HAW HAMBURG

BACHELORTHESIS

Jannes Kasper

Autonome Erkennung und Interaktion mit Türen in der Robotik

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jannes Kasper

Autonome Erkennung und Interaktion mit Türen in der Robotik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 16.12.2021

Jannes Kasper

Thema der Arbeit

Autonome Erkennung und Interaktion mit Türen in der Robotik

Stichworte

Robotik, Autonomie

Kurzzusammenfassung

Das Öffnen von Türen stellt seit jeher eine Herausforderung in der Robotik dar. Aufgrund der Komplexität und Diversität der auftretenden Szenarien und der durchzuführenden Bewegungen, wird aktuell an unterschiedlichen Lösungsansätzen entwickelt und geforscht. Es gibt bereits Ansätze und Versuche, die die erforderlichen Aufgaben, meist unter gewissen Einschränkungen, erfolgreich lösen. In der vorliegenden Arbeit wird nach der Untersuchung des aktuellen Stands der Technik eine neuartige Lösung vorgeschlagen und deren Realisierung erläutert.

Diese Lösung basiert auf einer bestehenden Fahrplattform (Husky Roboter) mit einem Universal-Robots Arm und ist in einer Simulation entwickelt und getestet. Es wird erläutert, wie die Detektion von Türen und deren Ausrichtung mit Hilfe von Bezugsmarkierungen gelöst werden kann. Außerdem werden weitere Algorithmen vorgestellt, die zum Lösen der Aufgabe beitragen. Des Weiteren wird die Umsetzung des gewünschten Verhaltens mit Behaviour Trees genauer beschrieben. In diesem Zusammenhang wird auch erklärt, wie das Fehlermanagement in einem solchen System aussehen kann.

Zum Abschluss wird die entwickelte Lösung quantitativ bewertet. Hierbei werden Problematiken in Bezug auf die Entwicklung in einer Simulation und in Bezug auf externe Softwareelemente aufgezeigt. Außerdem werden die Vor- und Nachteile des Systems erörtert. Zudem wird darauf eingegangen, wie das System erweitert beziehungsweise verbessert werden kann.

Jannes Kasper

Title of Thesis

Autonomous detection and interaction with doors in robotics

Keywords

Robotics, Autonomy

Abstract

Opening doors has always been a challenge in robotics. Due to the complexity and diversity of the scenarios and the difficult movements that have to be performed by the robots different approaches are being developed up until today. There already are some solutions that solve the required tasks successfully. Most of these work under certain restrictions. After examining the current state of technology, the presented work will propose a new solution to the problem and will explain how it is implemented.

This solution is based on an existing robot platform (Husky robot) in combination with a Universal Robots arm and is implemented and tested in a virtual environment. It is explained how using fiducial markers can help with the detection of doors and the calculation of their orientation. Additionally, other algorithms necessary for the system will be presented and explained. Besides that, the implementation of the process with Behaviour Trees will be discussed. In that context a possible fault management of such a system is presented.

Finally, there will be a quantitative evaluation of this solution. Different kind of problems regarding the development with a simulation and the use of external software will be discussed. At last different positive and negative aspects of the system will be pointed out and possible improvements and extensions will be presented.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AI	Artifizielle Intelligenz
BT	Behaviour Tree
FSM	Finite State Machine
HSM	Hierarchical State Machine
ROS	Robot Operating System

Glossar

Endeffektor	Ein Aktor am Ende eines Armes der zur Interaktion mit der Umgebung genutzt wird.
Gripper	Ein Endeffektor in Form eines Greifers.
Quaternion	Gibt die Rotation um eine Achse im 3-dimensionalen Raum an
Pose	Kombination von 3D Koordinate und Orientierung in Form eines Quaternions
Gazebo	Eine Open-Source 3D Robotik Simulation.

1 Einleitung

Wie sieht ein Prozess aus, der einem Roboter ermöglicht zuverlässig und gefahrenlos Türen zu öffnen? Welche Sensoren werden für eine solch komplexe Aufgabe benötigt? Wie würden Algorithmen aussehen, die diesen Ablauf ermöglichen? Fragen wie diese werden auf den folgenden Seiten untersucht und beantwortet. Hierfür wird der aktuelle Forschungsstand bezüglich Interaktion mit Türen in der Robotik untersucht. Dieser liefert eine breite Palette an Möglichkeiten, um das vorliegende Problem zu lösen.

Die Relevanz des Themas ist unbestritten, denn autonome Geräte sind immer weiter auf dem Vormarsch. Seien es Autos, Drohnen oder Roboter, die sich führerlos durch die Umgebung bewegen sollen. Gerade Roboter haben hierbei eine hohe Relevanz, um zum Beispiel Aufgaben wie die „letzte Meile“ der Postzustellung zu übernehmen. Um jedoch Aufgaben wie diese erledigen zu können, müssen sie alle alltäglichen und menschengemachten Hindernisse überwinden können. Zu diesen zählen unter anderem Türen, welche aufgrund ihrer unterschiedlichen Erscheinung und dem dahinterstehenden Öffnungsmechanismus nicht trivial zu bewältigen sind. Um diese Problemstellung trotz dessen lösen zu können, wird ein robuster Ablauf und eine sehr generelle, aber trotzdem genaue Detektion von Tür und Klinke benötigt.

Damit die Umsetzung, der im Rahmen dieser Arbeit entstandenen Lösung, nachvollziehbar dargestellt werden kann, wird folgender Aufbau gewählt. Angefangen wird mit den Use-Cases und Anforderungen, um aufzuzeigen, warum man dieses Thema untersuchen sollte und welche Eigenschaften das System aufweisen sollte. Im Anschluss werden Aspekte von vergleichbaren Arbeiten vorgestellt, um zu zeigen, welche Lösungsansätze bereits gefunden wurden und auf welchem Stand der Technik sich diese befinden. Daraufhin wird der Husky Roboter samt aller Komponenten vorgestellt. Dazu gehört der Aufbau des Roboters, die dazugehörige Simulation, die Sensoren und die unterschiedlichen Steuerungsmechanismen. Nachdem alle grundlegenden Informationen über das Projekt dargelegt wurden, geht es um die Erklärung, wie der Prozess

zum Öffnen von Türen modelliert und umgesetzt werden kann. Hierzu werden alle wichtigen Algorithmen beschrieben, vorgestellt und in den Kontext des Prozesses eingeordnet. Dazu gehört auch das Fehlermanagement in dem System. Im Anschluss an die Erklärung des Systems folgt die Auswertung mit definierten Testszenarien. Zum Abschluss werden die Ergebnisse des Projektes zusammengefasst und ein Fazit gezogen.

1.1 Anforderungen und Use-Cases

Für die autonome Erkennung und Interaktion mit Türen gibt es ein breites Anwendungsspektrum.

Konkrete Use-Cases für dieses Thema sind beispielsweise:

- Das Öffnen von Türen für Menschen mit Behinderung
- Das Öffnen von Türen in Situationen, in denen es für Menschen gefährlich wäre, sie zu öffnen (Rettung aus Häusern)
- Service Roboter, die sich durch Häuser bewegen müssen
- Die Anlieferung von Frachten über die „letzte Meile“

Die Akteure, die mit diesen Systemen in Verbindung stehen sind unter anderen:

- Endverbraucher (belieferte Person, hilfsbedürftige Person, etc.)
- Vertreibende Firmen
- Personen, die mit den Systemen Hand in Hand arbeiten (Pflegekräfte, Zulieferer, etc.)
- Firmen, die von den autonomen Systemen profitieren (Transportfirmen, Pflegezentren, etc.)

Aus diesen Use-Cases lassen sich Anforderungen für das System schließen.

Da es in den meisten der Use-Cases um Tätigkeiten geht, die normalerweise Menschen ausführen würden, ist eine der wichtigsten Anforderungen die Zuverlässigkeit und Robustheit des Systems. Das bedeutet, dass das System mit einer Vielzahl an Situation umgehen können muss und auch bei Problemen kein unvorhersehbares Verhalten an den Tag legen darf. Hierfür ist eine solide Ablaufstruktur, welche die Grundlage für das Verhalten des Roboters legt, essenziell. Einher mit der Zuverlässigkeit geht die Sicherheit, denn da der Roboter oftmals in der Umgebung von Menschen agieren wird, ist es wichtig, dass dieser Menschen und Umgebung keinen Schaden zufügen kann.

2 Technische Grundlagen

Einen Prozess zum autonomen Öffnen einer Tür, kann grob in zwei Teilaufgaben unterteilt werden. Zum einen ist es notwendig durch Sensorik Informationen aus der Umgebung zu sammeln und auszuwerten. Zu diesen gehören zum Beispiel, wo sich die Tür befindet, in welche Richtung sie sich öffnen lässt und wo die Türklinke positioniert ist. Zum anderen ist es wichtig einen stabilen Ablauf zu kreieren, durch den der Roboter, in den unterschiedlichsten Szenarien, in der Lage ist die Tür zuverlässig zu öffnen.

Zum Lösen dieser zwei komplexen Teilaufgaben gibt es bereits eine Vielzahl an Ansätzen und Versuchen. Im Folgenden werden ein paar vergleichbare Lösungen vorgestellt.

2.1 Sensorik und Auswertungsverfahren

Es gibt eine Vielzahl an Sensoren und Verfahren, die für das Problem der Informationsbeschaffung in diesem Anwendungsbereich in Frage kommen. Hauptsächlich haben sich folgende Sensoren bei der Recherche herauskristallisiert: Drucksensoren (bzw. allgemein Kraftsensoren), Lidar Sensoren, 3D-Scanner und normale 2D-Kameras (mit ggf. Stereo-Vision). Die Nutzung eines Sensors schließt die Nutzung eines anderen nicht aus, wodurch sie in vielen Fällen in Kombination genutzt werden. Unterschiedliche Sensoren haben verschiedene Vorteile, wegen derer sie eingesetzt werden und bringen unterschiedliche Auswertungsverfahren mit sich.

Eine normale 2D-Kamera gehört oftmals zur Grundausstattung (siehe [1] [2]) wenn man aus einer Umgebung Informationen extrahieren will. In Kombination damit kommen meist Verfahren wie Maschinelles Lernen mit CNNs oder klassische Bildverarbeitungsalgorithmen wie „Gradient Descent“, „Canny Edge Detector“ (siehe [3]) oder „Template Matching“ zum Einsatz. Diese Verfahren dienen dazu, die vielen Informationen in einem Bild auf wenige relevante Features zu reduzieren, um dann schlussendlich die gesuchten Informationen extrahieren zu können. Die Autoren in [3] nutzen besagte Filter zur Vorverarbeitung, um den Umriss von Türen identifizieren zu können.

Neben den klassischen 2D-Kameraverfahren ist die 3D-Sensorik in der Robotik weit verbreitet. Der Vorteil hierbei liegt darin, dass neben der Identifikation von Objekten, ohne weitere Informationen auch auf die Position dieser im Raum geschlossen werden kann. Außerdem können 3-dimensionale Daten von Vorteil sein, wenn die gesuchten Objekte nicht mehr gut durch 2D Bildverarbeitung von der Umgebung zu unterscheiden sind. Ein Beispiel hierfür wäre eine weiße Tür in einer weißen Wand. Um 3D-Informationen zu sammeln, werden unterschiedlichste Sensoren und Verfahren genutzt.

Zum einen gibt es Stereo-Kameras wie die Realsense „Realsense d435“, die neben normalen Farbbildern auch Tiefenbilder und RGB-Punktwolken liefern kann. Die Autoren von Paper [4] nutzen diese 2D und 3D Informationen der Realsense in ihrem Projekt. Sie stellen mehrere Verfahren zur Erkennung von Türen mit unterschiedlichen Informationsquellen (2D, 3D, RGB) vor.

Des Weiteren gibt es zur Erfassung von 3D Informationen noch Lidar-Sensoren und 3D-Scanner. Unter Diesen gibt es unterschiedliche Verfahren, wie die Daten erhoben werden. Das Ergebnis sind oftmals Punktwolken oder Laserscans aus denen 3D Informationen generiert oder extrahiert werden. Beispiele hierfür sind in [3], [4], [5] detailliert nachzulesen. Der Vorteil dieser Art von Verfahren ist, dass man aus diesen Daten eine große Menge an Informationen herausziehen kann. Aus dieser großen Menge an Informationen müssen jedoch erstmal die wichtigen extrahiert, sinnvoll interpretiert und genutzt werden. Andernfalls hat dieser hohe Informationsgehalt keinen Mehrwert gegenüber Daten mit einer geringeren Informationsdichte. Ein Nachteil bei 3D Methoden kann die Performance sein, denn wie in [5] beschrieben, dauert in diesem Fall ein hochauflösender Scan der Umgebung mit dem neigbaren Lidar (Hokuyo UTM-30) circa 10 Sekunden. Außerdem ist das Verarbeiten von 3D Punktwolken keine triviale Aufgabe, diese kann zum Beispiel mit Hilfe von Segmentierungs- und Clustering-Algorithmen gelöst werden [5]. Heutzutage gibt es bereits bei 3D-Daten einige Ansätze hinsichtlich maschinellen Lernens, wie die Autoren von [4] anhand von „Point Net“ zeigen.

Drucksensoren oder allgemein kraftbasierte Sensoren können ohne visuelle Ergänzung Informationen über den Status der Türklinke oder der Tür liefern, wenn der Kontakt hergestellt wurde. Der Endeffektor ist in dem Anwendungsgebiet der Türinteraktion meist ein Gripper und wie in [6] beschrieben, kann dieser zusätzlich mit Drucksensoren ausgestattet werden. Dies hat

zur Folge, dass wie bei einer menschlichen Hand, die Stärke und die Richtung des Drucks an den Kontaktpunkten gemessen werden kann. Der große Vorteil an dieser Methode ist, dass sie sehr robust gegenüber unterschiedlichen Umgebungen ist und dass die Bewegung allein durch die Sensoren an der Hand überwacht werden kann. Bei anderen Verfahren, ohne Drucksensoren in der Hand, müssen externe Sensoren den Bewegungsverlauf überwachen. Hierbei kann dann wieder der hohe Grad an Diversität in Farbe, Form und Reflexionsgrad einer Tür und ihrer Klinke Probleme bereiten. Dafür sind die Informationen der Drucksensoren recht beschränkt und sind alleinstehend nicht ausreichend für das Lösen komplexer Aufgaben. Schließlich muss der Kontakt zwischen den Sensoren und dem Ziel erst einmal hergestellt werden. Die Autoren von [6] nutzen deswegen ergänzend zu den Drucksensoren noch eine Stereo-Kamera, eine normale Kamera und einen Laser Range Finder.

In vielen Experimenten in diesem Anwendungsgebiet werden Teilaufgaben wie die Erkennung der Position der Tür oder der Klinke durch bestimmte Maßnahmen vereinfacht. Manche nutzen bestimmte Tools, um dem Roboter das Finden der Türklinke zu vereinfachen. Dadurch wird der Umfang der Aufgabe vereinfacht und es kann mehr Fokus auf andere Elemente der Aufgabe konzentriert werden. Die Autoren von Paper [6] nutzen hierfür beispielsweise einen grünen Laser Pointer in Kombination mit einem Grünfilter und einer Stereo-Kamera. Mit dem Laserpointer wird dem Roboter die Position der Türklinke gezeigt.

Manche Papers stellen neben der Sensorik und Auswertungsverfahren zusätzlich noch ein mathematisches Modell zum Öffnen von Türen vor [3] [7]. Diese Modelle werden genutzt, um aus Sensordaten auf einen genauen Zustand der Tür zu schließen. Diese Informationen können dann für eine exakte Planung des Ablaufs genutzt werden.

2.2 Durchfahrtsalgorithmus

Die zweite große Teilaufgabe zum Öffnen von Türen ist der Ablauf, der für eine möglichst fehlerfreie Manipulation der Umwelt durch den Roboter verantwortlich ist. Um einen solchen Ablauf gestalten zu können, wird dieser in der Regel in einzelne Aufgaben unterteilt, die dann in einem Prozess miteinander verknüpft werden. Diese Aufgaben können einfach sein, wie zum Beispiel eine Bewegung zu einer Zielkoordinate oder komplexer, wie die Aufgabe „drücke Klinke“. Diese komplexeren Teilaufgaben können dann wiederum in noch feinere

Teilaufgaben unterteilt werden. In dem Fall von „drücke Klinke“ wären die minimalen Aufgaben das nacheinander Anfahren von unterschiedlichen Koordinatenpunkten.

Umgesetzt und modelliert werden diese Abläufe meistens mit Modellen, die sich aufgrund ihrer Struktur, Robustheit und Flexibilität etabliert haben. Hauptsächlich werden hierbei Finite State Machines (FSM) [8] oder Behaviour Trees (BT) [9] genutzt.

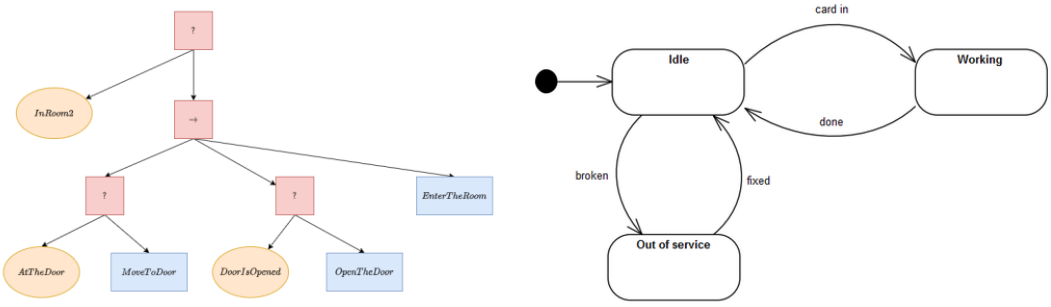

Abbildung 1: Behaviour Tree (links), Finite State Machine (rechts)

In Abbildung 1 kann man links eine Modellierung eines einfachen Behaviour Trees sehen. Rechts kann man ein Modell einer Finite State Maschine erkennen. Inhaltlich modellieren die beiden Beispiele unterschiedliches Verhalten. Die Abbildung soll zur Veranschaulichung der unterschiedlichen Modelle dienen.

3 Vorstellung des Systems

Nachdem andere Lösungsansätze vorgestellt wurden, geht es in den folgenden Seiten um die Lösung, die im Rahmen dieser Ausarbeitung erarbeitet wurde. Die Entwicklung wurde hauptsächlich mit Hilfe einer Simulation der Robotikplatform „Husky“ durchgeführt. Um ein genaues Bild der Simulation und dem Husky selbst zu vermitteln, wird dieser im Folgenden genauer vorgestellt. Des Weiteren werden zusätzliche Hilfsmittel und Methodiken erläutert, die für den späteren Ablauf essenziell sind.

3.1 Rahmenbedingungen

Das Ziel des Projektes ist es, eine robuste Lösung zu kreieren, die auf dem Husky eingesetzt werden kann. Performance wurde hierbei größtenteils vernachlässigt. Die Kommunikation zwischen den Komponenten, samt Überwachung des Systems, wird über das Robot Operating System (ROS) umgesetzt. Aufgrund der niedrigen Relevanz der Performance des Systems, wurden die einzelnen Softwarekomponenten größtenteils in Python realisiert. Dies ermöglicht ein schnelles Prototyping und Entwickeln, vor allem in Kombination mit der Python Version von ROS: „rospy“.

Der betrachtete Rahmen, für den das System entwickelt wurde, umfasst, dass der Husky in einem Raum steht, eine Tür detektiert, zu dieser hinfährt und sie öffnet. Dies entspricht dem maximalen Aufgabenumfang, der bei diesem Prozess möglich ist. Hierbei soll es außerdem irrelevant sein, in welche Richtung die Tür zu öffnen ist und auf welcher Seite sich die Türklinke befindet. Türen die bereits zum Teil oder ganz geöffnet sind werden nicht behandelt.

3.2 Die Fahrplattform

Der Husky ist ein Industrieroboter, der in der intelligenten Quartiersmobilität eingesetzt werden soll. Intelligente Quartiersmobilität bedeutet, das autonome Zurechtfinden in unterschiedlichsten Quartieren. Quartiere können alle möglichen Umgebungen sein, wie Wohngebiete, Fußgängerzonen, Büroräume oder auch ein ganzer Campus [10].

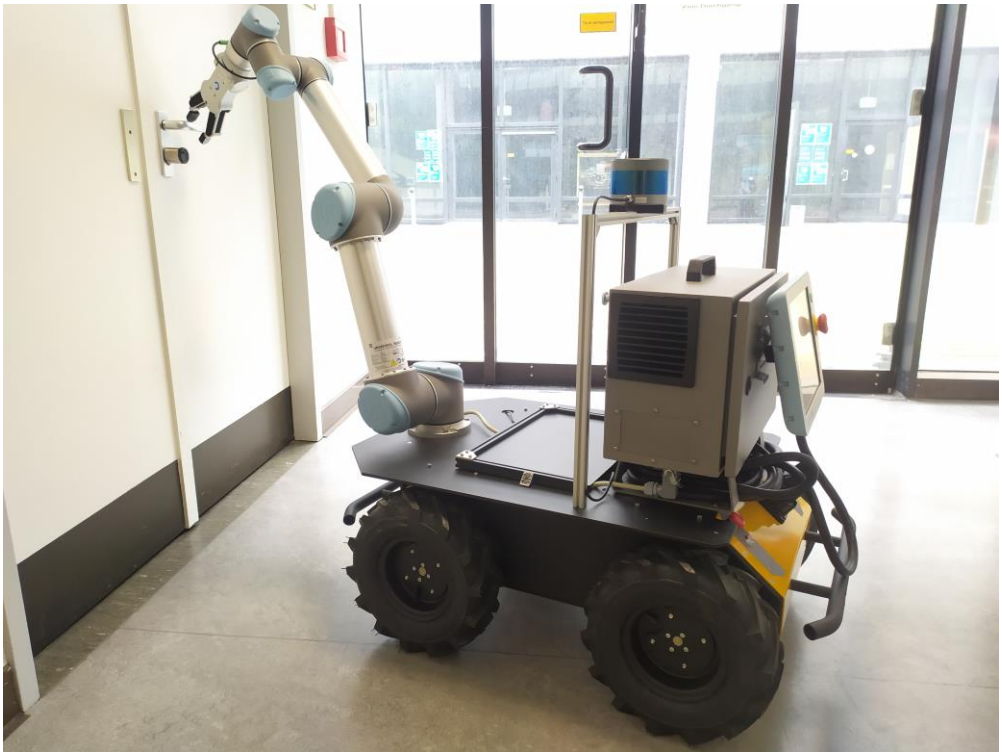

Abbildung 2: Der HAW-Husky [10]

Der Husky besteht aus einer Grundplattform mit vier Rädern. Die Räder sind pro Seite separat ansteuerbar (Panzersteuerung) und haben ebenfalls pro Seite eine Radencoder, um die Odometrie messen zu können. Angetrieben wird das System von einem 24 Volt / 44 Amperestunden Lithium-Ionen-Akku. Zur Orientierung dienen zwei „Intel Realsense 435d“ und ein „Robosense Rs16“ Lidar. Der Lidar ist, wie in Abbildung 2 zu sehen, oben auf dem Rahmen montiert, um ein möglichst freies Sichtfeld zu ermöglichen. Eine der beiden Kameras ist unten an der Front der Grundplattform montiert und die andere an dem Endeffektor des Arms. Der Arm ist von Universal Robots (UR-5), besitzt sechs Gelenke und einen Gripper als Endeffektor. Des Weiteren sind noch eine IMU „UM-7“ und ZOTAC Mini-PC verbaut.

Somit verfügt der Husky über zwei Stereo-Kameras, die normale Bilder und Tiefeninformationen aufnehmen können, sowie über einen High-Performance 16-Ebenen Lidar, der detaillierte 3D-Pointclouds liefern kann. Basierend auf diesen Sensoren wird der Lösungsansatz zur Erkennung der Umgebung aufgebaut.

3.2.1 Orientierung und Steuerung der Fahrplattform

Die Fahrtplanung zur Bewegung des Husky wird durch die ROS Implementierung von „move_base“ [11] übernommen. Dieses Package nutzt lokale und globale Planer in Kombination mit dem Gmapping Algorithmus [12], um Navigationsaufgaben einer Fahrplattform zu lösen. Gmapping ist ein laser-basierter SLAM Algorithmus. Das Gmapping Package erstellt lokale und globale „Costmaps“, die zeigen, welche Bereiche der Umgebung frei befahrbar, beziehungsweise geblockt sind. Die in Abbildung 3 gezeigte „Costmap“ entsteht durch die Auswertung des Lidar Scans, wenn der Husky frontal vor einer Wand steht, wie es in später Abbildung 4 zu sehen ist.

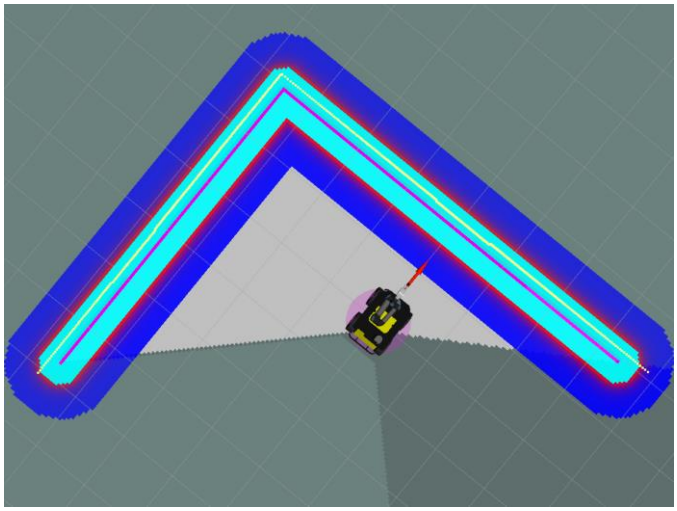

Abbildung 3: gmapping Costmap

Zusätzlich zur Steuerung und Pfadplanung des Husky liefert das „move_base“ Package Feedback über den Status der aktuellen Bewegung.

```
# Pose Message
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

Listing 1: Pose Message Aufbau

Um den Husky an eine bestimmte Position fahren zu lassen, muss dem „move_base“ Server eine Pose gesendet werden. Hierbei sind die „position“ und „orientation“ die Position und Rotation im Raum in Relation zu dem Koordinatensystem von „move_base“ Namens „odom“.

Es ist zudem möglich den Motorkontroller des Husky direkt über ROS-Topics anzusteuern. Dies ist für manuelle Korrekturen oder feste Bewegungsabläufe von Vorteil. Dafür wird eine Twist Message an das „/cmd_vel“ Topic gesendet.

```
# Twist Message
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

Listing 2: Twist Message Aufbau

Der Husky Motorkontroller hat für das Empfangen solcher Messages ein Timeout, sodass er nach 0.5 Sekunden ohne Empfang einer neuen Message automatisch die Bewegung anhält. Der „linear“ und „angular“ Teil der Twist Message beschreiben die Geschwindigkeit, mit der sich der Husky entweder linear, angular oder einer Kombination aus beidem bewegen soll. Ein Beispiel für eine Vorwärtsbewegung mit einem Meter pro Sekunde wäre eine Twist Message mit einem „linear“ Wert von (1,0,0).

3.2.2 Steuerung des Arms

Das Bewegen eines Armes mit mehreren Gelenken, wie dem UR-5 des Husky, ist keine triviale Aufgabe. Es gibt dafür drei Möglichkeiten:

Die erste und einfachste besteht darin alle Gelenke per Winkelangaben direkt anzusteuern. Diese Methode ist jedoch nur für sehr einfache oder im Vorfeld definierte Bewegungen geeignet.

Für komplexere Bewegungen oder das Anfahren einer bestimmten Position ist es notwendig, dass der Anwendung eine Pose übergeben werden kann. Um diese Pose erreichen zu können, muss die Anwendung in der Lage sein, für alle Gelenke eine Sequenz von Winkeln zu errechnen, die nacheinander angefahren werden müssen. Dieser Vorgang wird „motion planning“ genannt. Wenn die Berechnung erfolgreich war, entsteht ein eindeutiger Plan, wie der Arm sich

bewegen muss, um das Ziel zu erreichen. Dieser Vorgang wird erschwert durch zusätzliche Bedingungen, wie zum Beispiel das Vermeiden von Kollisionen mit sich selbst oder der Umgebung.

Als letzte Möglichkeit, gibt es das Planen einer Bewegung mit Hilfe von mehreren Posen, die der Endeffektor während der Bewegung passieren soll. Diese Posen bilden einen kartesischen Pfad, der dann durch das „motion planning“ interpoliert und im Anschluss wieder in einen Ablaufplan umgerechnet wird.

Der Prozess des „motion planning“ und das Ausführen der Bewegung wird über das Motion Planning Framework „move_it“ [11] durchgeführt, welches ROS Wrapper in C++ und Python bereitstellt.

3.2.3 Gazebo Simulation

Zum Durchführen der Experimente, wurde eine Simulation des Husky in Gazebo [13] mit ROS aufgesetzt.

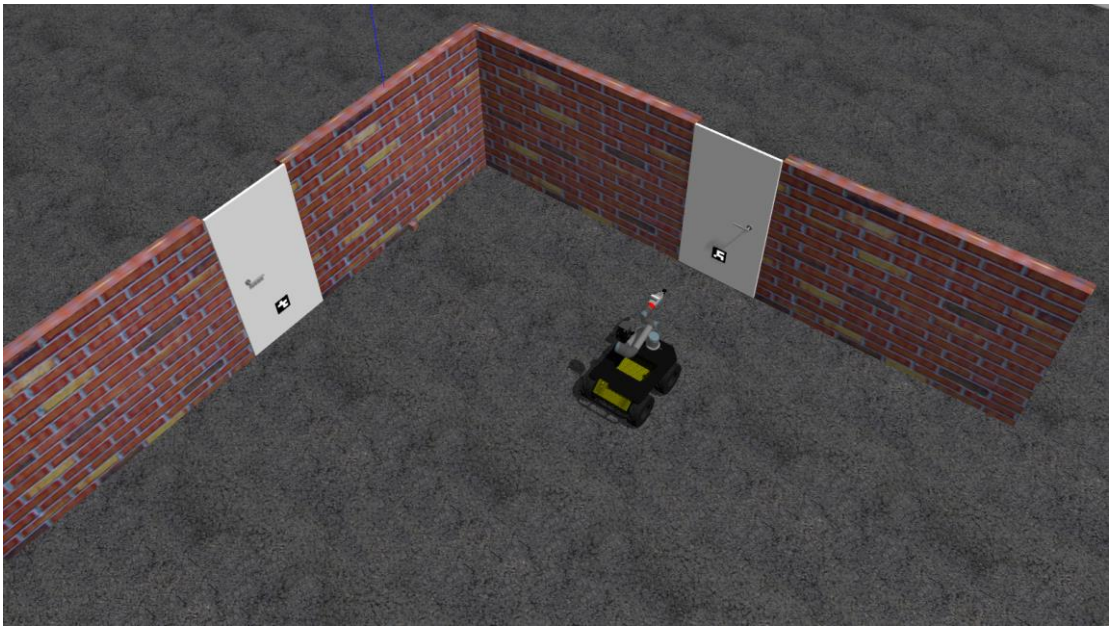

Abbildung 4: Simulationsumgebung

Neben dem Husky, wurden in der Welt Wände und Türen platziert, um eine reale Umgebung simulieren zu können (siehe Abbildung 4). Des Weiteren wurde ein Gazebo-Plugin implementiert, durch das die Türen bewegt und mit der Klinke ent- und verriegelt werden können. Man kann in Abbildung 4 die ArUco-Marker an den Türen erkennen, welche im Folgenden Abschnitt vorgestellt werden.

3.3 ArUco Marker

Die Lokalisierung der Türen im Raum wird mit Hilfe von ArUco-Markern durchgeführt. Wie sie konkret in dem System genutzt werden wird in dem Teil der Umsetzung 5.3.3 (Abschnitt „Aruco Marker detektieren“) erläutert.

ArUco-Marker wurden von der Augmented Reality University of Cordoba entwickelt und sind sogenannte Bezugsmarkierungen (eng. fiducial markers). Essenzielle Merkmale dieser Marker sind die quadratische Form, der breite schwarze Rand und eine binäre Matrix innerhalb des Randes, die die ID des Markers festlegt (siehe Abbildung 5). Diese Form, insbesondere der schwarze Rand, sorgen für eine schnelle und robuste Erkennung [14].

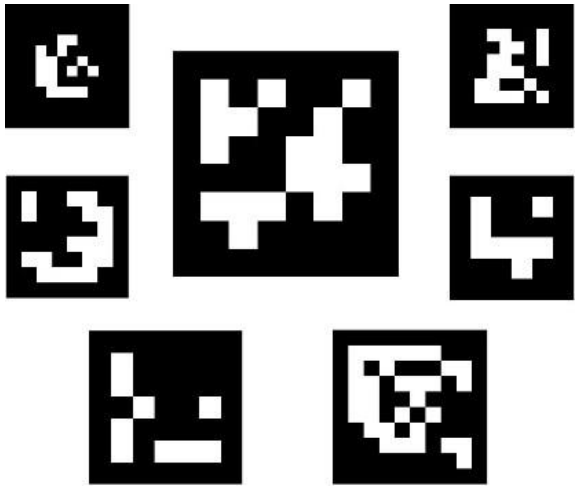

Abbildung 5: ArUco Marker Beispiele

Um eine Menge von ArUco-Markern in einer spezifischen Anwendung klassifizieren zu können, werden diese in einem „Dictionary“ zusammengefasst. Dieses „Dictionary“ ist eine Menge von Markern gleicher Größe mit unterschiedlichen Matrizen. Umso ein „Dictionary“

generieren zu können, werden zwei Parameter benötigt: die Anzahl an unterschiedlichen Markern und die Größe der Matrix (Menge an Bits), die die ID beschreiben. Diese generierten Marker kann man ausdrucken und in der Umgebung platzieren. Mit Hilfe des „Dictionary“, der Größe der Marker und den Kamera Informationen (Kamera Matrix, Distortion Coefficients) ist es möglich, die Position der Marker im Raum mit einer normalen Kamera zu erkennen.

3.4 Behaviour Trees

Für dieses Projekt werden Behaviour Trees zur Umsetzung des Prozesses genutzt, weswegen diese im Folgenden kurz erläutert werden.

Behaviour Trees basieren auf einer Baum Struktur, die immer von oben nach unten durchlaufen (getickt) wird. Der Beginn eines Behaviour Trees ist immer die „Root Node“, bei der ein Tick gestartet wird. Zum Aufbau des Baumes gibt es eine begrenzte Auswahl an Bausteinen aufgeteilt in zwei Kategorien. Die erste Kategorie sind die „Execution Nodes“ (in Tabelle 1 mit „E“ gekennzeichnet), die für das Ausführen von Aktionen verantwortlich sind und immer am Ende eines Astes, als Blatt, platziert werden. Zu der zweiten Kategorie gehören die „Control Flow Nodes“ (in Tabelle 1 mit „CF“ gekennzeichnet), die für den Kontrollfluss durch den Baum verantwortlich sind. Alle Nodes können nach einem Tick einen von vier Zuständen zurückgeben: „success“ (Aktion erfolgreich beendet), „running“ (Aktion läuft noch), „invalid“ (Aktion ist undefiniert) oder „failure“ (Aktion war nicht erfolgreich). Dieser Rückgabewert beeinflusst rückwirkend das Verhalten der übergeordneten Nodes und des Baumes [9].

In Tabelle 1 werden die unterschiedlichen Nodes mit ihrer Bedeutung dargestellt.

Name	***Bedeutung***
Action-Node (E)	Führt Aktionen aus (Beispiel: Bewege Arm, Fahre nach x)
Condition-Node (E)	Prüft Zustandsvariablen. (Beispiel: Counter == 5?)
Sequence-Node (CF)	Führt die „Child Nodes“ der Reihe nach aus und gibt SUCCESS zurück, wenn alle erfolgreich waren. Bricht ab, sobald eine der Nodes fehlgeschlagen ist und gibt FAILURE zurück.

Fallback-Node (CF)	Führt die „Child Nodes“ der Reihe nach aus, bis eine erfolgreich war und gibt dann SUCCESS zurück. Der Rest wird nicht ausgeführt. Wenn alle fehlschlagen wird FAILURE zurückgegeben.
Parallel-Node (CF)	Führt alle „Child Nodes“ parallel aus und gibt SUCCESS zurück, wenn eine gegebene Anzahl oder alle erfolgreich waren.
Decorator-Node (CF)	Modifiziert den Rückgabewert einer „Child Node“. (Beispiel: FAILURE ist SUCCESS, SUCCESS ist RUNNING)

Tabelle 1: Behaviour Tree Nodes [9]

Mit dieser Menge an Nodes ist es möglich fast jedes Verhalten zu beschreiben.

Die Aktion Nodes implementieren die unterschiedlichen Aufgaben und haben alle eine grundlegende Menge an Methoden, die durch das Framework vorgegeben wird (siehe Tabelle 1)

Name	***Rückgabe***	***Bedeutung***
Setup	Bool	Einmaliger Aufruf bei Start des Baums
Initialise	-	Aufruf bei jedem Neustart der Node (bedeutet Node hat vorher nicht „running“, sondern „failure“ oder „sucess“ zurückgegeben)
Update	Status	Aufruf bei jedem Tick, wenn Node erreicht wird. Führt Aufgabe aus und gibt Status dieser zurück.
Terminate	-	Zum Unterbrechen der Aufgabe durch Node mit höherer Priorität
Shutdown	-	Definiert Verhalten bei Herunterfahren des Baumes. Gegenteil von Setup

Tabelle 2: Aktion Node Struktur [15]

Ein weiteres wichtiges Feature von Behaviour Trees sind Blackboards. Diese dienen als zentraler Speicherungsort, an dem Daten gespeichert und ausgelesen werden können. Das

Blackboard ist im gesamten Baum bekannt und kann somit als Datenbank des Baums gesehen werden.

Der Vorteil der oftmals mit der Nutzung von Behaviour Trees verbunden wird ist, dass sie stark modular aufgebaut sind und der Kontrollfluss nur durch die Struktur des Baums beeinflusst wird (siehe [16], [17]). Dies hat zur Folge, dass die einzelnen Nodes komplett unabhängig voneinander existieren und einfach ausgetauscht werden können. Bei FSMs hingegen sind die Zustände durch die Transition aneinandergekoppelt und mit wachsender Größe wird der Aufwand größer, wenn man Teile modifizieren oder austauschen will. Aufgrund dessen haben die einzelnen Nodes eines Behaviour Trees einen hohen Wiederverwendungsgrad. Des Weiteren skalieren Behaviour Trees dadurch gut, dass die einzelnen Nodes ebenfalls komplexere Sub-Bäume sein können, wie es bei Hierarchischen State Machines mit den Hierarchischen States der Fall ist.

Aufgrund dieser Vorteile wurde sich für Behaviour Trees und gegen State Machines als grundlegendes Programmierschema entschieden.

4 Modellierung des Ablaufs

Bevor man einen Prozess in Form von State Machines oder Behaviour Trees modelliert, ist ein Modell des Ablaufs von Nöten. Dieses hilft dabei zu definieren, welche Schritte in dem Prozess beachtet werden müssen, um das gewünschte Ziel zu erreichen. In den bereits vorgestellten Lösungen werden solche Modelle ebenfalls genutzt. Beispiele hierfür kann man in Abbildung 6 und 7 sehen.

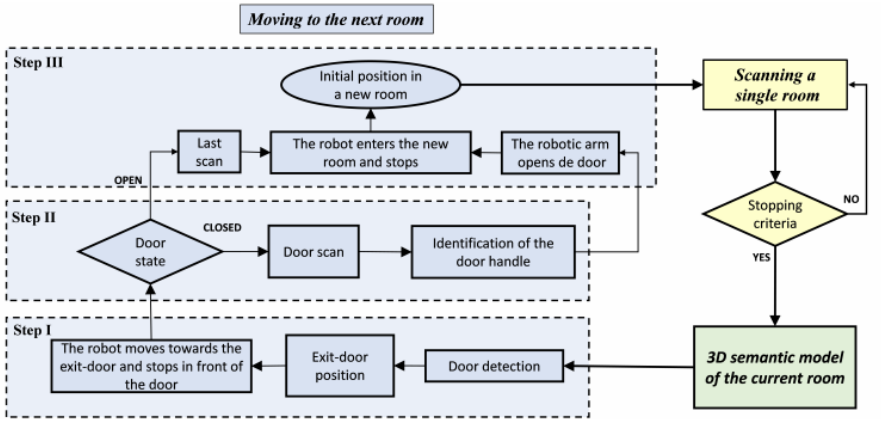

Abbildung 6: Tür öffnen mit einem 3D-Scan Roboter [3]

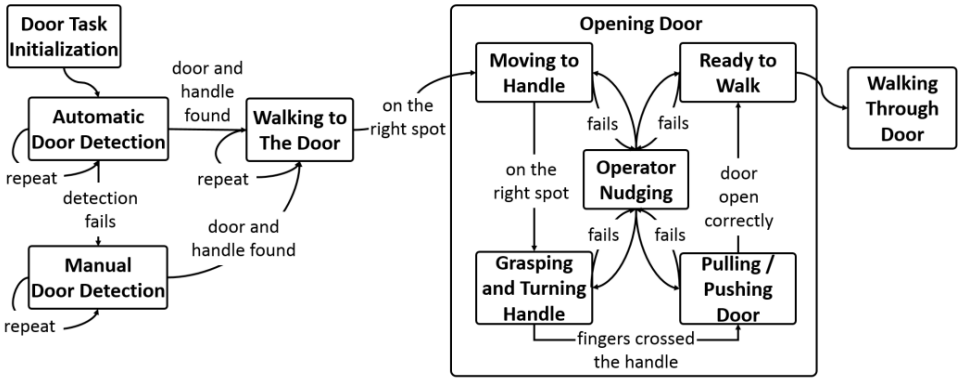

Abbildung 7: Tür öffnen mit einem Humanoiden Roboter [1]

Aus den hier gezeigten Beispiellösungen lassen sich zentrale Punkte entnehmen, die den Öffnungsprozess grundlegend beschreiben:

1. Anfahren einer Ausgangsposition (beziehungsweise Positionierung des Roboters an Ausgangsposition)
2. Identifizieren der Türklinke (gegebenenfalls samt Tür)
3. Anfahren einer Position zum Öffnen der Tür (wenn diese Aufgabe noch nicht durch Schritt eins übernommen wurde)
4. Gripper in Position bringen
5. Tür öffnen
6. Tür passieren

Die Gemeinsamkeiten zwischen den Lösungen lassen sich darauf zurückzuführen, dass man bei dem Ablauf eine Tür zu öffnen, prinzipiell keine großen Änderungen vornehmen kann. Die größten Unterschiede sind darauf zurückzuführen, dass unterschiedliche Paper unterschiedliche Umfänge betrachten. Einige lassen den Roboter unter stark kontrollierten Bedingungen nur durch eine Tür fahren, wie die Autoren von Paper [6]. Wohingegen andere Autoren den Anspruch haben, dass der Roboter in der Lage ist, sich autonom durch mehrere Räume zu bewegen [3]. Des Weiteren kann man Unterschiede feststellen, die durch die unterschiedlichen Methoden bei der Erkennung oder durch den Aufbau des Roboters entstehen.

Basierend auf diesen Informationen wurde der folgende Ablauf, der in Abbildung 8 zu sehen ist, entwickelt.

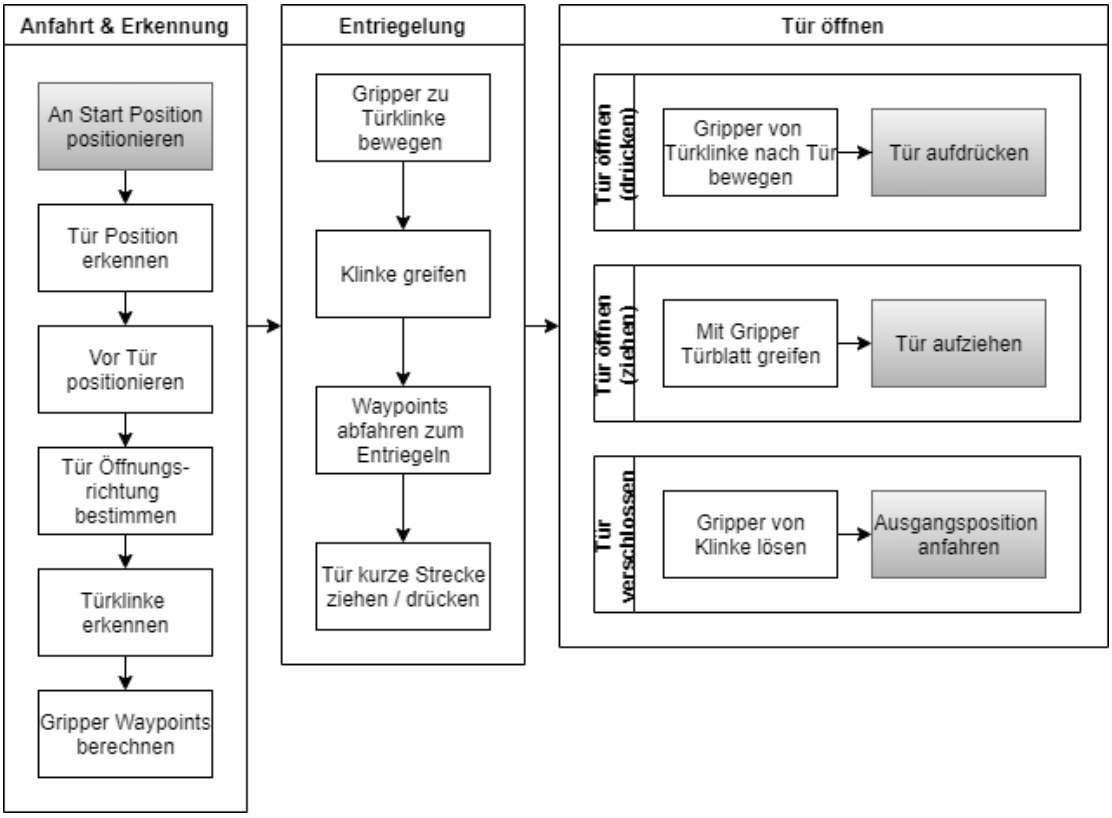

Abbildung 8: Tür öffnen Ablaufplan

Die grauen Felder beschreiben die Start- und Endpunkte des Prozesses. Außerdem wurde der Ablauf in drei grobe Abschnitte unterteilt: „Anfahrt und Erkennung“, „Entriegelung“ und „Tür öffnen“.

4.1 Phase 1: Anfahrt und Detektion

Der Prozess beginnt mit der Phase „An Start Position positionieren“, was bedeutet, dass das Öffnen der Tür in einer beliebigen Position gestartet werden kann, aus der der Husky eine Tür sehen kann. Im Anschluss geht es in der ersten Phase darum, den Husky in eine Ausgangsposition zu bewegen, aus der die Türklinke gut detektiert und die Tür geöffnet werden kann. Nachdem diese Position erreicht wurde, soll die Öffnungsrichtung bestimmt, die Türklinke erkannt und der Pfad zum Entriegeln der Klinke errechnet werden.

Der Endzustand nach der „Anfahrt und Detektion“ Phase beinhaltet, dass der Husky möglichst senkrecht vor der Tür steht und alle relevanten Informationen (Tür Position, Klinken Position, Entriegelungs-Wegpunkte, Öffnungsrichtung der Tür) besitzt, um mit dem Entriegelungsprozess zu beginnen.

4.2 Phase 2: Entriegelung

Der Teilprozess der Entriegelung besteht aus dem Ausführen von Steuerungsbefehlen, die auf den in der ersten Phase berechneten Daten basieren. Diese Befehlskette beginnt mit dem Positionieren des Grippers an der Türklinke. Wenn die Position erfolgreich erreicht wurde, wird versucht den Gripper zu schließen, um die Türklinke zu greifen. Nachdem dieser Schritt abgeschlossen wurde, wird versucht die Tür durch das Abfahren der berechneten Koordinaten zu öffnen. Nachdem die letzte Koordinate erreicht wurde, kann die Tür theoretisch geöffnet werden.

Da die Öffnungsrichtung der Tür bereits ermittelt wurde, wird im Folgenden versucht die Tür in die detektierte Öffnungsrichtung zu bewegen. Wenn das erfolgreich ist, gilt die Tür als entriegelt und die Phase als abgeschlossen. Wenn es jedoch fehlschlägt, wird der Vorgang wiederholt oder die Tür als verschlossen angesehen.

Der Endzustand nach der Entriegelungsphase besteht darin, dass der Roboter die Tür entweder entriegelt hat und sie nun in einem bestimmten Winkel geöffnet ist oder darin, dass er festgestellt hat, dass sie verschlossen ist.

4.3 Phase 3: Das Öffnen der Tür

Basierend auf den Ergebnissen des Entriegelungsprozesses wird in der „Tür öffnen“ Phase der Prozess abgeschlossen. Je nachdem, ob die Tür zu öffnen ist oder nicht, wird der passende Prozess ausgeführt, um die Aufgabe zu abzuschließen. Im Falle dessen, dass die Tür verschlossen ist, fährt der Husky die Ausgangsposition des Experimentes an, um sicherzustellen, dass er die Tür nicht blockiert. Falls die Tür erfolgreich entriegelt wurde, versucht er sie je nach Öffnungsrichtung komplett zu öffnen.

Nachdem die letzte Phase beendet wurde, befindet sich der Roboter in einem von zwei möglichen Zuständen. Eine Möglichkeit ist, dass er die Tür geöffnet hat und nun passieren kann. Die andere Möglichkeit besteht darin, dass er die Tür nicht öffnen konnte und wieder zu seiner Ausgangsposition zurückgekehrt ist.

5 Umsetzung des Prozesses

Für die Implementierung des Behaviour Trees wurde die Open-Source-Bibliothek „py-trees“ [15] genutzt. Diese bietet mit der Erweiterung „py-trees-ros“ [18] alle grundlegenden Funktionalitäten, die man zum Implementieren eines Behaviour Trees braucht. Die Erweiterung „py-trees-ros“ liefert eine allgemeine Schnittstelle zu ROS, durch die das Konzept von asynchronen ROS-Callbacks in Behaviour Trees integriert werden kann.

Im Folgenden werden die zentralen Komponenten des Systems und wichtige Basisklassen vorgestellt, die für die Erklärung der einzelnen Nodes von Relevanz sind. Danach wird kurz der Modellierungssyntax für Behaviour Tree Diagramme erläutert. Anschließend werden alle Teile des Behaviour Trees vorgestellt und bei erstem Erscheinen einer Nodes wird die Umsetzung dieser erklärt. Zum Schluss wird auf das Fehlermanagement in dem System eingegangen.

5.1 Zentrale Steuerungselemente und Basisklassen

5.1.1 Gripper Controller Action Server

Der Gripper des Husky ist unabhängig vom Arm steuerbar und muss somit auch extra angesprochen werden. Um das zu erreichen wurde ein Action-Server [19] implementiert. Dieser entkoppelt das Ausführen der Bewegung von dem Aufruf, wodurch der aufrufende Code asynchron weiterlaufen kann. Dies ist für die Kombination mit Behaviour-Trees essenziell, weil diese mit einer konstanten Frequenz ticken und blockierende Aufrufe dieses Tick Verhalten stören würden.

Die Action-Server sind eigenständige ROS-Nodes, die in einem bestimmten Pattern mit Goal, Result und Feedback kommunizieren. Um die Kommunikation zu spezifizieren wurde folgendes Action-File definiert.

```
# Goal
float32 target_width
float32 stepsize
uint32 tick_time
---
# Result
float32 current_width
---
# Feedback message
float32 current_width
```

Listing 3: Gripper Action Message Aufbau

Das Goal besteht aus der Ziel Breite („target_width“), der Schrittgröße des Grippers pro Tick („stepsize“) und der Zeit pro Tick („tick_time“). Die letzten beiden Parameter werden in der Simulation genutzt, um die Bewegungsgeschwindigkeit des Grippers zu definieren. Als Feedback und Result wird die momentane Öffnungsweite des Grippers gewählt.

Die Bewegung des Grippers wird mit dem Senden eines Goals an den Action Server gestartet und mit dem periodischen Abfragen des Status des Servers überwacht.

5.1.2 Arm Controller

Damit die Bewegungen des Armes einfach und sicher ausgeführt werden können, wurde ein zentraler Controller implementiert, der alle Funktionalitäten bündelt. Dazu gehört das Aufbauen der Verbindung zu dem „move_it“ Server, das Setzen von grundlegenden Parametern und das Anbieten von Methoden wie dem Zurücksetzen des Armes. Des Weiteren werden die unterschiedlichen Bewegungsarten (Winkel angeben, Pose angeben, kartesischen Pfad angeben) des Armes abstrahiert und können somit leichter von allen anderen Komponenten aufgerufen werden.

5.1.3 Single Arm Movement (Basisklasse)

Diese Klasse bildet die Basis für eine Behaviour-Tree Node, die eine Arm Bewegung jeglicher Art ausführt. Eine Node die von dieser Klasse erbt, muss lediglich den Typ der Bewegung und das Ziel festlegen. Mögliche Bewegungstypen und deren Datenformat kann man in Tabelle 3 sehen.

Bewegungstyp	***Ziel Typ***
Gelenke setzen	Liste von sechs Winkeln angegeben in Radianten Beispiel: [PI, -2.79, 2.44, 3.45, -PI/2, 0]
Pose setzen	ROS geometry_msgs/Pose [20] bestehend aus: - Position (x,y,z) - Ausrichtung in Quaternion (x,y,z,w)
Kartesischen Pfad setzen	Liste von Posen

Tabelle 3: Arm Bewegungstypen

Die Bewegung wird im Anschluss über den Arm Controller gestartet und kann auch über diesen pro Tick überwacht werden. Je nachdem was der Arm Controller für einen Status liefert, passt sich der Rückgabewert der Node an.

5.1.4 Single Drive Movement (Basisklasse)

Von dieser Basis Node erben alle Action Nodes, die die Husky-Fahrplattform über „move_base“ an eine bestimmte Position bewegen wollen. Die erbenden Nodes müssen nur die Ziel Pose, wie in Abschnitt 4.2.1 beschrieben, in der Initialisierung setzen. Die Basis Node übernimmt dann in der Update-Methode das Senden an den „move_base“ Action Server und überwacht die Bewegung durch das Abfragen des Status pro Tick. Auch hier verändert sich abhängig von dem Server Status der Rückgabewert der Node.

5.2 Behaviour Tree Modellierungssyntax

Um das Verständnis der folgenden Diagramme zu erleichtern, werden in Tabelle 4 die Symbole zur Modellierung der Behaviour Trees vorgestellt.

Zeichen	***Bedeutung***
	Fallback-Node
	Fallback-Node (Ohne Gedächtnis)
	Sequenz-Node
	Sequenz-Node (Ohne Gedächtnis)
	Parallel-Node
	Decorator-Node (Inverter)
	Decorator-Node
	Action-Node
	Condition-Node
	Set-Condition-Node
	Subbaum

Tabelle 4: Modellierungssyntax

5.3 Behaviour Tree Implementierung

Im Anschluss wird die konkrete Umsetzung des Behvaiour Trees mit allen dazugehörigen Algorithmen vorgestellt. Gegliedert ist die Erklärung nach dem Hauptbaum und den einzelnen Subbäumen: Daten erheben, Anfahrt und Detektion, Entriegelung, Tür passieren.

5.3.1 Hauptbaum

Der Hauptbaum beschreibt die oberste Ebene des Prozesses. Hier wird das grundlegende Verhalten durch das Zusammenfügen aller Komponenten festgelegt. Er besteht aus normalen Action-Nodes wie der Fehleranalyse, aber auch aus einer Menge von Subbäumen wie dem Baum „Daten erheben“.

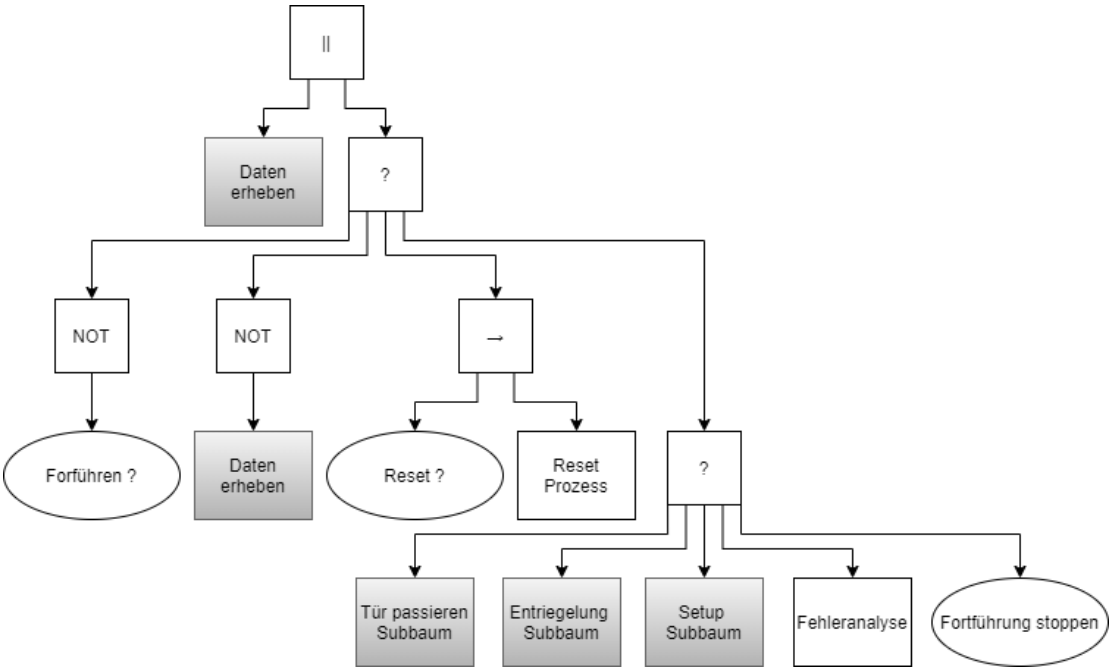

Abbildung 9: Hauptbaum

Es gibt mehrere Möglichkeiten, die Nodes unterschiedlich anzuordnen und trotzdem das gleiche Verhalten zu modellieren. Die Lösung, für die sich entschieden wurde und welche in

Abbildung 9 zu sehen ist, ist möglichst kompakt und bildet das gewünschte Verhalten komplett ab.

Es fällt auf, dass der „Daten erheben“ Subbaum mehrmals genutzt wird, und zwar zum einen in der obersten Ebene in Kombination mit einer Parallel-Node und zusätzlich noch in einer Ebene darunter nach dem „Fortführen ?“ Check. Das hat den Grund, dass bei manchen Bewegungen konstant Daten gesammelt werden müssen. Das wird durch die parallel Node gewährleistet, die bei jedem Tick, unabhängig von dem Kontrollfluss des restlichen Baumes, ausgeführt wird. Diese Umsetzung hat jedoch den Nachteil, dass nicht garantiert ist, dass die Daten bereits im Blackboard gespeichert sind, bevor sie zum ersten Mal durch andere Komponenten abgefragt werden. Der „Daten erheben“ Subbaum, der auf der unteren Ebene liegt, schützt davor und speichert bei Start oder Reset des Systems alle Daten mindestens einmal ab, bevor sie gebraucht werden.

Abseits davon ist der Kontrollfluss des Hauptbaums überschaubar. Zu Beginn wird überprüft, ob der Ablauf fortgeführt werden soll. Im Anschluss wird geprüft, ob die Daten des Systems zurückgesetzt werden sollen oder ob auf den bestehenden weitergearbeitet werden soll. Schlussendlich wird der eigentliche Öffnungsalgorithmus ausgeführt.

Dieser ist, wie in der Modellierung beschrieben, in drei Schritte unterteilt, welche durch Subbäume repräsentiert werden. Falls in diesem Ablauf etwas nicht ordentlich funktioniert, wird der Prozess durch die Fehleranalyse aufgefangen. Wenn das Korrigieren des Fehlers auch fehlschlagen sollte, wird der ganze Prozess abgebrochen und die Fortführung sofort gestoppt.

5.3.2 Subbaum: Daten erheben

Dieser Subbaum sorgt für die Datenaufnahme in das System. Er besteht aus einer Parallel-Node die auf den Erfolg von allen Child-Nodes wartet (siehe Abbildung 10) und diesen Child-Nodes.

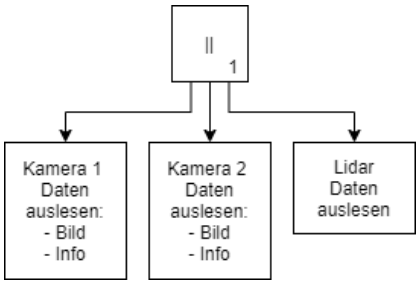

Abbildung 10: Subbaum Daten erheben

Besagte Child-Nodes sind dafür verantwortlich die Daten in dem Blackboard zu updaten. Da mit einem ROS-System gearbeitet wird, wird diese Aufgabe durch die Bibliothek „py-trees-ros“ übernommen. In einem ROS-System wird mit asynchronen Callbacks gearbeitet, was bedeutet, dass der Baum diese Daten normalerweise nicht in seiner Tick Frequenz abfragen kann. Um diese beiden Verhalten trotzdem kombinieren zu können, dienen die Action-Nodes des „Daten erheben“ Subbaums als Adapter zwischen Behaviour Trees und ROS Callbacks. Die einzelnen Nodes hören auf gewünschte ROS-Topics und speichern in dem dazugehörigen Callback die Daten zwischen. Wenn sie durch den Tick des Baumes aufgerufen werden und bereits neue Daten durch den Callback abgespeichert wurden, werden diese Daten in das Blackboard übertragen. Dadurch kann sichergestellt werden, dass trotz der Asynchronität eines ROS-Systems, mindestens einmal pro Tick alle Daten erneuert werden.

5.3.3 Subbaum: Anfahrt und Detektion

Dieser Subbaum (siehe Abbildung 11) setzt den ersten Teil des Öffnungsprozesses um. Das Ziel ist es, den Roboter in den Zustand zu versetzen, der in der Beschreibung des Öffnungsalgorithmus am Ende der „Anfahrt und Detektion“ Phase (Abschnitt: 5.1) gefordert wird.

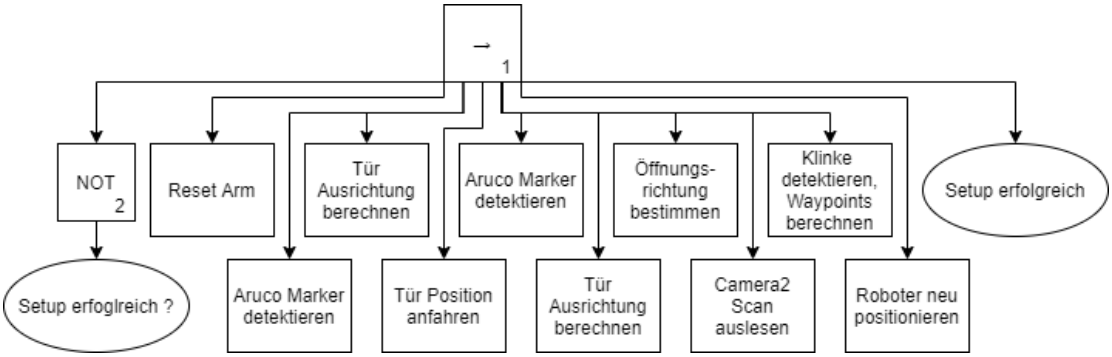

Abbildung 11: Subbaum Anfahrt und Detektion

Die Struktur des Baumes besteht aus einer Sequenz von Checks und Aktionen. Zu Beginn wird überprüft, ob der Baum bereits erfolgreich ausgeführt wurde und wenn nicht, wird der Ablauf fortgesetzt. Anschließend wird der Arm in eine Ausgangsposition gebracht, die Marker und die Tür detektiert und die initiale Ausgangsposition vor der Tür eingenommen. Nachdem der Roboter dort angekommen ist, wird zur Sicherheit erneut die Position und die Ausrichtung der Tür berechnet. Woraufhin weitere Informationen, wie die Öffnungsrichtung der Tür und die Position der Türklinke, bestimmt werden. Je nachdem wie diese Informationen ausfallen wird der Roboter erneut ausgerichtet. Wenn dieser gesamte Ablauf erfolgreich war, wird das im Blackboard festgehalten.

Im Folgenden werden die Umsetzungen der einzelnen Nodes für den „Anfahrt und Detektion“ Baum vorgestellt und erläutert. Manche von Ihnen kommen in späteren Subbäumen erneut vor, werden dann jedoch nicht noch einmal erklärt.

Reset Arm

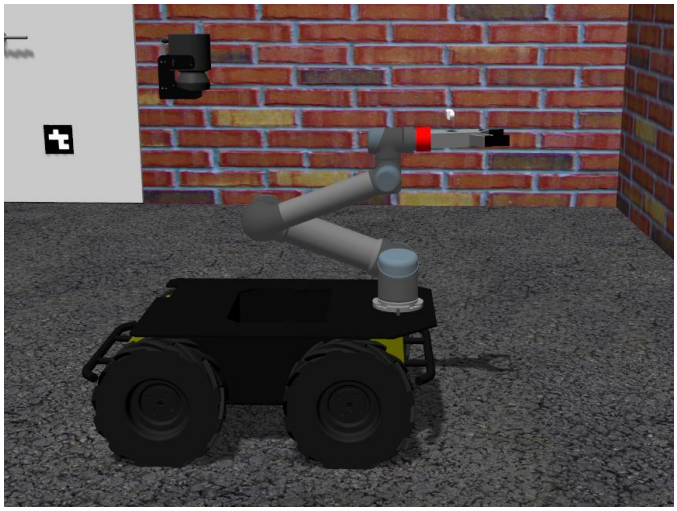

Abbildung 12: Arm Ausgangsposition

Durch diese Node wird der Arm in eine spezifizierte Ausgangsposition geführt (siehe Abbildung 12). Es geht um eine einzelne Bewegung, weswegen sie von der Basis Node „SingleArmMovement“ erben kann. Der Bewegungstyp ist hierbei das direkte Ansteuern der Gelenke. Um die Position aus Abbildung 12 zu erreichen, werden die Gelenkwinkel, wie in Tabelle 5 beschrieben, angefahren.

Gelenk	***Winkel (in Radianten)***
ur_arm_shoulder_pan_joint	3.14
ur_arm_shoulder_lift_joint	-2.79
ur_arm_elbow_joint	2.44
ur_arm_wrist_1_joint	3.52
ur_arm_wrist_2_joint	-1.57
ur_arm_wrist_3_joint	0.0

Tabelle 5: Arm Ausgangsposition Gelenkwinkel

Durch diese Winkel wird garantiert, dass der Gripper samt Kamera möglichst parallel zum Boden positioniert ist und sich auf einer Höhe befindet, die ungefähr der Höhe der Türklinke (0.6-0.8m) entspricht. Das vereinfacht später das Detektieren der Klinke.

Aruco Marker detektieren

Um die Aufgabe zum Identifizieren der Türen zu vereinfachen, werden ArUco-Marker an ihnen befestigt. Zum Erkennen der Marker wird die Bilderverarbeitungs-Bibliothek „Opencv“ genutzt, welche mehrere Methoden bezüglich ArUco-Markern bereitstellt [14].

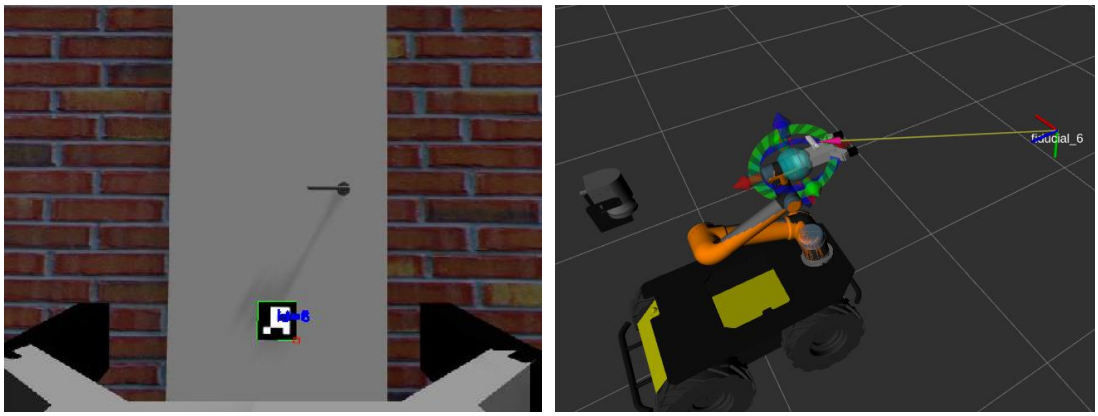

Abbildung 13: ArUco Marker Detektion

Für das Dictionary, welches in dem Projekt genutzt wird und aus dem auch der Marker aus Abbildung 13 stammt, wird eine Matrix Größe von 4x4 und eine Anzahl von 50 Markern genutzt. Außerdem ist die Größe der Marker auf 0.17x0.17m festgelegt (Kantenlänge der Matrix inklusive schwarzem Rand), damit ein Marker auch auf ein DIN-A4-Blatt passen würde.

```
bboxs, ids, reject = detectMarkers(gray, dict, arucoParam)
```

Listing 4: Opencv Marker detektieren Methoden Syntax

Zuvor stehend ist die Methode aus der Opencv-Bibliothek, die zur Detektion der Marker verwendet wird. Die Parameter von links nach rechts sind: das Bild, das gewählte Dictionary und die Parameter, die aus dem Dictionary generiert werden. Zurückgeliefert werden Bounding-Boxen, die IDs der identifizierten Marker und Bounding-Boxen von Marker Kandidaten, bei

denen keine ID identifiziert werden konnte. Die Visualisierung der Ergebnisse einer Detektion kann man in dem linken Bild aus Abbildung 13 sehen (grüner Rand um den Marker mit der ID).

Mit Hilfe der Kamera Informationen „distortion paramters“, „intrinsic matrix“ und der Größe der Marker ist es möglich die Position und Rotation der Marker im dreidimensionalen Raum zu bestimmen.

```
rot, trans, mark = estimatePoseSingleMarkers(bboxs[i], 0.17, k, d)

sendTransform(trans, quaternion_from_euler(rot[0],rot[1],rot[2]),
rospy.Time.now(), tf_name, camera_tf_name)
```

Listing 5: Opencv Marker Pose Estimation und Koordinatensystem Verbreitung

Mit dem hier gezeigten Code Abschnitt wird das Ergebnis erzeugt, welches im rechten Bild der Abbildung 13 zu sehen ist. Hierbei wird die Position und Rotation der Bounding-Box (bboxs[i]) mit der realen Größe von 0.17 x 0.17 Metern und den Kamera Informationen k (für intrinsic matrix) und d (für distortion coefficients) im dreidimensionalen Raum bestimmt. Im Anschluss wird eine Methode aufgerufen, die diese Ergebnisse in einem Koordinatensystem („tf_name“) vereint und für das restliche System in Relation zu einem übergeordneten Koordinatensystem („camera_tf_name“) bereitstellt. Dies vereinfacht den weiteren Umgang mit diesen Daten, weil die Library „tf“ von ROS [21] es ermöglicht, zu jeder Zeit Rotation und Translation zwischen zwei beliebigen Koordinatensystemen, abzufragen.

Tür Ausrichtung berechnen

Nachdem mit Hilfe der ArUco-Marker die Position der Tür im Raum bestimmt wurde, wird mit den Tiefendaten des Lidars, die genaue Ausrichtung der Tür berechnet. Mit dieser und der Position der Tür kann ein Punkt vor der Tür errechnet werden, der als erster Ausgangspunkt angefahren werden soll.

Der Algorithmus zur Berechnung der Ausrichtung beginnt mit dem Errechnen der Punkte im Laserscan, die auf der Tür liegen. Dafür wird zuerst der Winkel zwischen der Ausrichtung des

Husky (x_1, y_1), die durch den Vektor (1,0) repräsentiert wird, und der Richtung zum Marker an (x_2, y_2) der Tür berechnet.

$$signed\_radiant = \arctan\left(\frac{y_1}{x_2}\right) - \arctan\left(\frac{y_2}{x_1}\right) \quad (1)$$

Im Anschluss kann bestimmt werden, welche Punkte des Laserscans auf der Türebene liegen. Um dies zu erreichen, wird die Anzahl und Richtung der Schritte von der Mitte des Scans errechnet. Die Berechnung besteht darin, den Winkel durch die Winkelschrittgröße des Laserscans zu teilen. Das Ergebnis ist die Anzahl an Schritten, die von dem mittleren Wert im Laserscan in eine bestimmte Richtung gegangen werden muss, um den besagten Messwert zu finden. Nun werden zehn Werte rechts und links davon genutzt (insgesamt 20), um eine Gerade zu bilden. Um diese Gerade aufzuspannen, werden die Distanzen (d) der ausgewählten Punkte in kartesische Koordinaten (x, y) umgerechnet, wobei α der Winkel vom Mittelpunkt des Laserscans zu dem gerade betrachteten Messwert ist.

$$x = d * \cos(\alpha) \quad (2)$$

$$y = d * \sin(\alpha) \quad (3)$$

Der Winkel zwischen der Normalen dieser aufgestellten Geraden und der Ausrichtung des Husky, ist die Ausrichtung der Tür aus Sicht des Roboters. Diese Ausrichtung samt der bereits bekannten Position der Tür wird wieder als Koordinatensystem dem restlichen System zur Verfügung gestellt.

Mit Hilfe dieser Daten und einer festgelegten Distanz, die der Husky von der Tür entfernt sein soll (1,5 Metern), wird nun der Punkt berechnet, der als erstes angefahren werden soll.

Abbildung 14: Anfahrtsprozess (Teil 1)

Diese Position kann man beispielhaft in Abbildung 14 erkennen. Die Berechnung für den anzufahrenden Punkt (x_z, y_z) mit Hilfe der Position (x_d, y_d) und Ausrichtung (x_a, y_a) der Tür und der Distanz von der Tür (d_d) sieht wie folgt aus:

$$x_z = x_d + x_a * d_d \tag{4}$$

$$y_z = y_d + y_a * d_d \tag{5}$$

Es muss darauf geachtet werden, dass die Richtungen der unterschiedlichen Koordinatensysteme beachtet werden, damit der resultierende Punkt nicht hinter der Tür liegt.

Tür Position anfahren

Nachdem das Ziel errechnet wurde, fährt der Husky dieses an. Diese Node liest dafür die berechnete Pose aus dem Blackboard aus und setzt sie als Ziel. Da sie eine einfache Fahrplattform Bewegung repräsentiert, kann sie von der „Single Drive Movement“ Basis Node erben.

Öffnungsrichtung bestimmen

Jetzt steht der Roboter vor der Tür und damit im späteren Verlauf bekannt ist, in welche Richtung die Tür geöffnet werden muss, wird dies nun bestimmt. Hierbei gibt es nur zwei mögliche Richtungen, die bei normalen Türen mit Türrahmen leicht voneinander zu unterscheiden sind. Wenn eine Tür zu ziehen ist, ist der Rahmen nicht zu erkennen. Wenn sie hingegen mit Drücken

geöffnet werden muss, sind die Kanten des Rahmens klar durch den Laserscan erkennbar (siehe Abbildung 15).

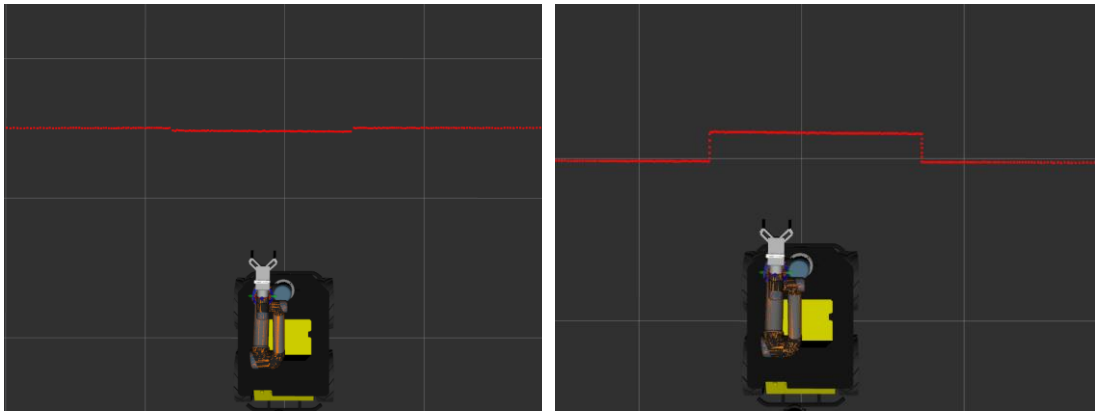

Abbildung 15: Scan geschlossene Tür (links: Tür ziehen, rechts: Tür drücken)

Aufgrund des Sichtradius und der Genauigkeit der Messwerte werden die Daten des Lidar-Scans genutzt. Zu Beginn werden die Distanzen des Laserscans in kartesische Koordinaten umgerechnet, wie es auch bei der Berechnung der Tür Ausrichtung gemacht wurde (siehe Formel (2) und (2)(3)). Um nicht den gesamten Laserscan betrachten zu müssen und um mögliche Irritationen ausschließen zu können, werden im Anschluss nur die 200 mittleren Punkte des Scans genutzt, um mit dem Algorithmus fortzufahren. Die Mitte beschreibt in diesem Kontext den Punkt, der den niedrigsten absoluten y-Wert besitzt, also möglichst gerade vor dem Husky liegt. Dadurch, dass der Husky sich bereits vor der Tür positioniert hat, ist garantiert, dass die gewählten Punkte den Bereich der Tür abdecken.

Das Ziel des Algorithmus ist es, einen Türrahmen in den extrahierten Punkten zu finden. Wenn einer gefunden wird, wird angenommen, dass die Tür durch Drücken zu öffnen ist. Schlägt der Algorithmus fehl, wird davon ausgegangen, dass die Tür zu ziehen ist.

Um den potenziellen Türrahmen in den Punkten zu finden, werden über vier Punkte gemittelt Richtungsvektoren berechnet und mit Hilfe dieser der Winkel zwischen ihnen und der Husky Ausrichtung (1,0) berechnet (siehe Formel (1)).

Die Indizes aller Punkte, die einen Richtungsvektor bilden, der einen größeren Winkelwert als 50 Grad hat, werden abgespeichert, weil sie theoretisch zu dem Türrahmen gehören könnten.

Hierbei wird direkt überprüft, ob der y-Wert des Punktes positiv oder negativ ist, was Auskunft darüber gibt, ob er zur rechten oder linken Seite des Rahmens gehört.

Nachdem alle Vektoren und Winkel berechnet wurden, gibt es zwei Listen mit Indizes von Punkten die potenziell zu dem Türrahmen gehören, aufgeteilt in links und rechts. Falls ein Rahmen erkennbar ist, sollten die Listen beide mindestens drei Einträge haben, sonst gilt die Detektion als fehlgeschlagen und die Tür wäre durch Ziehen zu öffnen. Wenn jedoch in beiden Listen mehr als drei Einträge zu finden sind, wird zusätzlich noch die Distanz zwischen dem Punkt, der am weitesten links ist unter den Rechten und dem Punkt der am weitesten rechts ist unter den Linken berechnet. Wenn diese Distanz zwischen 0.6 und 1.0 Metern liegt, gilt auch dieser Test als bestanden. Der letzte Test besteht darin zu überprüfen, wie viele aufeinander folgende Punkte es in die Listen geschafft haben, die direkt an die Engsten beiden Anknüpfen. Dadurch kann ausgeschlossen werden, dass es nicht zufällig zwei kleine Kanten im richtigen Abstand waren. Der Test gilt als bestanden, wenn beide Seiten mindestens drei aufeinander folgende Punkte aufweisen können. Wenn das der Fall ist, wurde ein Türrahmen gefunden. In allen anderen Fällen wurde kein Türrahmen gefunden.

Klinke detektieren und Waypoints berechnen

Nachdem der Roboter die Öffnungsrichtung der Tür bestimmt hat, wird die Türklinke identifiziert. Dafür wird der Laserscan der Kamera am Arm ausgewertet und daraus die Klinke extrahiert. Im Anschluss werden die Koordinaten berechnet, die abgefahren werden müssen, um die Tür zu entriegeln. Das erwartete Ergebnis ist in Abbildung 16 zu sehen. Man kann die Pointcloud der Realsense, in Kombination mit den Detektionen der ersten Phase erkennen. Dazu gehört das Koordinatensystem der Tür (unten mittig), die Punkte, die die Klinke beschreiben (grüner Punkt: Rotationsachse, blauer Punkt: Greifpunkt) und die Entriegelungskoordinaten (kleine rote Punkte).

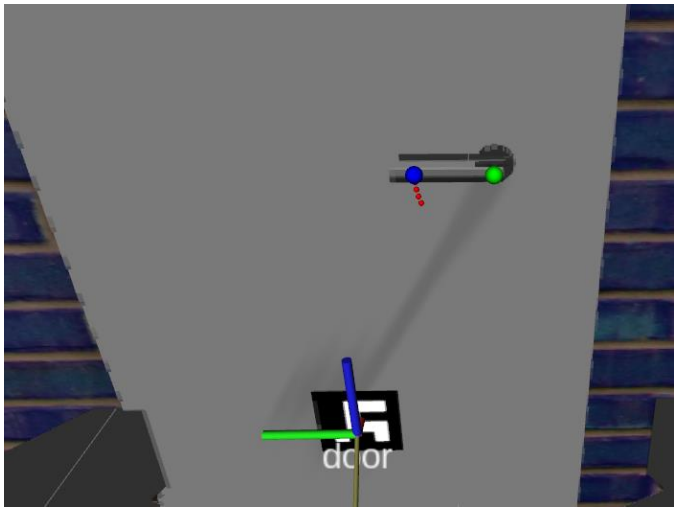

Abbildung 16: Anfahrt und Detektion Phase Ergebnis

Aufgrund immer ähnlichen Ausgangsposition vor der Tür, bleibt der Input, auf dem der Algorithmus zur Klinkendetektion basiert, relativ ähnlich (siehe Abbildung 17).

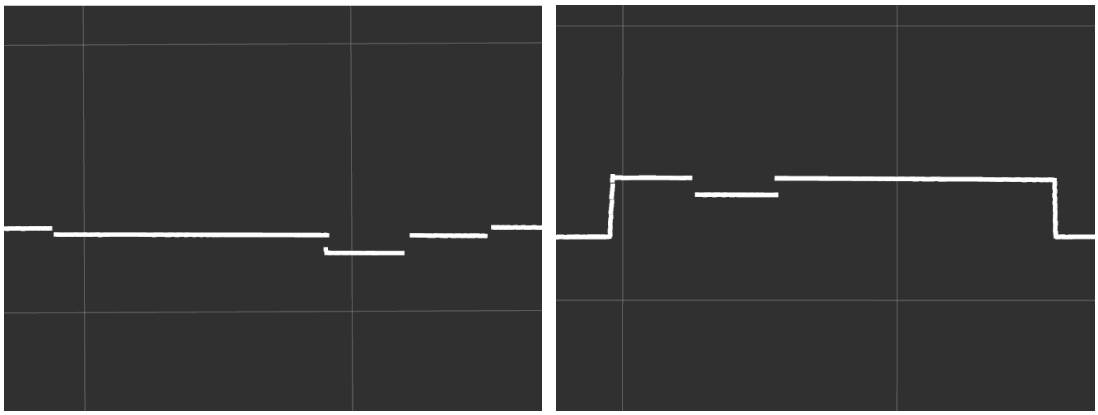

Abbildung 17: Klinken Detektion Laserscan (links: ziehen, rechts: drücken)

Der Algorithmus zum Extrahieren der Klinke ist in drei Abschnitte aufgeteilt: Kanten im Scan detektieren, das Mappen der Türmitte in den Scan und das Detektieren von möglichen Klinken.

Die Kanten Detektion beginnt mit einem Median Filter, der dafür sorgt, dass die Kanten möglichst glatt sind und nicht aus mehreren Punkten bestehen. Dafür wird die „medfilt“ Methode von „scipy“ mit einer Größe von fünf genutzt.

```
filtered = np.array(medfilt(distances, 5))
```

Listing 6: Scipy Median Filter Methoden Syntax

Danach wird der Laserscan durchlaufen und die einzelnen Distanzen mit ihren Nachbarn verglichen. Wenn die Differenz zwischen ihnen einen Grenzwert von 0.01 Metern überschreitet, wird das als Kante anerkannt und der Index abgespeichert. Trotz des Median Filters wird im Anschluss noch nach Kanten gesucht, die mehr als einen Punkt breit sind. Überprüft wird dies, indem die Indizes der Kanten miteinander verglichen werden und wenn sie direkt aufeinander folgen, wie zum Beispiel 134 und 135, werden sie zu einer Kante zusammengefasst.

Zum Mappen der Türmitte in den Scan, werden die Distanzen wieder in kartesische Koordinaten umgerechnet, wie es bereits bekannt ist. Nun wird die Position der Tür, die als Koordinatensystem verfügbar ist, mit den einzelnen Punkten verglichen. Da es nur um die Einordnung in der y-Achse geht (links, rechts), werden auch nur die y-Werte betrachtet. Der Index des Punktes im Laserscan, der der Türmitte am nächsten ist, wird als Mitte angesehen.

Nachdem die Mitte der Tür und alle Kanten im Scan erkannt wurden, können die möglichen Türklinken extrahiert werden. Dafür werden die Distanzen zwischen nebeneinander liegenden Kanten untersucht. Damit zwei Kanten als eine Türklinke angesehen werden, müssen sie zwischen 0.1 und 0.15 Metern auseinander liegen. Außerdem wird erwartet, dass eine der beiden Kanten der potenziellen Türklinke die erste Kante in eine Richtung (links oder rechts) von der Mitte ist. Das ist auf die generell glatte Oberfläche von Türen zurückzuführen, denn bis auf die Klinke, sollten sich zwischen Mitte und der Klinke im Normalfall keine weiteren Kanten dieser Größe befinden. Wenn nach diesen Kriterien eine Klinke feststeht, wird zusätzlich noch entschieden, ob sie vom Typ links oder Typ rechts ist. Dies ist später wichtig, um die Öffnungskoordinaten für die Tür und die Klinke zu berechnen.

Nachdem sichergestellt wurde, wo sich die Türklinke befindet, ist der letzte Schritt die Informationen in einige beschreibenden Punkten zu komprimieren. Zum Positionieren der Rotationsachse wurde ein Parameter festgelegt, der beschreibt wie viele Indizes von der äußeren Kante der Detektion sie sich befinden soll (für alle Tests: zwei). Ähnlich ist es auch für den Griffpunkt, also den Punkt, den der Gripper greifen soll, um die Tür zu entriegeln. Hierfür

wurde ebenfalls ein Parameter festgelegt, der eine prozentuale Verschiebung nach Innen in Abhängigkeit von der äußeren Kante und Klinkenlänge beschreibt (für alle Tests: 80%).

Zum Abschluss werden mit Hilfe dieser beiden Punkte die Entriegelungskoordinaten berechnet. Dafür wird der Griff Punkt (x_1, y_1) um die Rotationsachse (x_2, y_2) in vier Schritten (5,10,15,20) bis zu einem maximalen Winkel (α) von 20 Grad rotiert.

$$x = x_2 + \cos(\alpha) * (x_1 - x_2) - \sin(\alpha) * (y_1 - y_2) \quad (6)$$

$$y = y_2 + \sin(\alpha) * (x_1 - x_2) + \cos(\alpha) * (y_1 - y_2) \quad (7)$$

Die resultierenden Punkte (x, y) für die unterschiedlichen Winkel sind die Entriegelungskoordinaten. Da zur Steuerung des Arms Posen benötigt werden, wird der Rotationsteil mit der momentanen Ausrichtung des Endeffektors befüllt. Die funktioniert, da der Husky vor der Tür positioniert wurde.

Roboter neu positionieren

Als letzten Schritt in der Sequenz wird der Roboter noch einmal repositioniert, wie es beispielhaft in Abbildung 18 zu sehen ist. Diese Positionsänderung ist notwendig, weil es sonst zu Problemen mit dem Aktionsradius des Arms kommt.

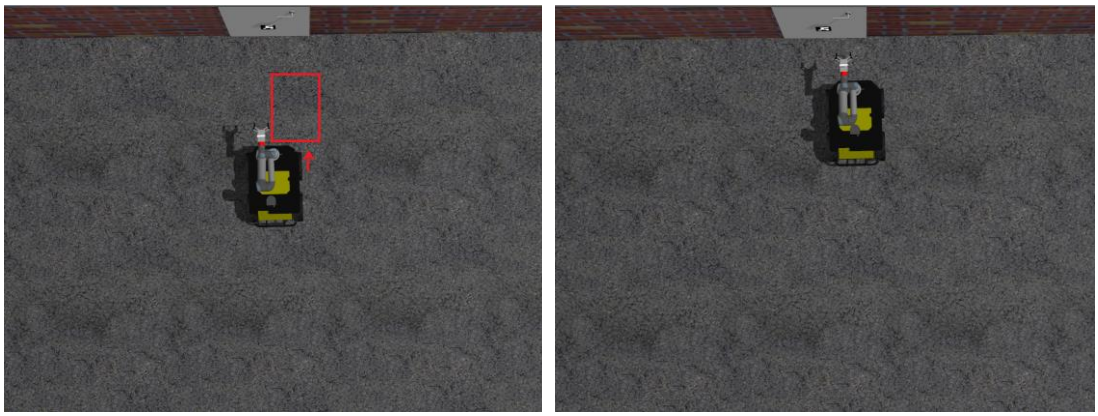

Abbildung 18: Anfahrtsprozess (Teil 2)

Diese Node ist wie die „Tür Position anfahren“ eine Fortbewegungs-Node und nutzt somit die „Single Drive Movement“ Node als Basis. Neben dem Setzen des Ziels, wird es hierbei

zusätzlich vorher noch berechnet. Die Auslagerung dieser Berechnung in eine andere Node ist aufgrund ihrer geringen Komplexität nicht notwendig.

In Abhängigkeit von der Öffnungsrichtung, die vorher bestimmt wurde, wird die Position vor der Tür gewählt.

Wurde kein Türrahmen detektiert ist das Ziel, eine Kombination aus der Türausrichtung und der Position der Türklinken-Rotationsachse. Mit der Position der Rotationsachse als Ausgangspunkt und der Türausrichtung als Richtung wird in einem Abstand von einem Meter das Ziel bestimmt.

Wurde ein Türrahmen entdeckt, wird die Türausrichtung in Kombination mit der Türmitte und einer Distanz von 0.7 Meter zur Berechnung der neuen Position verwendet. Das hat zur Folge, dass der Husky näher und mittig vor der Tür steht.

Die Berechnung hierfür erfolgt wie im Schritt „Tür Ausrichtung berechnen“ mit den Formeln (4) und (5).

Nun ist die erste Phase abgeschlossen und die Entriegelung der Tür kann beginnen.

5.3.4 Subbaum: Entriegelung

Der folgende Subbaum befasst sich mit der Entrieglung der Tür, wie sie im Öffnungsalgorithmus dargestellt wird.

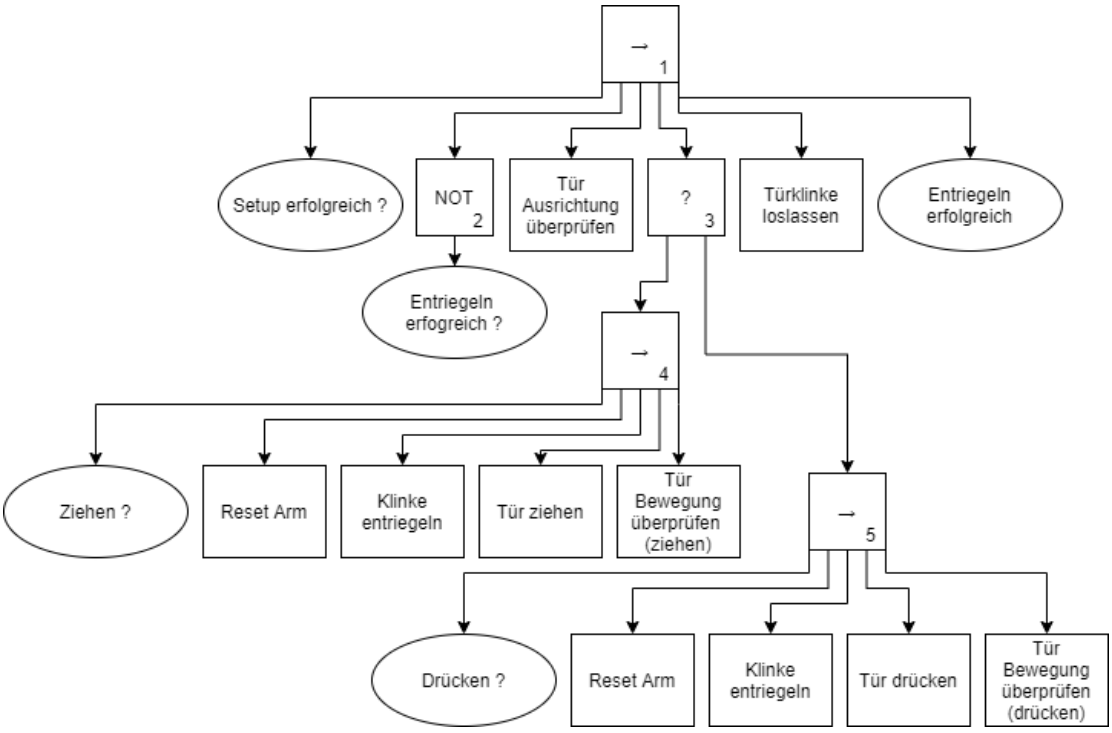

Abbildung 19: Subbaum Entriegelung

Um sicherzustellen, dass die Voraussetzungen für die Ausführung dieses Subbaums vorliegen, werden diese durch die ersten beiden Nodes („Setup erfolgreich?“ und „Entriegeln erfolgreich?“) in der Sequenz überprüft. Dabei wird zum einen geprüft, ob in dem Verlauf des Prozesses zu einem vorherigen Zeitpunkt die Anfahrt und Detektion Phase erfolgreich beendet wurde. Zum anderen, ob dieser Subbaum noch nicht erfolgreich beendet wurde. Wenn diese Bedingungen zutreffen, kann davon ausgegangen werden, dass der Husky die Position vor der Türklinke eingenommen hat und alle notwendigen Daten berechnet wurden.

Nach der initialen Überprüfung der Tür Ausrichtung wird je nach Status der Checks „Ziehen?“ und „Drücken?“ entschieden, ob versucht werden soll die Tür durch Drücken oder durch Ziehen zu öffnen. Beide Abläufe sind grundlegend gleich, bis auf die Richtung, in welche sich der

Arm nach dem Entriegeln der Klinke bewegt und dem Vorzeichen des Winkels bei der Überprüfung des Türblattwinkels.

Wenn einer der beiden Entriegelungsprozesse erfolgreich war, wird die Klinke losgelassen und der Erfolg des Subbaums im Blackboard festgehalten.

Tür Ausrichtung überprüfen & Tür Bewegung überprüfen

Damit überprüft werden kann, ob die Entriegelung in eine Richtung erfolgreich war oder nicht, wird zu Beginn des Prozesses eine Referenzmessung der Tür Ausrichtung gemacht. Die Berechnung dahinter funktioniert wie die Berechnung in der Node „Tür Ausrichtung berechnen". Es wird mit Punkten um die vorher berechnete Türmitte herum eine Gerade aufgestellt und der Winkel zwischen dieser und dem Husky ist die momentane Ausrichtung der Tür. Als Daten werden hierfür der Laserscan des Lidars und die Position des Tür Koordinatensystems genutzt.

Die beiden Aktionen „Tür Ausrichtung überprüfen" und „Tür Bewegung überprüfen" wurden in einer Node zusammengefasst, weil sie beide die gleiche Berechnung durchführen. In dem Fall der Aktion „Tür Ausrichtung überprüfen" wird das Ergebnis im Blackboard als Referenz gespeichert. In dem Fall von „Tür Bewegung überprüfen" wird nach der Berechnung das Ergebnis mit dem Referenzwert verglichen. Zwischen Referenz- und Messwert wird eine Mindestdifferenz von plus/minus fünf Grad erwartet. Je nach Bewegungsrichtung soll die Differenz positiv (ziehen) beziehungsweise negativ (drücken) sein.

Klinke entriegeln

Nachdem alle Vorbereitungen getroffen wurden, kann der Ablauf zur Klinken-Entriegelung beginnen. Abbildung 20 zeigt diesen am Beispiel einer Tür, die durch Ziehen zu öffnen ist. Da diese Node nur für die Entriegelung der Klinke verantwortlich ist, sprich für alle Bewegungen bis das Türblatt theoretisch bewegt werden kann, unterscheidet sich der Ablauf nicht zwischen Türen die zu ziehen, zu drücken oder sogar verschlossen sind.

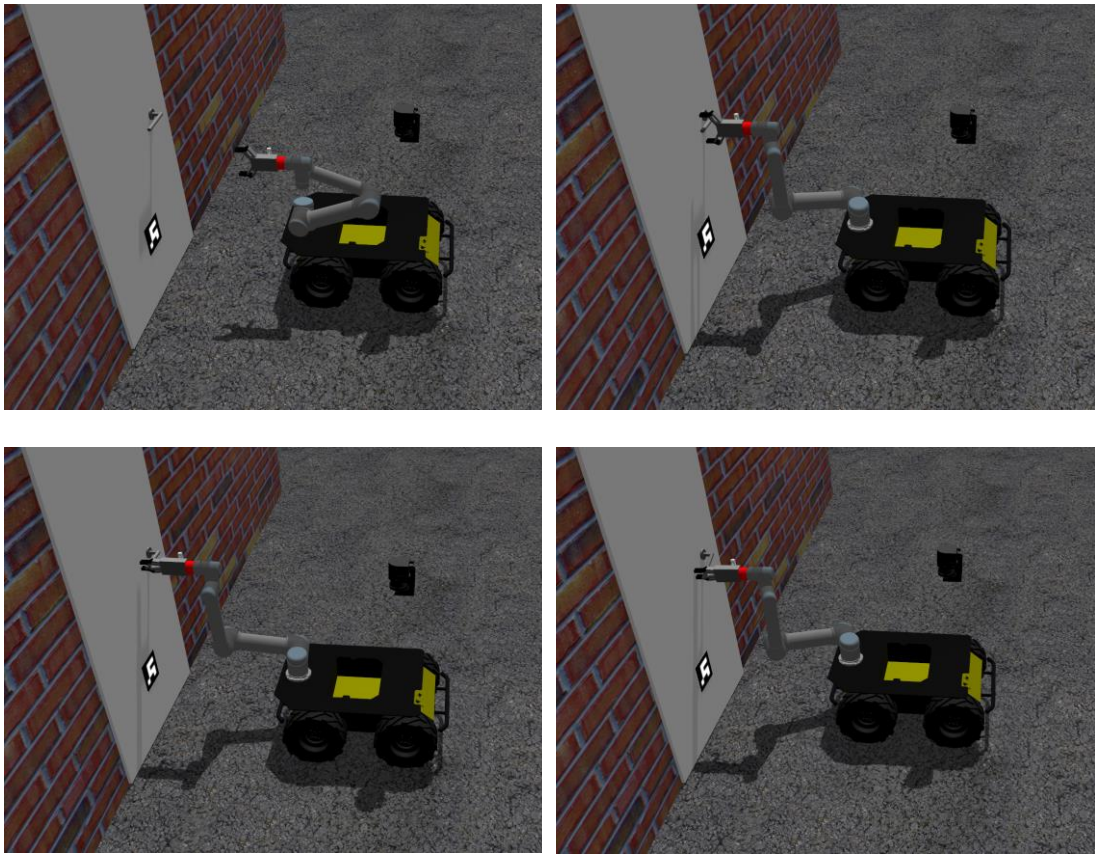

Abbildung 20: Klinken Entriegelungsprozess

Der Ablauf setzt sich aus fünf Schritten zusammen, wobei der erste darin besteht den Gripper zu öffnen. Die folgenden vier werden durch die einzelnen Bilder in Abbildung 20 repräsentiert und bestehen darin den Gripper in die Vertikale zu bringen (Bild oben links), den Griff Punkt anzufahren (Bild oben rechts), den Gripper zu schließen (Bild unten links) und schlussendlich die Entriegelungskoordinaten abzufahren (Bild unten rechts).

In diesem Ablauf werden alle Bewegungsarten des Arms genutzt. Das Drehen des Grippers wird umgesetzt durch das Festlegen eines Winkels für das „ur_arm_wrist_3_joint“ Gelenk, an dem der Gripper befestigt ist. Das Anfahren des Grip Punktes wird durch das Setzen einer Ziel Pose für den Gripper umgesetzt, sodass „move_it“ mit Hilfe von Reverse Kinematics selbst die Winkel für die Gelenke errechnen muss. Die letzte Armbewegung wird über das Festlegen eines kartesischen Pfades erreicht. Dieser besteht aus mehreren nah beieinander liegenden Posen, welche in diesem Fall die Entriegelungskoordinaten sind.

Der Ablauf selbst ist über ein Switch-Case Pattern geregelt, in dem die einzelnen Schritte nacheinander durchgeführt werden. Wenn einer der Schritte fehlschlägt, wird der gesamte Ablauf abgebrochen.

Tür ziehen & Tür drücken

Um die Tür aufzuziehen oder zu drücken ist eine einfache gerade Bewegung ausreichend, da die Verformung der Geraden bei kleinen Öffnungswinkeln von fünf bis zehn Grad und einem großen Radius nicht signifikant ist.

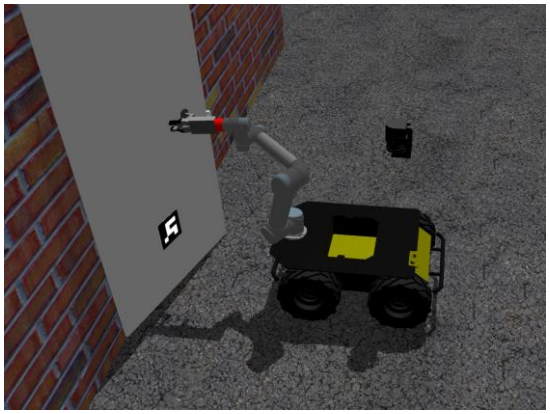

Abbildung 21: Tür entriegeln (ziehen)

Da die Nodes „Tür drücken“ und „Tür ziehen“ nur eine Arm Bewegung durchführen, erben sie von der „Single Arm Movement“ Base Node. Infolgedessen muss in der Initialisierung lediglich die Ziel Pose gesetzt werden.

Anhand der DIN-Norm für Türmaße ergibt sich eine Breite zwischen 0.61 und 0.985 Meter [22]. Das Ziel ist eine Öffnung der Tür um fünf bis zehn Grad. Dieser Bereich wurde gewählt, damit die Tür sich durch ihre Ausrichtung anschließend klar von der Wand abhebt, aber nicht zu weit geöffnet ist, dass sie das Sichtfeld des Husky verlässt oder einschränkt. Aus diesen Werten resultiert die Distanz, um die die Tür minimal beziehungsweise maximal aufgezogen werden muss.

$$\max = \sin(10) * 0.985m = 0.171m$$

$$\min = \sin(5) * 0.61m = 0.053m$$

Anhand dieser Ergebnisse wurde sich für eine Öffnungsdistanz von 0.1 Meter entschieden. Das resultiert darin, dass die größten Türen 5.83 Grad geöffnet werden und die kleinsten Türen bis zu 9.43 Grad. Somit liegen beide Grenzwerte im Toleranzbereich.

Daraus resultiert die Ziel Pose, die in Relation zu der Endpose aus der „Klinke entriegeln“ Node nur um 0.1 Meter verschoben ist. Im Fall der „Tür ziehen“ Node wird die Ziel Pose in Richtung des Husky verschoben, im Fall der „Tür drücken“ Node in die entgegengesetzte Richtung.

5.3.5 Subbaum: Das Öffnen der Tür

Der letzte Subbaum des Prozesses befasst sich mit der Aufgabe die Tür nach der Entriegelung komplett zu öffnen, sodass diese durchfahren werden kann und setzt somit den Teil „Tür passieren“ des Öffnungsalgorithmus um.

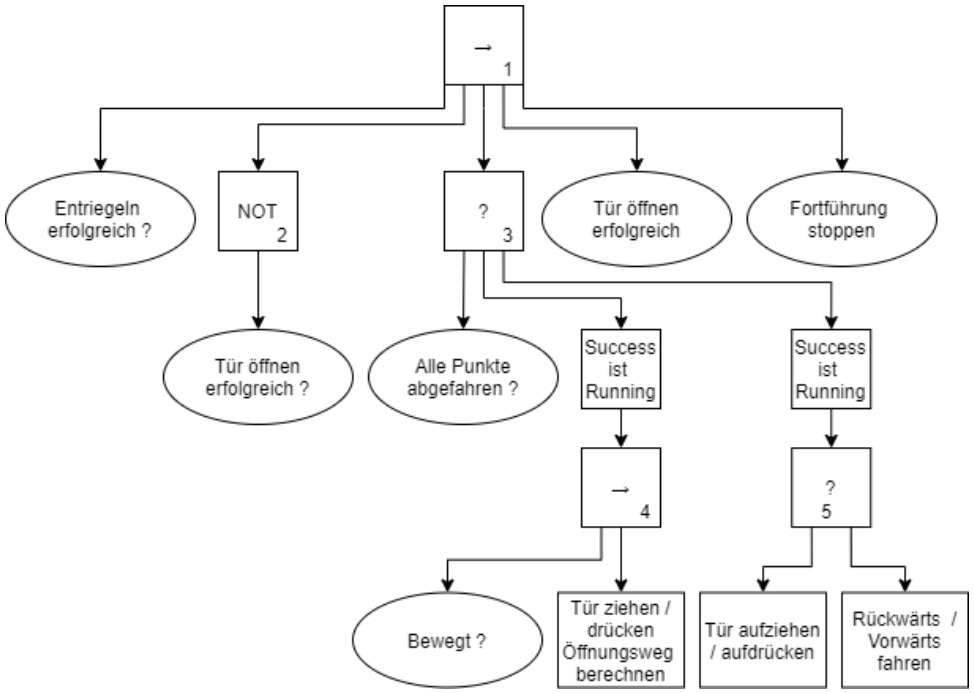

Abbildung 22: Subbaum Tür öffnen

Die oberste Ebene ähnelt der des Entriegelungs-Subbaums. Hierbei wird geprüft, ob die Konditionen zum Ausführen des Baums zutreffen. Außerdem wird auch in diesem Subbaum festgehalten, ob die Ausführung bis zum Ende erfolgreich war. Wenn das der Fall ist, wird

zusätzlich noch gespeichert, dass der Ablauf nicht weiter fortgeführt werden soll, da die Aufgabe erledigt ist.

Die Fallback-Node drei umfasst den geregelten Ablauf zum Öffnen der Tür. Zu Beginn wird überprüft, ob bereits alle berechneten Koordinaten abgefahren wurden. Falls ja gilt der Prozess als erfolgreich beendet und die Tür kann durchfahren werden.

Falls nicht wird im Anschluss überprüft, ob der Husky sich bewegt hat. Dieser Check wird in der ersten Iteration immer positiv sein, damit initial die Öffnungskoordinaten berechnet werden. Nach der Berechnung wird diese Zustandsvariable auf falsch gesetzt. Wenn der Husky sich nicht bewegt hat und die Öffnungskoordinaten bereits berechnet wurden, wird die Tür geöffnet.

Dafür wird vor jedem neuen Punkt, der angefahren werden soll, überprüft, ob er zu weit weg oder zu nah dran ist, um ihn zu erreichen. Falls dieser Test, das Planen der Bewegung oder die Bewegung selbst fehlschlägt, wird versucht das Problem durch eine Positionsänderung der Fahrplattform zu lösen. Hierbei wird die eben erwähnte Zustandsvariable auf wahr gesetzt und im nächsten Tick neue Öffnungskoordinaten berechnet. Durch dieses überwachte iterative Verfahren soll die Tür nach und nach geöffnet werden.

Die Nodes „Success ist Running" sind Decorator Nodes, die dafür sorgen, dass der Ablauf trotz Erfolg der einzelnen Schritte solange in dem Regelungskreis gefangen ist, bis er entweder fehlschlägt oder alle Punkte erfolgreich abgefahren wurden.

Tür Öffnungsweg berechnen

Damit die Tür komplett geöffnet werden kann, werden wie auch bei der Entriegelung, Posen berechnet, die der Gripper abfahren soll. Anhand der Bilder in Abbildung 23 kann man die Ergebnisse dieser Berechnungen erkennen. Der grüne Punkt repräsentiert die Rotationsachse der Tür, der blaue Punkt die Tür Kante und bei den kleinen roten Punkten handelt es sich um die Koordinaten Teile der errechneten Posen.

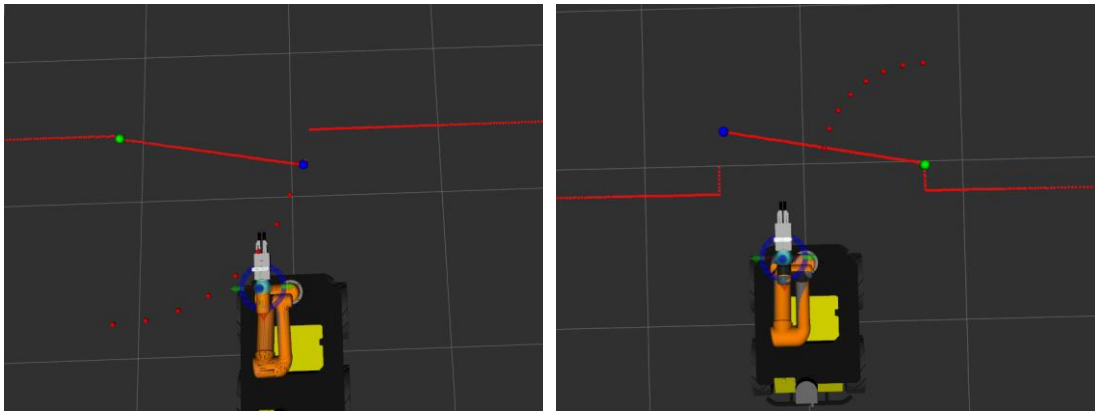

Abbildung 23: Öffnungskoordinaten berechnen (links: ziehen, rechts: drücken)

Die Berechnung dieser Features ist unabhängig von vorherigen Ergebnissen und es werden keine Hilfsmittel, wie zum Beispiel die ArUco-Marker, benötigt. Da die Tür in dieser Phase entriegelt sein sollte, müsste sie sich wie in Abbildung 23 zu sehen, aufgrund ihrer Ausrichtung von dem Rest der Wand unterscheiden.

Als Daten für die Berechnung wird erneut der Laserscan des Lidars genutzt, weil dieser einen großen Sichtbereich besitzt und relativ weit hinten auf dem Husky sitzt, wodurch auch beim nach vorne Fahren noch lange die Tür klar erkannt werden kann. Die Vorbereitung der Daten erfolgt wie bei „Öffnungsrichtung bestimmen“ aus der ersten Phase, sodass nach der Vorverarbeitung die 200 mittleren Punkte des Laserscans weiterverarbeitet werden. Zuerst wird erklärt wie pro Öffnungsrichtung die Tür Kante und Rotationsachse erkannt werden und danach wie die Öffnungskoordinaten errechnet werden.

Wurde festgestellt, dass die Tür aufgedrückt werden muss, wird wie in „Öffnungsrichtung bestimmen“ vorgegangen und nach dem Türrahmen gesucht. Wird dieser gefunden, werden die Punkte, die am weitesten vom Husky entfernt sind, aber trotzdem als Teil des Türrahmens klassifiziert wurden, als Rotationsachse und Tür Kante klassifiziert. Der Punkt, der die Tür Kante repräsentiert, wird trotz der Lücke zwischen dem echten Türrahmen und sich selbst durch den Algorithmus als Türrahmen klassifiziert. Das liegt daran, dass dieser sich nur mit den Winkeln der Vektoren zwischen den Punkten und der Ausrichtung des Husky befasst. Das hat den Vorteil, dass die Detektion wie eben beschrieben funktioniert.

Unterscheiden kann man die Rotationsachse und die Türk Kante durch die vorher detektierte Position der Türklinke. Befindet sich die Türklinke links ist der rechte Punkt die Rotationsachse und vice versa. Beim Drücken der Tür wird mit dem Gripper gegen die Tür gedrückt (siehe Abbildung 24).

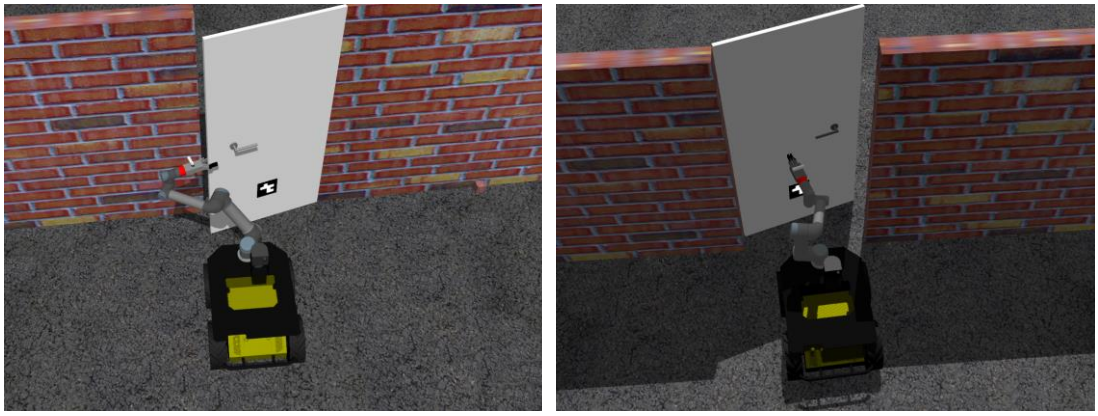

Abbildung 24: Tür öffnen Arm Position (links: ziehen, rechts: drücken)

Die Berechnung der Koordinaten zum Aufziehen der Tür kann nicht auf dem Türrahmen basieren. Hierfür wird versucht die Lücke zwischen Wand und Tür Kante, sowie den Knick zwischen Türblatt und Türrahmen zu nutzen. Die Tür Kante wird gefunden, indem die Punkte nach Lücken mit einer Mindestgröße von 0.05 Metern durchsucht werden. Falls mehrere solcher Lücken erkannt werden ist es höchstwahrscheinlich, dass der Punkt, der am nächsten am Husky ist, die Tür Kante ist. Das ist auf die Repositionierung des Husky vor der Türklinke zurückzuführen. Nachdem die Tür Kante identifiziert wurde, werden die Winkel zwischen der Husky Ausrichtung und den über vier Punkte aufgespannten Vektoren berechnet. Diese Winkel werden im Anschluss mit ihren Nachbarn verglichen und damit versucht den Knick zwischen Türblatt und Türrahmen zu finden. Hierfür ist ein Threshold von fünf Grad gewählt worden, um eine Differenz zwischen zwei Winkeln als potenziellen Knick durchgehen zu lassen. Zum Aufziehen der Tür umschließt der Gripper die Tür mit geöffneten Fingern, damit sie geführt bewegt werden kann. Dies ist beispielhaft in Abbildung 24 zu sehen.

In beiden Szenarien ist die Anzahl der Öffnungskoordinaten abhängig von dem zu öffnenden Restwinkel. Das Ziel ist immer eine Öffnung um 90 Grad, wovon der momentane

Öffnungswinkel der Tür abgezogen, durch zehn geteilt und aufgerundet wird. Das Ergebnis repräsentiert die Anzahl an Öffnungskoordinaten.

Berechnet werden diese Öffnungskoordinaten, indem ein neuer Punkt gefunden wird, der dann um die Rotationsachse rotiert wird. Dieser Punkt liegt auf der Geraden zwischen Rotationsachse und Tür Kante. Zum Aufdrücken soll dieser mittig liegen, sodass er definiert wird durch die Rotationsachse plus den halben Richtungsvektor zwischen Rotationsachse und Tür Kante. Beim Aufziehen könnte theoretisch auch die Tür Kante rotiert werden, um jedoch sicherzustellen, dass der Gripper richtig in die Tür greift wird der Punkt nach innen versetzt. Somit definiert sich dieser Punkt durch die Rotationsachse plus 0.98-mal den Richtungsvektor.

Da Posen, die zum Steuern des Arms genutzt werden, nicht nur aus einem Koordinaten Teil, sondern auch aus einer Rotationsangabe bestehen, muss diese noch zusätzlich berechnet werden. Beim Aufdrücken der Tür ist die Zielrotation immer senkrecht zu dem Richtungsvektor zwischen Rotationsachse und der nächsten Zielkoordinate. Beim Aufziehen ist die Zielrotation parallel zu diesem Vektor, weil der Gripper in das Türblatt greifen soll. Daraus resultiert immer ein Winkel α, der um die z-Achse rotiert. Da die Rotationsangabe der Pose in Quaternion Format angegeben werden muss, ist eine Umrechnung von Nöten. Diese wurde mit Hilfe von der „tf“ Bibliothek durchgeführt, die Methoden wie „quaternion_from_euler“ und „euler_from_quaternion“ bereitstellt.

Nach Abschluss der Berechnung werden alle wichtigen Daten im Blackboard gespeichert. Dazu gehören die errechneten Posen und die Position der Tür Kante. Außerdem werden die Zustandsvariable, ob der Husky sich bewegt hat, auf falsch und der Index, welcher Punkt als nächstes angefahren werden soll, auf null gesetzt.

Nun kann die Tür Öffnung beginnen beziehungsweise fortgesetzt werden.

Tür aufziehen / aufdrücken

Diese Node, die für die Steuerung des Arms bei der Öffnung verantwortlich ist, ist vom Typ „SingleArmMovement“ mit dem Bewegungstyp Pose. Die gesamte Bewegung zum Öffnen der Tür ist zwar keine einfache Bewegung zu einer einzelnen Pose, aber durch die Struktur des Baumes wird diese in mehrere einfache Bewegungen unterteilt.

Dafür wird bei jedem erneuten Aufruf der Node mit Hilfe eines Indexes eine neue Pose aus dem Blackboard geladen und als Ziel eingesetzt. Bevor das Ziel tatsächlich versendet wird, wird überprüft, ob es ungefähr dem Bewegungsradius des Arms vor dem Husky entspricht oder ob eine präventive Korrektur nötig ist. Dafür wird die Distanz zwischen Husky Mittelpunkt und der Ziel Pose berechnet. Diese Distanz sollte zwischen 0.7 und 1.4 Metern liegen.

Nach erfolgreichem Abschluss der Bewegung wird der Index erhöht und geprüft, ob dies die letzte Pose in der Liste war. War es die letzte Pose wird die Blackboard Variable die durch „Alle Punkte abgefahren?“ abgefragt wird auf wahr gesetzt und der Ablauf beendet.

Rückwärts / Vorwärts fahren

Damit die eben genannten Korrekturen durch die Fahrplattform möglich sind, setzt diese Node eine einfach Vorwärts- beziehungsweise Rückwärtsbewegung um. Hierbei wird nicht wie vorher ein Ziel an den „move_base“ Server gesendet, sondern die Möglichkeit des Sendens von ROS-Messages an den Motorkontroller genutzt. Hierfür wurde sich für eine Bewegungsgeschwindigkeit von 0.1 Meter pro Sekunde entschieden, die über eine Dauer von einer Sekunde plus die 0.5 Sekunden Timeout gefahren wird. Die Twist Message besteht dementsprechend aus (+-0.1, 0, 0) als linear Part und (0, 0, 0) als angular Part. Diese wird in einem Abstand von 0.01 Sekunden an den Motorkontroller gesendet bis die Sekunde abgelaufen ist.

Nach Abschluss der Bewegung wird die Zustandsvariable, die eine Positionsänderung des Husky repräsentiert, wieder auf wahr gesetzt.

5.4 Fehleranalyse und -handling

Es wurde sich für ein zentrales Fehlermanagement entschieden. Aus diesem Grund befindet sich dies in der obersten Ebene des Prozesses und wird durch die Nodes „Fehleranalyse“ und „Fortführung stoppen“ durchgeführt. In der folgenden Tabelle sind alle behandelten Fehlerquellen und deren Lösungsvorschläge zu sehen.

Phase	***Grund***	***Lösung***
Generell	Reset Arm schlägt fehl	Nochmal versuchen (2-mal)
Generell	Anfahrt einer Position schlägt fehl	Nochmal versuchen (2-mal)
Anfahrt und Detektion	Keine Marker gefunden	Abbruch
Anfahrt und Detektion	Keine Klinke gefunden	Kamera repositionieren (2-mal)
Entriegelung	Tür Position nicht erkannt	Stück zurückfahren (2-mal)
Entriegelung	Klinke entriegeln schlägt fehl	Nochmal versuchen (2-mal)
Entriegelung	Tür ziehen schlägt fehl	Ausgangsposition anfahren
Entriegelung	Tür drücken schlägt fehl	Ausgangsposition anfahren
Entriegelung	Gripper von Klinke gerutscht	Nochmal versuchen (1-mal)
Tür öffnen	Öffnungskoordinaten Berechnung schlägt fehl	Stück zurückfahren (2-mal)
Tür öffnen	Tür aufziehen/drücken schlägt fehl	Nochmal versuchen (1-mal)

Tabelle 6: Fehlerquellen und Lösungen

Damit die Fehleranalyse-Node weiß, welcher Fehler aufgetreten ist, wird beim Aufruf von wichtigen Nodes im Blackboard gespeichert, dass diese gerade aufgerufen wurden. Zu diesen Nodes gehören zum Beispiel die „ArUco Marker detektieren“ Node oder die „Tür ziehen“

Node. Aufgrund der feingranularen Aufspaltung der Aufgaben ist es möglich, bei fast allen Nodes auf ein Problem zu schließen. Eine Ausnahme davon ist beispielsweise die „Klinke entriegeln“ Node aus der Entriegelung. Dieser Vorgang besteht aus mehreren Schritten, wobei auch diese im Blackboard festgehalten werden.

Aufgrund der gespeicherten Phasen erkennt die Fehleranalyse, weshalb der Ablauf gescheitert ist. Gelöst werden die Probleme, wie es in Tabelle 6 beschrieben wird. Alle Lösungen erfordern Wiederholungen von Nodes oder Teilbäumen, wofür die Abfragen von Statusvariablen im Blackboard von Relevanz sind (beispielsweise: Abb. 10, „Ziehen?“ und „Entriegeln erfolgreich?“). Mit Hilfe dieser Abfragen kann die Fehleranalyse den Kontrollfluss des Baumes beeinflussen und somit Teilbäume wiederholen oder überspringen. Für die Lösungen, die zusätzlich noch Beeinflussung der Husky Position oder des Arms benötigen, steuert die Fehleranalyse die Bewegungen selbst.

6 Vorstellung der Versuche und Ergebnisse

Um das System in der Simulation zu testen, wurden mehrere Versuche mit unterschiedlichen Startpositionen durchgeführt.

Aufgrund von Fehlschlägen der Abläufe und Problemen mit der Simulation, die später genauer erläutert werden, wird die Validierung nicht für den Ablauf als Ganzes, sondern einzeln für die wichtigen Teilprozesse (Anfahrt und Detektion, Entriegelung und Tür passieren) durchgeführt. Dafür wird das System in die spezifizierte Ausgangslage für den zu testenden Teilprozess gebracht und im Anschluss nur dieser ausgeführt. Wenn nach Abschluss des Tests das erwartete Endresultat vorliegt, gilt dieser Versuch als erfolgreich. Dadurch ist es möglich, trotz der Schwierigkeiten, den Ablauf und die Algorithmen zu testen. Außerdem wird ein gezielter Fokus auf die errechneten Koordinaten und Detektionen gelegt, um die Korrektheit des Ablaufs und der Algorithmen zu bestimmen, als auf den tatsächlichen Erfolg der daraus resultierenden Bewegungen. Um die Ergebnisse zu bewerten wird „rviz“ [23] genutzt. Rviz ermöglicht es ROS-Messages jeglicher Form zu visualisieren. Bilder aus vorherigen Kapiteln, die Detektionen des Systems zeigen, sind ebenfalls mit Hilfe von Rviz entstanden. Mit Markern, wie sie in vorherigen Bildern zu sehen sind, kann man visuell die Korrektheit der Berechnungen verifizieren.

Im Folgenden werden die Tests für die einzelnen Phasen erläutert und ausgewertet.

6.1 Anfahrt und Detektion Ergebnisse

In der ersten Phase wird erwartet, dass der Husky mit allen vier Szenarien (Tür ziehen/drücken, Klinke rechts/links) wie vorher beschrieben konstant umgehen kann. Zudem soll er aus möglichst vielen Anfahrtspositionen das Ziel erfüllen. Diese Positionen werden in Abbildung 25 durch die unterschiedlich gefärbten Zonen repräsentiert.

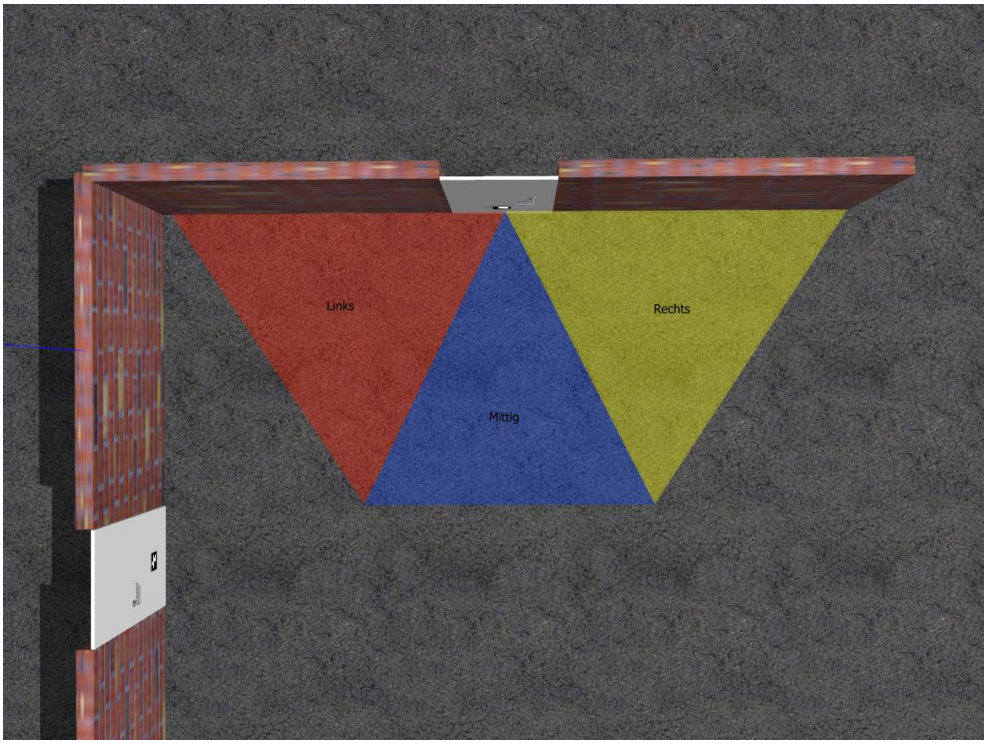

Abbildung 25: Anfahrtszonen Anfahrt und Detektion Test

Aus jeder Zone wurden zehn Testläufe pro Szenario durchgeführt. Bei jedem Testlauf wurde der Winkel und die Position des Husky innerhalb der Zone variiert. Die einzige Einschränkung hierbei war, dass der Marker gesehen werden musste. Die Ergebnisse dieser Versuche sind in den folgenden Tabellen zu sehen. Die Daten beschreiben, wie oft welche Phase bei den 120 Versuchen fehlgeschlagen ist und somit zum Fehlschlag des Ablaufs geführt hat

Startposition	***Erste Ausgangsposition erreicht***	***Öffnungsrichtung bestimmt***	***Türklinke detektiert***	***Total fehlgeschlagen***
Links	2	0	2	4 / 40
Mittig	1	0	3	4 / 40
Rechts	2	1	1	4 / 40
Total	5 / 120	1 / 120	6 / 120	12 / 120

Tabelle 7: Anfahrt und Detektion Ergebnisse

Tabelle 7 zeigt, dass die Algorithmen stabil funktionieren. Auch wenn die erste Ausgangsposition nicht immer senkrecht und mittig angefahren wurde, wurden meist alle weiteren Features richtig erkannt. Probleme beim Anfahren der ersten Position vor der Tür traten speziell bei sehr spitzen Anfahrtswinkeln auf. Diese ungenaue erste Anfahrtsposition wäre zudem noch durch die Repositionierung korrigiert worden.

Die Repositionierung des Husky je nach Öffnungsrichtung und Türgriff Position war jedoch selten erfolgreich. Die Endposition für das Aufdrücken der Tür wurde nie erreicht. Die Endposition bei den Zieh Szenarien wurde 14/108-mal erreicht. Die Berechnung der Zielkoordinate für die Endposition war jedoch 105/108-mal erfolgreich.

Zurückzuführen ist die geringe Erfolgsquote auf die Pfadplanung mit „move_base“. Diese lässt nicht zu, dass der Husky nah an Wänden navigiert. Die erste Ausgangsposition ist 1.5 Meter von der Wand entfernt, sodass „move_base“ frei navigieren kann. Die Ziele für die Repositionierung sind 1 Meter und 0.7 Meter von der Wand entfernt. Die daraus resultierende Pose wird zwar von der Pfadplanung nicht abgelehnt, jedoch entstehen Pläne, die nicht zielführend sind. Teilweise fährt der Husky hierbei im Kreis oder wackelt nur von links nach rechts. Das Ziel erreicht er dadurch nicht.

Ein weiterer Aspekt ist, dass die Panzersteuerung des Husky für diese Art von Bewegung nicht förderlich ist. Drehungen sind hierbei immer mit einem Rutschen der Räder verbunden, was zum Einen schwer zu steuern ist und zum Anderen zu Ungenauigkeiten in der Positionsbestimmung führen kann.

6.2 Entriegelung Ergebnisse

Die Entriegelung der Tür wird getestet, indem der Husky für alle vier Szenarien (Tür ziehen/drücken, Klinke rechts/links) in die richtige Ausgangsposition gestellt wird. Beim Ziehen wäre das mittig vor der Türklinke mit einer Entfernung von circa einem Meter. Beim Drücken mittig, circa 0.7 Meter von der Tür entfernt. Diese Positionen werden in den Tests immer ein wenig variiert, um einem richtigen Ablauf näher zu kommen. Jedes Szenario wird zehnmal getestet. Die Grenzen für die Variationen der Ausgangsposition wurden den Beobachtungen

der Endposition aus 6.1 angepasst (maximal circa zehn Grad in eine Richtung rotiert und ungefähr in der Nähe der definierten Position).

Wie bei den Ergebnissen der ersten Phase zeigt die folgende Tabelle, wie oft welcher Schritt fehlgeschlagen ist.

Tür Typ	***Klinke gegriffen***	***Klinke entriegelt***	***Tür entriegelt***	***Total fehlgeschlagen***
Ziehen / Links	0	2	3	5 / 10
Ziehen / Rechts	1	3	3	7 / 10
Drücken / Links	0	3	4	7 / 10
Drücken / Rechts	1	1	4	6 / 10
Total	2 / 40	9 / 40	14 / 40	25 / 40

Tabelle 8: Entriegelung Ergebnisse

Die initiale Berechnung der Tür Ausrichtung war in 38/40 Fällen korrekt. Ähnlich lief es bei der zweiten Berechnung nach dem Entriegeln der Tür, die 14/15-mal erfolgreich war. Somit war auch in dieser Phase die Detektion robust.

Die Versuche sind alle mit richtig berechneten Daten der Anfahrts- und Detektions-Phase durchgeführt worden. Wie man jedoch sehen kann, schlug trotz dessen ein Großteil der Versuche fehl.

Die Hauptprobleme liegen hier bei der Simulation an sich, sowie der Arm Steuerung durch „move_it“. Ersteres hat zur Folge, dass die Arm Gelenke und Finger nicht so robust und stabil sind wie in der Realität, was zu schwammigen Bewegungen führt, sobald Kraft auf sie ausgewirkt wird. Das ist der Fall beim Greifen und Drücken der Klinke.

Neben diesem Aspekt hat „move_it“ Probleme manche Bewegungen korrekt zu planen und auszuführen, wodurch Abläufe nur aufgrund dessen fehlschlagen. Nicht korrekte Pläne sind hierbei Pläne, die theoretisch auch das Ziel erreichen aber auf dem Weg dahin große Umwege

in Kauf nehmen und gegebenenfalls Kollisionen verursachen. Des Weiteren werden Bewegung teilweise als fehlerhaft angesehen, obwohl sie in der Simulation richtig ausgeführt werden.

Diese Probleme sind größtenteils simulationsspezifisch und auch die fehlerhafte Steuerung des Arms durch „move_it“ kann durch Input der Simulation und die „weichen“ Gelenke des Arms hervorgerufen werden.

6.3 Tür öffnen Ergebnisse

Auch in dieser Testreihe wird der Husky an den definierten Ausgangspunkten positioniert. Diese entsprechen denen der Entriegelungsphase, weil der Husky sich vom Anfang der Entriegelung bis zum Start dieser Phase nicht bewegt. Wie in den anderen Tests wird auch hier jedes Szenario zehnmal getestet und die Ausgangsposition wird wie in 6.2 beschrieben variiert.

Tür Typ	***Koordinaten korrekt berechnet***	***Tür geöffnet***	***Total fehlgeschlagen***
Ziehen / Links	1	9	10 / 10
Ziehen / Rechts	1	9	10 / 10
Drücken / Links	1	5	6 / 10
Drücken / Rechts	0	7	7 / 10
Total	3 / 40	30 / 40	33 / 40

Tabelle 9: Das Öffnen der Tür Ergebnisse

Wie man in Tabelle 9 sehen kann war die Berechnung der Öffnungsposen mit einer Erfolgsquote von über 90% erfolgreich. Die fehlerhaften Versuche wurden zudem durch eine schlechte Ausgangsposition provoziert. Alle Detektionen aus einfachen Positionen ohne große Rotation und Verschiebung heraus, waren erfolgreich.

Der Öffnungsablauf, wie er in 5.3.5 beschrieben wurde, hat nicht den Erwartungen entsprochen. Beim Zieh Szenario gab es zwei Probleme. Zum einen wurden manche Bewegungen ohne triftigen Grund abgebrochen, was im Testteil der Entriegelungs-Phase auch schon

thematisiert wurde. Zum anderen ist das Problem aufgetreten, dass die Tür sich aufgrund ihrer freien Beweglichkeit und geringen Reibung im Scharnier ohne Einfluss des Husky bewegt hat. Das hat zur Folge, dass die Punkte, die nach einer Repositionierung berechnet wurden, nicht mehr mit der Position der Tür übereinstimmen. Wenn die Bewegung nicht abgebrochen wurde, war der Ablauf bis zur Repositionierung erfolgreich, was bedeutet, dass das Konzept mit dem Gripper die Tür zu führen funktioniert.

Das Drück Szenario hatte ebenfalls Probleme mit der sehr lockeren Bewegung der Tür. Jedoch ist dies hierbei weniger relevant, weil der Gripper die Tür nur aufdrücken und nicht genau eine bestimmte Position treffen muss. Der Husky ist zwar in fast allen Versuchen durch die Tür gekommen, jedoch wurden nur die Versuche als erfolgreich gewertet, in denen er die Tür nur mit dem Gripper berührt hat. Letztendlich könnte er in diesem Szenario, nach der erfolgreichen Entriegelung, auch ohne den Gripper zu nutzen durch die Tür fahren.

7 Zusammenfassung und Ausblick

Wie die Ergebnisse der Versuche gezeigt haben, ist die Erkennung der Features in jeder Phase gelungen. Mit einer Erfolgsrate von circa 90 Prozent über alle drei Phasen hinweg, können die Algorithmen hinsichtlich Erkennung als robust eingestuft werden. Diese Erfolgsquote ist durch die vereinfachten Sensorwerte der Simulation im Vergleich zur Realität gesteigert. Die Ergebnisse zeigen, dass die entwickelten Algorithmen grundlegend eine Möglichkeit darstellen, die unterschiedlichen Aufgaben zu bewältigen.

Die Implementierung des Ablaufs in Form eines Behaviour Trees erwies sich in den Versuchen und der Entwicklung als durchaus sinnvoll. Es konnte während der Entwicklung leicht Änderungen am Verhalten durchgeführt werden, ohne die einzelnen Nodes abzuändern. Außerdem konnte das Verhalten während den Tests problemlos verstanden und nachvollzogen werden. Probleme setzen vor allem in der dritten Phase bei der Umsetzung des Regelkreises ein. Dies zeigt sich auch in den schlechten Ergebnissen dieser Phase. Das konstante Sammeln von Daten in den Baum einzubinden, erweist sich als schwierig, wie es beispielsweise in einem Regelkreis benötigt wird. Das Konzept, welches in Kapitel 5.3 diesbezüglich vorgestellt wurde, hat nur teilweise funktioniert.

Die Husky Panzersteuerung in Kombination mit „move_base“ als steuernde Instanz hat sich zudem negativ auf die Erfolgsquote der Tests, vor allem in Phase eins und drei, ausgewirkt. Die Panzersteuerung funktioniert auf weiten Flächen, wo Präzision nicht von hoher Relevanz ist, gut. Ist aber für einen Roboter, der in engeren Umgebungen navigieren soll, eher weniger gut geeignet. Ähnliches gilt für „move_base“. Die Navigation und die Ground Truth durch die Odometry und den Laserscan funktioniert grundlegend zuverlässig. Jedoch scheint die Navigation in der Nähe von Wänden oder engem Terrain kein Anwendungsgebiet von „move_base“ zu sein. Dementsprechend muss bei weiterer Entwicklung in dieser Richtung eine eigene Navigation entwickelt werden, um die Aufgaben zuverlässig erfüllen zu können.

Ebenfalls sollte die Steuerung des Armes überarbeitet werden. Es ist unklar wie viele der Probleme durch die Simulation in Kombination mit „move_it“ verursacht wurden. Jedoch kann man

festhalten, dass die Anzahl der fehlerhaften Aktionen des Armes ein robustes und konsistentes Verhalten des Ablaufs verhindern.

In Zukunft ist zu empfehlen, das System auf den realen Roboter anzupassen, da die Möglichkeiten der Simulation ausgeschöpft wurden. Hierfür sind speziell die Algorithmen in ihrer Toleranz gegenüber unsauberen Daten zu erweitern, da diese in der Simulation nicht vorkommen. Außerdem sollten die zuvor beschriebenen Probleme mit der Steuerung der Fahrplattform und des Armes in der Realität überprüft und gegebenenfalls behoben werden.

Je nachdem wie erfolgreich die Algorithmen auf dem echten System sind, können noch ergänzende Methode, wie zum Beispiel Maschine Learning zum Detektieren der Klinke oder der Tür eingesetzt werden. Das Weiteren könnte man die ArUco Marker durch eine Karte der Umgebung ersetzen oder den Ablauf in ein größeres System einbinden. Außerdem wäre eine Verbindung des Fahrplattform Controllers mit dem Arm Controller interessant. Das hätte den Vorteil, dass Bewegungen möglich wären wie, dass der Endeffektor an einer globalen Position stehen bleibt, obwohl sich die Fahrplattform bewegt. So könnte der Gripper immer an der Tür Kante bleiben, obwohl der Husky seine Position korrigiert. Damit wäre das Problem der sich bewegenden Türen gelöst.

Abschließend kann man festhalten, dass das Projekt konzeptionell erfolgreich war und alle notwendigen Schritte realisiert und durch Simulationen der einzelnen Phasen verifiziert werden konnten. Eine Simulation des Gesamtablaufs war auf Grund der erläuterten Problematiken wie der erreichten Grenzen der Simulation und Schwierigkeiten mit der Steuerung des Husky nicht realisierbar. Im Rahmen der Entwicklung der Algorithmen und der Simulation wurden viele Erkenntnisse über den Prozess und dessen Komplexität, die Grenzen einer Simulation und die Wichtigkeit der Verifikation in der Realität gesammelt.

Ausblickend kann man feststellen, dass eine weitergehende Optimierung des vorgestellten Verfahrens und eine Übertragung auf das Echt-System ein gutes Potential zur praktischen Lösung der Tür-Öffnungs-Aufgabe bietet.

Literaturverzeichnis

[1] N. Banerjee, X. Long, R. Du und F. Polido, „Human-Supervised Control of the ATLAS Humanoid Robotfor Traversing Doors," November 2015. [Online]. Available: https://www.researchgate.net/publication/308850554_Human-supervised_control_of_the_ATLAS_humanoid_robot_for_traversing_doors. [Zugriff am 5 August 2021].

[2] E. Klingbeil, A. Saxena und A. Y. Ng, „Learning to Open New Doors," [Online]. Available: http://www.cs.cornell.edu/~asaxena/openingnewdoors/klingbeil-saxena-ng-opennewdoors.pdf. [Zugriff am 6 August 2021].

[3] S. A. Prieto, A. Adán, A. S. Vázquez und B. Quintana, „Passing through Open/Closed Doors: A Solution for 3D Scanning Robot," 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/21/4740. [Zugriff am 1 August 2021].

[4] J. G. Ramôa, V. Lopes, L. A. Alexandre und S. Mogo, „Real-time 2D–3D door detection and state classification on a low-power device," 29 April 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42452-021-04588-3. [Zugriff am 2 August 2021].

[5] W. Meeussen, M. Wise, S. Glaser und S. Chitta, „Autonomous Door Opening and Plugging In with a Personal Robot," 3 Mai 2010. [Online]. Available:

https://www.researchgate.net/publication/221076655_Autonomous_Door_Opening_and_Plugging_In_with_a_Personal_Robot. [Zugriff am 2021 August 2].

[6] A. Jain und C. C. Kemp, „Behaviors for Robust Door Opening and Doorway Traversal with a Force-Sensing Mobile Manipulator," 2008. [Online]. Available: https://www.semanticscholar.org/paper/Behaviors-for-Robust-Door-Opening-and-Doorway-with-Jain-Kemp/4c3342ad8f947bf62c4a25d1861f72dc5f5fd970. [Zugriff am 1 August 2021].

[7] T. Winiarski und K. Banachowicz, „Opening a door with a redundant impedance controlled robot," Juli 2013. [Online]. Available: https://www.researchgate.net/publication/261022988_Opening_a_door_with_a_redundant_impedance_controlled_robot. [Zugriff am 5 August 2021].

[8] isaac computer science, „Finite state machines," [Online]. Available: https://isaaccomputerscience.org/concepts/dsa_toc_fsm. [Zugriff am 4 August 2021].

[9] D. Nirwan, „Designing AI Agents' Behaviors with Behavior Trees," 6 Dezember 2020. [Online]. Available: https://towardsdatascience.com/designing-ai-agents-behaviors-with-behavior-trees-b28aa1c3cf8a. [Zugriff am 4 August 2021].

[10] HAW Hamburg, „Industrieroboter „Husky" für die Autonomone Quartiersmobilität," 8 April 2021. [Online]. Available: https://www.haw-hamburg.de/hochschule/technik-und-informatik/departments/maschinenbau-und-produktion/forschung/forschungsgruppen/elektrische-mobilitaet/tiq-fahrplattform-husky/. [Zugriff am 5 August 2021].

[11] MoveIt ROS, „MoveIt," [Online]. Available: https://moveit.ros.org/. [Zugriff am 5 August 2021].

[12] ROS Wiki, „gmapping,“ [Online]. Available: http://wiki.ros.org/gmapping. [Zugriff am 13 12 2021].

[13] Gazebo, „Gazebo - Robot simulation made easy,“ [Online]. Available: http://gazebosim.org/. [Zugriff am 5 August 2021].

[14] Opencv, „Detection of ArUco Markers,“ [Online]. Available: https://docs.opencv.org/4.5.2/d5/dae/tutorial_aruco_detection.html. [Zugriff am 3 8 2021].

[15] D. Stonier, „Py Trees,“ [Online]. Available: https://py-trees.readthedocs.io/en/devel/. [Zugriff am 14 Oktober 2021].

[16] K.-S. Chang und D. Zhu, „Hierarchical Finite State Machine (HFSM) & Behaviour Tree (BT),“ [Online]. Available: https://web.stanford.edu/class/cs123/lectures/CS123_lec08_HFSM_BT.pdf. [Zugriff am 4 August 2021].

[17] M. Estrada, „Finite State Machine vs Behaviour Tree, A True Story,“ 23 Februar 2014. [Online]. Available: https://coffeebraingames.wordpress.com/2014/02/23/finite-state-machine-vs-behaviour-tree-a-true-story/. [Zugriff am 4 August 2021].

[18] D. Stonier, „Py Trees for ROS,“ [Online]. Available: https://py-trees-ros.readthedocs.io/en/devel/. [Zugriff am 14 Oktober 2021].

[19] ROS Wiki, „actionlib,“ [Online]. Available: http://wiki.ros.org/actionlib. [Zugriff am 15 11 2021].

[20] ROS Wiki, „geometry_msgs/Pose Message,“ [Online]. Available: http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Pose.html.

[21] ROS Wiki, „tf,“ [Online]. Available: http://wiki.ros.org/tf. [Zugriff am 14 11 2021].

[22] Deine Tür, „Wiki-Wissen: Innentür messen und DIN-Normen für Türen und Zargen,“ [Online]. Available: https://www.deinetuer.de/wiki/innentuer-messen-din-normen. [Zugriff am 16 11 2021].

[23] ROS Wiki, „rviz,“ [Online]. Available: http://wiki.ros.org/rviz. [Zugriff am 7 12 2021].

[24] NobleProg, „UML State Machine Diagram,“ 7 April 2016. [Online]. Available: https://training-course-material.com/training/UML_State_Machine_Diagram. [Zugriff am 5 August 2021].

[25] ROS Wiki, „ROS move_base,“ 3 September 2020. [Online]. Available: http://wiki.ros.org/move_base. [Zugriff am 5 August 2021].

[26] „Türklinke,“ [Online]. Available: https://www.elton.nl/de/produkte/fingerschutzprofile/ellen-salvadita-fingerschutz .

[27] Tensorflow, „Training Custom Object Detector,“ [Online]. Available: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html. [Zugriff am 2021 Oktober 12].

[28] ROS Wiki, „pointcloud_to_laserscan,“ [Online]. Available: http://wiki.ros.org/pointcloud_to_laserscan. [Zugriff am 12 Oktober 2021].

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original