und Ausführung der Trajektorien unzuverlässig und unvorhersehbar.

Das Verhalten äußerte sich dadurch, dass sie berechneten Wegpunkte teilweise von einer sinnvollen Trajektorie zur Zielpose abwichen, manchmal sogar in Kollision lagen. Außerdem konnten LIN-Befehle teilweise gar nicht mehr ausgeführt werden.

Dies konnte auf den verwendeten Inverskinematik-Solver zurückgeführt werden. Anfangs wurde der Standard MoveIt IK-Solver „KDLKinematics“ genutzt, der auch in den Standard UR-Packages verwendet wird.

Dieser wurde durch den externen IK-Solver „Trac-IK“ ersetzt, welcher einen leicht verschiedenen Berechnungsansatz für die Inverskinematik verwendet. Nach der Installation der benötigten ROS2-Packages und der entsprechenden Konfiguration des IK-Solvers in der kinematics.yaml Konfigurationsdatei waren die vorherigen Probleme gelöst.

Ausschlaggeben hierfür war speziell die Konfiguration „Optimization Objective = Distance“.

8.7.3 Optimierung der Zielposen im Joint Space

Manche der fest definierten, also von der konkreten Becherposition unabhängigen Posen, wurden im Joint Space definiert, um auf die Berechnung der Inverskinematik zu verzichten. Einzelne dieser definierten Posen wurden zwar präzise und wiederholgenau angefahren, jedoch waren die Trajektorien nicht sehr intuitiv bzw. geometrisch sinnvoll.

Dieses Problem konnte gelöst werden, indem alle Gelenkstellungen in der Zielpose in dem Bereich +/- Pi angegeben wurden, was zuvor nicht der Fall war. So liegt die Zielpose mittiger im Gelenkraum von +/- 2*Pi pro Achse und kann daher schneller und zuverlässiger angefahren werden.

8.7.4 Interferenzen auf den Topics

Beide Roboter (Neobotix und UR) funktionierten im isolierten Betrieb tadellos. Sobald die beiden Robotersysteme allerdings softwareseitig kombiniert wurden, traten unvorhersehbare Fehler auf.

Dies konnte nach einer gründlichen Analyse auf folgenden Grund zurückgeführt werden: Zu Beginn wurden teilweise ROS2-Packages verwendet, die die Hersteller zur Verfügung stellen. Diese sind lediglich auf einen isolierten Betrieb ausgelegt. Im Speziellen bedeutet das, dass jedes Robotersystem sein eigenes URDF-Modell hatte und dessen Kinematik durch einen robot_state_publisher Node auf das Topic /robot_description gepublisht hat. Somit wurden im Kombinierten Betrieb von zwei verschiedenen Nodes zwei verschiedene Roboterkinematiken auf das gleiche Topic gepublisht. Die entsprechenden Nodes (MoveIt, …), die diese Roboterkinematik von dem Topic lesen, konnten demzufolge nicht mehr zuverlässig arbeiten.

Gelöst wurde dieses Problem, indem der Sourcecode der Hersteller-Packages angepasst wurde, sodass im aktuellen System nur noch ein URDF-Modell, welches beide Systeme direkt kombiniert, von einem Publisher Node auf das entsprechende Topic gepublisht wird.

Explizit wird die gesamte Roboterkinematik (Neobotix + UR) nun ausschließlich vom neobotix_mmo_500 bringup container publisht und alle Nodes des ur_driver_humble containers subscriben auf dieses Topic. Dies bedeutet im Umkehrschluss nun allerdings auch, dass es nicht mehr möglich ist, den UR zu nutzen, ohne die entsprechenden Systeme der mobilen Neobotix Plattform zu starten.

8.8 Rechenleistung

Bei Tests der gleichzeitigen Navigation der Neobotix-Plattform und der Steuerung des UR5-Roboters wurde die CPU des Neobotix-PC vollständig ausgelastet, was sich durch unplanmäßiges Verhalten des Roboters äußerte. Diese Probleme traten nicht auf, wenn die Navigation oder die Steuerung des Roboters einzeln verwendet wurden. Die CPU-Auslastung auf dem Neobotix-PC kann nach einigen Versuchen wie folgt aufgegliedert werden:

- 5-10% Kamera
- 30 % Navigation (mit Rviz 50-60%)
- 5 % SSH-Verbindung zu weiterem Laptop
- 40 % UR (mit Rviz 60-70%)
- 70% UR während Bahnplanungsphase

Da es im Rahmen dieser Projektarbeit nicht möglich war, die Rechenleistung des in die mobile Plattform integrierten PCs zu erhöhen, wurde beschlossen, einen Teil der Tasks auf mehrere externe Laptops auszulagern. Der Laptop, welcher den UR steuert, muss sich auf der mobilen Plattform befinden, darf jedoch den Bewegungsraum des Roboters nicht einschränken und nicht im Scanfeld der Laserscanner liegen.

Wie in Abbildung 8-1 zu sehen ist, wurde hierfür eine Halterung konstruiert. Der Laptop wird von vier 3D-gedruckten Ecken gehalten und daran gehindert, seitlich herunterzurutschen. Die vorderen Befestigungspunkte sind fest mit dem Aluprofil verschraubt, während die Befestigungspunkte zur mobilen Plattform hin verschiebbar auf das Aluprofil aufgesteckt sind. Dadurch ist es möglich, die Befestigungspunkte nach außen zu schieben, wenn kein Laptop darauf liegt, um so den Zugang zum Schaltschrank zu ermöglichen.

Damit die Software (Nav2-Stack) diesen Anbau bei der Pfadplanung inklusive Kollisionsvermeidung durch die Costmaps mit berücksichtigt, musste der Footprint des Neobotix in der *navigation.yaml* Konfigurationsdatei entsprechend nach hinten erweitert werden.

Abbildung 8-1: Laptophalter

9. Zusammenfassung und Ausblick

Mit diesem Projekt konnte ein gemeinsam interagierendes Robotersystem bestehend aus der mobilen Neobotix-Plattform und dem Roboterarm UR-5 des Unternehmens Universal Robots in ROS2 aufgesetzt und dessen Funktionalität an einer beispielhaften Aufgabe getestet werden. Der gesamte Ablauf konnte in mehreren Versuchen mit ausreichender Zuverlässigkeit absolviert werden.

Die Anwendung wurde komplett in ROS2 integriert und erlaubt die Ansteuerung der mobilen Plattform sowie des Roboterarms über Behavior Trees. Die Arucocodes können mittels Kamera zuverlässig gelesen und die gestapelten Becher aus einer selbst entwickelten und gedruckten Halterung sowie Vereinzelung zuverlässig entnommen werden. Durch eine Website können User mit dem System interagieren, indem sie den zu bestückenden Tisch auf einer grafischen Oberfläche intuitiv auswählen können.

Das Projekt erforderte ein nahezu vollständiges Neuaufsetzen des Systems. Diverse ROS2-Packages von Neobotix und UR konnten genutzt werden, mussten jedoch stark modifiziert werden, um im Gesamtsystem einwandfrei zu funktionieren. Dies ermöglichte einen intensiven Einblick in die Anpassung und Integration von ROS2-Komponenten und förderte das Verständnis für die Feinheiten und Komplexitäten der Systementwicklung in der Robotik.

Da innerhalb dieses Projektes nicht auf einer bereits funktionierenden Basisimplementierung/ Docker Container, wie bei den Schulungszellen, aufgebaut werden konnte, können einige Lessons Learned nachgewiesen werden. Um diese Erkenntnisse an die nächsten Generationen RKIM Studenten weiterzugeben, wurde der gesamte Code in einem GitHub Repo abgelegt und veröffentlicht:

https://github.com/mathias31415/neobotix_beerpong.git.

Dieses Projekt bietet eine ausgezeichnete Gelegenheit für zukünftige Studierende, die bereits über Erfahrung mit ROS2 verfügen, ihr Wissen weiter zu vertiefen und komplexere Herausforderungen in Bezug auf Robotik in der Logistik zu meistern.

In Zukunft könnte dieses System auf neue Use-Cases angepasst werden, z.B. in Restaurants oder auch zum Medikamententransport in Krankenhäusern dienen. Als Grundlage steht zukünftigen Arbeiten nun die hier entwickelte Systemintegration zur Verfügung.

10. Literaturverzeichnis

[1] Neobotix, „Mobiler Manipulator MMO-500," 2024. [Online]. Available: https://www.neobotix-roboter.de/produkte/mobile-manipulatoren/mobiler-manipulator-mmo-500. [Zugriff am 29 Mai 2024].

[2] Universal Robots, „Der UR5e," 2024. [Online]. Available: https://www.universal-robots.com/de/produkte/ur5-roboter/. [Zugriff am 29 Mai 2024].

[3] Intel Realsense, „Intel® RealSense™ Depth Camera D415," 2024. [Online]. Available: https://www.intelrealsense.com/depth-camera-d415/. [Zugriff am 29 Mai 2024].

[4] Zimmer Group, „GEP5010IO-00-A," 2024. [Online]. Available: https://www.zimmer-group.com/de/technologien-komponenten/komponenten/handhabungstechnik/greifer/elektrisch/2-backen-parallelgreifer/serie-gep5000/produkte/gep5010io-00-a. [Zugriff am 29 Mai 2024].

[5] Auryn Robotics, „Behavior Tree - About," 2024. [Online]. Available: https://www.behaviortree.dev/docs/intro/. [Zugriff am 25 Juni 2024].

[6] Neobotix, „Nav2 - ROS 2 Navigation Stack," 2024. [Online]. Available: https://neobotix-docs.de/ros/ros2/autonomous_navigation.html. [Zugriff am 25 Juni 2024].

[7] OpenCV, „Detection of ArUco Markers," 2024. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. [Zugriff am 25 Juni 2024].