OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
FACULTY OF COMPUTER SCIENCE
Communication and Networked Systems

Communication and Networked Systems

Bachelorarbeit

Haptische Kommunikation im MIoT-Lab: Ein Framework für hochskalierbare, automatisierte Netzwerkexperimente mit haptischen Daten

Johannes Behrens

Matr. 000000

Betreuer: Prof. Dr. rer. nat. Mesut Güneş
Betreuender Assistent: MSc. Frank Engelhardt

Institut für Intelligente Kooperierende Systeme, Otto-von-Guericke-Universität Magdeburg

02. November 2019

Kurzzusammenfassung

Die Bachelorarbeit befasst sich mit der Übertragung haptischer Daten über drahtlose Netzwerke. Ziel ist es, für die haptische Kommunikation ein Framework für Netzwerkexperimente zu schaffen. Mit diesem Framework können Algorithmen zur Steuerung von Robotern, Netzwerke für die haptische Kommunikation, aber auch Kodierungsverfahren entwickelt und evaluiert werden. Dazu gehört ein Softwarebestandteil, der Roboter(-simulationen) aus der Ferne über ein drahtloses Netzwerk steuerbar macht. Genauer wird hier die Robotersimulation virtual robot experimation platform (v-rep) betrachtet. Voraussetzung für dieses Framework ist, dass es die Anforderungen an die haptische Kommunikation einhält und einen geringen Overhead in Form von Rechenzeit und Netzwerklast verursacht. Dadurch wird eine Verfälschung der Testergebnisse verhindert. In Tests konnte gezeigt werden, dass das Framework diesen Anforderungen entspricht. So konnte etwa eine Frequenz von 1 kHz, wie auch Latenzen von weit unter 1 ms eingehalten werden. Auch ist eine einfache Anpassung des Frameworks an bestimmte Szenarien und Algorithmen möglich. Gleichzeitig bleibt für den Entwickler eine große Freiheit in der eigentlichen Umsetzung der Algorithmen erhalten. Weiterhin ist ein möglichst automatischer Ablauf der Tests über das Magdeburg Internet of Things Lab (MIoT-Lab) erreichbar.

Die Übertragung der Daten soll unter anderem über das MIoT-Lab erfolgen, um die Netzwerkbedingungen einer realen Welt zu nutzen. Dafür wurde ein Konzept entwickelt, wie das Framework mit dem MIoT-Lab kombiniert und die einzelnen Bestandteile dort ausgeführt werden können.

Abstract

The bachelor thesis deals with the transmission of haptic data via wireless networks. The aim is to create a framework for network experiments for haptic communication. This framework can be used to develop and evaluate algorithms for controlling robots, networks for haptic communication, as well as coding methods. This includes a software component that makes robot (simulation) remotely controllable via a wireless network. The robot simulation v-rep is examined in more detail here. The prerequisite for this framework is that it meets the requirements of haptic communication and causes a low overhead in the form of computing time and network load. This prevents falsification of the test results. Tests have shown that the framework meets these requirements. For example, a frequency of 1 kHz and latencies of well below 1 ms could be maintained. It is also possible to easily adapt the framework to specific scenarios and algorithms. At the same time, the developer retains great freedom in the actual implementation of the algorithms. Furthermore, the MIoT-Lab can be used to automate the tests as much as possible.

The transmission of the data is to take place among other things over the MIoT-Lab, in order to use the network conditions of a real world. Therefore a concept was developed, how the framework can be combined with the MIoT-Lab and how the single components can be executed there.

Genderhinweis

Es wurde in dieser Bachelorarbeit versucht, vorwiegend geschlechtsneutrale Bezeichnungen zu verwenden. Sollte an der einen oder anderen Stelle doch, wie im deutschsprachigen Raum üblich, eine maskuline Bezeichnung auftauchen, so schließt diese selbstredend auch die feminine Form mit ein.

Danksagungen

Der Autor möchte sich hiermit bei Prof. Dr. Mesut Güneş und M.Sc. Frank Engelhardt für dieses Thema und die geleistete Unterstzützung bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Akronyme

4G Vierte Generation im Mobilfunk. 10

5G Fünfte Generation im Mobilfunk. 10

API Application Programming Interface. 4, 8, 10, 13, 14, 19, 20, 26, 29

CPU Central Processing Unit. 11

DES Distributed Embedded Systems. 5, 13, 17–19

IP Internet Protocol. 18

MIoT-Lab Magdeburg Internet of Things Lab. iii, 2, 5, 7, 11, 13, 16, 17, 41

Mutex Mutual exclusion. 11, 14, 27, 34

POSIX Portable Operating System Interface. 2, 11, 21

QoE Quality of Experience. 2, 8, 10, 37

ROS Robot Operating System. 19

TBMS Testbed Management System. 2, 4, 5, 7, 13, 17, 41, 42

TCP Transmission Control Protocol. 13, 42

UDP User Datagram Protocol. 12, 13, 42

v-rep virtual robot experimation platform. iii, 4, 7, 8, 11, 13–15, 17–21, 23–26, 32, 41

XML Extensible Markup Language. 13, 17

KAPITEL 1

Einleitung

1.1 Motivation

Bereits bei der Durchführung eines Softwareprojekts während meines Studiums hatte ich die erste Berührung mit dem Thema "Haptische Kommunikation". Bei dem Projekt ging es inhaltlich darum, dass eine Drohne über ein drahtloses Netzwerk aus Routern gesteuert werden sollte.

Bei dem Softwareprojekt stieß ich auf mehrere Probleme. Ich war bei der Durchführung von Experimenten immer auf den Flug einer realen Drohne angewiesen. Dies ist mit einem hohen Aufwand verbunden. Der Drohnenflug musste vorbereitet werden, wozu unter anderem das Laden der Akkus und das Aufsuchen eines Fluglabors gehörte. Durch sich ändernde Testbedingungen war es nicht möglich, wiederholbare Testszenarien durchzuführen und diese objektiv auszuwerten.

1.2 Ziel der Arbeit

Aus dieser Motivation heraus soll hier ein Framework geschaffen werden, das es ermöglicht, eine Kommunikation mit haptischen Daten zu evaluieren. Bei dieser haptischen Kommunikation geht es um die Kommunikation zwischen einem Computer und einem Menschen oder zwischen zwei Menschen, bei denen Daten für ein taktiles Erlebnis übertragen werden. Dabei wird über haptische Geräte, welche in der Regel Ein- und Ausgabegeräte, wie ein Joystick oder Datenhandschuhe sind, die Verbindung zu einem Computer hergestellt. Diese Geräte nehmen die Körperbewegungen auf, mit denen Software oder wiederum andere Hardware, wie Roboter gesteuert werden können. Bei der Nutzung dieser haptischen Kommunikation werden nicht nur durch den Nutzer Aktionen ausgeführt, sondern der Nutzer erhält auch Informationen in Form von fühlbaren Empfindungen, wie etwa Vibrationen [1]. Ein Beispiel hierfür ist die Steuerung einer Drohne mit Datenhandschuhen, die ein Feedback über Vibratoren geben können. Die Steuerung erfolgt über Bewegungssensoren in den Handschuhen. Fliegt die Drohne in die Nähe einer Wand, kann ein Warnsignal in Form von Vibrationen ausgegeben werden. Diese Vibrationen werden stärker, je näher die Drohne an die Wand herankommt. In diesem Beispiel befindet sich die Drohne zusammen mit der

Steuerung in der so genannten Control Domain. Dieser Bereich ist dadurch gekennzeichnet, dass dort physikalische Größen aus der echten Welt eine Rolle spielen. Dies ist beispielsweise die Distanz der Drohne von der Wand oder die Steuerungsbefehle. Das Netzwerk fällt hingegen in den davon getrennten Bereich der digitalen Network Domain. Dieser Bereich ist durch abstrakte Informationen gekennzeichnet. Die haptische Kommunikation fällt durch die Übertragung haptischer Daten über ein Netzwerk genau in diese Lücke zwischen den beiden Bereichen.

Durch die haptische Kommunikation ergeben sich eine Reihe an Anforderungen an ein Netzwerk und die Software, die bei der Implementierung berücksichtigt werden müssen. Einerseits ist dies eine Sendefrequenz von bis zu 1 kHz. Diese ermöglicht es dem Nutzer, hochfrequente Effekte wahrzunehmen. Gleichzeitig muss die Latenz so niedrig wie möglich gehalten werden. Eine zu hohe Latenz kann die Stabilität des System gefährden oder die Performance einschränken. Auch sollten die Paketheader so klein wie möglich gehalten werden. Bei einer hohen Frequenz, wie 1 kHz, wächst der Overhead sonst stark und wird zu einem kritischen Faktor [2]. Um möglichst wenig Daten trotz der hohen Frequenz zu übertragen, soll es möglich sein, verschiedene Kodierungen auf die Kommunikation anzuwenden. Dadurch wird das Netzwerk weniger belastet. Auch können so verschiedene Kodierungsalgorithmen für die haptische Kommunikation getestet werden.

Das Framework soll Teil des vorhandenen MIoT-Lab werden. Das ermöglicht die Nutzung der bereits vorhandenen Netzwerkinfrastruktur und des Testbed Management System (TBMS) für den Ablauf der Experimente. Bei der Implementierung des Frameworks soll generell darauf geachtet werden, dass es sich möglichst einfach an sich ändernde Rahmenbedingungen anpassen lässt. Dazu gehört die freie Wahl des Netzwerks und des Steuerungsalgorithmus beziehungsweise des Anschlusses eines Controllers oder Datenhandschuhs. Das Testbed eignet sich dadurch für das Evaluieren verschiedener Bestandteile einer haptischen Kommunikation. Dazu gehören unter anderem das Netzwerk (Latenz, Paketverlust,...), Routingalgorithmen, Kodierungsverfahren und Steuerungsalgorithmen. Um die Güte der Lösung von Problemen zu bewerten ist es nötig, sich von der alleinigen Nutzung von klassischen Netzwerkmetriken zu lösen und dafür die Quality of Experience (QoE) zu betrachten. Ein Ziel des Testbeds ist es unter anderem herauszufinden, welche Einflüsse die Netzwerkmetriken, wie zum Beispiel die Latenz, auf die Lösung des eigentlichen Problems haben. Es müssen neue Metriken für die Qualität der Steuerung gefunden werden. Bei einem linienfolgenden Roboter ist dies beispielsweise die Abweichung vom optimalen Pfad.

Damit das Framework mit möglichst viel Hard- und Software kompatibel ist, wird auf die Portable Operating System Interface (POSIX)-Kompatibilität geachtet.

KAPITEL 2

Related Work

Im Folgenden werden Arbeiten vorgestellt, die sich bereits mit Forschungsfragen auseinandersetzen, die eng mit der Fragestellung dieser Arbeit verknüpft sind. Eine Gegenüberstellung mit der Fragestellung und dem Ziel dieser Arbeit zeigt auf, dass eine Forschungslücke existiert, die erst geschlossen werden muss.
Eine Recherche ergab, dass es noch keine Arbeiten gibt, die genau dieses Thema abdecken. Allerdings gibt es bereits Ergebnisse zu ähnlichen Themen und zu einzelnen Bestandteilen dieser Arbeit. Diese Lösungen werden hier vorgestellt und mit der Forschungsfrage unter Zuhilfenahme der Kriterien aus Auflistung 2.1 verglichen. Einerseits ergeben sich die Lösungen aus der Auseinandersetzung mit der Control Domain, also Experimenten mit Robotern und der Steuerung dieser. Andererseits beruhen sie auf der Beschäftigung mit Testbeds und Simulationen im Bereich der Network Domain.

Auflistung 2.1: Kriterien für den Vergleich der bestehenden Lösungen mit der Forschungsfrage dieser Arbeit.

1. **Steuerung der Roboter(-simulation) über ein Netzwerk**
   Die Lösung ermöglicht es, dass ein Roboter oder eine Robotersimulation über ein Computernetzwerk gesteuert wird.

2. **Erfüllung der Anforderungen an haptische Kommunikation**
   Die Lösung erfüllt die Anforderungen an die haptische Kommunikation. Dazu gehört die Einhaltung einer Frequenz von 1 kHz, eine Latenz unter 1 ms und eine hohe Zuverlässigkeit (siehe Kapitel 3.1.1).

3. **Unterstützung von Robotersimulationen**
   Es ist mit der Lösung möglich, Robotersimulationen im Bereich der Control Domain einzusetzen. Dies ist wichtig, da es nicht immer möglich ist, einen realen Roboter einzusetzen.

4. **Unterstützung realer Roboter**
   Es ist mit der Lösung möglich, reale Roboter im Bereich der Control Domain einzusetzen.

5. **Steuerung kann frei gewählt werden (z.B.: Hardware, Algorithmus)**

Die Lösung erlaubt die freie Wahl bei der Steuerung. Es kann sowohl Hardware, wie Datenhandschuhe oder ein Joystick, als auch Software in Form eines Algorithmus genutzt werden.

6. **Steuerung der Experimente durch TBMS**
   Die durchgeführten Experimente werden durch ein TBMS gesteuert. Dazu gehört die Ausführung der Experimente unter wiederholbaren Bedingungen und das Sammeln der Ergebnisse.

2.1 Robotersimulation (v-rep)

Eine Control Domain besteht aus einer Master Domain, welche die Steuerung der Roboter übernimmt und einer Slave Domain, welche den Roboter in Form einer Simulation oder als echten Roboter darstellt.

Ein Vertreter dieses Ansatzes ist v-rep [3], eine Simulationsumgebung für Roboter. v-rep steht hier mit seinem großen Funktionsumfang beispielhaft für viele Robotersimulationen. v-rep ist vielseitig, skalierbar, aber dennoch leistungsstark und universell einsetzbar. Trotz des Umfangs ist v-rep einfach zu bedienen und bietet verschiedene Ansätze, um durch Programme erweitert zu werden.

Es können Sensoren, Mechanismen, Roboter und ganze Systeme modelliert werden. Dabei baut die Software auf drei zentrale Objekte auf: Szenenobjekte, Berechnungsmodelle und Kontrollmechanismen. Bei Szenenobjekten handelt es sich um 14 verschiedene einfache Bauklötze (z.B. Kameras, Lichter, Verbindungen, Formen, Graphen, Abstandssensoren, ...), die miteinander kombiniert werden können. Daraus ergeben sich zusammen mit den Berechnungsmodulen und Kontrollmechanismen komplexe Systeme. Berechnungsmodelle sind 5 grundlegende Algorithmen, die miteinander kombiniert werden können. Diese sind zur Erkennung von Kollisionen, für die Berechnung von Abständen, Pfadplanung, Kinematik und Physics/Dynamics.

Zu den Kontrollmechanismen gehören sechs Methoden und Schnittstellen, um die Simulation zu steuern. Diese Methoden können gleichzeitig genutzt werden und decken insgesamt über 7 verschiedene Programmiersprachen ab [3]. Dazu gehört auch eine remote Application Programming Interface (API), die einen Zugriff aus der Ferne über ein Netzwerk bietet. Allerdings erfüllt diese nicht die Anforderungen an die haptische Kommunikation, da die Frequenz zum Senden der Daten nicht frei gewählt werden kann. Auch ist der entstehende Overhead an Daten nicht klar.

Die Steuerung kann frei gewählt werden. Durch ein Plugin kann hier sogar ein externer Controller angesteuert werden. Da es sich hierbei um eine Software zur Robotersimulation handelt, können keine echten Roboter genutzt werden. Die Simulation verfügt über kein integriertes System zur Durchführung von Experimenten. Damit die Simulationen auch auf schwächerer Hardware ausgeführt werden können, bietet v-rep einen headless Modus, in dem die Simulation nur in einem Terminal ohne das Rendern der Bilder ausgeführt wird. Für die Nutzung an Bildungseinrichtungen existiert eine kostenlose Version: V-REP PRO EDU. Diese bietet einen vollständigen Funktionsumfang ohne Einschränkungen. Ein Vergleich zwischen Robotersimulationen und eine ausführlichere Begründung, warum sich für v-rep entschieden wurde, ist in Kapitel 3.1.6 zu finden.

2.2 Google Robotics

Ein weiteres Projekt im Bereich der Control Domain, mit echten Robotern, wird von Google betrieben [4]. Dabei handelt es sich um ein Labor mit 14 echten Robotern und weiteren Robotersimulationen. Die Ziele sind Robotik durch Machine Learning und Machine Learning durch Robotik zu verbessern. Leider sind zu diesem Projekt nicht viele Informationen über das TBMS verfügbar, da dieses hier nur ein Mittel zum Zweck ist. Möglicherweise existiert hier sogar gar kein TBMS, und die Experimente werden manuell ausgeführt. Allerdings wird klar, dass es hier nicht um das Forschungsgebiet der haptischen Kommunikation geht, sondern darum, wie Roboter lernen. Dabei werden unter anderem Experimente durchgeführt, bei denen Roboter lernen, Objekte zu greifen und diese durch Räume zu befördern [5].

2.3 MIoT-Lab

Im Bereich der Network Domain existiert bereits eine Arbeit im Rahmen eines Testbeds für Experimente mit vermaschten Netzwerken. Das MIoT-Lab ist ein Testbed für Forschungszwecke an der Otto-von-Guericke Universität. Aktuell besteht es aus 35 Knoten und soll in der Zukunft aus mehr als 100 Knoten bestehen, die über den gesamten Universitätscampus verteilt sind. Jeder Knoten besteht aus einem embedded System mit bis zu drei IEEE 802.11 Netzwerkkarten. Die Experimente werden durch das integrierte TBMS gesteuert, welches es erlaubt Experimente im Distributed Embedded Systems (DES)-Cript Format festzulegen.

Der Forschungsmittelpunkt besteht aus dem Vergleich zwischen Simulationen und Experimenten unter den Bedingungen der echten Welt. Dies ist notwendig, da in reinen Simulationen Faktoren, die das drahtlose Medium betreffen, schwer zu modellieren sind. Auch wird das Betriebssystem auf den einzelnen Knoten in der Regel nicht berücksichtigt. Hier spielen unter anderem verschiedene Scheduling Strategien bei Multi-Threading eine Rolle. Mithilfe des MIoT-Labs können so Routing-Algorithmen ausgewertet und verbessert werden. Diese Informationen können zum Beispiel in ländlichen Regionen mit schlechter Infrastruktur hilfreich sein, um dort den Internetzugriff zu verbessern oder erst zu ermöglichen [6].
Da es sich bei dem MIoT-Lab um ein reines Testbed für Netzwerke und Sensorknoten handelt, unterstützt es direkt noch keine Funktionen bezüglich der Simulation oder Steuerung von Robotern.

2.4 Mobile Emulab

Bei dem Mobile Emulab [7] handelt es sich um ein Testbed an der Univeristät von Utah, USA, in dem sich mobile Roboter in einem drahtlosen Netzwerk bewegen. Diese Roboter bestehen aus Sensorknoten und Single Board Computern. Die Software auf den Robotern wird durch den Nutzer ausgewählt. Mit der Software können die Nutzer bestimmen, wie die Roboter aus der Ferne positioniert, die Netzwerkschnittstellen und Computer gesteuert und Daten geloggt werden können. Das Testbed basiert auf der Emulab Testbed Software. Die Nutzung dieser Software ermöglicht einen hohen Grad an Automatisierung. Auch ist das Testbed online erreichbar und kann damit geteilt werden.

Das Ziel des Testbeds ist neue Netzwerkprotokolle, Anwendungen und Systeme zu evaluieren. Inwiefern die Anforderungen an die haptische Kommunikation erfüllt werden, wird aus den zur Verfügung gestellten Informationen nicht deutlich. Es kann die Szene nicht frei gewählt werden, und auch die Roboter können nicht durch eine Simulation ersetzt werden. Dadurch ist das mobile Emulab nicht als allgemeine Testplattform geeignet. Auch ist das Ziel hier ein anderes, denn es geht nicht speziell um die haptische Kommunikation, sondern darum, wie sich Systeme durch drahtlose Netzwerke bewegen.

2.5 Fazit zu Related Work

Es ist deutlich geworden, dass keine Arbeiten bestehen, die den geplanten Umfang dieser Arbeit umfassen. Die bestehenden Arbeiten decken immer nur spezielle Teile ab. Die Tabelle 2.1 zeigt eine Zuordnung der Kriterien aus Auflistung 2.1 zu den näher betrachteten Arbeiten.

Kriterium	*1*	*2*	*3*	*4*	*5*	*6*
v-rep	✓	✗	✗	✗	✓	✗
MIoT-Lab	✗	✗	✗	✗	✗	✓
Google Robotics	❍	❍	✓	✓	✓	❍
Mobile Emulab	✓	❍	✗	✓	✓	✓

Tabelle 2.1: Die Tabelle zeigt die Funktionalität ähnlicher Arbeiten. ✓- Ja, ✗- Nein, ❍- Unklar. Die Nummerierung bezieht sich auf die Kriterien aus der Auflistung 2.1.

KAPITEL 3

Thesis-Beitrag

Das Kapitel widmet sich der Beantwortung der Forschungsfrage. Es wird ein Konzept erstellt, das aufzeigt, wie die Forschungslücke geschlossen wird. Dazu gehört das Ermitteln der Anforderungen und eine Beschreibung, wie diese Anforderungen erfüllt werden. Abschließend werden wichtige Details zur Implementierung des Frameworks erläutert.

3.1 Konzept

Das Ziel ist die Entwicklung eines Frameworks, das Experimente mit Bezug zur haptischen Kommunikation vom Start bis zum Ende umfasst. Dafür wird ein Konzept entwickelt, und die Implementierung für eine neue Software zur Anbindung eines Steuerung(-algorithmus) an eine Roboter(-simulation) durchgeführt. Teilweise wird auch auf bereits bestehende Komponenten zurückgegriffen. Ein beispielhafter Ablauf zur Durchführung eines Experiments in dem Framework mit der Robotersimulation v-rep und dem MIoT-Lab Testbed ist in Abbildung 3.1 zu erkennen. Jedes Experiment beginnt mit Überlegungen über die Modellierung der Roboterszene in v-rep. Dann wird die Roboterszene dort umgesetzt. Anschließend beginnt die Anpassung des Frameworks an diese Szene. Dort müssen die benötigten Daten festgelegt, API-Aufrufe, die Steuerung und Kodierungsalgorithmen implementiert und ein DES-Cript-Skript erstellt werden. Dieses wird in das TBMS geladen und das Experiment ausgeführt. In der Abbildung 3.1 ist auch ersichtlich, welche Bestandteile bereits bestanden und welche im Rahmen der Bachelorarbeit konzeptioniert und implementiert werden.

Bevor die einzelnen Komponenten im Verlauf erläutert werden, folgt eine kurze Erklärung und Übersicht über das allgemeine Konzept. Das Testbed lässt sich konzeptionell in drei Domänen aufteilen: Master Domain, Network Domain und Slave Domain. Die Master Domain stellt den Bereich der Steuerung des Roboters dar. Dazu gehört ein Ein- und Ausgabegerät. Einerseits werden hier die Sensordaten als haptische Signale ausgegeben, aber auch Steuerungsdaten eingegeben. Dies kann ein Joystick, ein Controller oder im einfachsten Fall für Experimente auch ein Steuerungsalgorithmus sein, der die eingehenden Daten verarbeitet und in Abhängigkeit davon Steuerungsdaten ausgibt. Das Ein- und Ausgabegerät ist mit dem Framework verbunden, oder als Algorithmus Teil des Frameworks. Das Frame-

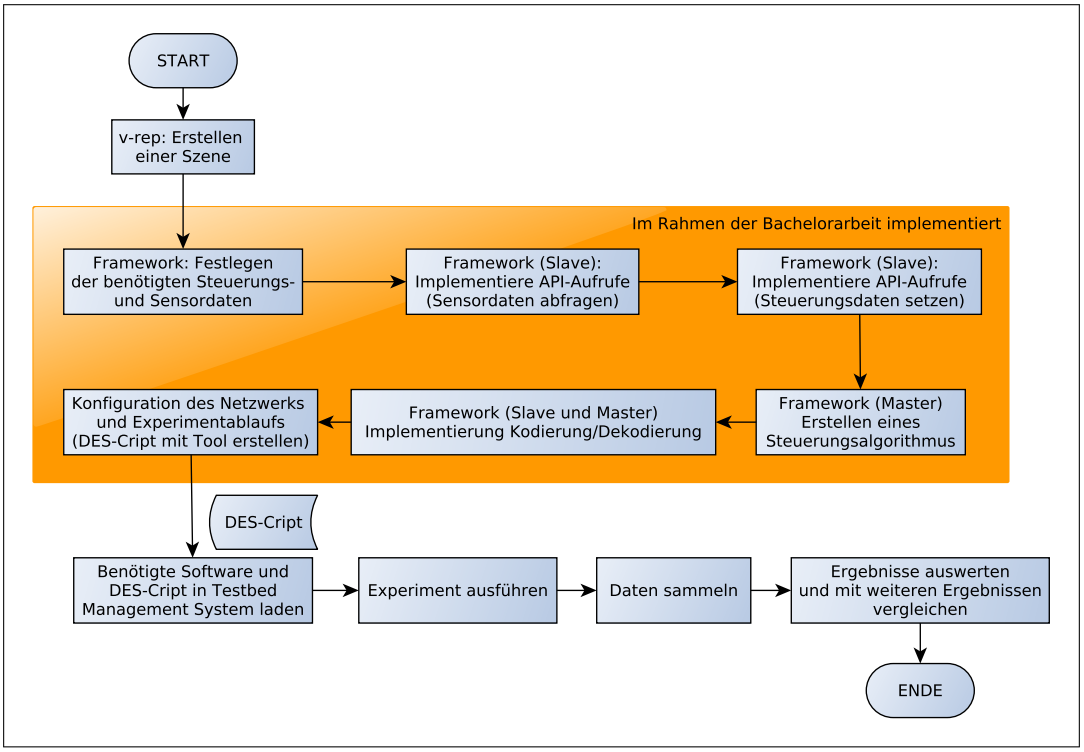

Abbildung 3.1: Möglicher Ablauf eines Experiments (eigene Darstellung).

work sorgt im Bereich der Master Domain für die Kodierung der Daten, den Aufruf der Steuerung(-algorithmus) und die Netzwerkanbindung.

Auf der anderen Seite, im Bereich der Slave Domain befindet sich der Roboter, beziehungsweise eine Robotersimulation. Diese wird über die Steuerungsdaten gesteuert und gibt Sensordaten aus. Dies geschieht über das Framework, welches mit der Master Domain verbunden ist. Im Beispiel der Robotersimulation v-rep geschieht dies über API-Aufrufe. Ebenfalls können im Framework auf der Seite der Slave Domain die Daten kodiert werden. Verbunden werden die Master und Slave Domain über die Network Domain. Hier befindet sich ein Netzwerk, welches frei konfigurierbar und wählbar ist. Insgesamt lassen sich durch den Nutzer so die Steuerung, das Netzwerk und der Roboter frei wählen und das Framework daran anpassen. Eine Übersicht über dieses Konzept gibt die Abbildung 3.2.

3.1.1 Anforderungen an das Framework

Für ein Framework, das eine haptische Kommunikation evaluieren soll, ergeben sich eine Reihe an Anforderungen. Diese Anforderungen basieren größtenteils auf den Anforderungen an das taktile Internet, welche sich wiederum aus den biologischen Hintergründen und der Wahrnehmung des Menschen ergeben. Die Anforderungen werden benötigt, um die QoE für den Nutzer sicherzustellen. Dabei handelt es sich um ein Maß dafür, wie der Nutzer die Qualität einer Anwendung wahrnimmt [8]. Bei der haptischen Kommunikation möchte der Nutzer beispielsweise das Gefühl haben, dass er in die entfernte Umgebung eintaucht und

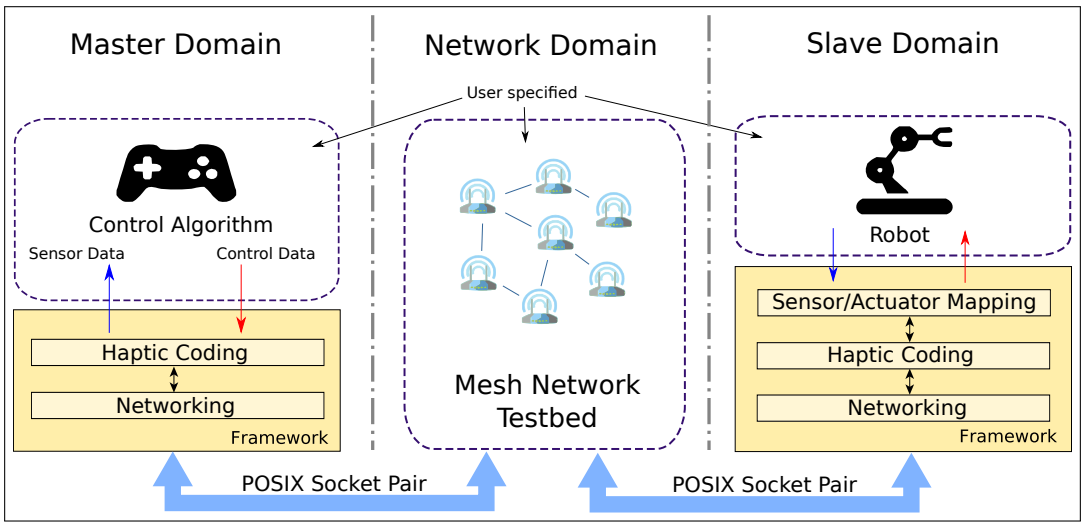

Abbildung 3.2: Überblick über das Konzept: Steuerung, Netzwerk und Roboter sind frei auswählbar (eigene Darstellung).

die gewünschten Aktionen ohne eine Distanz ausführt [9]. Diese Anforderungen werden hier kurz erklärt. Im Laufe der Arbeit wird darauf eingegangen, wie sichergestellt wird, dass die Anforderungen eingehalten werden.

Frequenz

Bei der haptischen Kommunikation werden sehr hohe Frequenzen benötigt. Ursächlich hierfür ist das menschliche haptische Wahrnehmungssystem. Die Tabelle 3.1 zeigt vier verschiedene Typen an mechanischen Rezeptoren. Ersichtlich ist dort, dass die menschliche Haut Frequenzen von bis zu 1 kHz wahrnehmen kann. Um entsprechende Signale kontinuierlich zu übertragen, müssen dementsprechend Frequenzen bis 1 kHz durch das Framework unterstützt werden.

	Merkel-Zellen	*Ruffini-Körperchen*	*Meissner-Körperchen*	*Vater-Pacini-Körperchen*
Bester Stimulus	Druck, Kanten, Ecken, Punkte	Dehnung	Querbewegung	Hochfrequente Vibrationen
Beispiel	Blinden-schrift	Halten großer Objekte	Rutschen von Objekten (Reibung)	Wahrnehmen haptischer Struktur
Frequenzbereich	0-100	/	1-300	5-1000
Sensitivste Freq.	5	/	50	200

Tabelle 3.1: Funktion, Rollen und Frequenzbereiche von vier verschiedenen mechanischen Rezeptoren der menschlichen Haut [10].

Latenz

Im Kern des Designs des taktilen Internets, wovon die haptische Kommunikation ein Teil ist, steht die sogenannte 1 ms-Challenge [11]. Hiermit ist die Latenz zwischen dem Zeitpunkt, zu dem ein Nutzer eine Aktion ausführt und er sie wahrnimmt, gemeint [12]. Hier wurde durch mehrere Studien gezeigt, dass für die haptische Kommunikation ab einer Frequenz von 1 ms die Latenz wahrnehmbar wird [13]. Wird diese Verzögerung spürbar, kann es für den Nutzer zu unerwünschten Effekten kommen, wie *Cybersickness*. Werden Systeme gesteuert, die möglichst schnell eine aktuelle Antwort benötigen, kann es auch hier zu Problemen kommen.

Einer der Hauptfaktoren für die entstehende Latenz liegt in Übertragungslatenzen in dem Netzwerk. Und dort soll der Großteil der Latenz auch weiterhin entstehen und nicht bereits durch die Nutzung des Frameworks. Weitere Faktoren für die Entstehung von Latenzen bei der Kommunikation sind Warteschlangenbildung, Verarbeitung, Codierung und API-Aufrufe für die Verbindung mit einem Controller oder einem Endgerät. Die Größe der Pakete spielt hier auch eine Rolle. Denn je größer die Pakete sind, desto länger dauert auch die Verarbeitung, Codierung und Übertragung der Daten [9]. Realistisch ist eine Latenz von unter 1 ms nur auf kurzen Distanzen. Hier ist die Ausbreitungsgeschwindigkeit des Lichts eine physikalische Grenze. Alleine dadurch entsteht beispielsweise beim Senden eines Pakets auf der Distanz zwischen Amsterdam und New York eine Latenz von 20 ms [9]. Die Vierte Generation im Mobilfunk (4G) hat auf kurzen Distanzen typischerweise eine Latenz von 15 ms [14] und die Fünfte Generation im Mobilfunk (5G) verspricht eine Latenz von unter 5 ms [15].

Zuverlässigkeit

Die Wichtigkeit der Zuverlässigkeit der Übertragung ist abhängig von der Anwendung. Das menschliche Gehirn ist in der Lage fehlende und fehlerhafte Informationen bis zu einem bestimmten Grad auszugleichen beziehungsweise mit visuellen Signalen zu ergänzen. Auch ist das Gehirn dazu in der Lage den Schwerpunkt auf die zuverlässigeren Informationen zu legen. Problematisch ist hier nur, dass das Gehirn ab einem bestimmten Punkt an falschen Informationen nicht mehr weiß, welche Information die Wahre ist. Unter der hohen Belastung des Gehirns leidet im Weiteren auch die QoE [2]. Weiterhin kann jede fehlerhaft empfangene Information potentiell zu einer Fehlentscheidung des Nutzers auf die entfernte Umgebung führen. Gerade bei kritischen Umgebungen ist dies nicht zu akzeptieren, da dort jeder Fehler zu großen Problemen führt [9].

3.1.2 Erweiterbarkeit

Das Framework soll ein modulares Design aufweisen, leicht erweiterbar und anpassbar sein. Dadurch kann es vielfach wiederverwendet werden, und die eingesparte Zeit in die Anwendungsfälle und Experimente investiert werden. Dazu gehört, dass auf der Seite des Slaves ohne großen Aufwand unterschiedliche Simulationen und auch echte Roboter genutzt werden können. Um dies zu erreichen, wird die Kommunikation in einzelne Softwarebestandteile ausgegliedert. Diese müssen später nicht mehr verändert werden. So muss beim Wechsel der Slaveanwendung nur noch die Logik verändert werden, die die zu übertragenden Daten in

die entsprechenden Strukturen ablegt und aus den Strukturen auf den Roboter umsetzt. Die Strukturen werden in Form von Pointern übergeben. Konkurrierende Zugriffe auf die Daten werden durch die Verwendung von Mutual exclusion (Mutex) verhindert. Diese Prinzip wird in der Abbildung 3.3 verdeutlicht.

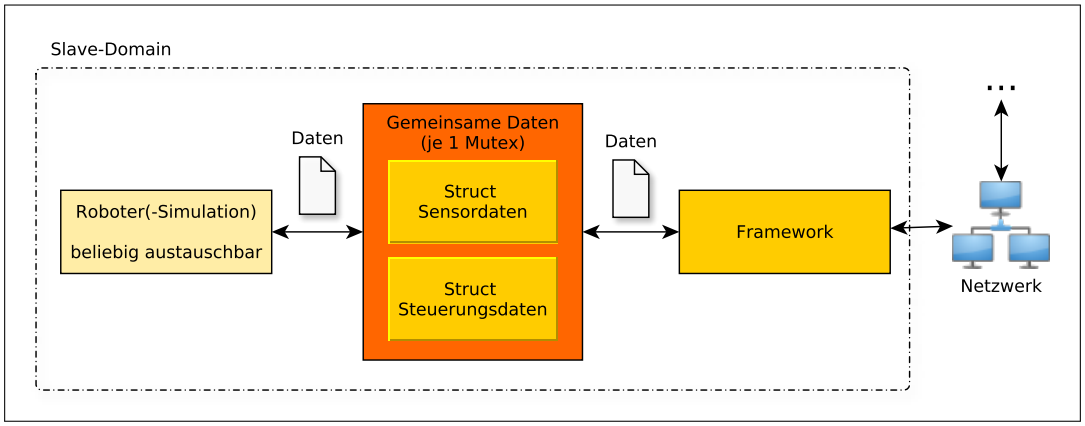

Abbildung 3.3: Datenaustausch zwischen Roboter und Framework im Bereich des Slaves. Durch die Verbindung über die gemeinsamen Datenstrukturen wird die Austauschbarkeit des Roboters gewährleistet (eigene Darstellung).

Auf der Seite des Masters soll die Software zur Steuerung frei festlegbar und austauschbar sein. Es soll auch die Möglichkeit bestehen Hardware anzuschließen, zum Beispiel Datenhandschuhe oder Controller. Hier werden die Daten ebenfalls über gemeinsame Datenstrukturen zwischen den Kommunikationsbestandteilen des Frameworks und der eigentlichen Steuerung ausgetauscht. Soll Hardware zur Steuerung und Ausgabe von Signalen genutzt werden, so muss anstelle eines Algorithmus eine Software zur Ansteuerung dieser Hardware genutzt werden.

Das Netzwerk soll frei wählbar sein. Hier wird eine Kompatibilität durch die Nutzung von POSIX-Sockets gewährleistet. Damit die Software auf vielen verschiedenen Systemen lauffähig ist, wird generell auf die Nutzung von POSIX-kompatiblen Systemfunktionen geachtet. Um die Robotersimulationen auszuführen, darf die Hardware allerdings nicht zu schwach sein. Im Entwicklerforum von v-rep werden von einem der Entwickler beispielsweise folgende grobe Anforderungen genannt: "anständige" Grafikkarte, hoher Central Processing Unit (CPU)-Takt, 4 CPU-Kerne [16]. Für diese Anwendung müssen beispielsweise dem MIoT-Lab weitere leistungsfähigere Testbedknoten hinzugefügt werden.

3.1.3 Freie Spezifikation von Applikationen

Um eine freie Spezifikation von Applikationen zu ermöglichen und das Framework trotzdem allgemeingültig und einfach zu halten, werden als Obermenge ausschließlich Fließkommawerte übertragen. Unterschiedliche Datentypen hätten eine größere Komplexität verursacht. Beim Übertragen von digitalen Werten, zum Beispiel bei einem Kollisionssensor (0 - keine Berührung, 1 - Berührung) entsteht hier ein gewisser Overhead an Daten. Häufig handelt es sich bei den zu übertragenden Daten (z.B.: Kräfte, Positionen, Rotationen, Beschleu-

nigungen) jedoch um Werte von analogen Sensoren, die auf Fließkommazahlen abgebildet werden.

3.1.4 Kodierung

Bei der haptischen Kommunikation ist eine hohe und gleichmäßige Frequenz wichtig. Bereits geringe Latenzen können die Steuerung destabilisieren und die Performance beeinträchtigen. Durch die hohe Frequenz wird allerdings der Kommunikationskanal stark beeinträchtigt, was zu Problemen bei der Steuerung führen kann. Um diesem Problem entgegenzuwirken, können verschiedene Kodierungsverfahren genutzt werden. Speziell für die haptische Kommunikation wurde hier bereits viel Forschungsarbeit betrieben. Zum Beispiel kann eine wahrnehmende Kodierung genutzt werden. Hier gibt es den Deadband-Ansatz. Diese Kodierung macht sich zunutze, dass die hohe Frequenz nicht immer benötigt wird. Es gibt immer wieder Zeitpunkte, zu denen keine oder nur eine geringfügige Änderung der Daten vorliegt. Zu diesen Zeitpunkten kann das Senden der Daten übersprungen werden und erst nach einer größeren Änderung wieder fortgesetzt werden [17].

Bei der Kodierung von haptischen Daten ergeben sich im Gegensatz zur Kodierung von Audio- und Videosignalen neue Herausforderungen. Der Trick bei der Kodierung ist es, dem Nutzer eine gute Erfahrung zu bieten und gleichzeitig bei der Datenübertragung sparsam zu sein: Dafür werden die Grenzen des menschlichen haptischen Wahrnehmungssystems ausgenutzt. Es müssen nicht mehr Daten übertragen werden, als der Mensch überhaupt wahrnehmen kann. Es reichen gerade so viele Daten, dass der Nutzer nicht merkt, dass etwas fehlt. Bei Bildern reichen beispielsweise 30 Bilder pro Sekunde, damit eine fließende Bewegung erkannt wird. Für eine einfache Sprachtelefonie reichen ca. 4 kHz Samplingrate. Für die haptische Kommunikation fehlen solche festen Grenzwerte. Die Herausforderung bei der haptischen Wahrnehmung im Gegensatz zum Hören und Sehen ist, dass sie über den ganzen Körper verteilt stattfindet. Hinzu kommt eine große Bandbreite an verschiedenen möglichen Signalen, wie Druck, Reibung und Vibrationen. Es stellt allein eine große Herausforderung dar, Grenzen für die einzelnen Bestandteile zu finden. Es müssen die verschiedenen Signale jedoch gemeinsam betrachtet werden, um ein uneingeschränktes Benutzererlebnis zu erreichen [18].

Beide Seiten des Frameworks bieten die Möglichkeit, die übertragenen Daten vor dem Senden zu kodieren und die empfangenen Daten zu dekodieren. Dem Nutzer werden dabei durch den vollen Zugriff auf die zu übertragenen Daten und die zuletzt übertragenen Daten viele Möglichkeiten und Freiheiten gelassen.

3.1.5 Datenübertragung

Die Steuerungs- und Sensordaten werden komplett ohne jeglichen zusätzlichen Overhead über das User Datagram Protocol (UDP) übertragen. Zu Beginn der Kommunikation wird lediglich von der Steuerungssoftware eine Initialisierungsnachricht solange gesendet, bis eine Antwort empfangen wird. Die Initialisierungsnachricht beinhaltet den Namen der Szene. Der Name wird auf der Seite der Robotersimulation mit der dort gewählten Szene verglichen. Stimmt dieser Name überein, wird dies bestätigt. Danach ist die Steuerungssoftware bereit für den Empfang von Sensordaten. Die Entscheidung für UDP wurde getroffen, da bei

haptischer Kommunikation ein großer Wert auf eine geringe Latenz gelegt wird, wohingegen der Verlust eines einzelnen Pakets bei der hohen Frequenz häufig keine sehr große Rolle spielt. UDP ist ein verbindungsloses Protokoll, das keine Checks bezüglich des Versands der Nachrichten durchführt. Dadurch kann auch der Header klein gehalten werden. Dies ermöglicht die gewünschte niedrige Latenz. Auch finden keine erneuten Übertragungen statt, was hohe Variationen in der Latenz verhindert [19]. Ebenfalls ist UDP ein gebräuchliches Protokoll für Anwendungen im taktilen Internet. Transmission Control Protocol (TCP) als verbindungsorientiertes Protokoll ist hier nicht geeignet. Der Overhead durch den größeren Header summiert sich bei der hohen Frequenz schnell auf. Auch führt der TCP-Algorithmus zu einem Jitter der Latenz.

3.1.6 Bereits existierende Komponenten

Für die Arbeit wurden mehrere bereits existierende Komponenten benutzt. Einerseits wurde für die beispielhafte Implementierung mit einer Robotersimulation der Simulator v-rep genutzt. Diese Robotersimulation wurde bereits im Kapitel 2 vorgestellt. Die Entscheidung für diese Robotersimulation wurde aus mehreren Gründen getroffen, die hier kurz erläutert werden. Der ausschlaggebende Grund war, dass v-rep sechs verschiedene Programmieransätze bietet, um die Software anzupassen und zu erweitern. Dabei ist v-rep trotzdem kompatibel zu mehreren Plattformen (MacOS, Linux und Windows). Die Software ist jeweils bereits als Binärpaket erhältlich und kann so über das Testbed auf die Simulationsknoten verteilt werden. v-rep erlaubt die leichte Erstellung von austauschbaren Inhalten. So reicht es, dass eine einzige Datei weitergegeben wird, welche das komplette funktionierende Modell inklusive des Steuerungscodes beinhaltet. So muss für die Durchführung der Experimente neben dem DES-Cript-Skript, der v-rep-Szene und gegebenenfalls dem v-rep-Binärpaket nicht mehr auf die Knoten verteilt werden. Weiterhin ist die Bedienung von v-rep leicht zu erlernen und es existiert eine große Community mit vielen Anleitungen und Tutorials [20].

Alternative Robotersimulationen sind hier beispielsweise Gazebo[1] und ARGoS[2]. Diese bieten den Vorteil, dass die Szenen nicht in einem speziellen Format, sondern als Extensible Markup Language (XML)-Datei abgespeichert werden. Dies würde ermöglichen, dass die Szenen durch Skripte zwischen den Simulationen angepasst werden. Allerdings bieten diese Simulationen durch eine kleinere und schlechter dokumentierte API weniger Möglichkeiten einer Erweiterung für die externe Kommunikation. ARGoS bietet nur eine kleine Community, die Entwicklung ist dazu unregelmäßig und Updateanfragen benötigen eine lange Zeit [21]. Ein genauerer Vergleich ist der Tabelle 3.2 zu entnehmen.

Als Netzwerkkomponente und für die Steuerung der Experimente fiel die Entscheidung auf das MIoT-Lab Testbed, welches ebenfalls in Kapitel 2 vorgestellt wurde. Die Entscheidung war einfach, da dieses Testbed eine Reihe an Vorzügen für diese Arbeit bietet. Das TBMS nimmt die Steuerung und Beschreibung der Experimente ab. So muss diese Funktionalität nicht direkt in das Framework implementiert werden. Die Experimente können über die offene Sprache DES-Cript, welche XML-basiert ist, erstellt und modifiziert werden. Auch

[1] `http://gazebosim.org/`
[2] `http://argos-sim.info/`

	v-rep	*Gazebo*	*ARGoS*
Betriebssystem	MacOS, Linux, Windows	MacOS, Linux, Windows	MacOS, Linux
Binärpaket	alle Plattformen	Linux Debian	Linux
Dateiformat	Proprietär	XML	XML
Erweiterb.	Plugins, Scripts, ROS Interface, Add-ons, Remote API	kompilierte Plugins, ROS keine Scripts	Lua Scripts Plugins
Dokumentation	gute API Dokumentation, viele Beispiele, Tutorials	umfassende Doku, Tutorials, Beispiele (nicht alle funkt.)	gute Dokumentation, RobDokumentation lückenhaft
Updates	regelmäßig	regelmäßig	unregelmäßig
Community	groß	groß	klein
Lizenz	kommerziell freie Educational Lizenz	Apache-Lizenz	MIT-Lizenz

Tabelle 3.2: Die Tabelle zeigt einen Vergleich zwischen den Robotersimulationen v-rep, Gazebo und ARGoS[21].

ist eine einfache Verteilung der für das Experiment benötigten Software über das System möglich.

3.1.7 Slave Domain

Auf der Seite der Slave Domain muss eine Verbindung zwischen dem Framework und der Roboter(-simulation) hergestellt werden. Einerseits müssen die Steuerungsdaten umgesetzt und andererseits die Sensordaten abgefragt werden. Am Beispiel der Robotersimulation v-rep ergibt sich hier die Herausforderung, dass für den Zugriff auf die API ein Plugin erstellt werden muss. Dieses Plugin wird beim Start geladen und bietet mehrere Einstiegsprozeduren. Dies wird in Kapitel 3.2.1 genauer erklärt. API-Aufrufe zum Umsetzen und Abfragen von Daten sind nur in Folge von Prozeduraufrufen möglich. Aus einem in dem Plugin gestarteten Thread ist dies allerdings nicht erlaubt [22]. Dies hätte zur Folge, dass sich die Frequenz zum Senden von Datenpaketen nach dem Aufruf der Einstiegsprozeduren richtet und nicht konstant gehalten werden kann.

Um trotzdem eine Kommunikation mit einer festen Sendefrequenz, unabhängig von dem Empfang von Simulationsnachrichten zu ermöglichen, wurden die hierfür benötigten Programmteile in einzelne Threads ausgegliedert. Ein weiterer Grund hierfür ist die dadurch erreichbare leichte Anpassungsfähigkeit an andere Roboter(-simulationen). Ein Thread ist für das Senden der Sensordaten verantwortlich und ein weiterer Thread für das Empfangen der Steuerungsdaten.

Die Sensor- und Steuerungsdaten befinden sich je in einem eigenen Struct, der beim Starten der Threads als Referenz an diese weitergegeben wird. Gleichzeitig wird je ein Mutex mitgegeben, der vor dem Nutzen der Structs gesperrt wird. Dadurch können konkurrierende Zugriffe auf Daten verhindert werden.

Jetzt können in dem Hauptthread (nach einem Prozeduraufruf durch v-rep) die Sensordaten ausgelesen und die Steuerungsdaten umgesetzt werden. Die Sensordaten können aber trotzdem unabhängig davon in einem regelmäßigen Intervall versendet werden. Dabei ist aber nicht garantiert, dass bei jedem Senden der Daten auch wirklich neue Daten verschickt werden.

Das Sequenzdiagramm 3.4 zeigt einen Überblick über den Aufbau und Ablauf der Implementierung. Zu beachten ist, dass das Diagramm den Ablauf nur vereinfacht zeigt. Es wird davon ausgegangen, dass der Slave und Master bereits gestartet worden sind. Der Datenaustausch innerhalb des Plugins ist für die Übersichtlichkeit nicht modelliert worden. Der Zustand der Simulation und die Information darüber, ob die Threads beendet werden sollen, wird den Threads über Atomics mitgeteilt.

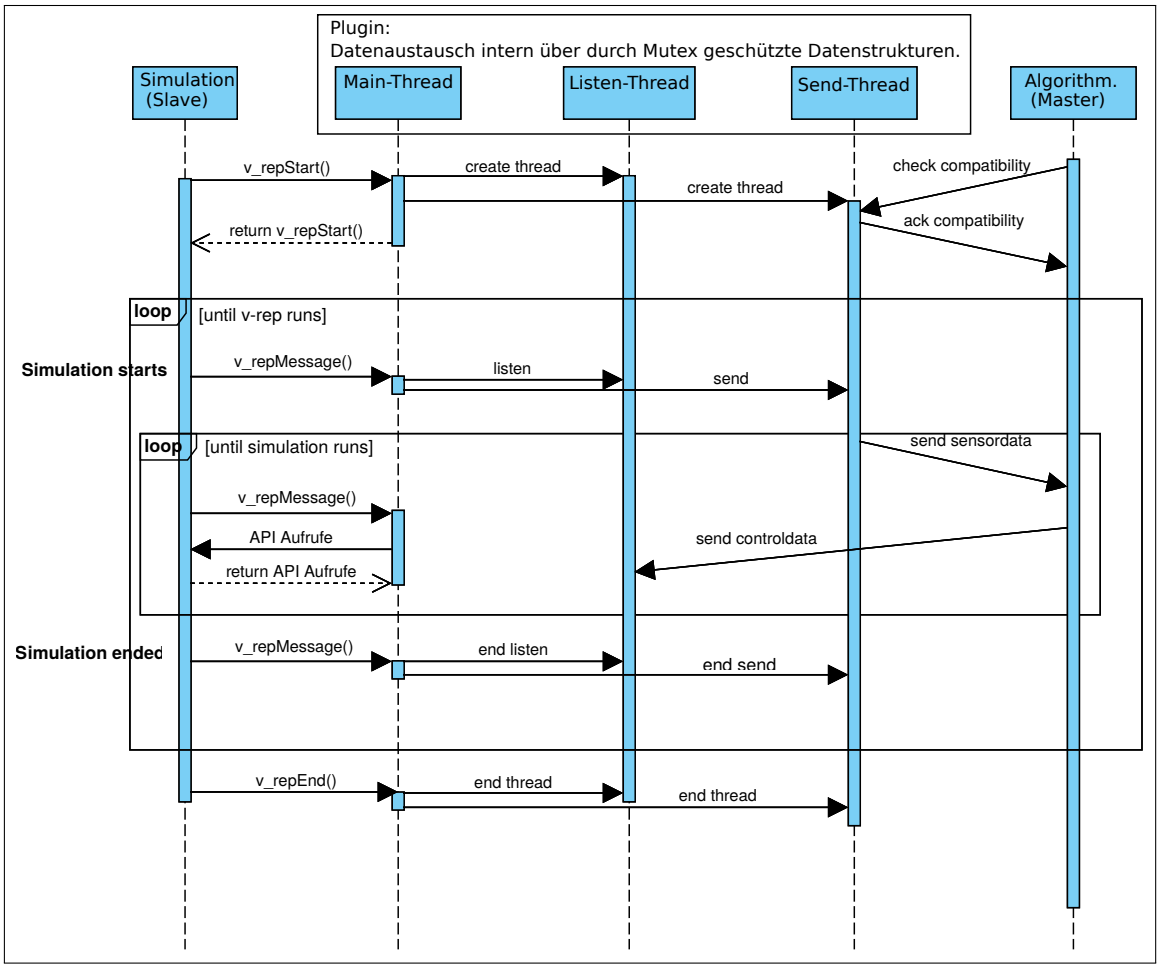

Abbildung 3.4: Das Sequenzdiagramm gibt einen Überblick über den Aufbau und Ablauf der Implementierung (eigene Darstellung).

3.1.8 Master Domain

Das Framework im Bereich der Master Domain ist ähnlich aufgebaut. Hierbei handelt es sich jedoch um eine alleinstehende Anwendung und nicht um ein Plugin. Aber auch hier besteht eine Trennung zwischen der Ausführung der Steuerung und dem Kommunikationsteil der Software. Dies erleichtert die Anpassung an andere Szenarien. Auch die Verbindung zwischen dem Framework und der Steuerung besteht hier über Datenstrukturen. Allerdings wird hier kein Multithreading benötigt, da sich die Frequenz nach den empfangenen Paketen richtet: Immer wenn Sensordaten empfangen werden, wird die Steuerung ausgeführt und anschließend werden die Steuerungsdaten zurückgesendet. Der Programmablaufplan in Abbildung 3.5 zeigt den Ablauf am Beispiel der Robotersimulation, die über einen Steuerungsalgorithmus gesteuert wird.

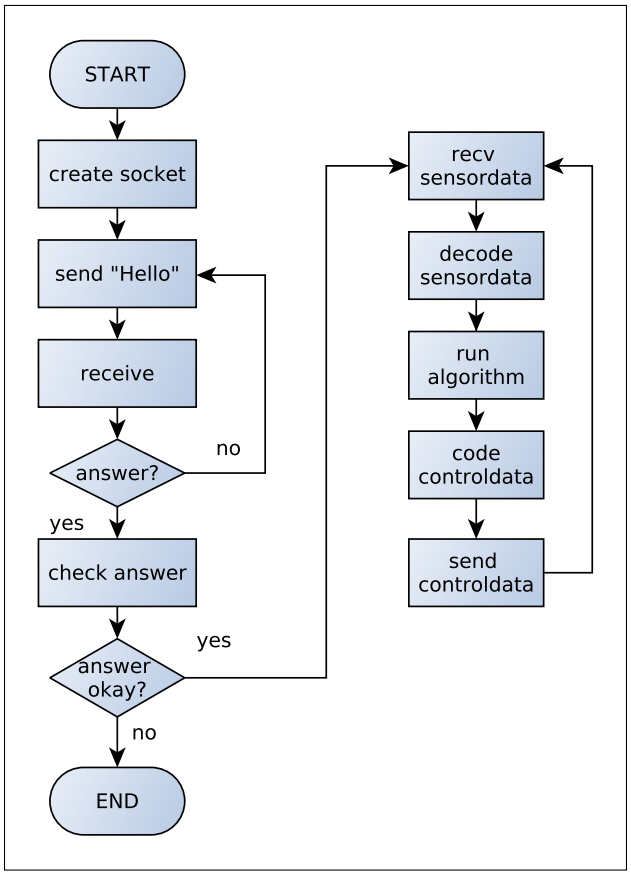

Abbildung 3.5: Zeigt den Weg des Client-Programms für einen Steuerungsalgorithmus (eigene Darstellung).

3.1.9 Network Domain

Das verwendete Netzwerk im Bereich der Network Domain kann prinzipiell frei gewählt werden. Diese Arbeit beschäftigt sich tiefer mit dem MIoT-Lab als Netzwerk und zur Steue-

rung der Experimente. Daher wird im Folgenden die Verwendung des MIoT-Labs und der Skriptsprache DES-Cript genauer erläutert.

MIoT-Lab

Experimente mit der Roboter(-simulation) sollen über das MIoT-Lab möglich sein. Um eine exakte Beschreibung der Experimente zu ermöglichen und die Experimente damit wiederholbar und vergleichbar zu gestalten, existiert die auf XML aufgebaute Skriptsprache DES-Cript. Ein DES-Cript-Skript wird als Eingabe für das TBMS genutzt. Das DES-Cript-Skript kann bei der Erstellung eines Experiments hochgeladen und anschließend über ein Webinterface weiter bearbeitet werden. Aufbauend darauf kann das TBMS das Experiment automatisiert ausführen und dafür sorgen, dass unter anderem das Timing zuverlässig eingehalten wird. DES-Cript wurde so entwickelt, dass es nicht nur für das MIoT-Lab funktioniert, sondern auch auf anderen Experimentierplattformen angewendet werden kann [23].

Aufbau des DES-Cript-Skripts

Ein DES-Cript-Skript ist in zwei große Bereiche aufgeteilt. Der erste Bereich nennt sich *general* und enthält Konfigurationen für das Testbed und allgemeine Informationen über das durchzuführende Experiment. Dazu gehören unter anderem der Name und eine Beschreibung für das Experiment, ein Startzeitpunkt und die Anzahl an Wiederholungen [23]. Auch werden in diesem Bereich die einzelnen Testknoten in verschiedene Gruppen organisiert. Jede Gruppe besitzt einen Namen, eine Rolle (*Server*, *Servent*, *Client*) und beliebig viele Mitglieder. Für das Testframework gibt es eine große Gruppe, die alle Knoten enthält, die einen Routingalgorithmus ausführen sollen. Diese Knoten besitzen die Rolle *Servent*. Für jeden Knoten, der die v-rep-Simulation ausführt, gibt es je eine Gruppe mit der Rolle *Server*. Für jeden Knoten, der die Steuerung übernehmen soll, gibt es je eine Gruppe mit der Rolle *Client*. Diese Aufteilung in die vielen Gruppen ist nötig, um mehrere Simulationen parallel auszuführen. Befehle werden immer Gruppen zugewiesen und dadurch können den einzelnen Simulationen verschiedene Startparameter übergeben werden.

Der zweite große Bereich nennt sich *actions* und enthält alle Befehle, die während des Experiments ausgeführt werden. Innerhalb des *actions* Bereichs gibt es wiederum einzelne Bereiche, die sich *action_block* nennen. Diese Blöcke enthalten wiederum die Befehle, die tatsächlich auf den Knoten ausgeführt werden. Für jeden *action_block* kann ein Ausführungsmodus festgelegt werden. Zur Verfügung stehen hier unter anderem *Server*, *Serial* und *Parallel*. Als erster *action_block* wurde ein serieller Block gewählt, der für die Initialisierung der Knoten zuständig ist. Hier wird das Archiv entpackt, das alle für das Experiment benötigten Dateien inklusive des v-rep-Simulators enthält. Daraufhin wird auf allen Knoten der Routingalgorithmus gestartet.

Für jede v-rep-Simulation, die ausgeführt werden soll, existiert ein eigener *action_block*. Diese Action-Blöcke werden parallel ausgeführt und starten v-rep mit den entsprechenden Startparametern, die an das Plugin weitergegeben werden. Dies sind der Port und die gewünschte Sendefrequenz. Als Gruppe wird die für den gewünschten Knoten zuvor angelegte Gruppe ausgewählt. Auch wird hier ein Evaluationsskript festgelegt. Hier wird das Skript *global:save-stdout.py* genutzt. Das Skript speichert alle Ausgaben, die über die Standard-

ausgabe getätigt werden. Für den Steuerungsalgorithmus wird ebenfalls je ein *action_block* erstellt. Hierin wird allerdings der Steuerungsalgorithmus gestartet. Die Internet Protocol (IP)-Adresse muss nicht direkt angegeben werden. Diese kann über String Interpolation von dem Testbed über den Knoten- und Interfacenamen festgelegt werden. Der Port muss allerdings festgelegt werden. Ebenfalls wird hier das Skript *global:save-stdout.py* genutzt. Eine vereinfachte Darstellung des Aufbaus findet sich in Abbildung 3.6.

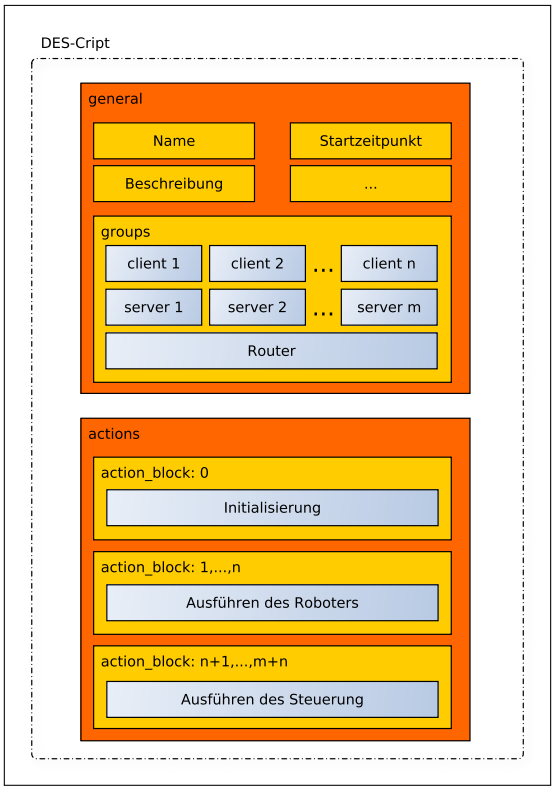

Abbildung 3.6: Integration eines Experiments in DES-Cript. Die Darstellung ist stark vereinfacht und zeigt nicht alle Attribute und Elemente (eigene Darstellung).

Automatische Erstellung des Experiments

Um dem Nutzer des Testframeworks die Erstellung des DES-Cript-Skripts zu erleichtern, wurde ein Pythonprogramm geschrieben, das ein DES-Cript-Skript mit wenigen Eingaben automatisch erstellt. Das ist möglich, da sich die Experimente auf der Basis des DES-Cript-Skripts nur gering unterscheiden. Hier geht es hauptsächlich um die Zuordnung der Knoten zueinander. Es ist möglich, dass auf einem Knoten sowohl mehrere Steuerungsalgorithmen, als auch v-rep-Simulationen ausgeführt werden. Dabei steuert aber ein Controller immer nur einen Roboter, und ein Roboter wird von genau einem Controller gesteuert. Zur Konfiguration lädt das Tool eine Config-Datei. Diese enthält den Namen, eine Beschreibung, den Startzeitpunkt, die Anzahl an Wiederholungen und die gewünschte Sendefrequenz. Auch werden hier die Router festgelegt, sowie Zuordnungen von Serverknoten zu Clientknoten,

jeweils inklusive des genutzen Ports. Eine Konfiguration, um ein DES-Cript-Skript zu erstellen findet sich in dem Quellcode 3.1.

```
1  name:V-REP Simulation
2  description:Ein Experiment mit v-rep Simulation
3  start:2019-07-16 10:00:00
4  iterations:1
5  freq:50
6  #### router ####
7  router1
8  router2
9  router3
10 router4
11 #END_ROUTER
12 #### SERVER  CLIENT   PORT ####
13 server_node1 client_node1 8080
14 server_node1 client_node2 8081
15 server_node2 client_node2 8080
16 server_node2 client_node2 8282
```

Quellcode 3.1: Config zur automatischen Erstellung eines DES-Cript-Skripts

3.2 Implementierung

Der Abschnitt widmet sich erwähnenswerten Implementierungsdetails, der für die Bachelorarbeit notwendigen Softwarebestandteile unter der Verwendung der Robotersimulation v-rep. Die komplette Implementierung ist in dem zur Bachelorarbeit gehörenden git[3] und auf der beiliegenden CD-ROM zu finden.

3.2.1 v-rep-Plugin (Slave Domain)

Zur Steuerung der Robotersimulation v-rep stehen mehrere Schnittstellen zur Verfügung. Dazu gehören unter anderem die *regular-API*, zwei *remote-APIs* und ein *Robot Operating System (ROS)-Interface*. Der Zugriff auf die *regular-API* ist nur innerhalb des v-rep-Frameworks möglich. Auf die *remote-APIs* kann bereits aus externen Anwendungen zugegriffen werden [22]. Trotzdem wurde sich für die Nutzung der *regular-API* und der damit verbundenen Entwicklung eines Plugins entschieden. Dies liegt darin begründet, dass das Testframework so einfach wie möglich und mit einem möglichst geringen Overhead gehalten werden soll. Des Weiteren ist so sichergestellt, dass die durchgeführten Tests nicht durch die *remote-API* beeinflusst werden. Diese bringt im Gegensatz zur *regular-API* bereits einen Kommunikationslag mit sich. Auch wird dadurch ein größerer Funktionsumfang ermöglicht. Die *regular-API* besteht aus mehr als 500 Funktionen, wohingegen die *remote-API* nur ca. 100 Funktionen bereitstellt. Eine Herausforderung ist, dass das Plugin keine asynchrone Ausführung erlaubt. Es dürfen zwar mehrere Threads gestartet werden, ein API Aufruf ist allerdings nur aus dem Main-Thread erlaubt [24].

Ein v-rep-Plugin ist eine shared library. Diese muss sich im gleichen Ordner, wie v-rep befinden und wird beim Start von v-rep automatisch geladen. Unter Linux muss folgende Namenskonvention eingehalten werden: libv_repExtXXXX.so. Dabei ist XXXX der Name

[3] `https://code.ovgu.de/comsys-group/students/bsc/2019-behrens-johannes`

des Plugins. Dieser muss mindestens vier Zeichen lang sein und darf keinen Unterstrich beinhalten. Ein Plugin muss die drei Prozeduren aus Quellcode 3.2 als Einstiegspunkte implementieren [25], welche im Folgenden genauer erläutert werden.

```
1 extern "C" __declspec(dllexport) unsigned char v_repStart(void* reserved,int
      reservedInt);
2 extern "C" __declspec(dllexport) void v_repEnd();
3 extern "C" __declspec(dllexport) void* v_repMessage(int message,int*
      auxiliaryData,void* customData,int* replyData);
```

Quellcode 3.2: Einstiegsprozeduren

v_repStart

Diese Prozedur wird einmalig nach dem Laden des Plugins aufgerufen. Hier können alle Schritte durchgeführt werden, bei denen sichergestellt sein muss, dass sie vor dem weiteren Programmverlauf ausgeführt werden. Dazu gehört zum Beispiel der Start von Threads oder das überprüfen, ob die richtige v-rep-Version vorliegt [25].

v_repEnd

Diese Prozedur wird einmalig beim Beenden von v-rep aufgerufen. Sie kann genutzt werden, um Threads zu beenden, Speicher freizugeben oder Auswertungen der Testdaten durchzuführen und diese auszugeben [25].

v_repMessage

Diese Prozedur wird während der Simulation immer wieder aufgerufen. Sie ist dafür da, um Nachrichten aus der Simulation zu empfangen und auf sie zu reagieren. Anhand der Nachricht kann ermittelt werden, um welches Event es sich handelt. Dazu gehören Events, wie der Start oder das Ende einer Simulation, das Laden von Szenen, aber auch regelmäßige Events, die bei jedem Simulationsschritt aufgerufen werden. Diese Aufrufe können dann wiederum genutzt werden, um Befehle mit der *regular-API* auszuführen [25].

C++-Framework

Um bei dem Plugin auf eine vorhandene Basis aufzubauen, wurde ein bestehendes C++-Framework benutzt [26]. Dieses ist nach dem objektorientierten Paradigma aufgebaut und implementiert bereits die benötigten Prozeduren. Außerdem werden die verschiedenen Nachrichtentypen bereits aufgeschlüsselt, sodass je Event eine eigene Methode bereitsteht, die beim Eintritt des Events aufgerufen wird. Um das Testframework trotzdem übersichtlich zu halten, wurde eine neue Klasse erstellt, die von dem C++-Framework erbt und nur die Methoden implementiert, die für die Steuerung aus der Ferne auch wirklich benötigt werden.

3.2.2 Steuerung (Client)

Die Software zur Steuerung der Simulation ist alleinstehend. Bei dieser Anwendung wird nur ein Thread benötigt, da direkt auf die empfangenen Nachrichten reagiert werden muss. Auch hier wurde eine objektorierentierte Programmierung in C++ gewählt. Für den Algorithmus wird eine neue Instanz erzeugt. Dieses Vorgehen ermöglicht, in den Klassenvariablen für den Algorithmus, Daten über die einzelnen Ausführungen hinweg zu speichern. Gerade für Algorithmen, die bestimmte Zustände, wie Zeiten, speichern müssen, ist dies hilfreich. Beim Start der Steuerungssoftware können der Hostname und Port der Robotersimulation als Parameter übergeben werden. Wenn die Software an einen Algorithmus angepasst wird, muss nur in der Datei *Algorithm.cpp* der Algorithmus innerhalb der Funktion *void Algorithm step(struct ctrlData &robCtrl, struct sensData &robSens)* implementiert werden.

Die Software nimmt die Sensordaten entgegen, berechnet mit dem vorgegeben Algorithmus die entsprechenden Steuerungsdaten und sendet diese zurück an die Simulation. Wenn eine Kodierung implementiert wurde, werden die Daten nach dem Empfangen dekodiert und vor dem Senden kodiert.

3.2.3 Kodierung

Sowohl für den Master, als auch für den Slave existieren jeweils die Funktionen *encode(...)* und *decode(...)*.

Die Funktion *encode(...)* nimmt einen Pointer auf den ursprünglichen Buffer inklusive der Größe der dort gespeicherten Daten entgegen. Ein weiterer Pointer zeigt auf den Speicher, in dem die kodierten Daten abgelegt werden sollen. Auch existiert eine Referenz auf eine Variable, die die Größe der Daten nach der Kodierung angibt und innerhalb der Funktion gesetzt werden soll. Wird der Wert dieser Variablen auf *0* gesetzt, so werden in diesem Schritt keine Daten gesendet. Dies kann bei Kodierungen hilfreich sein, die Daten nur versenden, wenn eine bestimmte Änderung existiert. Um diese Änderung festzustellen, wird ebenfalls ein Pointer auf den Speicher der zuletzt versendeten Daten übergeben.

Die Funktion *decode(...)* nimmt die empfangenen Daten über einen Pointer auf den Buffer entgegen. Gegebenenfalls werden die Daten verändert und an eine Speicherposition gespeichert, die durch einen weiteren Pointer übergeben wird. Auch hier wird die Größe vor der Dekodierung übergeben und die Größe nach der Dekodierung gesetzt.

3.2.4 Zeitliche Abstimmung

Eine Besonderheit der haptischen Kommunikation ist die sehr hohe Sendefrequenz. Damit diese hohen Frequenzen eingehalten werden können, spielt das Timing innerhalb des v-rep-Plugins eine entscheidende Rolle. Um diese Frequenzen zuverlässig gewährleisten zu können, wurde die POSIX-Uhr *CLOCK_MONOTONIC* gewählt. Dieser Zeitgeber ist unabhängig von der Systemuhr und vollzieht keine Zeitsprünge. Er ist nicht von außen setzbar und misst die absolute Zeit ausgehend von einem fixen Punkt in der Vergangenheit [27]. Gespeichert werden die Zeiten in einem *struct timespec*, welches das Abspeichern von Zeiten in einer Nanosekunden-Auflösung ermöglicht. Um die Zeit abzurufen, wird die auf POSIX kompatiblen Systemen zur Verfügung stehende Funktion *clock_gettime()* genutzt. Problematisch

kann die Einhaltung der Frequenz aufgrund der Nutzung von Mutexen werden. Hier besteht die Gefahr, dass der Programmablauf zu lange blockiert wird. Ob die Frequenz trotzdem gehalten werden kann, wird durch die folgenden Experimente geprüft.

KAPITEL 4

Evaluierung

Zur Evaluierung des Konzepts werden zuerst einige Experimente durchgeführt. Diese zeigen die Leistungsfähigkeit und Eignung der Implementierung für die haptische Kommunikation. Anschließend folgt eine kurze theoretische Betrachtung, die die Eignung der Implementierung und des Konzepts zeigt.

4.1 Experimente

Um die Vorgehensweise zu evaluieren wurden mehrere Experimente durchgeführt. Dabei soll gezeigt werden, dass das Framework die Anforderungen an die haptische Kommunikation erfüllt. Unter anderem wird nachgewiesen, dass die durch das Framework erzeugte Latenz durchgehend unter 1 ms bleibt, die Sendefrequenz von 1 kHz erreicht wird und, dass diese Frequenz ohne große Schwankungen gehalten werden kann. In einem weiteren Experiment wird mit einer Beispielszene die Qualität der Steuerung unabhängig von den klassischen Netzwerkmetriken, wie zum Beispiel Latenz oder Jitter bewertet. Die Experimente werden in diesem Kapitel genauer erläutert und ausgewertet.

Für die Experimente wurde eine spezielle Simulations-Szene in v-rep erstellt, welche sich für die Evaluierung eignet. Auch dieses wird im Folgenden, inklusive der Anwendung des Frameworks für diese Szene, genau erklärt. Zusätzlich wird der benötigte Algorithmus für das Problem implementiert und erklärt. Hierbei wird ersichtlich, wie einfach die Anpassung des Frameworks an sich wechselnde Gegebenheiten ist.

Das für die Experimente benutzte System wird in Tabelle 4.1 charakterisiert.

Hardware	Intel® Xeon(R) CPU E3-1230 v3
Betriebssystem	Ubuntu 18.04.2 LTS, 4.15.0-58-generic (64-bit)
Auflösung clock	1ns (CLOCK_MONOTONIC)
Compilerflags	-std=c++11 -O3

Tabelle 4.1: Informationen über das für die Experimente genutzte System.

4.1.1 v-rep-Szene

Zur Durchführung der Experimente ist es sinnvoll eine Szene zu wählen, die es erlaubt, die Qualität der Steuerung durch den Algorithmus über das Framework zu bestimmen. Gleichzeitig sollte die Szene möglichst einfach aufgebaut sein, damit nicht die Szene selbst, beziehungsweise der Algorithmus zur Lösung des Problems zum Hindernis wird. Wünschenswert ist zur Anschauung auch ein reales Problem, wie es in der echten Welt vorkommt.

Mit den zuvor genannten Kriterien wurde ein stationärer Industrieroboter gewählt, dessen Roboterarm einen vorgegeben Pfad folgen soll. In der echten Welt könnte dies ein Roboter sein, der eine Schweißnaht zieht. Die Sensordaten sind in diesem Fall das Bild einer Kamera (Auflösung 256*256 Pixel), die in der Mitte des Roboterarms befestigt ist.

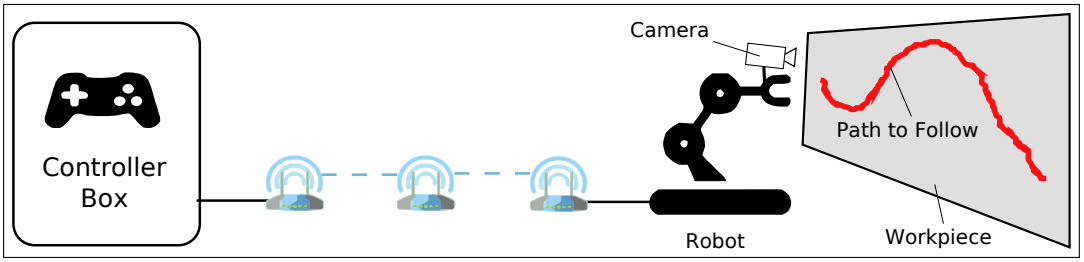

Abbildung 4.1: Das Diagramm zeigt den Aufbau des Experiments (eigene Darstellung).

Zur Vereinfachung der Steuerung des Roboters besteht in v-rep die Möglichkeit, den Roboterarm mit einem so genannten Dummy zu verbinden. Dieser steht parallel zum Pfad. Der Dummy gibt die Position und Orientierung des Roboterarms vor. So muss nur noch die Position auf zwei Achsen des Dummys verändert werden, anstelle aller Freiheitsgrade des Roboters. Der Pfad ist in schwarzer Farbe vor einer weißen Wand montiert, um einen möglichst hohen Kontrast und damit eindeutige Daten zu erreichen. Die horizontale Bewegung des Roboters zwischen dem Start- und Endpunkt wird lokal ausgeführt und muss nicht über den Algorithmus gesteuert werden. Ansonsten wäre keine gleichmäßige Bewegung garantiert. Um eine endlose Simulation zu ermöglichen, fährt der Roboter in horizontaler Richtung immer vor und zurück. Durch den Algorithmus muss also nur noch die vertikale Bewegung gesteuert werden. Als Maß für die Qualität der Steuerung wird die Abweichung der Mitte des Roboterarms von der optimalen Linie über die gesamte Zeit der Simulation genommen.

Algorithmus

Zur Steuerung des Roboters wird ein Proportional-Regler (P-Regler) implementiert. Ein P-Regler zeichnet sich dadurch aus, dass die Stellgröße (Veränderung der Position auf der vertikalen Achse) in linearer Abhängigkeit zu der Abweichung von der gewünschten Position steht. Das bedeutet, dass je stärker die Abweichung des Roboterarms von dem Pfad ist, desto größer ist die Stellgröße [28].

Der Algorithmus bekommt als Eingabe das Kamerabild. Das Kamerabild (Grautöne) wird als ein Array dargestellt. Ein Datum in dem Array ist eine Gleitkommazahl mit einem Wert

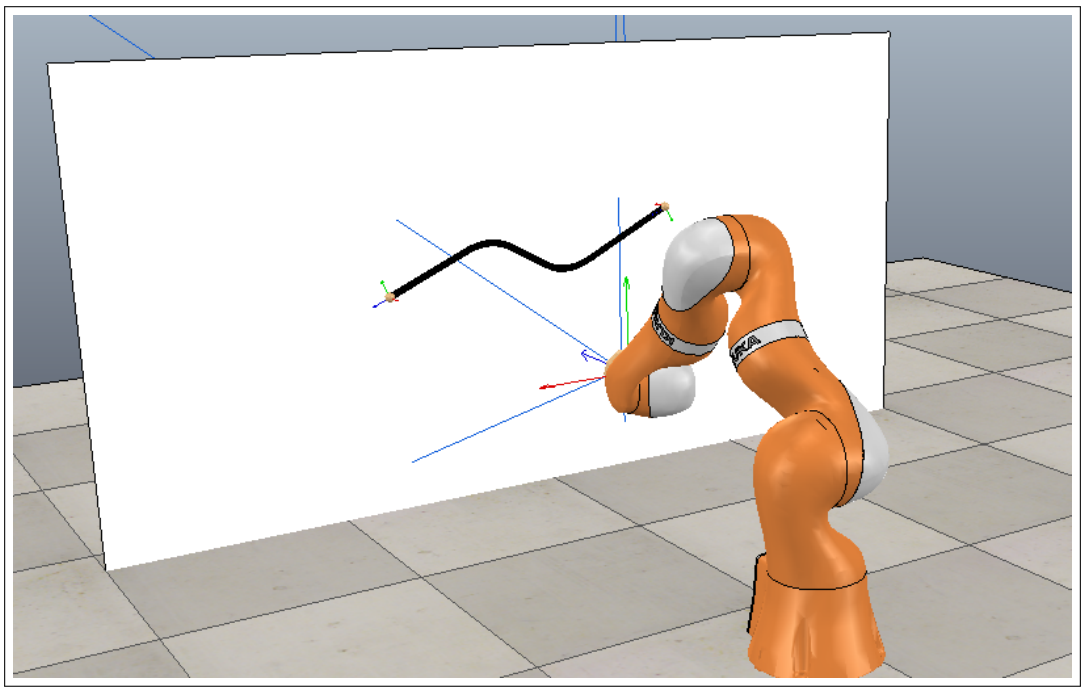

Abbildung 4.2: Screenshot aus v-rep mit Roboter, der mit seinem Arm einen Pfad abfährt (eigener Screenshot).

zwischen 0 und 1. Dabei ist nur die mittlere Spalte von Interesse, da sich der Roboterarm, beziehungsweise die Kamera mittig auf dem Pfad befinden soll. Über diese Spalte wird in einer Schleife iteriert. Dabei werden die Zeilen oberhalb der Linie und dann die Zeilen unterhalb der Linie gezählt. Dadurch, dass in diesem vereinfachten Fall die Linie schwarz ist, entspricht ein Pixel, der zur Linie gehört, einer 0. Ein Pixel, der zum weißen Hintergrund gehört, besitzt den Wert 1. Die Größe und Richtung der Abweichung ergeben sich aus dem Verhältnis aus den Zeilen über der Linie und den Zeilen unter der Linie. Diese Abweichung multipliziert mit einer Konstanten bildet die Stellgröße. Die Konstante muss so gewählt werden, dass die Abweichung korrigiert werden kann, ohne dass es zu großen Schwankungen um das Optimum kommt. Falls der Sonderfall eintritt, dass in der Spalte keine Linie gefunden wird, so beträgt die Stellgröße 0.

4.1.2 Anwendung des Frameworks

Im Folgenden Kapitel wird aus Sicht des Slaves (v-rep-Plugin) und des Masters (Algorithmus) erklärt, wie das Framework an die für das Experiment nötige Szene angepasst wird. Dies soll die Anwendung und Einfachheit des Frameworks verdeutlichen. Dazu werden stellenweise auch Codeausschnitte gezeigt.

Zu Beginn sollte bedacht werden, welche Sensordaten und welche Steuerungsdaten zwischen den Komponenten ausgetauscht werden müssen. Auch müssen die Handle, welche Referenzen zu Objekten in v-rep sind, zum Zugriff auf die Roboterkomponenten festgelegt werden. Dies geschieht alles in der durch den Master und Slave gemeinsam genutzten Headerdatei *RobData.h*.

Da der Algorithmus nur eine Spalte des Bildes benötigt, sind die Sensordaten in diesem Fall nur diese eine Spalte. Die weiteren Spalten des Bildes müssen nicht übertragen werden. Als Steuerungsdaten wird nur die Stellgröße übertragen. Zur Steuerung des Roboters wird ein Handle für den Dummy benötigt. Zum Auslesen des Kamerabilds ein Handle für die Kamera und für die horizontale Bewegung Handles für den Start- und Endpunkt des Graphen. In der *RobData.h* wird auch der Szenen-Name für das automatische Laden der Szene und die Buffergröße für die Sende- und Empfangsbuffer festgelegt. Dies ergibt den im Quellcode 4.1 gezeigten Code.

```
1  #define sceneName "scene_Path.ttt"
2
3  // ############## Handles of parts of the Rob (int) ####################
4  struct handles
5  {
6      int camera;
7      int target;
8      int startPoint;
9      int endPoint;
10 };
11 // ############## Sensor- and Controldata (only float) ################
12 struct sensData
13 {
14     float imageColumn[256];
15 };
16 struct ctrlData
17 {
18     float dstellgroesse;
19 };
```

Quellcode 4.1: RobData.h: Festlegung der Handle, Szene, Steuerungs- und Sensordaten

Anpassungen im Bereich des Slaves

Der Slave beziehungsweise das v-rep-Plugin hat die Aufgabe, die Sensordaten abzufragen und die Steuerungsdaten auf den Roboter umzusetzen. Um das Framework an die Szene anzupassen, müssen hier nur Änderungen an der Datei *Slave.cpp* vorgenommen werden. Da es sich hierbei um die Erweiterung einer Klasse handelt, von der beim Start von v-rep eine Instanz erzeugt wird, besteht die Möglichkeit Daten zwischen den Funktionen über Klassenvariablen auszutauschen. Die Kommunikation mit v-rep erfolgt über die *regular-API* [1].
Als Erstes sollte hier die Methode *getHandle()* implementiert werden. Diese Funktion wird einmalig ausgeführt, bevor die Simulation startet. Hier können die Handle mit der Funktion *simGetObjectHandle()* über die Namen der Szenenbestandteile abgefragt und in den *struct robHandles* gespeichert werden. Um den Code übersichtlicher zu halten, wurde in diesem Auszug auf die Kontrolle der Rückgabewerte verzichtet. Der Quellcodeauszug hierzu findet sich in dem Quellcode 4.2.

```
1  void getHandle()
2  {
3      // Get handles (You should check values. Returns -1 when error occurs.)
4      robHandles.camera = simGetObjectHandle("Vision_sensor");
```

[1] http://www.coppeliarobotics.com/helpFiles/en/apiOverview.htm

```
5       robHandles.startPoint =  simGetObjectHandle("StartPoint");
6       robHandles.endPoint =  simGetObjectHandle("EndPoint");
7       robHandles.target = simGetObjectHandle("Target");
8
9       // Get start- and endpoint of graph.
10      simGetObjectPosition(robHandles.startPoint, -1,startPosition);
11      simGetObjectPosition(robHandles.endPoint, -1,endPosition);
12
13      // Set position of dummy
14      startPosition[0] += -0.2;
15      simSetObjectPosition(robHandles.target,-1,startPosition);
16
17      // initialize controldata
18      robCtrl.dstellgroesse = 0.0f;
19  }
```

Quellcode 4.2: getHandle()

Bei einem Fehler gibt die Funktion *simGetObjectHandle()* einen Wert von -1 zurück. Außerdem wird in der Methode *getHandle()* die Start- und Endposition des Graphen einmalig abgerufen, sowie die Stellgröße in dem *struct robCtrl* initialisiert. Auch wird der Roboterarm über den Dummy an seine Startposition gesetzt.
In der Funktion *getSensData()*, welche in einen Mutex für den Schutz des *struct robSens* gehüllt ist, wird das Kamerabild mit der Funktion *simGetVisionSensorImage()* abgerufen. Beim Übergeben des Handles wird die Konstante *sim_handleflag_greyscale* hinzu addiert. Dadurch gibt die Funktion das Kamerabild in Grautönen zurück. Aus diesem Bild wird die mittlere Spalte herausgezogen und diese in das *struct robSens* gespeichert. Danach darf nicht vergessen werden, den Buffer für das Bild mit der Funktion *simReleaseBuffer()* wieder freizugeben. Auch wird hier die Position des Dummys mit der Funktion *simGetObjectPosition()* aktualisiert. Dieser Codeausschnitt befindet sich in dem Quellcode 4.3.

```
1   void getSensData ()
2   {
3       // get image in Greyscale (256*256)
4       image=simGetVisionSensorImage(robHandles.camera + sim_handleflag_greyscale);
5
6       // save one column in struct for sensordata
7       for(int i = 0; i<resxy;i++)
8       {
9           robSens.imageColumn[i] = image[i*resxy + resxy/2];
10      }
11      simReleaseBuffer((simChar*)image); //release memory for image
12
13      // get position of dummy (error: returns -1)
14      simGetObjectPosition(robHandles.target, -1,targetPosition);
15  }
```

Quellcode 4.3: getSensData()

In der Funktion *setCtrlData()*, die in einen Mutex für den Schutz des *struct robCtrl* gehüllt ist, werden die Steuerungsdaten in die Simulation übertragen. In diesem Fall ist dies die Korrektur der Position auf der vertikalen Achse. Dafür wird die Stellgröße auf den zuvor bestimmten Wert addiert. Auch wird in diesem Fall die Bewegung auf der horizontalen Achse umgesetzt. Dies ist unabhängig von den empfangenen Steuerungsdaten. Diese neuen Werte werden mit der Funktion *simSetObjectPosition()* in der Simulation umgesetzt. Gezeigt wird dies in dem Quellcode 4.4.

```
1  void setCtrlData ()
2  {
3      float newPosition[3] = {targetPosition[0],targetPosition[1],targetPosition
           [2]};
4
5      // constant movement on the horizontal-axis
6      if(newPosition[1]<endPosition[1] || newPosition[1]>startPosition[1])
7      {
8          step *= -1;
9      }
10     newPosition[1] += step;
11
12     // correction on vertical-axis
13     newPosition[2] += robCtrl.dstellgroesse;
14
15     // set new Position
16     simSetObjectPosition(robHandles.target,-1,newPosition);
17 }
```

Quellcode 4.4: setCtrlData()

Anpassungen im Bereich des Masters

Auf der Seite des Masters wird der zuvor beschriebene Algorithmus implementiert. Dies geschieht in der Datei *Algorithm.cpp* in der Funktion *step(struct ctrlData &robCtrl, struct sensData &robSens)*. Diese wird aufgerufen, sobald Daten empfangen worden sind. Der *struct robSens* bietet hier Zugriff auf diese empfangenen Daten und die Steuerungsdaten werden nach den Berechnungen in den *struct robCtrl* gespeichert. Nach dem Durchlauf des Algorithmus werden diese an den Slave gesendet. Sollen Daten über mehrere Durchläufe hinweg gespeichert werden, so können Variablen in der Klassendefinition in der Datei *Algorithm.h* deklariert werden.

Kodierung

Beispielhaft wurde auch eine sehr einfache Kodierung nach dem Deadband-Ansatz implementiert. Das heißt, dass die Sensordaten nur gesendet werden, wenn sie sich tatsächlich geändert haben. Für die Experimente wurde diese Kodierung jedoch deaktiviert, damit die gewünschten Frequenzen nicht durch die Kodierung beeinflusst werden. Um die Kodierung zu implementieren wurde in der Datei *Coding.cpp* auf der Seite des Slaves die Funktion *encode(...)* angepasst. Zu sehen ist dies in dem Codeausschnitt 4.5. Haben sich die Daten im Vergleich zum vorherigen Durchlauf verändert, so werden die aktuellen Daten in den Speicher an die Position kopiert, auf die der Zeiger *bufferEncoded* zeigt. Falls die Daten unverändert sind, so wird die Variable *sizeEncoded* auf 0 gesetzt. Jetzt weiß das Framework, dass die Daten nicht versendet werden müssen.

```
1  void encode(char* buffer, char* bufferEncoded, size_t size, size_t &sizeEncoded,
       char* dataLastSend)
2  {
3      // Only send Data, if changed!
4      if(memcmp(dataLastSend,buffer,size)==0)
5      {
6          sizeEncoded=0;
7      } else
```

```
8       {
9           memcpy(bufferEncoded, buffer, size);
10          sizeEncoded = size;
11      }
12  }
```

Quellcode 4.5: encode()

4.1.3 Einhaltung der gewünschten Frequenz

Ein wichtiges Kriterium, um vergleichbare und aussagekräftige Experimente zu garantieren, ist die Einhaltung der vorgegebenen Frequenz. Die Einhaltung der Frequenz wird in diesem Versuch genauer untersucht. In diesem ersten Versuch geht es nur um die absolute Frequenz und nicht um die Zeiten der einzelnen Intervalle. In diesem Experiment soll gezeigt werden, dass die implementierte Software in der Lage ist, Daten mit einer Frequenz bis zu 1.000 Hz zu versenden.

Versuchsaufbau

Für das Experiment wird die zuvor beschriebene Szene mit dem pfadfolgenden Roboter genutzt. Die gewählte Frequenz ist 1.000 Hz. Die Software wird so angepasst, dass sie genau 30 Sekunden lang die Sensordaten sendet. Das bedeutet, dass in diesem Zeitraum genau 30.000 Datenpakete gesendet und empfangen werden sollten, das entspricht 3.000 mal 10 ms lang je 10 Pakete. Jedes Datenpaket mit Sensordaten, das gesendet wird, ist 1024 Bytes groß. Die empfangenen Steuerungsdaten sind 4 Bytes groß.

Alle 10 ms wird die Anzahl der in den vergangenen 10 ms gesendeten und empfangenen Datenpakete in je einem Array hinterlegt. Durch das häufige Messen der Werte werden zufällige Messfehler ausgeglichen und damit die Aussagekraft und Genauigkeit erhöht. Diese Arrays werden zum Ende der Simulation ausgegeben und gespeichert. Das Zählen der Pakete geschieht in den Kommunikationsthreads, um eine Unabhängigkeit von den API-Aufrufen zu erreichen. Die Datenpakete werden in dem Plugin für die Robotersimulation gezählt. Das Experiment wird lokal ausgeführt: Sowohl die Steuerung, als auch die Simulation werden auf dem gleichen Rechner ausgeführt. Im Anschluss daran wird die Frequenz unter dem gleichen Versuchsaufbau schrittweise erhöht, bis die Frequenz nicht mehr gehalten werden kann.

Auswertung

Aufgrund der Vielzahl der Messwerte werden diese hier nicht exakt aufgelistet. Alle Messwerte zusammen mit dem Experiment sind in dem zur Bachelorarbeit gehörenden git[2] und auf der beiliegenden CD-ROM zu finden.

Die einzelnen Messwerte der Anzahl der empfangenen Pakete x_i und der gesendeten Pakete y_i pro Zeitintervall i sind alle gleich. Es wurden immer 10 Pakete je 10 ms gesendet und empfangen. n ist die Anzahl der gemessen Werte (Zeitintervalle).

Damit ergeben sich die folgenden Mittelwerte $\overline{x}, \overline{y}$ und Standardabweichungen s_x, s_y:

[2] `https://code.ovgu.de/comsys-group/students/bsc/2019-behrens-johannes`

$$\overline{x} = \frac{1}{n} \sum_{i=0}^{n} x_i = 10$$

$$\overline{y} = \frac{1}{n} \sum_{i=0}^{n} y_i = 10$$

$$s_x = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n} (x_i - \overline{x})^2} = 0$$

$$s_y = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n} (y_i - \overline{y})^2} = 0$$

Die Tabelle 4.2 zeigt die Ergebnisse der Fortführung des Experiments unter stufenweiser Erhöhung der Frequenz. Ersichtlich sind die gewählte Frequenz, die minimal gemessene Frequenz und der Mittelwert inklusive des Konfidenzintervalls bei einem Vertrauensniveau von 95% und dem entsprechenden STUDENT-Faktor $t = 2$ [29]. In der letzten Spalte ist die Standardabweichung ersichtlich.

Deutlich zu sehen ist, dass bei steigender Frequenz die Standardabweichung wächst und ab einem bestimmten Punkt (senden: ca. 48 kHz, empfangen: ca. 32 kHz) die Frequenz über einen längeren Zeitraum nicht mehr gehalten werden kann. Da die Empfangsfrequenz direkt von der Sendefrequenz abhängig ist, und die Pakete über den localhost erst zur Steuerung und dann wieder zurückgesendet werden, ist hier die Grenzfrequenz eher erreicht. Auch ist daher beim Empfangen die Standardabweichung in der Regel größer.

In dem Diagramm 4.3 wird der lineare Zusammenhang zwischen der gewählten Frequenz und der gemessenen Frequenz dargestellt. Dort sind die Punkte zu erkennen, an denen die Frequenz nicht mehr gehalten werden kann und die Kurve abflacht. Für diese Darstellung wurden die Mittelwerte gewählt.

Die Histogramme 4.4 und 4.5 zeigen die relative Häufigkeit von gemessenen Frequenzen bei einer gewählten Frequenz von 450 kHz und bei 200 kHz. Bei beiden gewählten Frequenzen stimmt zwar der Mittelwert fast mit der jeweils gewählten Frequenz überein, jedoch ist die Standardabweichung bei 450 kHz deutlich größer. In den Histogrammen lassen sich die unterschiedlichen Streubreiten deutlich erkennen. Bei 450 kHz treffen nur noch wenige Messungen direkt den gewünschten Wert.

Bei dem Experiment kommt heraus, dass eine Frequenz von 1.000 Hz gehalten werden kann. Auch ist durch die Fortführung des Experiments unter Erhöhung der Frequenz ersichtlich, dass hier noch Raum nach oben existiert. Dies spricht für die Nutzbarkeit des Frameworks für Netzwerkexperimente mit haptischen Daten. Durch diesen Versuch kann jedoch nicht allgemeingültig gesagt werden, dass es immer möglich ist, die Daten mit einer Frequenz von 1.000 Hz zu senden. Die Leistung der Hardware, die Genauigkeit der Uhr und das Betriebssystem spielen hier eine entscheidende Rolle. Auch bei einer außergewöhnlich großen Menge an Sensor- oder Steuerungsdaten kann es zu Problemen kommen.

	min [kHz]		*mean [kHz]*		*SD [kHz]*	
	snd	*rcv*	*snd*	*rcv*	*snd*	*rcv*
50 kHz	50,00	47,70	50,00 ± 0,00	49,99 ± 0,00	0,00	0,07
100 kHz	100,00	91,00	100,00 ± 0,00	99,99 ± 0,01	0,00	0,40
150 kHz	149,50	140,60	150,02 ± 0,00	149,98 ± 0,02	0,05	0,42
200 kHz	187,40	139,50	199,99 ± 0,08	199,74 ± 0,09	2,06	2,53
220 kHz	200,40	191,80	220,02 ± 0,09	219,22 ± 0,14	2,53	3,94
240 kHz	221,30	196,50	240,03 ± 0,07	239,29 ± 0,18	1,83	4,80
260 kHz	246,00	192,30	260,01 ± 0,02	257,83 ± 0,27	0,56	7,47
280 kHz	230,30	54,10	280,03 ± 0,08	276,54 ± 0,37	2,06	10,04
300 kHz	249,50	162,60	300,03 ± 0,34	292,32 ± 0,61	9,32	16,73
320 kHz	298,00	108,10	319,99 ± 0,04	312,13 ± 0,64	0,99	17,65
340 kHz	280,70	38,70	340,02 ± 0,18	327,19 ± 0,86	4,80	23,62
360 kHz	309,90	219,50	360,10 ± 0,26	346,39 ± 0,79	7,20	21,83
380 kHz	261,80	188,20	380,08 ± 0,52	345,46 ± 0,98	14,33	26,81
400 kHz	237,70	105,30	399,99 ± 0,72	349,54 ± 0,93	19,68	25,52
420 kHz	236,30	195,80	420,16 ± 0,65	352,14 ± 0,85	17,76	23,28
440 kHz	351,80	0,00	440,13 ± 0,88	347,73 ± 0,95	23,97	26,06
450 kHz	312,20	0,00	450,04 ± 0,84	342,64 ± 0,89	22,89	24,26
460 kHz	259,40	91,40	460,19 ± 1,41	342,86 ± 1,20	38,71	32,88
480 kHz	255,20	120,80	480,07 ± 1,35	349,56 ± 1,12	36,87	30,78
500 kHz	313,70	37,80	499,78 ± 1,25	350,61 ± 0,90	34,31	24,78
520 kHz	254,10	40,60	500,59 ± 1,40	346,13 ± 0,96	38,33	26,32
540 kHz	298,80	98,90	487,28 ± 1,20	348,17 ± 0,81	32,95	22,18
560 kHz	238,30	0,00	510,61 ± 1,24	351,41 ± 0,93	33,99	25,34
580 kHz	322,20	0,00	500,35 ± 1,32	347,11 ± 1,02	36,06	27,96
600 kHz	243,20	84,90	496,35 ± 1,72	345,18 ± 1,17	47,12	32,06

Tabelle 4.2: Zusammenfassung der Messwerte bei stufenweiser Erhöhung der Frequenz.

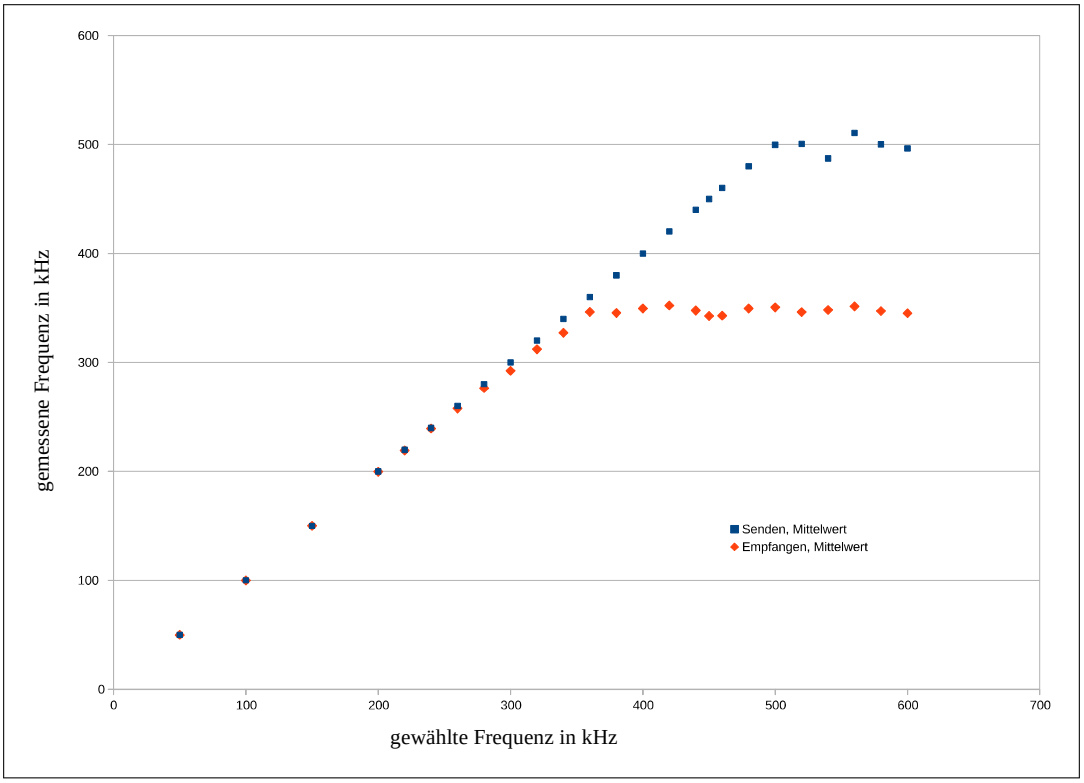

Abbildung 4.3: Zusammenhang zwischen gewählter Frequenz und gemessener Frequenz (eigene Darstellung).

4.1.4 Schwankungen der Genauigkeit im Übertragungstakt

Nicht nur die Frequenz über den gesamten Zeitraum spielt eine Rolle. Auch ist es wichtig, dass die Pakete gleichmäßig versendet werden. Das bedeutet, dass die Zeiten (Interarrival Time) zwischen dem Versenden von zwei aufeinanderfolgenden Paketen immer möglichst gleich sind. Ist dies nicht der Fall, würde das bedeuten, dass der Roboter immer wieder für bestimmte Zeiträume nicht gesteuert werden kann oder die Daten aufgrund einer Zeitabhängigkeit an Bedeutung verlieren. Mit dem Experiment soll gezeigt werden, dass keine nennenswerten Schwankungen existieren.

Versuchsaufbau

Für dieses Experiment wird die Szene mit dem pfadfolgenden Roboter genutzt. Die Frequenz wird fest auf 1.000 Hz eingestellt und der Roboter folgt für 1 Minute dem Pfad. Während der Ausführung werden in dem Plugin die Zeiten zwischen dem Versenden von zwei Paketen gemessen. Dafür wird jeweils die Zeit direkt nach dem Versenden eines Pakets innerhalb des v-rep-Plugins genutzt. Am Ende der Ausführung werden die Zeiten ausgegeben und ausgewertet. Die Anzahl der gesendeten Pakete nach einer Minuten sollte 60.000 betragen und die Zeit zwischen zwei Paketen sollte optimalerweise 1 ms sein.

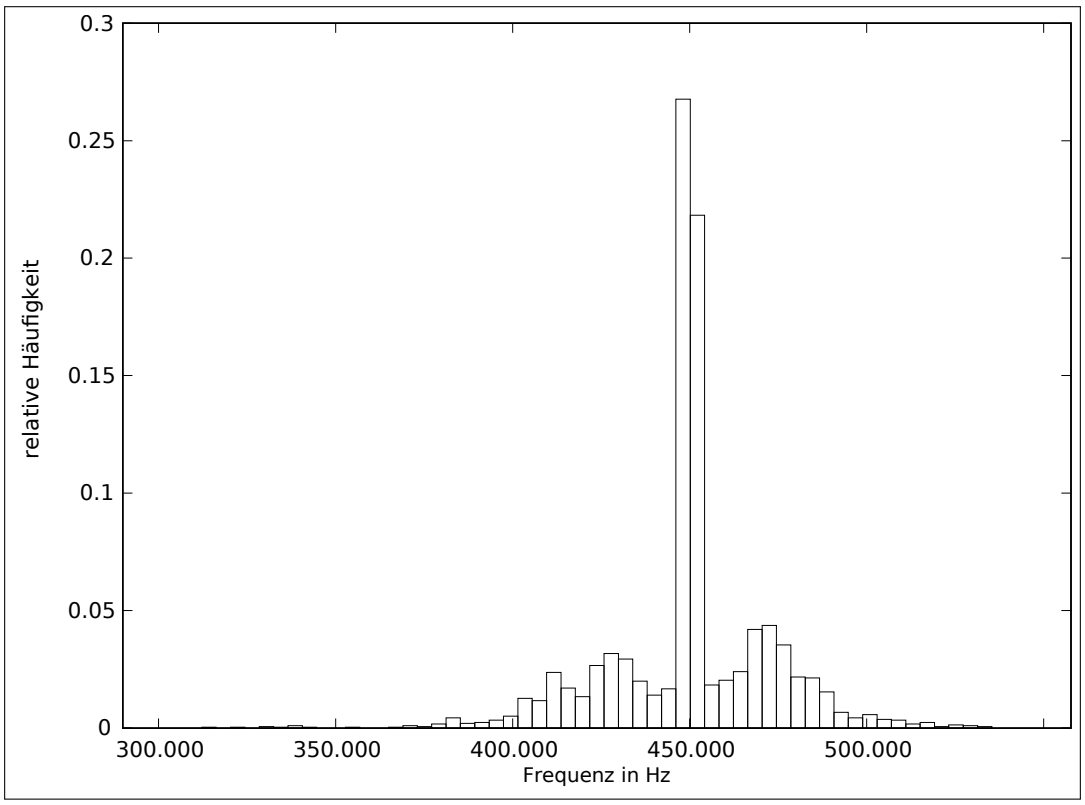

Abbildung 4.4: Das Histogramm zeigt die relative Häufigkeit der Anzahl an gemessenen Frequenzen bei einer Frequenz von 450 kHz (eigene Darstellung).

Auswertung

Es wurden entsprechend den 60.000 gesendeten Paketen 59.999 Zeiten gemessen. n steht in diesem Experiment wieder für die Anzahl an Messwerten, x ist die Zeit zwischen zwei gesendeten Paketen. Die vollständigen Messwerte sind dem git[3] und der CD-ROM zu entnehmen. Die längste gemessene Zeit ist 1,47125 ms und die kürzeste Zeit 0,528734 ms.

Aus den Messwerten ergibt sich folgender Mittelwert $\overline{x}$ und die Standardabweichung s_x:

$$\overline{x} = \frac{1}{n} \sum_{i=0}^{n} x_i = 1,0000000303011\,ms$$

$$s_x = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n} (x_i - \overline{x})^2} = 0,0060220843672515\,ms$$

Bei einem Vertrauensniveau von 95 % und dementsprechend dem STUDENT-Faktor $t = 2$ ergibt [29] sich eine zufällige Abweichung des Mittelwertes

$$\Delta\overline{x} = \frac{ts}{\sqrt{n}} = 0,000049171\,ms$$

[3] `https://code.ovgu.de/comsys-group/students/bsc/2019-behrens-johannes`

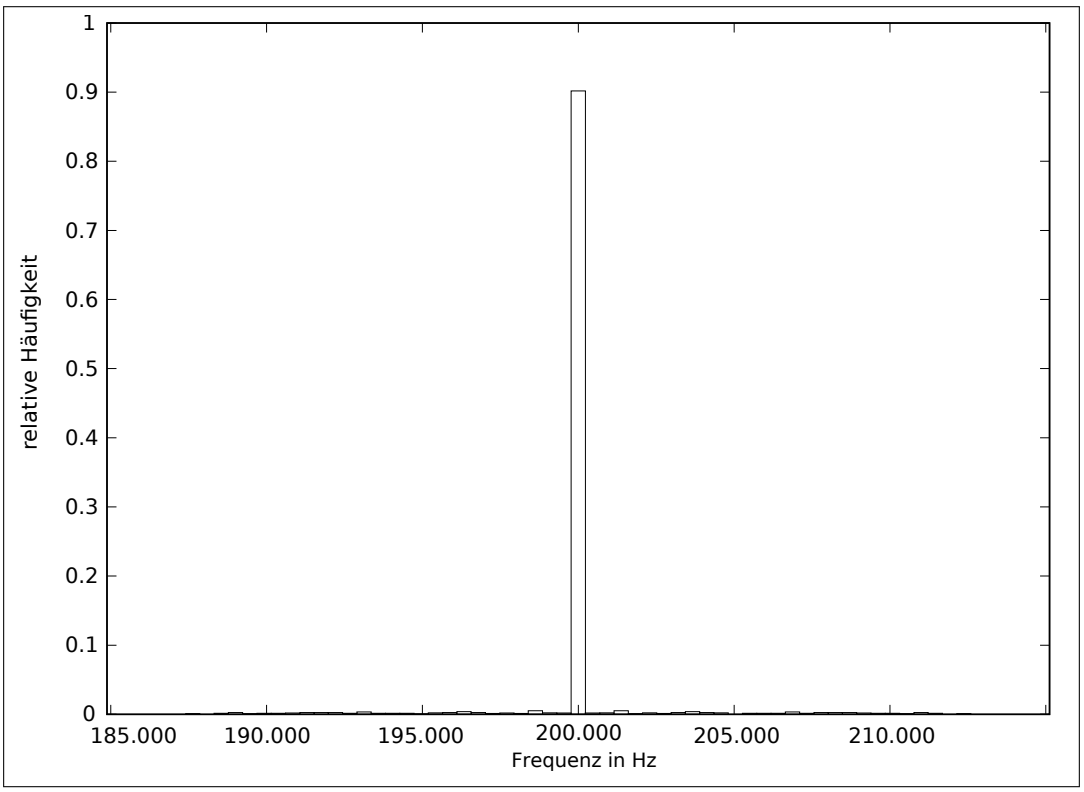

Abbildung 4.5: Das Histogramm zeigt die relative Häufigkeit der Anzahl an gemessenen Frequenzen bei einer Frequenz von 450 kHz (eigene Darstellung).

und somit als Vertrauensbereich

$$(1,0000000303011 \pm 0,000049171)\, ms$$

In dem Histogramm 4.6 ist gut zu sehen, dass die meisten Messwerte genau den erwarteten Wert von 1 ms treffen oder nur gering davon abweichen.

Größtenteils sind die gemessenen Zeiten gleichmäßig. Teilweise kam es zu Ausreißern, bei denen die Zeit zwischen zwei Paketen größer war. Dies kann zum Beispiel passieren, wenn die Mutex zu lange gesperrt sind oder der Rechner aktuell einen anderen Prozess bearbeitet. Da die Frequenz ausgehend von dem Startzeitpunkt gehalten wird, sind die nach einer langen Zeit gemessenen Zeiten niedriger. Dies ist gut in dem Diagramm 4.7 zu erkennen. Es zeigt einen Ausschnitt von ca. 100 Messwerten (58.000 bis 58.100). Die zufällige Abweichung des Mittelwertes bei einem Vertrauensniveau von 95 % ist mit ca. 49 ns sehr niedrig.

4.1.5 Overhead (Zeit)

Durch das Framework, insbesondere die Netzwerkkommunikation und die durch die genutzten Mutex entstehende Serialisierung, entsteht ein Zeitoverhead. Dadurch wird die Reaktion auf Ereignisse verzögert. Um diese Verzögerung zu ermitteln, wird ein weiteres Experiment

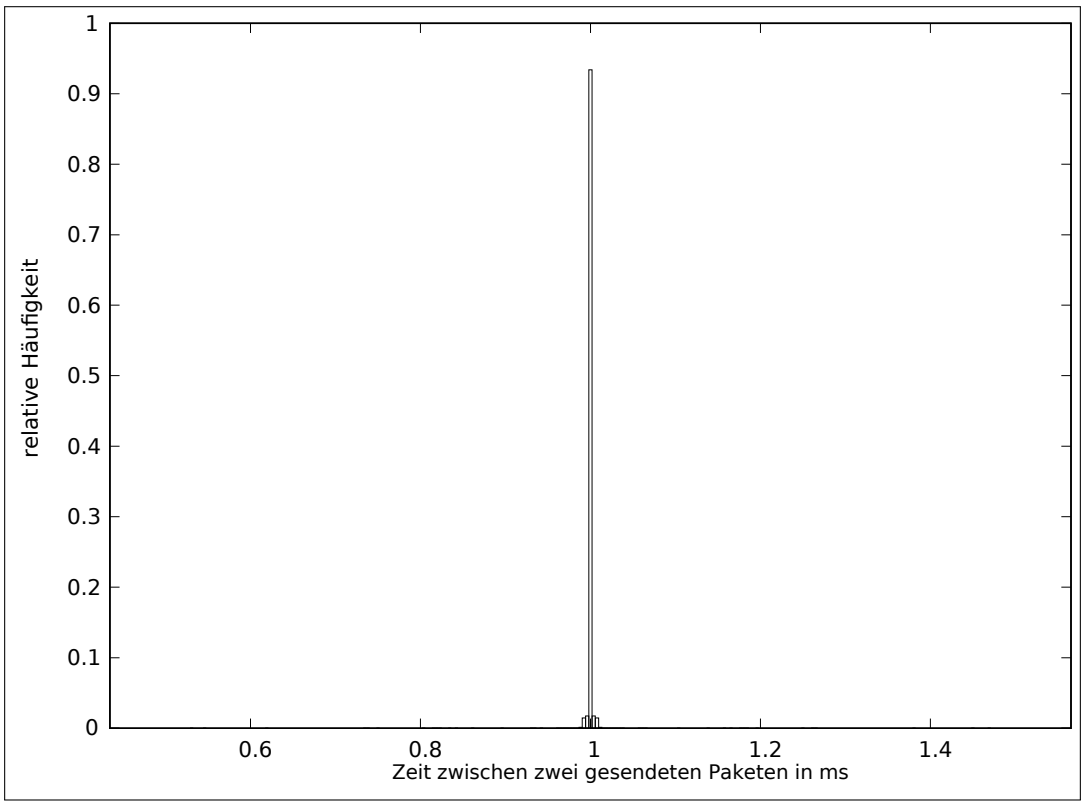

Abbildung 4.6: Das Histogramm zeigt die relative Häufigkeit der gemessenen Zeiten in ms zwischen dem Versand von zwei Paketen (eigene Darstellung).

durchgeführt. Dafür wird eine Round-Trip-Time ermittelt. Es wird erwartet, dass diese deutlich unter 1 ms ist, damit die 1 ms-Challenge für die haptische Kommunikation eingehalten werden kann.

Versuchsaufbau

Bei diesem Experiment wird zusätzlich zu den Sensordaten eine Zahl übertragen, die als Index für ein lokales Array dient und die bei jedem Versenden der Sensordaten um 1 erhöht wird. In diesem lokalen Array werden die Zeiten gespeichert, direkt bevor die Sensordaten versendet werden. Bei der Steuerung angekommen wird beim Ausführen des Algorithmus dieser Index wieder zu den Steuerungsdaten hinzugefügt und zurückgesendet. Der eigentliche Algorithmus wird nicht ausgeführt, da diese Zeit nicht mitgemessen werden soll. Sobald die Steuerungsdaten wieder am Plugin angekommen sind, wird die aktuelle Zeit bestimmt und die Differenz zu der Startzeit gespeichert. Zum Ende der Simulation werden die so ermittelten Latenzen ausgegeben. Für dieses Vorgehen wird der pfadfolgende Roboter gewählt und die Frequenz auf 1.000 Hz festgelegt. Es werden insgesamt 10.000 Zeiten gemessen.

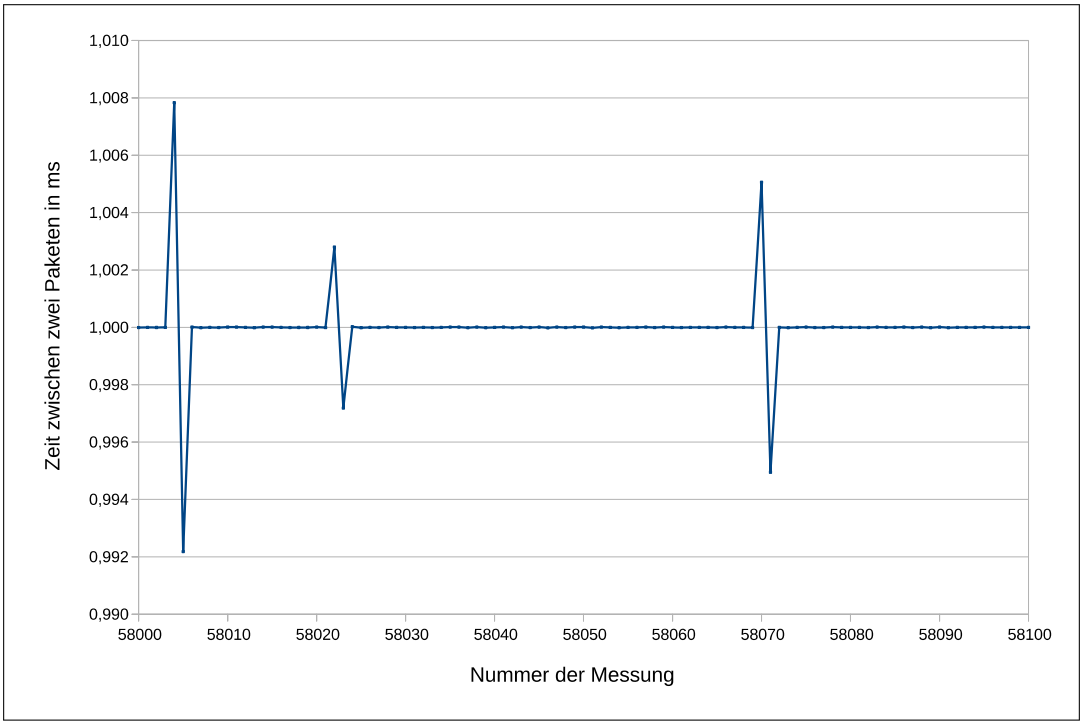

Abbildung 4.7: Das Diagramm zeigt Abweichungen von dem optimalen Wert inklusive der folgenden Korrektur (eigene Darstellung).

Auswertung

Es wurden $n = 10.000$ Zeiten gemessen, dabei wurde eine maximale Round-Trip-Time von 0,86 ms gemessen. Diese Zeit ist unter 1 ms und daher noch im Rahmen der Anforderung von 1 ms. Allerdings ist dieser maximale Wert schon nah an der Grenze und lässt nicht viel Spielraum für die weitere Verarbeitung und Übertragung der Daten. Jedoch handelt es sich bei den hohen Zeiten nur um wenige Ausreißer, die vermutlich auf das Scheduling zurückzuführen sind. Der Mittelwert

$$\overline{x} = \frac{1}{n} \sum_{i=0}^{n} x_i = 0,0250885\,ms$$

ist sehr niedrig und bestätigt die Erwartungen. Dabei kommt es zu einer Standardabweichung von

$$s_x = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n} (x_i - \overline{x})^2} = 0,0435509\,ms$$

Bei einem Vertrauensniveau von 95 % und dementsprechend dem STUDENT-Faktor $t = 2$ [29] ergibt sich eine zufällige Abweichung des Mittelwertes

$$\Delta\overline{x} = \frac{ts}{\sqrt{n}} = 0,0008710\,ms$$

und somit als Vertrauensbereich

$$(0,0250885 \pm 0,0008710)\, ms$$

Das Diagramm 4.8 zeigt die relative Häufigkeit der Round-Trip-Zeiten aufgeteilt in $\sqrt{n}$ Klassen. Zu erkennen ist, dass es hier nur wenige Ausreißer über 0,1 ms gibt. Die meisten Round-Trip-Zeiten sind in dem Bereich darunter.

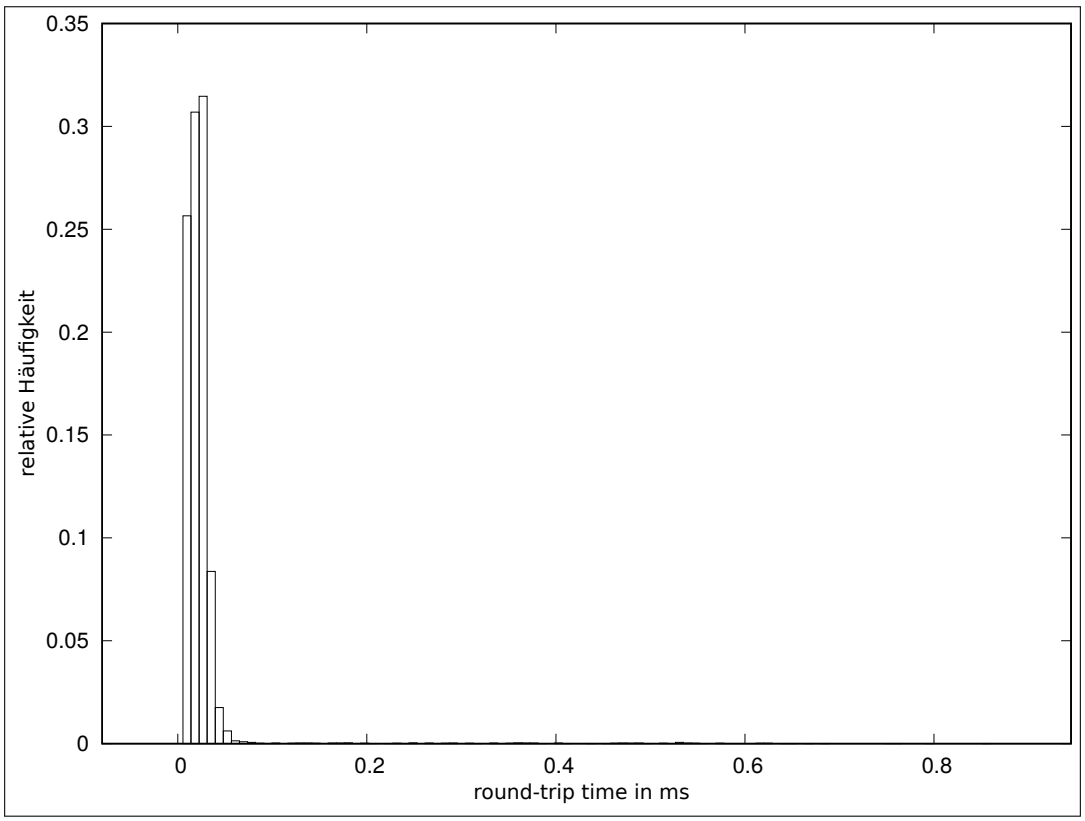

Abbildung 4.8: Das Diagramm 4.8 zeigt die relative Häufigkeit der Round-Trip-Zeiten aufgeteilt in $\sqrt{n}$ Teilintervalle (eigene Darstellung).

4.1.6 Qualität der Steuerung

Das letzte Experiment beschäftigt sich mit der Qualität der Steuerung anhand des Beispiels mit dem pfadfolgenden Roboter. Die Robotersimulation unterstützt minimal Zeitschritte von 10 ms. Dies entspricht einer maximalen Frequenz von 100 Hz. Erwartet wird daher, dass die Steuerung über das Framework erst ab 100 Hz schlechter wird, als die lokale Steuerung. Dieses Experiment ist ein Beispiel, um die QoE einer Lösung über das Framework festzustellen.

Versuchsaufbau

Der Roboter folgt dem Pfad für 250 Simulationsschritte zu je 10 ms. Über den gesamten Zeitraum wird die vertikale Abweichung der Mitte des Roboterarms von der Linie aufsummiert. Das Experiment wird für jede Frequenz f $n = 250$ mal wiederholt. Da die Kamera genau in der Mitte des Roboterarms befestigt ist, reicht die Verarbeitung einer Spalte des Bildes. Die Abweichung ergibt sich aus der Anzahl der Pixel über der Linie und der Anzahl der Pixel unter der Linie auf dem sichtbaren Bild. Sind über der Linie mehr Pixel, als unter der Linie, so wird die Anzahl der Pixel über der Linie durch die Anzahl der Pixel unter der Linie geteilt. Für den Fall, dass sich unter der Linie mehr Pixel befinden, gilt das Gegenteil. Der Versuch wird mit verschiedenen Frequenzen durchgeführt und zum Vergleich auch mit einem Plugin ohne Netzwerkkommunikation. Dadurch kann gezeigt werden, dass die Qualität der Steuerung mit dem Framework nicht schlechter ist, als eine direkte Steuerung.

Auswertung

Die Tabelle 4.3 zeigt in der ersten Spalte die maximale gemessene Abweichung für eine Frequenz. Die Mittelwerte inklusive der Konfidenzintervalle bei einem Vertrauensniveau von 95% und dementsprechend dem STUDENT-Faktor $t = 2$ [29] sind in der zweiten Spalte ersichtlich. Berechnet wurden die Werte hierfür mit folgenden Formeln:

Mittelwert $\overline{x}$:

$$\overline{x} = \frac{1}{n} \sum_{i=0}^{n} x_i$$

Zufällige Abweichung des Mittelwertes:

$$\Delta\overline{x} = \frac{ts}{\sqrt{n}}$$

Die dritte Spalte zeigt die Standardabweichung s_x von dem Mittelwert, welche jeweils wie folgt berechnet wurde:

$$s_x = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n} (x_i - \overline{x})^2}$$

Ersichtlich ist, dass die Abweichung von der Linie, wie erwartet, erst unter 100 Hz zunimmt. Interessanterweise wird die Abweichung genau bei 100 Hz minimal geringer. Dies ist vermutlich darauf zurückzuführen, dass der Algorithmus an Wendepunkten geringer überkorrigiert. Ab 15 Hz verliert der Roboter die Linie.

	Abweichung von der Linie (Verhältnis)		
	max	*mean*	*SD*
lokal	313,39	303,10 ± 0,25	1,94
1000 Hz	322,65	315,68 ± 0,30	2,37
900 Hz	322,65	315,69 ± 0,30	2,38
800 Hz	322,65	315,69 ± 0,30	2,38
700 Hz	322,65	315,69 ± 0,30	2,38
600 Hz	322,65	315,69 ± 0,30	2,38
500 Hz	322,65	315,69 ± 0,30	2,38
400 Hz	322,65	315,69 ± 0,30	2,38
300 Hz	322,65	315,69 ± 0,30	2,38
200 Hz	322,65	315,69 ± 0,30	2,38
100 Hz	322,65	315,67 ± 0,30	2,35
80 Hz	322,65	315,71 ± 0,30	2,38
60 Hz	323,13	315,82 ± 0,30	2,39
40 Hz	327,93	320,34 ± 0,41	3,21
20 Hz	334,13	324,35 ± 0,54	4,27
19 Hz	338,38	326,37 ± 0,63	4,97
18 Hz	341,18	326,95 ± 0,74	5,87
17 Hz	341,19	327,93 ± 0,74	5,88
16 Hz	344,84	329,46 ± 0,75	5,95
15 Hz	∞	∞	∞
14 Hz	∞	∞	∞

Tabelle 4.3: Die Tabelle zeigt die Abweichungen des Roboterarms von der Linie. Die Abweichung ergibt sich aus dem Verhältnis zwischen den Pixeln über und unter der Linie auf dem sichtbaren Bild. ∞ bedeutet, dass der Roboter die Linie während des Experiments verloren hat.

4.2 Overhead (Daten)

Ein Ziel ist es, dass die Daten mit möglichst wenig Overhead versendet werden, da sich die Menge dieser Daten bei der hohen Frequenz schnell aufsummiert und das Netzwerk zusätzlich belastet. Hier erfolgt eine kurze Betrachtung des entstehenden Overheads.

Zu Beginn der Kommunikation wird ausgehend vom Framework im Bereich der Master Domain ein Initialisierungspaket mit einer variablen Größe von z Bytes gesendet. Das Paket enthält den Namen der gewählten Szene, daher ist die Größe abhängig davon. Beantwortet wird dies mit einem Paket mit der Größe von 1 Byte. Diese Pakete sind der einzige Overhead $(z + 1)$ Bytes, der im Bereich der Anwendungsschicht entsteht. Anschließend werden nur die "reinen" Sensor- und Steuerungsdaten gesendet. Hinzu kommen jedoch die Header der darunterliegenden Schichten. Bei dem Beispiel unter der Nutzung von IPv4 und Ethernet sind dies 8 Bytes in der Transportschicht (UDP Header), 20 Bytes im Network Layer (IPv4 Header) und 14 Bytes auf der Sicherungsschicht (Ethernet). Damit ergibt sich folgende Formel für die Größe an zusätzlich zu den Steuerungs- und Sensordaten übertragenen Daten für x Sekunden bei einer Frequenz von 1000 Hz.

$$f(x) = 2 * 1000 \frac{1}{s} * x * (20\,B + 8\,B + 14\,B) + z + 1\,B$$

$$\Rightarrow f(x) = 84000 \frac{1}{s}\, x\,B + z + 1\,B$$

Bei der Beispielszene, mit z = 14 Bytes sind dies pro Minute (x = 60 s) ein Overhead von:

$$f(x) = 84000 \frac{1}{s} * 60\,s * B + 14\,B + 1\,B = 5040015\,B = 5,04\,MByte$$

Die in dieser Zeit anfallenden Steuerungs- und Sensordaten (jeweils 1024 B Sensordaten und 8 B Steuerungsdaten) d sind:

$$d = 60\,s * 1000 \frac{1}{s} * (1024\,B + 8\,B) = 61920000\,B = 61,92\,MByte$$

Insgesamt werden so pro Minute 66,96 MByte Daten über das Netzwerk versendet. Der entstehende Anteil an Overhead ist bei diesem Beispiel 8,14 %. Um diesen Anteil zu senken, müsste bei der Wahl der Protokolle auf den Schichten unter der Anwendungsschicht angesetzt werden. Der Anteil auf der Anwendungsschicht ist mit einer konstanten Größe sehr gering.

KAPITEL 5

Ergebnisse der Arbeit

5.1 Zusammenfassung und Fazit

Das Ziel dieser Bachelorarbeit ist es, ein Framework für Netzwerkexperimente mit haptischen Daten im MIoT-Lab zu schaffen. Dafür wurde zuerst ein Konzept erarbeitet. In einem zweiten Schritt wurde das Framework zur Umsetzung und Verifikation des Konzepts entwickelt. Für die Nutzung des MIoT-Labs wurde eine Software zur möglichen Integration implementiert. Diese Software erstellt die für die Experimente passenden DES-Cript-Skripte.

Das Framework wurde durch mehrere Experimente getestet und auf die Eignung für die haptische Kommunikation geprüft. Dafür wurde eine Beispielszene in der Robotersimulation v-rep mit einem passenden Steuerungsalgorithmus erstellt. Dies wurde ausführlich dokumentiert und erklärt, wodurch die Nutzung des Frameworks deutlich wird. Dabei hat sich gezeigt, dass die Nutzung des Frameworks einfach ist und die Anforderungen an die haptische Kommunikation eingehalten werden. Dazu zählt die benötigte Frequenz von 1 kHz und eine geringe Latenz von deutlich unter 1 ms. Damit hat sich gezeigt, dass die gewählte Vorgehensweise zielführend war und das Konzept in der Praxis aufgeht.

Ausgangspunkt war die Betrachtung früherer Lösungsansätze. Es wurden Kriterien angewandt, anhand derer diese bereits bestehenden Lösungsansätze miteinander verglichen wurden. Abschließend werden diese mit der hier geschaffenen Lösung verglichen. Es ist möglich, Roboter(-simulationen) über ein frei wählbares Netzwerk zu steuern. Aus Sicht des dafür implementierten Frameworks werden auch die für die haptische Kommunikation nötigen Anforderungen erfüllt. Es können aufgrund der Freiheit des Frameworks sowohl Roboter, als auch verschiedene Robotersimulationen genutzt werden. Das gleiche gilt für die Steuerung, welche ebenfalls frei gewählt werden kann. Unter der Nutzung des MIoT-Labs steht ein leistungsfähiges TBMS zur Verfügung.
Die folgende Tabelle 5.1 stellt den Vergleich mit den schon früher bestehenden Lösungsansätzen anhand der bereits in Kapitel 2 angewandten Kriterien dar.

Auflistung 5.1: Wiederholung der Kriterien für den Vergleich der Lösungen mit der Forschungsfrage dieser Arbeit.

1. Steuerung der Roboter(-simulation) über ein Netzwerk
2. Erfüllung der Anforderungen an haptische Kommunikation
3. Unterstützung von Robotersimulationen
4. Unterstützung realer Roboter
5. Steuerung kann frei gewählt werden (z.B.: Hardware, Algorithmus)
6. Steuerung der Experimente durch TBMS

Kriterium	*1*	*2*	*3*	*4*	*5*	*6*
v-rep	✓	✗	✗	✗	✓	✗
MIoT-Lab	✗	✗	✗	✗	✗	✓
Google Robotics	❍	❍	✓	✓	✓	❍
Mobile Emulab	✓	❍	✗	✓	✓	✓
Diese Arbeit	✓	✓	✓	✓	✓	✓

Tabelle 5.1: Die Tabelle zeigt die Funktionalität ähnlicher Arbeiten. ✓- Ja, ✗- Nein, ❍- Unklar. Die Nummerierung bezieht sich auf die Kriterien aus der Auflistung 5.1.

5.2 Ausblick

Für die Forschung wird es interessant sein, verschiedene Protokolle für die haptische Kommunikation auszuprobieren und näher zu erforschen. So kann möglicherweise eine höhere Zuverlässigkeit als mit UDP erreicht werden und gleichzeitig ein geringerer Overhead als mit TCP. Zukünftig sollte das Framework auch in Kombinationen mit echten Robotern und weiteren Steuerungsmöglichkeiten ausprobiert werden. Dies können Datenhandschuhe oder Joysticks mit Vibrationsmotoren für das Feedback sein.

Literatur

[1] V. Beal. *haptic*. URL: https://www.webopedia.com/TERM/H/haptic.html (besucht am 21. 09. 2019).

[2] E. G. Steinbach, S. Hirche, M. O. Ernst, F. Brandi, R. Gopal Chaudhari, J. Kammerl und I. Vittorias. „Haptic Communications“. In: *Proceedings of the IEEE* 100 (2012), S. 937–956.

[3] E. Rohmer, S. P. N. Singh und M. Freese. „V-REP: a Versatile and Scalable Robot Simulation Framework“. In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[4] *Google Robotics*. URL: https://ai.google/research/teams/brain/robotics/ (besucht am 05. 10. 2019).

[5] S. Levine, P. Pastor, A. Krizhevsky und D. Quillen. „Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection“. In: *arXiv e-prints*, arXiv:1603.02199 (März 2016), arXiv:1603.02199. arXiv: 1603.02199 [cs.LG].

[6] *MIoT-Lab*. 2018. URL: https://www.comsys.ovgu.de/MIOT_Lab.html (besucht am 25. 08. 2019).

[7] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci und J. Lepreau. *Mobile Emulab: A Robotic Wireless and Sensor Network Testbed*. 2006. URL: https://www.flux.utah.edu/download?uid=95 (besucht am 25. 08. 2019).

[8] F. Kuipers, R. Kooij, D. De Vleeschauwer und K. Brunnström. „Techniques for Measuring Quality of Experience“. In: *Wired/Wireless Internet Communications*. Hrsg. von E. Osipov, A. Kassler, T. M. Bohnert und X. Masip-Bruin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 216–227.

[9] D. Van Den Berg, R. Glans, D. De Koning, F. A. Kuipers, J. Lugtenburg, K. Polachan, P. T. Venkata, C. Singh, B. Turkovic und B. Van Wijk. „Challenges in Haptic Communications Over the Tactile Internet“. In: *IEEE Access* 5 (2017), S. 23502–23518.

[10] E. G. Steinbach, M. Strese, M. S. A. Eid, X. Liu, A. Bhardwaj, Q. Liu, M. Al-Ja'afreh, T. Mahmoodi, R. Hassen, A. El Saddik und O. Holland. „Haptic Codecs for the Tactile Internet“. In: *Proceedings of the IEEE* 107 (2018), S. 447–470.

[11] A. Aijaz, M. Dohler, A. Aghvami, V. Friderikos und M. Frodigh. „Realizing The Tactile Internet: Haptic Communications over Next Generation 5G Cellular Networks“. In: *IEEE Wireless Communications* PP (Dez. 2015).

[12] J. LaViola. „A discussion of cybersickness in virtual environments“. In: *ACM SIGCHI Bulletin* 32 (Jan. 2000), S. 47–56.

[13] M. Di Luca, T. Machulla und M. Ernst. „Recalibration of multisensory simultaneity: Cross-modal transfer coincides with a change in perceptual latency“. In: *Journal of vision* 9 (Nov. 2009), S. 7.1–16.

[14] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong und J. C. Zhang. „What Will 5G Be?“ In: *IEEE Journal on Selected Areas in Communications* 32.6 (Juni 2014), S. 1065–1082.

[15] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, H. Taoka, H. Tullberg, M. A. Uusitalo, B. Timus und M. Fallgren. „Scenarios for 5G mobile and wireless communications: the vision of the METIS project“. In: *IEEE Communications Magazine* 52.5 (Mai 2014), S. 26–35.

[16] coppelia. *Recommended hardware resources.* 2014. URL: `http://www.forum.coppeliarobotics.com/viewtopic.php?t=1707` (besucht am 01. 10. 2019).

[17] I. Vittorias, J. Kammerl und E. Steinbach. „Perceptual Coding of Haptic Data in Time-Delayed Teleoperation“. In: Apr. 2009.

[18] L. Zhang. „Real-time Coding for Kinesthetic and Tactile Signals“. In: 2016.

[19] N. Suzuki und S. Katsura. *Investigation of transport layer protocols for wireless haptic communication system.* 2011. URL: `https://ieeexplore.ieee.org/document/6147501/` (besucht am 25. 08. 2019).

[20] Coppelia Robotics. *[V-REP, Gazebo or ARGoS? A Robot Simulators Comparison].* URL: `http://lenkaspace.net/tutorials/programming/robotSimulatorsComparison` (besucht am 02. 10. 2019).

[21] L. Pitonakova, M. Giuliani, A. Pipe und A. Winfield. „Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators“. In: *Proceedings of the 19th Towards Autonomous Robotic Systems Conference (TAROS 2018), Lecture Notes in Computer Science* 10965 (2018), S. 357–368.

[22] CoppeliaRobotics. *V-REP API framework.* URL: `http://www.coppeliarobotics.com/helpFiles/en/apisOverview.htm` (besucht am 19. 08. 2019).

[23] M. Güneş, F. Juraschek, B. Blywis und O. Watteroth. „DES-CRIPT - A Domain Specific Language for Network Experiment Descriptions“. In: *Inproceedings of the International Conference on Next Generation Wireless Systems (NGWS) 2009.* Melbourne, Australia, Dez. 2009.

[24] CoppeliaRobotics. *Writing code in and around V-REP.* URL: `http://www.coppeliarobotics.com/helpFiles/en/writingCode.htm` (besucht am 19. 08. 2019).

[25] CoppeliaRobotics. *Plugins.* URL: `http://www.coppeliarobotics.com/helpFiles/en/plugins.htm` (besucht am 19. 08. 2019).

[26] CoppeliaRobotics. *C++ plugin framework for V-REP.* 2019. URL: `https://github.com/CoppeliaRobotics/v_repPlusPlus` (besucht am 17. 08. 2019).

[27] J. Quade. *Programmtechnischer Umgang mit Zeiten.* 2011. URL: `https://ezs.kr.hs-niederrhein.de/lectures/ezs/html/x2568.html` (besucht am 17. 08. 2019).

[28] H. Papp, R. Moros und F. Luft. *Grundlagen Regelung*. 2016. URL: http://www.chemgapedia.de/vsengine/vlu/vsc/de/ch/7/tc/regelung/grundlagen/regelung_grundlagen.vlu/Page/vsc/de/ch/7/tc/regelung/grundlagen/regler/p_ctrl.vscml.html (besucht am 13. 08. 2019).

[29] E. Specht. *Mittelwert, Standardabweichung und Streubreite*. 2019. URL: http://hydra.nat.uni-magdeburg.de/praktikum/mean.php (besucht am 09. 10. 2019).

Hiermit versichere ich, dass die vorliegende Arbeit mit dem Titel *Haptische Kommunikation im MIoT-Lab: Ein Framework für hochskalierbare, automatisierte Netzwerkexperimente mit haptischen Daten* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Magdeburg, 4. November 2019

(Johannes Behrens)