ING Fakultät Ingenieurwissenschaften

HTWK Hochschule für Technik, Wirtschaft und Kultur Leipzig

Masterarbeit

Thema: Simulation und Evaluation realistischer Bewegungsabläufe bei der Mensch-Roboter-Kooperation

vorgelegt von: Bastian Hofmann

Studiengang: Elektrotechnik und Informationstechnik
Studienprofil: Mechatronik

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. J.Jäkel
Betrieblicher Betreuer: Dr.rer.nat. A.Hoffmann

Ausgabetermin: 16.01.2023 Abgabetermin: 03.07.2023

Leipzig, 01.01.2023

Prof. Dr.-Ing. F. Illing
Prüfungsausschussvorsitzender

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Stand der Wissenschaft . . . 1
1.3 Welche Szenarien . . . 2
1.4 Welcher Nutzen / Contributions . . . 2

2 Konzept **3**
2.1 Simulation des Roboters . . . 3
2.2 Simulation des Menschen . . . 4
2.3 Behavior Trees . . . 4

3 Komponenten-/Softwareauswahl **6**
3.1 Dienstumgebung (ROS2) . . . 6
3.1.1 Auswahl . . . 6
3.1.2 Beschreibung . . . 6
3.2 Simulationsumgebung (Gazebo) . . . 8
3.2.1 Auswahl . . . 8
3.2.2 Robotersimulation . . . 9
3.2.3 Menschensimulation . . . 10
3.3 Roboterumgebung (MoveIt2) . . . 10

4 Umsetzung **12**
4.1 Grundlegender Systemaufbau . . . 12
4.2 Mensch . . . 12
4.2.1 Übersicht (Diagramme) . . . 12
4.2.2 Modellierung . . . 12
4.2.3 Programmierung . . . 13
4.3 Roboter . . . 14
4.3.1 Übersicht (Diagramme) . . . 14
4.3.2 Modellierung . . . 14
4.3.3 Details . . . 14

5 Szenarienbasierte Evaluation **15**
5.1 Simulation des Menschen . . . 15

5.2 Bewegung des Roboters 15
5.3 BehaviorTrees 15
5.3.1 Nodes 15
5.3.2 Kombinieren von Nodes zu einer Request 15

6 Diskussion **16**
6.1 Lessons Learned bei der Umsetzung 16
6.1.1 Erstellung des Robotermodells 16
6.1.2 Gazebo 16
6.1.3 ROS2 17
6.1.4 MoveIt2 18
6.2 Lessons Learned bei den Szenarien 18
6.2.1 Debugging 18

7 Zusammenfassung und Ausblick **19**
7.1 Ergebnisse 19
7.1.1 Graphische Repräsentation der Szenarien 19
7.1.2 Anpassung der Behavior Trees an Szenarien 19
7.2 Diskussion 19
7.3 Ausblick 19
7.3.1 Umsetzung in anderem Simulator 19
7.3.2 Simulation bewegter Objekte 19
7.3.3 Ergänzung von Umgebungserkennung 20

ING Fakultät
Ingenieurwissenschaften

HTWK
Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Aufgabenstellung zur Masterarbeit

für Herrn Bastian Hofmann, B.Eng.

Thema
Simulation und Evaluation realistischer Bewegungsabläufen bei der Mensch-Roboter-Kooperation

Beschreibung
Die Automatisierung von Prozessabläufen wird häufig in Etappen durchgeführt. Prozesse, welche selten oder oft unterschiedlich ausgeführt werden sollen, werden häufig nicht automatisiert, da der Aufwand der Automatisierung disproportional zum eigentlichen Aufwand der Aufgabe ist. Die Vollautomatisierung eines Prozesses hingegen ist sinnvoll, falls gleiche Aufgaben häufig wiederholt werden müssen. Hierbei ist der Aufwand der Automatisierung häufig kleiner als der Aufwand des gesamten Prozesses ohne Automatisierung.

Eine Art Zwischenschritt stellt das Feld der Mensch-Roboter-Kooperation dar, in welcher Roboter und Mensch zur selben Zeit am gleichen Ort arbeiten. Dabei wird die Ausdauer von Robotern durch die Flexibilität des Menschen ergänzt, welcher komplizierte oder kognitiv komplexe Tätigkeiten übernehmen kann. Bei diesen Projekten steht vor allem die Sicherheit der Arbeiter im Vordergrund, welche durch Roboter gefährdet werden können.

Für die Modellierung von Abläufen werden häufig Beschreibungssprachen verwendet. Diese Sprachen teilen die Abläufe in kleinere Schritte auf, welche das Verhalten darstellen. Leichte Modifizierbarkeit und Lesbarkeit sind dabei große Vorteile moderner Beschreibungssprachen.

In dieser Arbeit sollen die Vorteile einer Beschreibungssprache genutzt werden, um die Interaktion zwischen Mensch und Roboter in einer Simulation abzubilden. Hierzu sollen realitätsnahe Bewegungsabläufe erstellt werden, mit welchen Mensch und Roboter in Interaktion treten können, beispielsweise das Aufheben von Gegenständen, Kontrolle des Arbeitsbereichs und Übergabe von Gegenständen. Diese Aktionen sollen für die Beschreibungssprache verfügbar gemacht werden, um darin die Interaktion zwischen Mensch und Roboter modelliert werden kann.

Diese Bewegungsabläufe sollen in drei Szenarien mit unterschiedlichem Kollaborationsgrad genutzt werden. Hierfür sollen unterschiedliche Aufgaben für Koexistenz, Kooperation und Kollaboration erdacht und modelliert werden. Hierzu sollen für die Szenarien relevante Bewegungsabläufe, wie zum Beispiel das Greifen in den Arbeitsbereich oder die Übergabe von Objekten, bei der Roboter-Interaktion mit simulierten Personen dargestellt werden.

Diese sollen anhand vorgegebener Regeln in der Beschreibungssprache ausgelöst werden, um eine realitätsnahe Simulation zu ermöglichen. Außerdem sollen durch diese Aktionen bewegte Objekte auch durch die Beschreibungssprache in der Simulation erstellt und bewegt werden können und durch andere Aktoren im System verfolgt werden können. Außerdem ist die Anpassung der Beschreibungssprache auf zufälliges Verhalten vorzunehmen, da menschliches Verhalten nur selten durch lineare Abläufe beschrieben werden kann.

Der Projektablauf soll dabei zuerst mit einer Recherche zu möglichen Ausgangspunkten für eine Simulationsumgebung beginnen. Hierfür sollen einige Daten zu verschiedenen Umgebungen gesammelt und evaluiert werden. Aus diesen Daten soll die am besten passende Umgebung ermittelt werden. In dieser Umgebung soll ein System für menschliche Aktoren implementiert werden, um animierte Aktionen und Bewegungen im Raum darstellen und simulieren zu können. In dem so entwickelten System sollen dann mehrere Beispielszenarien implementiert werden, welche verschiedene Interaktionsgrade zwischen Mensch und Roboter darstellen, um das entwickelte System zu testen.

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. Jens Jäkel (HTWK Leipzig)
Betrieblicher Betreuer: Dr. Alwin Hoffmann (XITASO GmbH)

Leipzig, 13. Dezember 2022

1 Einleitung

1.1 Motivation

Das Feld der Mensch-Roboter-Kollaboration entwickelt sich mit zunehmender Geschwindigkeit fort. Viele Unternehmen bieten neue Lösungen für die unterschiedlichsten Einsatzszenarien der Endanwender.

Dabei ist eine Prüfung des Anwendungsfalls sinnvoll, um etwaige Probleme der Interaktion früh erkennen und beheben zu können. Diese Prüfung kann durch eine Simulation, in welcher der konkrete Anwendungsfall abgebildet wird, vereinfacht werden.

Außerdem bietet eine Simulation die Möglichkeit, die Aufgabe des Roboters, ohne dessen Anschaffung, evaluieren zu können. Das so gefertigte Modell des Anwendungsfalls könnte später auch als Grundlage für einen digitalen Zwilling dienen. Dieser kann später zur Wartung und Fehlerdiagnose des Systems dienen.

-MRK häufiger ein Thema -Anwendungsfälle sollen evaluiert werden -Erprobung von Szenarien ohne Roboter

->Simulation eine kompletten Szenarios mit Roboter und Mensch

1.2 Stand der Wissenschaft

Aktuelle Arbeiten:

-Planung von Interaktionen

-Parametervergleiche von maschinellen und menschlichen Bewegungen

-Vermeidung von Kollisionen und Strategie

-Steuerung von Robotern mit Behavior Trees

-> Keine allgemeine Simulation einen gesamten Szenarios mit Mensch.

`https://www.sciencedirect.com/science/article/pii/S2351978918311442` `https://www.researchgate.net/publication/319888916_Interactive_Simulation_of_Human-robot_`

Collaboration_Using_a_Force_Feedback_Device https://elib.dlr.de/120687/1/human_motion_projection.pdf https://www.researchgate.net/publication/220065749_Human-Robot_Collaboration_a_Survey

1.3 Welche Szenarien

Die drei Szenarien sollten so gewählt werden, dass vorher genutzte Bestandteile in späteren, komplexeren Szenarien weiter genutzt werden können. Hierfür kommen bestimmte Aufgaben, wie zum Beispiel die Manipulation von Objekten, besonders in Frage, da diese viele ähnliche Bestandteile haben, jedoch mehrere Szenarien denkbar sind.

Das erste abgebildete Szenario soll sich mit der Simulation einer bereits vollautomatisierten Fertigungsaufgabe handeln, in welcher ein Roboter im Arbeitsbereich eines Menschen Teile fertigt. Die zu erwartende Interaktion beschränkt sich hierbei auf die Anpassung der Fahrgeschwindigkeit bei Annäherung des Menschen.

Bei dem zweiten Szenario soll der Roboter Teile sortieren und auf ein Fließband legen, falls diese weiter genutzt werden können. Der Mensch muss nun nur noch den Ausschuss beseitigen, welcher vom Roboter in eine besondere Zone gelegt wird.

Die dritte simulierte Aufgabe stellt ein Colaborationsszenario dar, in welchem Mensch und Roboter an der selben Aufgabe arbeiten. Hierbei soll eine Palette entladen werden, wobei der Roboter nicht jedes Objekt ausreichend manipulieren kann. In diesen Fällen muss nun ein Mensch aushelfen, wodurch er mit dem Roboter in Interaktion tritt.

1.4 Welcher Nutzen / Contributions

- Erkennen von konzeptionellen Problemen vor Ersteinsatz

- Definition von Interaktion mit einfacheren Strukturen als State-Machines

2 Konzept

Die zu entwickelnde Simulation soll die bisher meißt separaten Zweige der Roboter- und Menschensimulation verbinden. Um die beiden Akteuren in der simulierten Umgebung zu steuern, werden Befehle von außerhalb der Simulation eingesetzt. Diese Befehle werden dabei von externer Software unter der Verwendung von Behavior Trees und Feedback aus der Simulation generiert.

Die zu erarbeitende Softwareumgebung soll einfach erweiterbar sein, um weitere Modifikationen und die Umsetzung anderer Projekte zuzulassen. Hierzu zählt die Austauschbarkeit und Erweiterbarkeit von Komponenten wie der simulierten Welt, dem Roboter oder dem simulierten Mensch. Um diese Möglichkeiten zu schaffen, sind die Systeme modular aufgebaut.

2.1 Simulation des Roboters

Der simulierte Roboter soll für viele unterschiedliche Szenarien nutzbar sein, was spezialisierte Robotertypen ausschließt. Außerdem ist die enge Interaktion mit Menschen interessant, was einen für Mensch-Roboter-Kollaboration ausgelegten Roboter spricht. Für diese beschriebenen Kriterien eignet sich der KUKA LBR iisy, welcher als Cobot vermarktet wird. Cobot ist dabei ein Portemanteau aus Collaborative und Robot, was die besondere Eignung für MRK-Szenarien noch einmal unterstreicht. Er besitzt auch einen modifizierbaren Endeffektor, um unterschiedlichste Aufgaben erfüllen zu können.

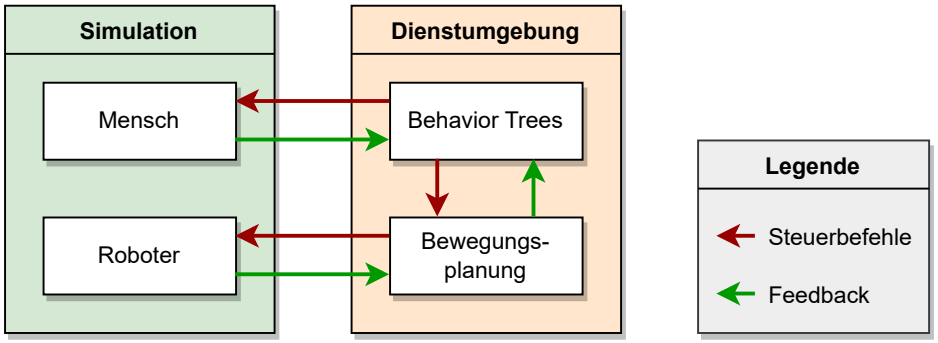

Abbildung 2.1: Visualisierung des Konzepts

Um den Kuka iisy in der Simulation verwenden zu können, muss ein Modell des Roboterarms erstellt werden. Dieses Modell sollte die physikalischen Eigenschaften des Roboters möglichst gut wiederspiegeln. Anhand dieses Modells kann der Roboter dann in der Simulation dargestellt werden und mit anderen Objekten interagieren.

2.2 Simulation des Menschen

Der Mensch soll in der Simulation typische Aufgaben erledigen und häufiges Verhalten abbilden können. Hierzu werden Animationen verwendet, welche die aktuelle Tätigkeit darstellen. Für komplexere Verhaltensweisen können Animationen und andere Aktionen, wie zum Beispiel eine Bewegung und Rotation kombiniert werden, um zum Beispiel die Aktion “laufen” auszuführen.

Auch hier wird ein Modell der Person für die Simulation benötigt. Außerdem werden mehrere Animationen und Übergänge zwischen diesen benötigt, um bestimmte Bewegungen darstellen zu können. Hinzu kommt noch eine Komponente, welche diese Animationen und andere Parameter von außen entgegennehmen kann, um sie in der Simulation ausführen zu können. Um die spätere Steuerung des Menschen von außerhalb zu erleichtern, müssen diese Aktionen im Fortschritt überwacht und abgebrochen werden können.

2.3 Behavior Trees

Häufig wird Verhalten in State-Machines ausgedrückt, welche jedoch einige Nachteile besitzen. State-Machines werden ab einer gewissen Größe schnell unübersichtlich. Dies erschwert die schnelle Erfassung von Abfolgen und Zustandsübergängen bei Änderungen am Code, welche jedoch essentiell für den Betrieb einer Maschine sind. Um diese Probleme zu adressieren, entstand das Konzept der Behavior Trees.

Ein Behavior Tree ist eine Struktur, um Verhalten als ein Baum zu beschreiben. Der Ablauf startet vom sogenannten Root, der Wurzel des Baums. Von dort an werden sogenannte Nodes, welche je nach Node unterschiedliches Verhalten abbilden, miteinander verbunden. Die Nodes werden untereinander angeordnet, welches die Relation der Nodes zueinander beschreibt. Jede Node hat dabei entweder die Root-Node oder eine andere Node über ihr im Baum und eine beliebige Anzahl an Nodes unter sich. Hierbei gibt es mehrere grundlegende Arten von Tree-Nodes.

Aktions-Nodes beschreiben einzelne ausführbare Aktionen. Mit Hilfe von Parametern kann ihr Verhalten von anderen Nodes beeinflusst werden.

Dekorations-Nodes können den Rückgabewert einer anderen Node modifizieren. Häufig existieren hier Negation, garantierter Erfolg und garantierter Fehler.

Sequenz-Nodes beschreiben eine nacheinander ausgeführte Abfolge von anderen Nodes, welche mit spezifischen Randbedingungen weiter fortschreitet.

Fallback-Nodes werden verwendet, um Verhalten zu definieren, welches nur bei Fehlern in vorherigen Nodes ausgeführt wird.

In dieser Arbeit sollen BehaviorTrees für die Steuerung von Mensch und Roboter verwendet werden. Die hierfür erstellten Nodes sollen universell gestaltet werden, um alle Szenarien, welche in dieser Arbeit bearbeitet werden, abzudecken.

3 Komponenten-/Softwareauswahl

3.1 Dienstumgebung (ROS2)

3.1.1 Auswahl

Durch eine Dienstumgebung werden häufig benötigte Funktionen bereitgestellt, welche in Programmen genutzt werden können. Dabei ist es irrelevant, ob diese für simulierte, aber auch echte Hardware, genutzt werden. Bei einer Dienstumgebung für Roboter gehören zu den grundlegendn Aspekten die Nachrichtenübergabe zwischen einzelen interagierenden Programmen, um eine gemeinsame Basis für ein einfach erweiterbares System zu schaffen.

In diesem Bereich sticht ROS als Dienstumgebung für Roboter hervor, da es sich um ein etabliertes, quelloffenes und häufig verwendetes System handelt. Es bietet die oben genannten Aspekte und einige weitere Verbesserungen, welche später näher beleuchtet werden. Die neuste Version ROS2 bietet dabei einige Verbesserungen im Vergleich zu früheren Version ROS1. Ein neues Nachrichtenformat mit Quality of Service kann zum Beispiel Nachrichten vorhalten und über sowohl TCP als auch UDP kommunizieren. Außerdem werden nun neben CMake auch andere Buildsysteme unterstützt, unter anderem auch Python.

Generell existieren im Feld der Roboter-Dienstumgebungen keine Alternativen mit ähnlichem Funktionsumfang und gleicher Reichweite, jedoch sind andere Systeme mit anderen Nachrichtenformaten denkbar. Vor allem die unzähligen ROS-Bibliotheken, welche von Nutzern des Systems über die Jahre erstellt wurden, machen das System so populär.[6]

-Alternative Ökosysteme mit gleichem Umfang wie ROS existieren nicht. -ROS2 -Andere (nur) Messagingsysteme -LCM -ZeroMQ

3.1.2 Beschreibung

ROS2[3], später auch einfach nur ROS genannt, beschreibt sich selbst als “a meta operating system for robots”[2]. Hierbei ist “operating system” nicht in seiner herkömmlichen Bedeutung eines vollständigen Betriebssystems zu verstehen. Es handelt sich dabei um eine gemeinsame Grundlage für Programme und Daten, welche durch ROS bereitgestellt wird.

Einzelne Bestandteile in der Umgebung sind dabei in Pakete gegliedert. Ein Paket kann beliebig viele Daten und Programme beinhalten, welche in zwei Dateien beschrieben werden. In CMakeLists.txt befinden sich Buildinstruktionen für den Compiler, falls sich Programme im Paket befinden. Außerdem können bestimmte Pfade aus dem Paket exportiert werden, sodass diese später im Workspace verfügbar sind. Programme, welche mit anderen Programmen in der Umgebung interagieren, werden in ROS "Nodes" genannt.

Zu den Aufgaben von ROS gehören dabei:

Buildumgebung

ROS benutzt colcon [1], um Pakete in den Workspaces reproduzierbar zu erstellen. Hierfür werden CMake und einige Erweiterungen, wie z.B. ament_cmake eingesetzt.

Workspaceverwaltung

Pakete können in verschiedenen Verzeichnissen installiert werden und müssen für andere Pakete auffindbar sein. ROS nutzt hierfür von colcon generierte Skripte, welche beim Erstellen eines Pakets und eines Workspaces mit angelegt werden. Das Skript des Pakets fügt nur dieses Paket der Umgebung hinzu, das Skript des Workspaces führt alle Skripte der enthaltenen Pakete aus, um diese der Umgebung hinzuzufügen.

Abhängigkeitsverwaltung

ROS kann durch die in den Paketen deklarierten Abhängigkeiten prüfen, ob diese in der aktuellen Umgebung verfügbar sind. Dies vermeidet Abstürze und undefiniertes Verhalten in der Ausführung von Nodes.

Datenübertragung

Nodes müssen miteinander auf einem festgelegten Weg kommunizieren können, um beliebige Verbindungen dieser zu unterstützen. Dieser wird durch ROS in Form mehrerer Bibliotheken für unterschiedliche Sprachen bereitgestellt.

Parameterübergabe

Nodes benötigen häufig problemspezifische Konfiguration, um in vorher nicht bedachten Szenarien eingesetzt werden zu können. ROS stellt diese durch deklarierfähige und integrierte Argumente bereit.

Startverwaltung

In sogenannten "launch"-Files können verschiedene Nodes und andere "launch"-Files zu komplexen Startvorgängen zusammengefasst werden.

3.2 Simulationsumgebung (Gazebo)

3.2.1 Auswahl

Als Simulationsumgebung können verschiedene Programme genutzt werden, welche sich in ihrem Funktionsumfang stak unterscheiden. Hierfür kommen dedizierte Werkzeuge zur Robotersimulation, aber auch zum Beispiel universell einsetzbare Gameengines in Frage. Diese Werkzeuge müssen hierfür auf ihre Tauglichkeit für die gesetzte Aufgabe geprüft werden. Auch andere Aspekte sind hierbei zu betrachten, wie Lizenzen oder schwer bewertbare Aspekte wie Nutzerfreundlichkeit. Für die Auswahl kommen verschiedene Prgramme in Frage, welche im folgenden weiter beleuchtet werden.

CoppeliaSim, früher auch V-REP genannt, ist eine Robotersimulationsumgebung mit integriertem Editor und ROS-Unterstützung. Es unterstützt viele Sprachen (C/C++, Python, Java, Lua, Matlab oder Octave) zur Entwicklung von Erweiterungen des Simulators. Der Simulator selbst unterstützt Menschliche Aktoren, jedoch können diese nur Animationen abspielen oder zusammen mit Bewegungen abspielen. CoppeliaSim existiert in 3 Versionen, welche sich im Funktionsumfang unterscheiden, jedoch hat nur die professionelle Version Zugriff auf alle Funktionen und Verwendungsszenarien.

Gazebo Ignition ist wie CoppeliaSim eine Robotersimulationsumgebung, jedoch ohne integrierten Editor und direkte ROS-Unterstützung. Gazebo setzt wie CoppeliaSim auf Erweiterungen, welche die gewünschten Funktionen einbinden können. Zum Beispiel existiert auch eine ROS-Brücke, welche die Anbindung an ROS ermöglicht. Auch hier unterstützt der Simulator nur Animationen für menschliche Aktoren. Das Projekt ist Open Source, unter der Apache Lizenz (Version 2.0), was die Verwendung in jeglichen Szenarien erleichtert.

Unity hingegen ist primär eine Grafikengine für Nutzung in Computerspielen. Es existieren mehrere Systeme zur Anbindung der Engine an ROS, vor allem das offizielle “Robotics Simulation”-Paket und ZeroSim. Beide Systeme erlauben die Erweiterung der Gameengine um die Simulation von Robotern. Unity besitzt eine gute Dokumentation, die vor allem auf die Nutzung im Einsteigerbereich zurückzuführen ist. Auch die Optionen zur Menschensimulation sind gut, da diese häufig in Spielen verwendet werden. Ein großer Nachteil hingegen ist die Lizenz, welche nur für Einzelpersonen kostenlos ist.

Die Unreal Engine ist wie Unity eine Grafikengine aus dem Spielebereich. Auch hier ist die Menschensimulation aufgrund oben genannter Gründe gut möglich. Jedoch existiert für Unreal Engine keine offizielle Lösung zur Anbindung an ROS2. Die Programmierung der Engine erfolgt in C++, was einer Drittlösung erlaubte, eine ROS-Anbindung für Unreal Engine zu erstellen. Die Lizenz der Unreal Engine erlaubt die kostenfreie Nutzung bis zu einem gewissen Umsatz mit der erstellten Software.

Eine weitere Möglichkeit zur Simulation stellt die Grafikengine Godot dar. Im Vergleich zu Unity und Unreal Engine ist Godot quelloffene Software unter der MIT-Lizenz. Auch hier stellt die Simulation von menschlichen Aktoren eine Standartaufgabe dar, jedoch befinden sich Teile des dafür verwendeten Systems derzeit in Überarbeitung. Auch für diese Engine existiert eine ROS2-Anbindung, jedoch ist diese nicht offiziell.

Jede der drei Gameengines besitzt ein integriertes Physiksystem, welches die Simulation von starren Körpern und Gelenken erlaubt. Aus diesen Funktionen könnte ein Roboterarm aufgebaut werden, welcher dann durch eine der ROS-Brücken der Engines gesteuert werden kann.

Die Wahl der Simulationsumgebung fiel auf Gazebo Ignition, da dieser Simulator bereits im ROS-Framework etabliert ist. Dabei erlauben die offizielle ROS-Anbindung und offene Lizenz eine zuverlässige Verwendung in unterschidlichsten Szenarien.

3.2.2 Robotersimulation

Für die Robotersimulation wird ein Modell des Roboters benötigt, in welchem dieser für die Siimulationsumgebung beschrieben wird. Gazebo nutzt hierfür .srdf-Dateien, welche auf xml basieren. In diesen werden die einzelnen Glieder des Arms und die verbindenden Gelenke beschrieben.

Jedes Glied des Modells besitzt eine Masse, einen Masseschwerpunkt und eine Trägheitsmatrix für die Physiksimulation in Gazebo. Außerdem werden Modelle für die visuelle Repräsentation in Gazebo und die Kollisionserkennung in der Physiksimulation hinterlegt. Für beide existieren einfache Modelle wie Zylinder, Boxen und Kugeln. Da diese Formen nicht jeden Anwendungsfall abdecken und in der visuellen Repräsentation nicht ausreichen, können auch eigene Modelle hinterlegt werden.

Gelenke werden separat von den Gliedern definiert und verbinden jeweils zwei Glieder miteinander. Durch das Aneinanderreihen von mehreren Gliedern und Gelenken kann so jeder Roboteraufbau beschrieben werden. Jedes Gelenk besitzt eine Position und Rotation im Raum, um dessen Effekte je nach Typ des Gelenks berechnen zu können. Aspekte wie Reibung und Dämpfung können auch für die Physiksimulation angegeben werden. Folgende Typen von Gelenken können in urdf genutzt werden:

freie Gelenke ermöglichen vollständige Bewegung in allen 6 Freiheitsgraden. Sie stellen den normalen Zustand der Gelenke zueinander dar.

planare Gelenke erlauben Bewegungen senkrecht zur Achse des Gelenks. Sie werden für zum Beispiel Bodenkollisionen eingesetzt.

feste Gelenke sperren alle 6 Freiheitsgrade und werden häufig zur Plazierung von Objekten in einer Szene genutzt.

kontinuierliche Gelenke erlauben die beliebige Rotation um die Achse des Gelenks. Sie sind nur selten in rotierenden Gelenken mit Schleifkontakten zu finden.

drehbare Gelenke verhalten sich wie kontinuerliche Verbindungen, haben jedoch minimale und maximale Auslenkungen. Sie sind die häufigste Art von Gelenken in Roboterarmen.

prismatische Gelenke ermöglichen die Bewegung entlang der Achse des Gelenks. Denkbare Anwendungsfälle sind simulierte lineare Aktuatoren.

3.2.3 Menschensimulation

Gazebo besitzt bereits ein einfaches Animationssystem für bewegliche Aktoren, welches auch für Menschen nutzbar ist. Es existiert bereits ein Modell eines Menschen mit mehreren Animationen, welche allein abgespielt, oder an Bewegungen gekoppelt werden können. Dadurch ist eine Laufanimation realisierbar, welche synchronisiert zur Bewegung abgespielt wird.

Jedoch ist dies nur unter der Bedingung möglich, dass der gesamte Bewegungsablauf zum Simulationsstart bekannt ist. Dies ist auf die Definition der Pfade, welche die Bewegung auslösen, zurückzuführen. Diese können nur in der .sdf-Datei des Aktoren definiert werden, was Veränderungen zur Laufzeit ausschließt. Durch diesen Umstand ist der mögliche Simulationsumfang nicht ausreichend.

Um diesen Umstand zu beheben, ist die Entwicklung eines eigenen Systems zum Bewegen und Animieren des Menschen unausweichlich. Dieses System muss, wie im Konzept beschrieben, Steuerbefehle von außen empfangen, umsetzen und Feedback liefern können.

3.3 Roboterumgebung (MoveIt2)

MoveIt2 ist das empfohlene ROS2 Paket für Bewegungsplanung von Robotern. Das System besteht aus meheren Komponmenten, welche in ihrer Gesamtheit den Bereich der Bewegungsplanung abdecken. Der Nutzer kann mit MoveIt auf mehreren Wegen Steuerbefehle für den Roboter absenden.

Die einfachste Art der Inbetriebnahme ist über das mitgelieferte RViz-Plugin und die demo-Launch-Files, welche durch den Setupassistenten für den Roboter generiert werden. Dort können Bewegungen durch das Bewegen von Markierungen oder in der Simulation geplant und ausgeführt werden.

Da sich ein solches System nur beschränkt zur Automatisierung durch Software eignet, existieren auch noch andere Schnitstellen. Für die Sprache Python existierte früher noch das moveit_commander Paket, welches den Zugriff auf MoveIt in Pyhon erlaubt, welches aber aktuell noch nicht portiert wurde. [4] Die direkte Nutzung der C++-API ist aktuell die einzige

offizielle Möglichkeit, mit MoveIt auf einer abstrakteren Ebene zu interagieren. Natürlich können die Befehle auch direkt an die entsprechenden Topics gesendet werden, was jedoch Erfahrung mit den verwendeten Datenstrukturen benötigt.

Durch diese Schnittstelle erhält die sogenannte MoveGroup ihre Informationen über die gewünschte Bewegung. Diese Daten können durch eine OccupancyMap ergänzt werden, welche die Bereiche beschreibt, welche sich um den Roboter befinden. Eine solche Erweiterung erlaubt die Nutzung von Kollisionsvermeidung mit Objekten im Planungsbereich.

Die Planung der Bewegung wird durch einen der zahlreichen implementierten Solver erledigt, welcher durch die MoveGroup aufgerufen wird. Um die generierte Bewegung umzusetzen, werden die gewünschten Gelenkpositionen als Abfolge an ros_control weitergegeben. Dabei können sowohl echte Hardwaretreiber, aber auch simulierte Roboter genutzt werden. Der Erfolg der gesamten Pipeline kann dabei durch einen Feedbackmechanismus überwacht werden.

Im Falle von Gazebo wird ign_ros_control genutzt, welches die benötigten ros_control Controller in die Simulation einbindet. Diese können dann wie normale Controller von ros_-control genutzt werden.

Dieser Ablauf ist auch im Anhang unter Abbildung 7.1 visualisiert.

4 Umsetzung

4.1 Grundlegender Systemaufbau

Der grundlegende Systemaufbau musste stark modifiziert werden, wie in Abbildung 4.1 zu sehen, um die gesetzten Aufgaben erfüllen zu können. Dabei fallen vor allem die überarbeitete Kommunikation mit dem simulierten Menschen und die komplexere Steuerung des Roboterarms auf.

Die komplexere Steuerung des Roboters ist auf die Struktur von MoveIt zurückzuführen, welches in viele einzelne Teilmodule aufgeteilt ist. Diese müssen durch ihren modularen Aufbau, welcher für die Vielseitigkeit verantwortlich ist, einzeln konfiguriert werden, um miteinander in Interaktion treten zu können.

Außerdem musste die Kommunikation des Modells des Menschen überarbeitet werden, da die ROS-Kommunikation in Gazebo nur mit Einschränkungen möglich ist.

-BehaviorTree -> ActorPluginServer -> ActorPluginServer

-BehaviorTree -> MoveIt -> ros_control -> Gazebo

4.2 Mensch

4.2.1 Übersicht (Diagramme)

4.2.2 Modellierung

Rerigging des Actor-Modells Animation in eigenem Rig Konflikte durch 'verschiedene' Rigs in Animationen Erstellung eines neuen Rigify-Rigs Erneutes Erstellen von Animationen Disconnect Bones in Rig Flatten Hierarchy

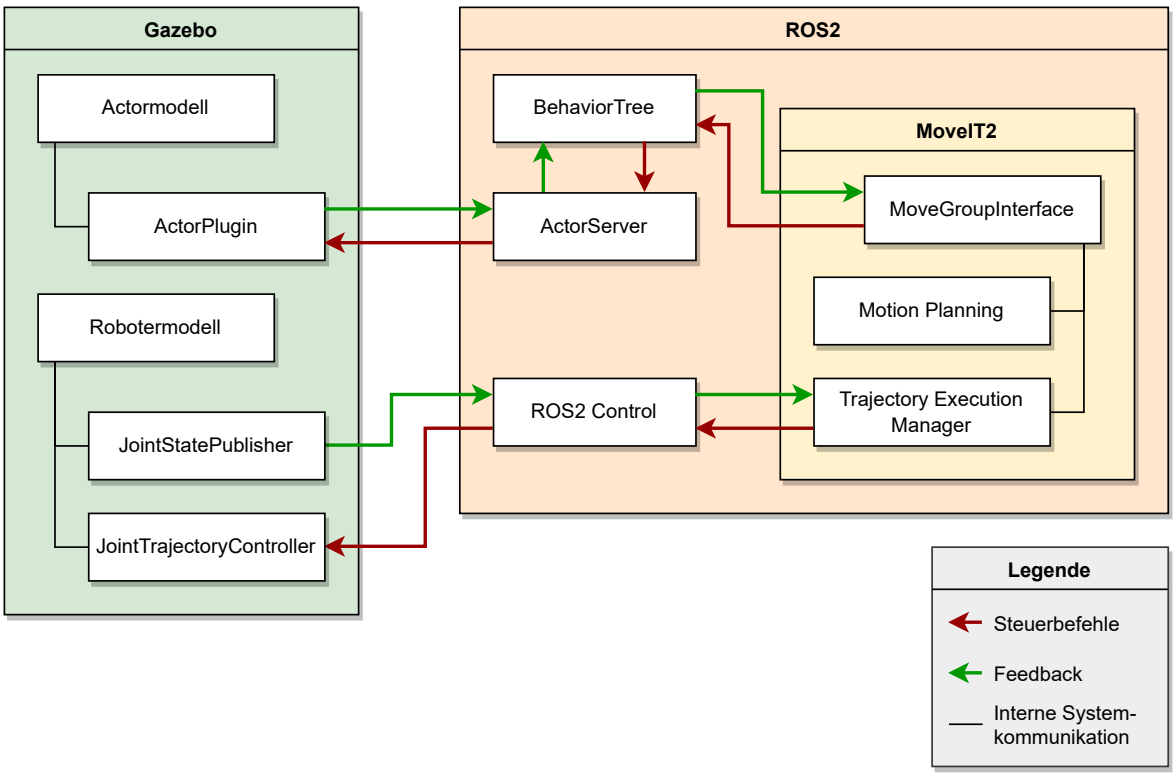

Abbildung 4.1: Visualisierung des überarbeiteten Konzepts

4.2.3 Programmierung

Message Queue

-2 rclcpp Instanzen kollidieren

-Anderes Protokoll nötig

-Möglichst einfach, von anderen Sprachen verwendbar und schnell

-Websocket zu langsam

-Shared memory schwer zu synchronisieren

-Message Queue guter Mix aus beiden

ROS-Server

-Transformieren von ROS2 action server in Message Queue

-Eigene state machine

Gazebo Plugin

-Relativ einfache Implementation

-Reagiert nur auf Nachrichten, kein Konzept der gesamten Abfolge

4.3 Roboter

4.3.1 Übersicht (Diagramme)

4.3.2 Modellierung

Erstellung der Robotermodelle aus Herstellerdaten

Kollision und Visualisierung

4.3.3 Details

5 Szenarienbasierte Evaluation

5.1 Simulation des Menschen

-Animationen und Bewegungen funktionieren

-Kollisionen möglich, aber mit Animationen schwer zu synchronisieren

5.2 Bewegung des Roboters

-Roboter kann sich bewegen

-Kollisionen verfügbar

-Interaktion technisch möglich, nicht implementiert. (Kooperation MoveIt/Gazebo nötig)

5.3 BehaviorTrees

5.3.1 Nodes

-Nodes für implementierte Funktionen verfügbar

-Einige andere, technisch relevante Nodes auch implementiert

5.3.2 Kombinieren von Nodes zu einer Request

-MoveIt Planung benötigt mehrere Parameter, sollen aber in einzelnen Nodes gesetzt werden

-Kombination nötig, Beschreibung hier

6 Diskussion

6.1 Lessons Learned bei der Umsetzung

-Viele kleine Unannehmlichkeiten, kombiniert ein großes Problemen

-Dokumentation für spätere Projekte

6.1.1 Erstellung des Robotermodells

-Keine direkten technischen Daten zum Roboter von Kuka

-Nur CAD-Dateien der Außenhülle

-Schätzung von Spezifikationen anhand Marketingmaterial

6.1.2 Gazebo

Upgrade auf Ignition

Gazebo ist zu diesem Zeitpunkt in mehreren, teilweise gleichnamigen Versionen verfügbar, welche sich jedoch grundlegend unterscheiden. Ein altes Projekt mit dem früheren Namen Gazebo, welches nun in Gazebo Classic umbenannt wurde, die Neuimplementation der Simulationssoftware mit dem Namen Gazebo Ignition und die nächste Version, welche nur noch den Namen Gazebo tragen soll. Dies ist darauf zurückzuführen, dass Gazebo Classic und Ignition eine Zeit lang gleichzeitig existierten, jedoch unterschiedliche Funktionen und interne Schnittstellen besitzen.

Das Upgrade von Gazebo Classic auf Gazebo Ignition gestaltete sich aus mehreren Gründen schwierig. Dies ist am leichtesten an der geänderten Nutzeroberfläche sichtbar, welche in Ignition nun modular ausgeführt ist. Um dieses neue modulare System nutzen zu können, wurde das Dateiformat für die Simulationen angepasst. In diesem werden die benötigten Physikparameter, Nutzeroberflächen, Plugins und die Simulationswelt definiert.

Um alte Simulationsdateien zu migieren, empfiehlt sich die Erstellung einer neuen, leeren Welt in Gazebo Ignition, sodass alle benötigten Physik, Nuteroberflächen und Pluginreferenzen

erstellt werden. Danach kann die Welt Stück für Stück in das neue Format übertragen werden, wobei nach jedem Eintrag ein Test zur Funktionsfähigkeit gemacht werden sollte, da sich bestimmte Parameterdeklarationen geändert haben. Die Arbeit in kleine Stücke aufzuteilen ist hierbei hilfreich, da Fehler während des Ladevorgangs im Log nicht weiter beschrieben werden.

Auch das alte Plugin musste auf das neue System angepasst werden, was auch hier zu einer kompletten Neuimplementation führte, da die internen Systeme zu große Unterschiede aufwiesen. Darunter fallen eine grundlegende Strukturänderung der unterliegenden Engine, welche auf ein Entity-Component-System umstellte, aber auch sahlreiche kleinere Änderungen

Pluginarchitektur

-Plugins werden im selben Kontext geladen, verwendung von Librarys mit globalem Kontext schwer

Doppelte Messagedienste

-Duplizierung von Messagediensten in ROS und Gazebo

-Potentielle Lösung in einer Präsentation angekündigt (mal sehen, ob ich die wiederfinde)

Fehlende Animationsgrundlagen

-Animationen werden als .col bereitgestellt

-Enthalten Textur, Mesh und Bones, jedoch nicht verbunden -> schlecht für Animation

6.1.3 ROS2

Nachrichten und deren Echtzeitfähigkeit

-TCP und UDP verfügbar, jedoch nicht sofort kompatibel

Änderung der Compilertoolchain

-Änderung des Buildsystems von rosbuild auf catkin

-Benötigte Änderungen in CMakeFile und package

6.1.4 MoveIt2

Upgrade auf MoveIt2

-Unterschiede in der Deklaration des Roboters selbst

-Änderungen der Deklaration der ros_control Anbindung

-Vorerst kein Setup, manuelle Migration erforderlich

->Lesson learned: Projekte häufig auf Updates prüfen

Fehlerhafte Generierung der Roboter

-Einige Aspekte der Generierung nicht erforderlich oder falsch

Controller

-Controller benötigte Anpassungen und manuelle Tests

6.2 Lessons Learned bei den Szenarien

6.2.1 Debugging

-async tty Option

-gdb ist ein Muss

-“Aufhängen” von trees

7 Zusammenfassung und Ausblick

7.1 Ergebnisse

7.1.1 Graphische Repräsentation der Szenarien

-Szenarien graphisch abgebildet

7.1.2 Anpassung der Behavior Trees an Szenarien

-Trees angepasst

7.2 Diskussion

-Viele Iterationen benötigt

-Funktionstüchtige Simulation der Szenarien

-Bewegung der Objekte schwerer als gedacht

-Synchronisationsmechanismus Gazebo <-> MoveIt benötigt

7.3 Ausblick

7.3.1 Umsetzung in anderem Simulator

-Einfachere ROS Anbindung -Potentiell einfacher ausbaubar auf Basis einer erweiterbaren Gameengine -Mangelhafte Dokumentation von Gazebo

7.3.2 Simulation bewegter Objekte

-Aufgrund von Komplexität des Prozesses nicht integriert

7.3.3 Ergänzung von Umgebungserkennung

-MoveIt hat Unterstützung

-Daten aus Gazebo extrahieren und in MoveIt einbinden

-Person in OctoMap erkennen

Literatur

[1] *colcon - collective construction*. letzter Zugriff: 02.04.2023. URL: https://colcon.readthedocs.io/en/released/.

[2] *GitHub - ros2/ros2: The Robot Operating System, is a meta operating system for robots.* letzter Zugriff: 02.04.2023. URL: https://github.com/ros2/ros2.

[3] Steven Macenski u. a. „Robot Operating System 2: Design, architecture, and uses in the wild“. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[4] *MoveIt Commander with ROS2 - Issue #337 - ros-planning/moveit2_tutorials*. letzter Zugriff: 10.04.2023. URL: https://github.com/ros-planning/moveit2_tutorials/issues/337.

[5] *moveit2_tutorials/moveit_pipeline.png at humble - ros-planning/moveit2_tutorials*. letzter Zugriff: 02.04.2023. URL: https://github.com/ros-planning/moveit2_tutorials/blob/humble/_static/images/moveit_pipeline.png.

[6] *Packages - /ros2/ubuntu/pool/main/ :: Oregon State University Open Source Lab*. letzter Zugriff: 10.04.2023. URL: http://packages.ros.org/ros2/ubuntu/pool/main/.

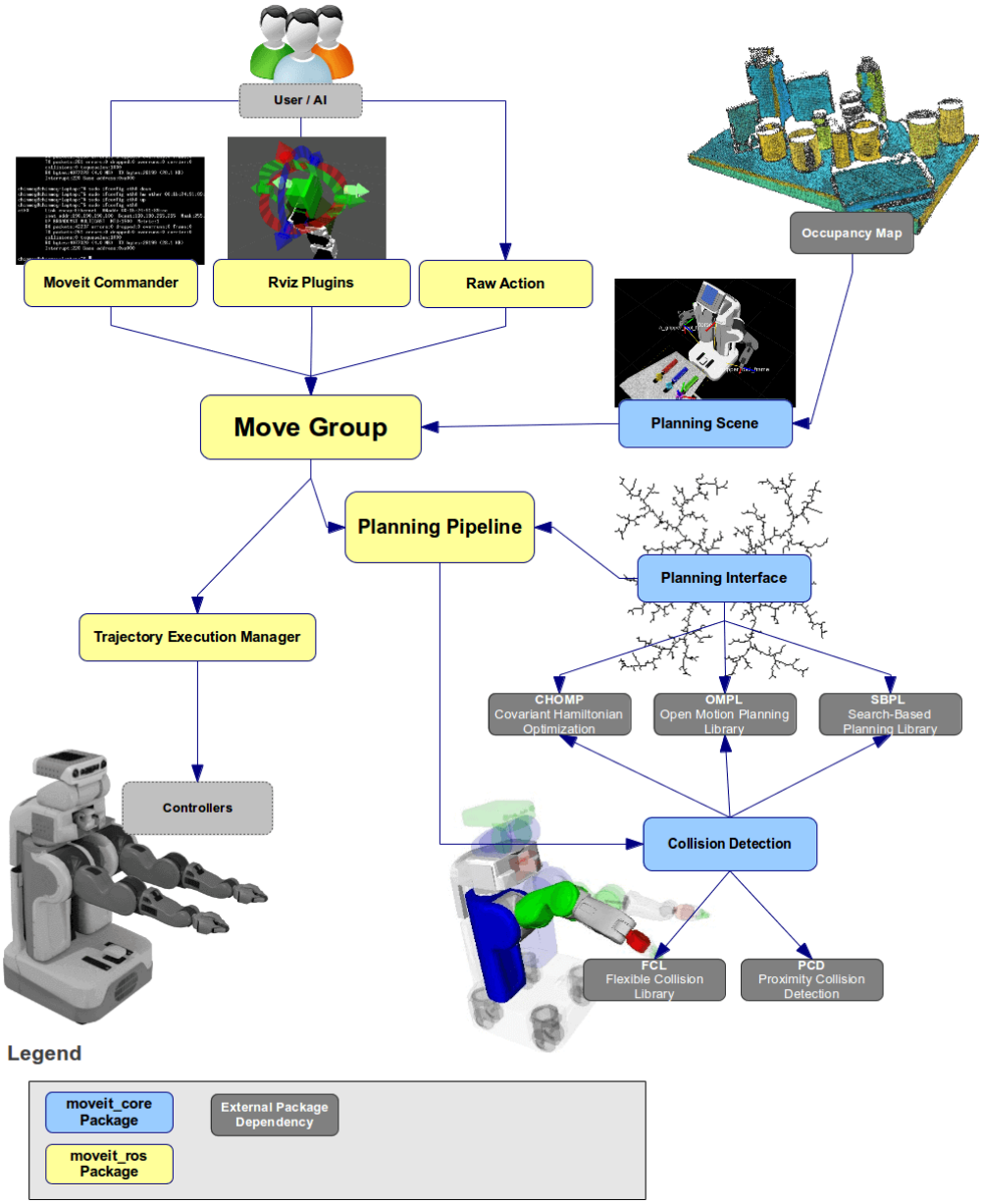

Abbildung 7.1: Visualisierung der MoveIt Pipeline[5]