Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Masterthesis

Frank Hoch

Konzeption und Entwicklung einer Steuerungsarchitektur für die Roboter-Mensch-Interaktion

*Fakultät Technik und Informatik*
*Department Informations-und Elektrotechnik*

*Faculty of Engineering and Computer Science*
*Department of Information and Electrical Engineering*

Frank Hoch

# Konzeption und Entwicklung einer Steuerungs-architektur für die Roboter-Mensch-Interaktion

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Jochen Maaß
Zweitgutachter: Prof. Dr.-Ing. Ulfert Meiners

Abgegeben am 29. März 2016

**Frank Hoch**

**Thema der Masterthesis**

Konzeption und Entwicklung einer Steuerungsarchitektur für die Roboter-Mensch-Interaktion.

**Stichworte**

Softwareentwicklung, Steuerungsarchitektur, Modulare Steuerung, Robotik, CoBots, , C++, Qt Framework

**Kurzzusammenfassung**

Die vorliegende Arbeit wurde im „Zentrum für Industrielle Robotik“ an der HAW Hamburg durchgeführt und soll als Grundlage für weitere Arbeiten dienen. Sie beschreibt die Konzeption und Entwicklung eines Steuerungsentwurfs in Verbindung mit dem CoBot UR5 der Firma Universal Robots. Neben der Erstellung eines modularen und erweitertbaren Steuerungskerns bestand die Kernaufgabe auch in der Validierung von diesem mit Hilfe exemplarischer Skills.

**Frank Hoch**

**Title of the paper**

Conception and development of a control architecture for the robot-human-interaction.

**Keywords**

Sofware development, control architecture, modular control, robotics, CoBots, C++, Qt Framework

**Abstract**

The present work was carried out at the "Centre for Industrial Robotics" of HAW Hamburg and will serve as a basis for further work and research. It describes the design process and development of a controller design in conjunction with the CoBot UR5 developed by “Universal Robots”. In addition to creating a modular and extendable control core, one of the main tasks was also its validation with help of exemplary skills.

# Inhaltsverzeichnis

**1 Einleitung ..... 1**
1.1. Motivation und Zielsetzung ..... 1
1.2. Inhalt und Herangehensweise ..... 2
**2 Grundlagen ..... 3**
2.1 Kooperativer Arbeitsplatz ..... 3
2.2 Kooperativer Roboter ..... 4
2.2.1 UR5-Roboter ..... 5
2.3 Qt-Framework und Qt-Creator ..... 10
**3 Aufgabenstellung ..... 11**
**4 Lösungsansätze ..... 12**
4.1 C-API ..... 12
4.2 Kontrolle über die Sockets via TCP/IP ..... 12
4.3 Senden von Skriptbefehlen über die Socketschnittstelle ..... 14
4.3.1 Manuelles Testen von Aktivitätsabläufen mit grafischer Testoberfläche ..... 17
4.3.2 Automatisiertes Starten der Motorcontroller ..... 18
4.3.3 Senden von variablen Trajektorien über die Socket Schnittstelle ..... 20
4.4 Remote Senden und Starten von Skriptprogrammen ..... 22
4.4.1 Scriptinjection ..... 22
4.4.2 Client-Server Kommunikation ..... 23
**5 Entwurf der aufgabenorientierten Steuerung ..... 26**
5.1 Modularisierung der Architektur ..... 26
5.2 Modul ProgManager ..... 28
5.3 Modul Settings ..... 29
5.4 Skillmodule ..... 30
5.4.1 Modul Skill als Interface ..... 31
5.4.2 Modul SkillInterpolationsMovement ..... 32
5.5 Validierung ..... 34
5.5.1 Test des Modules SkillInterpolationMovement ..... 34
5.5.2 Abarbeitung von mehreren Skills ..... 36
**6 Fazit und Ausblick ..... 37**
**Abbildungsverzeichnis ..... 38**
**Tabellenverzeichnis ..... 39**
**Verzeichnis der verwendeten Abkürzungen und Formelzeichen ..... 40**
**Literatur- und Quellenverzeichnis ..... 41**

**Anhang .....42**

# 1 Einleitung

Die vorliegende Arbeit entstand im Masterstudiengang Automatisierung an der Hochschule für angewandte Wissenschaften (HAW) in Hamburg im Department „Informations- und Elektrotechnik“ (I+E). Hierbei handelt es sich um einen konsekutiven Masterstudiengang des Bachelorstudiums Elektrotechnik. Die Hochschule und insbesondere das Department I+E ermöglicht es ihren Studenten, verschiedenste Praktika zu absolvieren und in hochschuleigenen Laboren vorlesungsbegleitend praktische Erfahrungen zu den theoretischen Lehrinhalten und Thematiken zu sammeln.
Zielsetzung hierbei ist, die aktuellen Entwicklungen in der Industrie mit den bestehenden Möglichkeit an der Hochschule mit in die Lehrinhalte aufzunehmen und im Umkehrschluss, theoretisches Wissen und Hochschulforschung praxis- und industrienah anzuwenden und auszubauen.
Unter diesem Gedanken entstand in Kooperation mit dem Department „Maschinenbau und Produktion“ (M+P) in den vergangenen Jahren das „Zentrum für industrielle Robotik“ (ZIR) als Erweiterung für das von M+P schon bestehenden Labor für Mechanik und Mechatronik. Im Umfeld dieses Labors wird Studenten die Möglichkeit gegeben, sich im Rahmen der Vorlesungen und Abschlussarbeiten mit den Thematiken der Arbeit der Robotik im Allgemeinen und mit Industrierobotern im Speziellen zu befassen.
Die Ausstattung des Labors umfasst insbesondere unterschiedliche Roboter verschiedener Baujahre, die eine Vielzahl von Ansätzen zur praxisnahen Lehre ermöglichen. Hierbei stehen die Entwicklung und Weiterentwicklung von Steuerungselektronik älterer Roboter, die Navigation von kleinen, autonomen Robotern sowie aktuelle Studien im Umfeld der Optimierung von Montagearbeitsplätzen durch das Hinzufügen eines Kooperativen Roboters im Fokus.

## 1.1. Motivation und Zielsetzung

Im Jahr 2009 veröffentlichte Herr Prof. Dr. Jochen Maaß seine Promotion „*Ein Beitrag zur Steuerungstechnik für parallelkinematische Roboter in der Montage“* [07]. Die Arbeit versteht sich als Beitrag, den Aufwand für den Einsatz parallelkinematischer Roboter zu senken, um das Leistungspotential dieser Kinematik auszuschöpfen und somit die Effizienz der Produktion zu steigern (ebd.). Aus diesem Grund befasst sie sich konkret mit der Entwicklung einer aufgabenorientierten Steuerung für die Handhabung und Montageaufgaben parallelkinematischer Roboter.
Aufbauend auf die bestehenden Strukturen der Hochschule und des Labors sowie in Anbetracht der Notwendigkeit, Erkenntnisse aus Wissenschaft und Forschung in die Industrie zu übertragen ergibt sich die Motivation der vorliegenden Arbeit. Die Zielsetzung beinhaltet, aufbauend auf die Forschung von Herrn Prof. Dr. Maaß, die Konzeption und Entwicklung eines Steuerungsentwurfs für den CoBot UR5 Roboter der Firma Universal Robots. Hierbei steht die Erstellung eines modularen und erweiterbaren Steuerungskerns im Vordergrund, der mit Hilfe exemplarischer Skills im Anschluss validiert werden soll. Damit soll sich die vorliegende Arbeit in den bestehenden Wissenschaftskörper einfügen und einen

Beitrag leisten, die Forschung auf dem Gebiet der Mensch-Maschine Interaktion, insbesondere bei 6-achsigen Industrierobotern, voranzutreiben.

## 1.2. Inhalt und Herangehensweise

Die vorliegende Arbeit teilt sich in 6 Kapitel mit einer zweiten und dritten Gliederungsebene. Dabei werden eingangs die Grundlagen für sowohl die Umsetzung der Ziels selbst gelegt, als auch Begrifflichkeiten und Konventionen definiert (Kapitel 2). Nach einer ausführlichen Beschreibung der Aufgabenstellung (Kapitel 3) in Anbetracht der Zielsetzung werden chronologisch Lösungsansätze präsentiert (Kapitel 4), die aufeinander aufbauend gewählt wurden, um sich der Problemlösung iterativ zu nähern. Abschließend wird neben der umfangreichen Präsentation des eigentlichen Entwurfs der aufgabenorientierten Steuerung (Kapitel 5) ein Fazit gezogen und ein Ausblick gegeben (Kapitel 6).

# 2 Grundlagen

Um zunächst einen anschaulichen Überblick zu verschaffen, werden im Folgenden die für die Arbeit maßgeblichen Techniken, Begrifflichkeiten und Grundlagen erklärt und dargestellt.

## 2.1 Kooperativer Arbeitsplatz

Der Begriff „Industrie 4.0“ steht für eine zunehmende Digitalisierung und Verknüpfung der Informationstechnik mit der Industrie. Kernelement ist hierbei eine fortschreitende Durchdringung der Industrie mit Software. Dies umfasst neben den Produkten und Dienstleistungen selbst auch deren Vernetzung über das Internet sowie interne Netze [01]. Ein bei dieser Arbeit im Fokus stehender Aspekt dieser Entwicklung ist die Automatisierung von Einzelarbeitsplätzen in der Produktionstechnik.

In der Vergangenheit wurden Roboter auf Grund ihrer Eigenschaften hauptsächlich verwendet, um eigenständig Fertigungs- oder Montageaufgaben durchzuführen. Der heutige Stand der Entwicklung ermöglicht darüber hinaus eine Arbeit in direktem Kontakt mit dem Roboter. Die HAW hat zur weiteren Erforschung dieses Themenfeldes zwei neue kooperative Roboter (UR5) der Firma „Universal Robots“ angeschafft. Für den Versuchsstand „Roboter-Mensch-Arbeitsplatz“ wurde ein Exemplar im ZIR auf einer mobilen Werkbank montiert (siehe Abbildung 2.1).

**Abbildung 2.1: Versuchsstand Roboter-Mensch-Arbeitsplatz**

Der mobile „Roboter-Mensch-Arbeitsplatz“ ist an keinen speziellen Lehrinhalt gebunden. Er ermöglicht eine breit gefächerte Erforschung der Zusammenarbeit zwischen Mensch und Roboter. Parallel zur Entstehung dieser Masterarbeit haben sich mehrere Gruppen von Studierenden sowohl innerhalb als auch außerhalb des Labors mit diesem Aufbau beschäftigt. Themenfelder waren unter anderem die Fertigung von „Greiferbacken“ für den am Roboter befestigten Parallelgreifer sowie Studien zum Einsatz des Arbeitsplatzes in der Montage. Darüber hinaus wurde eine Demonstrationsapplikation für die Nacht des Wissens an der HAW erarbeitet. Mit Hilfe dieser wurde ein Programm präsentiert. Es wurde gezeigt, wie der Roboter (UR5) einem Menschen in der Montage diverse Teile anreicht.

## 2.2 Kooperativer Roboter

Der Einsatz von Robotern in der Fertigung und Montage erfordert die Anwesenheit von Sicherheitskonzepte zum Schutz des Menschen. Im ZIR werden aus diesem Grund zur Demonstration unterschiedliche Sicherheitskonzepte angewandt. Das Sicherheitskonzept - „Käfig mit Magnetschalter im Türgriff“ -, welches am Roboter AR6 von Bosch Rexroth (Abbildung 2.2) dargestellt wird, zeigt eine strikte Trennung der Arbeitsbereiche von Roboter und Mensch. Der Roboterarbeitsplatz mit den zwei KR 6 SIXX (KR AGILUS) von KUKA (Abbildung 2.3) stellt ein komplexeres Sicherheitskonzept – „Lichtschranken in Verbindung mit SafetyEye (Decke)“ – dar, welches die Interaktion zwischen Mensch und Roboter bedingt zulässt. Das Auslösen jeglicher Sicherheitsvorrichtung hat die Aktivierung des Not-Aus des jeweiligen Systems oder den automatischen Wechsel des Roboters in einen Modus verringerter Geschwindigkeit zur Folge.

**Abbildung 2.2: Bosch Roboter im Käfig**

**Abbildung 2.3: Kuka Roboter mit Lichtschranken**

Das Wort „CoBot“ (hergeleitet von „cooperative Robot“, deutsch: „kooperativer Roboter“) ist, durch den kontinuierlichen Fortschritt der letzten Jahre, eine feste Begrifflichkeit in der Robotik geworden. Es bezeichnet Roboter, welche im Gegensatz zu den klassischen Industrierobotern in der Montage die Möglichkeit haben, ihre Umwelt durch verschiedene Maßnahmen wahrzunehmen. Dies passiert entweder durch Sensorik in den Gelenken zur Messung der Kräfte, die auf die Robotergelenke wirken oder durch hautähnliche auf dem Gehäuse des Roboters befindliche Sensorik, um ihn für Berührungen empfindlich zu machen.

Als einer der ersten Roboter wurde „Baxter“ (Abbildung 2.4) von „Rethink Robotics“ kommerziell erfolgreich im Geschäftsfeld CoBots in den USA vertrieben. Zu seinen Merkmalen gehört die einfache Bedienbarkeit und Einsetzbarkeit im direkten Umfeld zum Menschen.

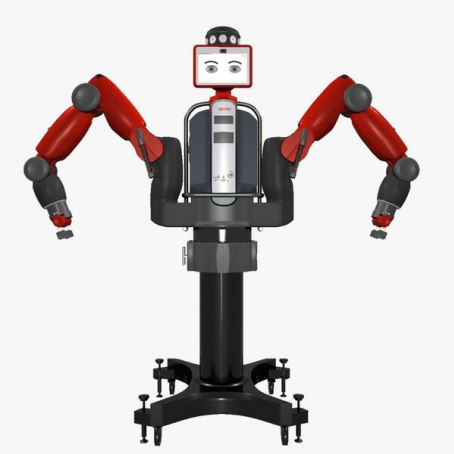


**Abbildung 2.4: Roboter Baxter der Firma „Rethink Robotics“**

### 2.2.1 UR5-Roboter

Die Firma „Universal Robots“ wurde im Jahre 2005 gegründet und vertreibt seit 2009 ihre 6-achsige UR Roboterreihe. Einfach bedienbare Leichtbauroboterarme wie UR3, UR5 und UR10 (die Zahl im Produktnamen steht für die maximal mögliche Zulast in kg) verhalfen der Firma schnell zu einem großen Bekanntheitsgrad. Mittlerweile vertreibt „Universal Robots“ seine Produkte durch ein weltweites Vertriebspartnernetz. Ein weiteres besonderes Merkmal der Roboter der Firma ist das Sicherheitsfeature „Force Sensitivity“ , durch welches der Roboter die Sicherheitszertifizierung „EN ISO 13849:2008 PL d“ erhält [02]. Die Zertifizierung bezeichnet, dass der Roboter ohne zusätzliche Sicherheitsvorkehrungen in der Nähe von Menschen arbeiten kann.

#### 2.2.1.1 Systembeschreibung

Allen von Universal Robots angebotenen Robotersystemen liegt der gleiche Aufbau bestehend aus drei Komponenten zu Grunde:

1. Controllerbox mit 230 V Netzanschluss (Abbildung 2.5)
2. Bedienpanel auch „Teachpanel“ genannt, welches zur Bedienung des Roboters verwendet wird (Abbildung 2.6)
3. Roboterarm, der an der Controllerbox angeschlossen wird (Abbildung 2.7)


**Abbildung 2.5: Controllerbox**

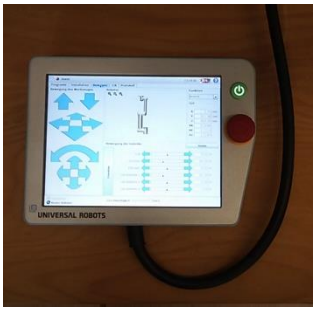



**Abbildung 2.6: Teachpanel**

**Abbildung 2.7: Roboterarm**

**Controller**

In der Controllerbox befindet sich ein Industrie PC mit einem auf Linux basierenden Betriebssystem. Das Programm welches auf dem Industrie PC läuft und für die Steuerung und Regelung des Roboters zuständig ist, wird im weiteren Verlauf als „Controller“ bezeichnet. Der Systemtakt des Controllers beträgt 8 ms. In jedem der 6 Gelenke des Roboters befindet sich jeweils zusätzlich ein Motorcontroller, der für die Steuerung der Motoren zuständig ist. Die Positionen werden in den Motorcontrollern und in dem Controller gleichzeitig berechnet. Zu jedem Systemtakt werden diese Positionen abgeglichen. Sollte der Fall eintreten, dass die Positionen nicht identisch sind, wird das System in einen „safety mode“ versetzt und gestoppt. Es kann nun ausschließlich händisch wieder in den Betriebszustand zurückversetzt werden [03].

**Sicherheit und Kraftmessung**

„Universal Robots“ bietet mit dem System zwei Sicherheitsfeatures an. Zunächst ermöglicht es das System „virtuelle Wänden“ (Bereiche) zu definieren, in denen der Roboter sich nicht bewegen kann oder darf. „Force Sensitivity“ ist ein weiteres Feature, mit dessen Hilfe sich der Grad an Gegenkraft festlegen lässt, ab welchem der Roboter automatisch in den „Safety Mode“ wechseln soll. Vermutlich werden für diese Kraftmessung die Ströme in den einzelnen Achsmotoren ausgewertet. „Universal Robots“ gibt hierzu zum Schutze ihrer Entwicklung keine Einzelheiten bekannt [03]. Zusätzlich sei erwähnt, dass die Kraftmessung eine Toleranz von ca. 10 N hat, sie dient daher primär zur Erfassung von Hindernissen statt als Hilfsmittel bspw. bei der Kraftführung von Aktoren in der Montage.

**Kommunikation**

Zur Kommunikation mit dem Controller bietet „Universal Robots“ Ports an (Tabelle 2.1), über die mit dem Controller via Ethernet kommuniziert werden kann [04].

**Tabelle 2.1: Controller Ports**

| Port-/Socketname | Portnummer | Frequenz |
|---|---|---|
| Dashboard Server | 29999 | - |
| Primary Client Interface | 30001 | 10 Hz |
| Secondary Client Interface | 30002 | 10 Hz |
| Real Time Client Interface | 30003 | 125 Hz |

Die Ports 30001-30003 senden in der angegebenen Frequenz dauerhaft Telegramme, die Informationen über den Zustand des Roboters geben. Auf die Ports 29999-30002 können darüber hinaus Befehle und Programme gesendet werden.

In den folgenden zwei Unterkapiteln werden die wesentlichen Wege zur Interaktion (grafische und textuelle) mit dem System dargestellt.

#### 2.2.1.2 Grafische Benutzeroberfläche Polyscope

Polyscope ist eine auf Java basierende Benutzeroberfläche, die auf dem Teachpanel läuft. Sie ermöglicht es dem Benutzer bei relativ klarer Menüführung einfache Programme, Bewegungs- und Arbeitsabläufe für den Roboter zu erstellen sowie Konfigurationen am System vorzunehmen.

Die Abbildungen Abbildung 2.8 bis Abbildung 2.11 zeigen die wichtigsten Oberflächen bei der Arbeit mit dem UR Roboter. Der Startbildschirm (Abbildung 2.8) stellt die Menüauswahl zur Verfügung, von der aus in unterschiedliche Programmbereiche navigiert werden kann. Der Bewegungsbildschirm (Abbildung 2.9) ermöglicht, zusätzlich zum händischen Führen des Roboters bei gleichzeitiger Bestätigung des „Freedrive“-Knopfes auf der Rückseite des Teachpanels, den Roboter kartesisch oder auf jeder Achse separat, durch Bestätigen der blauen Knöpfe oder alternativ durch Eingabe von Werten in den Textfeldern zu bewegen.

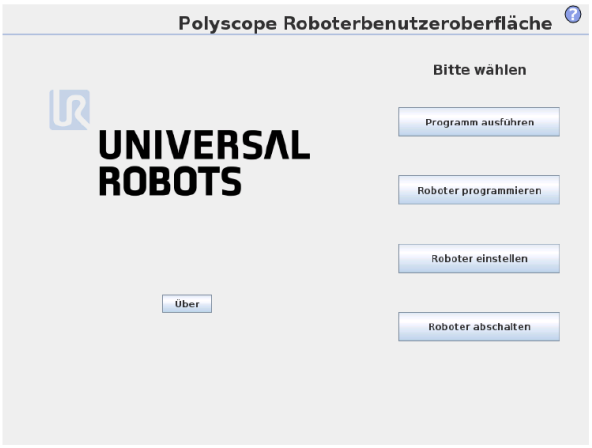



**Abbildung 2.8: Startbildschirm**

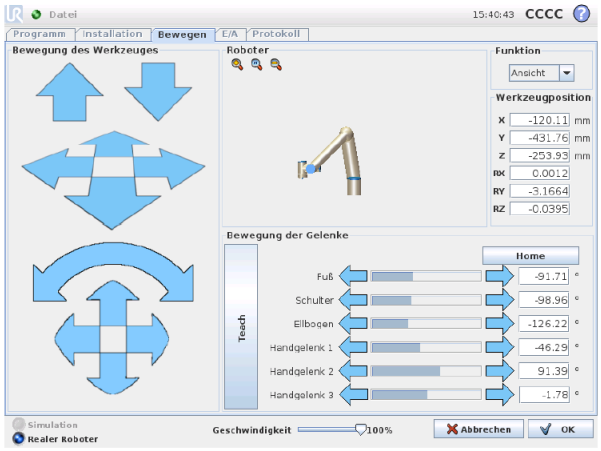



**Abbildung 2.9: Bewegungsbildschirm**

Die Programmoberfläche (Abbildung 2.10) ist die konventionelle Methode, Programme für den Roboter zu erstellen. Der Benutzer hat in dieser Oberfläche in unterschiedlichen Unterreitern die Möglichkeit, auf fertige Aktivitätsbausteine zurückzugreifen, mit deren Verwendung sich ein linear ablaufendes Programm erstellen lässt (erkennbar in linke Spalte in Abbildung 2.10).

Auf dem Bildschirm für Ein-/Ausgänge (E/A)(Abbildung 2.11) hat der Benutzer die Möglichkeit, die Zustände der unterschiedlichen sowohl digitalen als auch analogen Ein- und Ausgänge festzustellen sowie zu setzen.

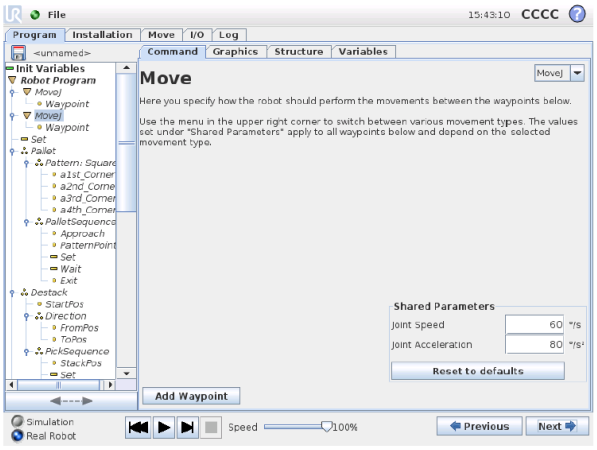



**Abbildung 2.10: Progammoberfläche**

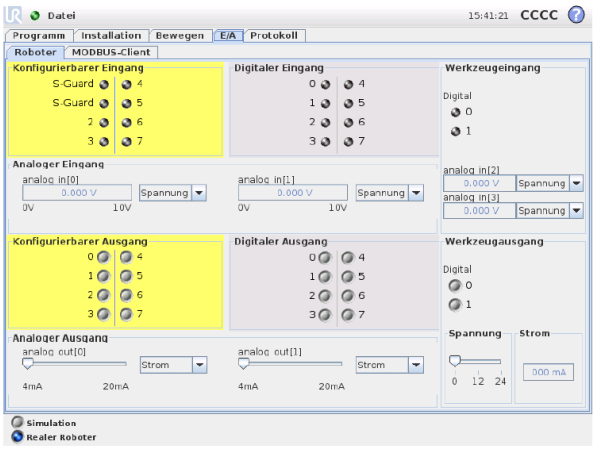



**Abbildung 2.11: Übersicht von Ein-/ Ausgängen**

#### 2.2.1.3 Textuelle Programmierung

Neben der grafischen Programmierung bietet „Universal Robots“ mit C-API und UR-Script zwei Möglichkeiten zur textuellen Programmierung.

**C-API**

Die hauptsächlich zu wissenschaftlichen Zwecken entwickelte C-API ermöglicht es dem Benutzer, auf hardwarenaher Ebene auf dem Controller selbst zu programmieren. Die Informationen zur API finden sich auf der durch ein Kundenpasswort geschützten Supportplattform des Herstellers. Zugriff darauf erhält man nur, soweit man ein Produkt erworben und ein Passwort für den geschützten Bereich angefragt hat. Alle relevanten Dokumente aus diesem Bereich finden sich der Vollständigkeit halber auf der beigelegten CD im PDF-Format. Die API stellt laut Dokumentation sowohl informationsgebende als auch manipulierende Funktionen bereit, mit denen sich der Roboterstatus, die E/As sowie die Bewegungsaufgaben kontrollieren lassen [05].

**Skriptsprache UR-Script**

Für den fortgeschrittenen Benutzer bietet „Universal Robots“ die eigens entwickelte Echtzeit-Skriptsprache „UR-Script“ an [06], die auch der Polyscope Oberfläche zugrunde liegt. In dieser Arbeit wurde die Controllerversion CB3 und Softwareversion SW3.1 verwendet (aktuellste Version: SW3.2). Programme, die über die Polyscope Oberfläche geschaffen werden, können exportiert werden, um im Klartext (.script-Datei) dargestellt zu werden. Ein solches Beispielprogramm ist auf Abbildung 2.12 zu sehen.

bewegungsBeispiel.script:1

```
1  def bewegungsBeispiel():
2    set_tool_voltage(24)
3    set_tcp(p[0.0,0.0,0.13,0.0,0.0,0.0])
4    set_payload(0.5, [0.0, 0.0, 0.07])
5    set_gravity([0.0, 0.0, 9.82])
6    while (True):
7      $ 1 "Roboterprogramm"
8      $ 2 "FahreLinear"
9      $ 3 "Wegpunkt_1"
10     movel([-1.2477, -1.5041, -1.7137, -0.2799, 1.3074, 0.7131], a=1.2, v=0.25)
11     $ 4 "Wegpunkt_2"
12     movel([-1.2703, -1.6446, -1.8804, 0.0293, 1.3311, 0.7040], a=1.2, v=0.25)
13     $ 5 "Halt"
14     halt
15   end
16 end
17
```

**Abbildung 2.12: Beispiel eines einfachen Bewegungsprogrammes in UR-Script**

## 2.3 Qt-Framework und Qt-Creator

Das Qt-Framework ist eine Open-Source C++-Klassenbibliothek, die für plattformübergreifende Programmierung verwendet wird. Ihr Anwendungsschwerpunkt liegt hauptsächlich in der Entwicklung von grafischen Oberflächen. Der Qt-Creator ist die mit dem Qt-Framework verknüpfte Entwicklungsumgebung.

# 3 Aufgabenstellung

Bezugnehmend auf die Promotion von Herrn Maaß [07] soll eine aufgabenorientierte Steuerung entwickelt werden, die das System UR5 von der bisherigen Bedienung über die grafische Oberfläche entkoppelt und eine Schnittstelle für Erweiterungen bildet. Die Steuerung soll die bisherigen Funktionen des Controllers beinhalten und erweitern. Weiterhin soll die Steuerung die Möglichkeit bieten, unterschiedliche (fest-)definierte Aufgaben in möglichst abstrakter Form zu verstehen und abzuarbeiten. Zu solchen Aufgaben zählen einfache Bewegungsbefehle sowie sich kontinuierlich verändernde Trajektorien, die dem Controller in Echtzeit übermittelt werden. Bei den Aufgaben im Allgemeinen und den Bewegungsaufgaben im Speziellen dient die Primitivdefinition aus der Promotion von Herrn Maaß als Entwicklungsansatz.

Die initiale Herangehensweise an oben genannte Aufgabenstellung wurde gegliedert in:

1. Einarbeitung in das CoBot-System UR5 von Universal Robots
2. Analyse der Programmiermöglichkeiten des UR5

Auf Grund im Laufe der Arbeit aufgetretener Probleme, musste die Herangehensweise zwischenzeitlich erweitert und angepasst werden, worauf im weiteren Verlauf eingegangen wird.

# 4 Lösungsansätze

Im Folgenden werden die verschiedenen Lösungsansätze dargestellt, die für die Entwicklung der aufgabenorientierten Steuerung herangezogen wurden.

## 4.1 C-API

Mit der zur Verfügung gestellten C-API sollte auf dem Industrie PC eine Steuerung geschrieben werden, die die bisherige Steuerung (den Controller) „verschachtelt“. Nach Möglichkeit sollte die bestehende Steuerung separiert und modularisiert werden.

In dem Lastenheft für die Beschaffung der zwei UR-5 Roboter über die Firma Lorenscheit Automatisierungstechnik wurde aufgeführt, dass ein System benötigt wird, welches eine C-API zur Verfügung stellt. Universal Robots stellt auf ihrer Webseite [05] die C-API mit Beispielen und einer Dokumentation zur Verfügung. Der Lösungsansatz scheiterte letztlich allerdings daran, dass die C-API sich (Softwarestand SW3.1) nicht kompilieren lässt. Auf Nachfrage des Autors teilte „Universal Robots“ mit, die Entwicklung des Features werde (seit dem letzten großen Software und Controllerupdate) nicht mehr weiterverfolgt und auch aus sicherheitsrelevanten Gründen solle davon abgesehen werden ein Downgrade des Systems vorzunehmen, um mit der C-API arbeiten zu können.

## 4.2 Kontrolle über die Sockets via TCP/IP

Aufgrund des Scheiterns des in Kapitel 4.1 beschriebenen Lösungsansatzes war ein neuer Ansatz erforderlich. Für den zweiten Lösungsansatz musste die Problemstellung geringfügig modifiziert werden. Die Programmierung konnte nicht mehr so hardwarenah erfolgen, wie ursprünglich angedacht war. Diese Tatsache führte zu der Entwicklung des Programmes „UrConn“ auf einem externen Host PC (Macbook OS X) mit Hilfe des Qt-Frameworks in C++, das auf von Universal Robots zur Verfügung gestellten Ports und Sockets aufbaut. Wie bereits in den Grundlagen beschrieben, stellt der Controller vier Ports zur Verfügung, über die mit dem System kommuniziert werden kann.

- Der Dashboard Server [08] ermöglicht es, dem Controller direkte Befehle zu senden. Hierdurch lässt sich der Controller in geringem Umfang steuern sowie Informationen anfordern.
- Über den Primary oder Secondary Client werden vom Controller dauerhaft Telegramme verschickt, die Informationsdaten (Positionswerte, Temperaturen, Sicherheitsmodi) zum aktuellen Status des Controllers und des Roboters liefern. Die Telegramme des Primary Client beinhalten mehrheitlich Systemnachrichten, wohingegen die Telegramme des Secondary Client nahezu alle Informationen des Systems enthalten.
- Das Real Time Client Interface liefert eine spezifizierte Auswahl an Systemdaten, diese aber am schnellsten (8ms Echtzeittakt) von allen Ports.

In einem ersten Schritt wurden die vom Controller gesendeten Daten in ein auslesbares Format gebracht. Zu diesem Zweck wurde ein Programm geschrieben, das sich per TCP/IP mit den jeweiligen Ports auf dem Controller verbindet und die entsprechenden Daten empfängt. Anschließend wurde für jeden Port ein spezifischer Parser entworfen und umgesetzt, der die einzelnen Byte-Ketten oder Zeichenfolgen empfängt, in individuelle Strukturen bzgl. der Telegrammart abspeichert und dem Programm zur Verfügung stellt.

**Entwurf einer einfachen GUI als Testumgebung**

Für die weitere Entwicklung der Software wurde eine einfache Oberfläche (Abbildung 4.1) kreiert, die ausgesuchte Werte aus den Telegrammen darstellt sowie Schaltflächen zur Interaktion zur Verfügung stellt. Die Schaltflächen dienten der Vereinfachung der Kommunikation mit dem System in Echtzeit und des Testens verschiedener Anwendungsfälle.

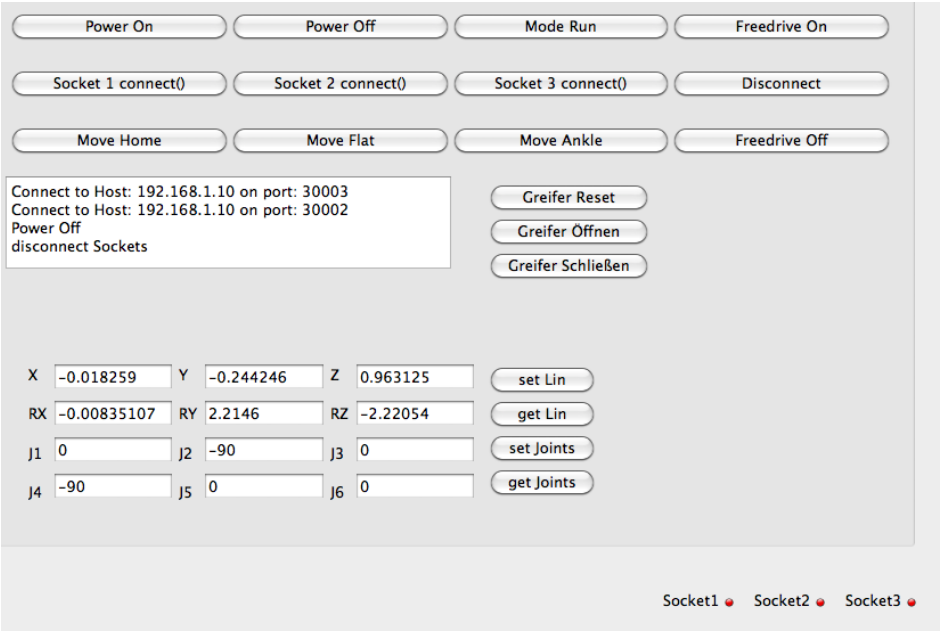



**Abbildung 4.1: Grafische Benutzeroberfläche zu Testzwecken**

## 4.3 Senden von Skriptbefehlen über die Socketschnittstelle

Die Telegramme des Controllers ermöglichen dem Anwender, sich über den Zustand des Roboters zu informieren. Eines der Pakete, die in den Telegrammen des Secondary Clients enthalten sind und übertragen werden, ist der sogenannte „Robot Mode“. Er gibt Auskunft über den momentanen Betriebszustand des Controllers. In der folgenden Tabelle 4.1 sind die verschiedenen Zustände aufgelistet. [09].

**Tabelle 4.1: Robot Modes**

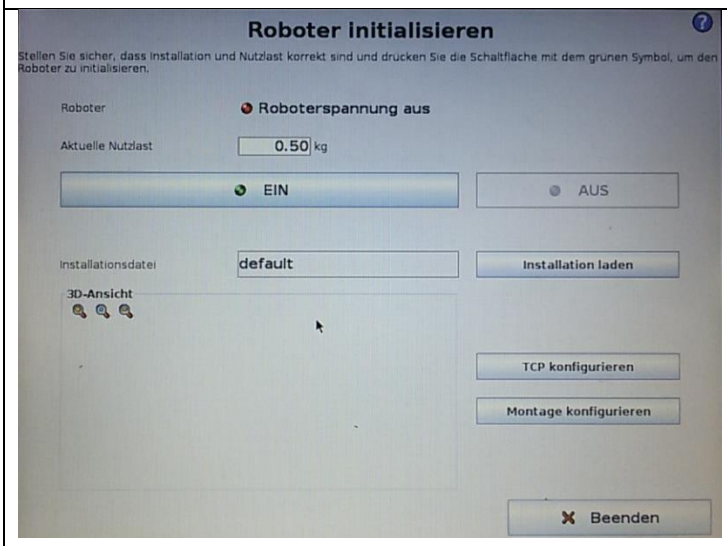
| Mode | Beschreibung |
|---|---|
| 0 | ROBOT_MODE_DISCONNECTED |
| 1 | ROBOT_MODE_CONFIRM_SAFETY |
| 2 | ROBOT_MODE_BOOTING |
| 3 | ROBOT_MODE_POWER_OFF |
| 4 | ROBOT_MODE_POWER_ON |
| 5 | ROBOT_MODE_IDLE |
| 6 | ROBOT_MODE_BACKDRIVE |
| 7 | ROBOT_MODE_RUNNING |
| 8 | ROBOT_MODE_UPDATING_FIRMWARE |

Beim händischen Starten des Roboters über das Teachpanel führen Texthinweise und Schaltflächen durch den Hochfahrvorgang des Roboters (Motorcontroller). Der Ablauf dazu wird durch Bildschirmausschnitte in Abbildung 4.2 bis Abbildung 4.5) illustriert. Zusätzlich zu den Bildschirmausschnitten sind die jeweiligen „Robot Modes“ dargestellt, die der Controller während der Vorgänge nacheinander einnimmt.

Nachdem der Hochfahrvorgang des Industrie PCs per Knopfdruck gestartet und die Oberfläche vollständig geladen wurde, erscheint folgendes Bild (Abbildung 4.2) auf der Oberfläche. Durch Bestätigen des Knopfes „Zum Initialisierungsbildschirm“ gelangt man auf diesen.

| Bildschirmausschnitt | Robot Mode |
|---|---|
|  **Abbildung 4.2: Startbildschirm nach erfolgreichen Hochfahren des Industrie PCs** | 3 |

Durch Bestätigen der Schaltfläche „EIN“ (Abbildung 4.3) werden die Motorcontroller mit Spannung versorgt und hochgefahren.

| Bildschirmausschnitt | Robot Mode |
|---|---|
|  **Abbildung 4.3: Initialisierungsbildschirm / Power Off** | 3 |

Anschließend befinden sich die Motorcontroller im Leerlauf und werden durch ein Bestätigen der Schaltfläche „START“ (Abbildung 4.4) in den „ROBOT_MODE_RUNNING“ versetzt.

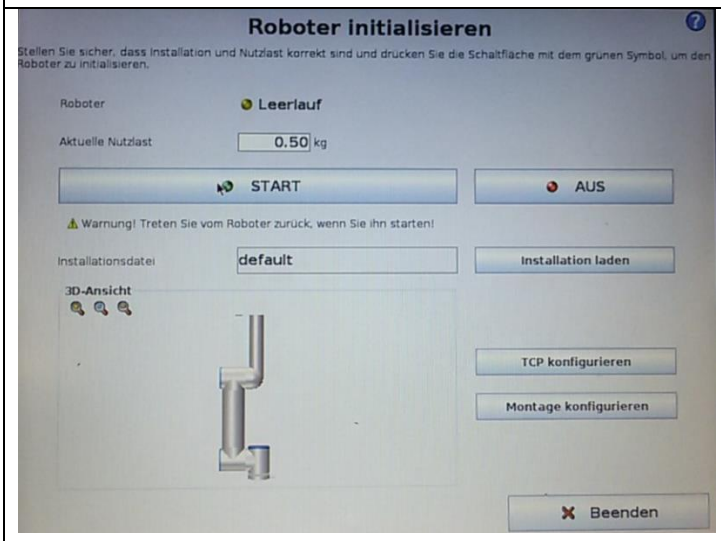
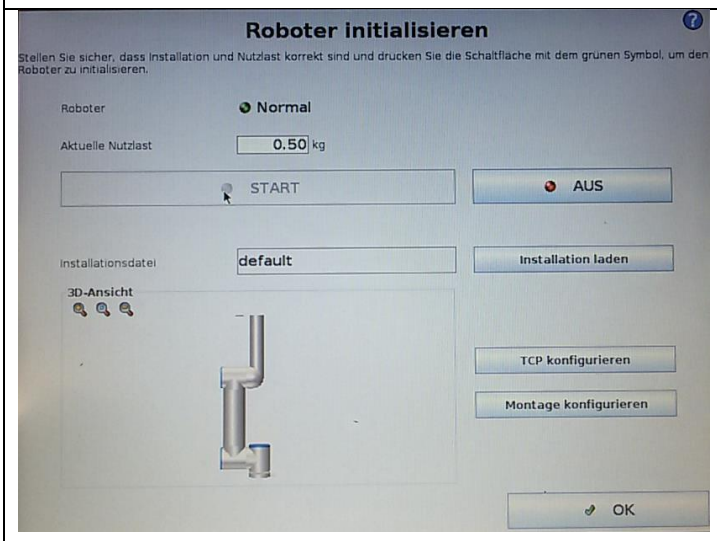
| Bildschirmausschnitt | Robot Mode |
|---|---|
|  <br>**Abbildung 4.4: Initialisierungsbildschirm / Leerlauf** | 5 |

Abbildung 4.5 stellt den erfolgreich abgeschlossenen Hochfahrvorgang des Roboters (Motorcontroller) dar.

| Bildschirmausschnitt | Robot Mode |
|---|---|
|  <br>Abbildung 4.5: Initialisierungsbildschirm / Normal | 7 |

In dieser Phase befindet sich der Controller in einem „Running“-Zustand, in dem er auch Bewegungsbefehle in der UR-Script-Sprache über die Socketschnittstellen annehmen und anschließend ausführen kann.

### 4.3.1 Manuelles Testen von Aktivitätsabläufen mit grafischer Testoberfläche

Um das Senden von Skriptbefehlen und deren Ausführung zu testen, wurden einzelne Schaltflächen auf der grafischen Testoberfläche mit Skriptbefehlen hinterlegt, die anschließend vom Entwicklungsrechner an den Controller über die Socketschnittstelle gesendet wurden.

Für den Vorgang lässt sich die folgende Schrittkette ableiten, die die Benutzerinteraktion darstellt:

1. Verbinden der Sockets auf dem Remoterechner mit dem Controller
2. Senden des „power on“ Befehls über die Socketschnittstelle, um die Motorcontroller mit Spannung zu versorgen
3. Senden des „set robotmode run“ Befehls über die Socketschnittstelle, um die Motorcontroller in den „running“- Zustand zu versetzen

Nachdem der Vorgang fehlerfrei ausgeführt werden konnte, wurde im nächsten Schritt getestet, den Roboter über die Remoteverbindung zu bewegen. Die Script-Dokumentation enthält unterschiedliche Befehle zur Bewegungssteuerung. Die zwei grundlegendsten Befehle hierbei sind „movej“ und „movel“. Die Suffixe „j“ oder „l“ stehen für „joint“ und „linear“. Die Bezeichnung „joint“ steht hier für die synchrone PTP-Interpolation der Gelenke, „linear“ für die lineare Interpolation der Bewegung im kartesischen Werkzeugbezugssystem. Die vollständigen Funktionsaufrufe sind in Tabelle 4.2 beschrieben.

**Tabelle 4.2: Erläuterung der Bewegungsbefehle „movej“ und „movel“**

| movej(q, a=1.4, v=1.05) | | movel(pose, a=1.2, v=0.25) | |
|---|---|---|---|
| q | 6 Gelenkwinkel (rad) | pose | Zielposition (m) |
| a | Vorgegebene Winkelbeschleunigung (rad/s²) | a | Vorgegebene Werkzeugbeschleunigung (m/s²) |
| v | Vorgegebene Winkelgeschwindigkeit (rad/s) | v | Vorgegebene Werkzeuggeschwindigkeit (m/s) |

Ein Beispielaufruf der Funktion „movej“ zur Bewegung des Roboters in die aufgestellte Ausgangsstellung lautet: „movej([0, -1.5707, 0, -1.5707, 0, 0], a=1.0, v=0.25)”

Die Schaltflächen „Move Flat“, „Move Home“ und „Move Ankle“ auf der Testoberfläche (Abbildung 4.1) sind mit dem „movej“ – Befehl und spezifischen Koordinaten hinterlegt, der auf Knopfdruck an den Controller gesendet wird. Der Roboter führte im Test jede dieser Bewegungen ausnahmslos aus.

### 4.3.2 Automatisiertes Starten der Motorcontroller

In Kapitel 4.3.1 wurde die Funktionsweise der Kommunikation mit dem Controller sowie die dazu erforderlichen Befehlsketten beschrieben. Aufbauend darauf sollte der Ablauf nun automatisiert umgesetzt werden.

Aus diesem Grund wurde im Programm ein Automat eingefügt, welcher die Abarbeitung des Startvorgangs der Motorcontroller bei gleichzeitiger Auswertung der Telegramme von dem Controller übernimmt. Äquivalent zu den „Robot Modes“ benötigt auch der Automat unterschiedliche Zustände, die er einnehmen kann. Tabelle 4.3 stellt die als „AutomatStates“ definierten Zustände dar.

**Tabelle 4.3: Zustände des Automaten**

| AutomatState | Beschreibung |
|---|---|
| AS_BOOT | Grundzustand |
| AS_POWER_ON, | Spannung der Motorcontroller wurde zugeschalten |
| AS_IDLE | Motorcontroller im Leerlauf |
| AS_RUNNING | Alle Controller sind betriebsbereit das Programm kann sich Aufgaben zuwenden |
| AS_FINISH | Automat wurde vollständig beendet |
| AS_ERROR | Fehlerzustand |

Der Automat wird nach einmaligen Starten zyklisch ausgeführt. Für die Taktung seiner Ausführung wird das Eingangssignal des Secondary Client Telegrammes verwendet. Hiermit ist sichergestellt, dass die im System hinterlegten Roboterinformationen, bei erneuter Ausführung des Automaten aktualisiert werden.

Der Automat wird durch die Betätigung der Schaltfläche „Socket 2 connect()“ auf der Testoberfläche (Abbildung 4.1) gestartet. Auf dem Aktivitätsdiagramm in Abbildung 4.6 ist der Entscheidungsverlauf dieses Vorgangs dargestellt.

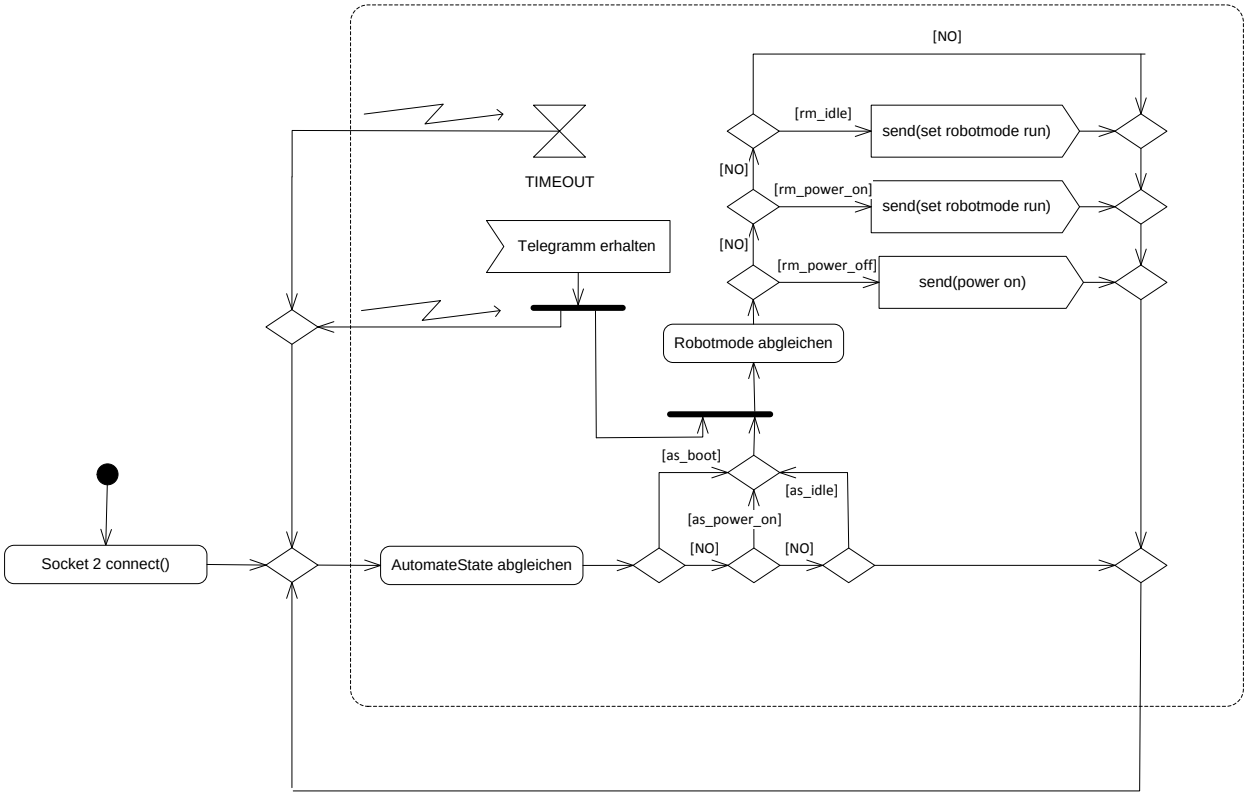



**Abbildung 4.6: PAP zum automatisierten Starten der Motorcontroller**

### 4.3.3 Senden von variablen Trajektorien über die Socket Schnittstelle

Der in Kapitel 4.3.2 beschriebene Test weist nach, dass die beidseitige Kommunikation mit dem Robotercontroller über ein externes System problemlos funktioniert.

Die Umsetzung der Aufgabenstellung erfordert grundlegend die Möglichkeit, variable Trajektorien über die Socketschnittstelle an den Roboter zu senden. Hierfür existiert in der UR-Script-Dokumentation der servoing Befehl „servoj“ (Tabelle 4.4).

**Tabelle 4.4: Erläuterung des servoing Befehls „servoj“**

| servoj(q, a, v, t=0.008, lookahead time=0.1, gain=300) | |
|---|---|
| q | 6 Gelenkwinkel (rad) |
| a | Momentan nicht in Verwendung |
| v | Momentan nicht in Verwendung |
| t | Zeitintervall (s) in welchem neue Punkte versendet werden |
| lookahead time | Glättungsfaktor (s), Bereich(0,03-0,2) |
| gain | Proportionaler Verstärkungsfaktor, Bereich (100-2000) |

Für die konkrete Umsetzung wurde dem Automaten ein weiterer Zustand „AS_SERVOJ“ hinzugefügt und dessen Taktung mit dem Empfang des Real Time Client Telegrammes synchronisiert (8 ms). Innerhalb des Automatenzustandes „SERVOJ“ wurde die aktuelle Position des Roboters mit jedem Takt um 0,01 rad für jede Achse inkrementiert, um eine ausreichend enge Positionsabdeckung zu erreichen. Anschließend wurde dieser neue Positionswert zusammen mit den anderen Parametern (Tabelle 4.5) innerhalb des „servoj“-Befehls auf den Port 30002 gesendet.

**Tabelle 4.5: servoj Parameter**

| Parameter | Wert |
|---|---|
| t | 0,0079 s |
| lookahead time | 0,15 s |
| gain | 100 |

**Beobachtungen während der Ausführung:**

Der Roboter beschrieb nachweislich die vorgegebene Bahn, verursachte jedoch bei der Bewegung ein zuvor nicht aufgetretenes Geräusch und erzeugte spürbare Vibrationen. Abbildung 4.7 zeigt die die Protokollmeldung des Teachpanels nach mehrfachem Senden des „servoj“-Befehls.

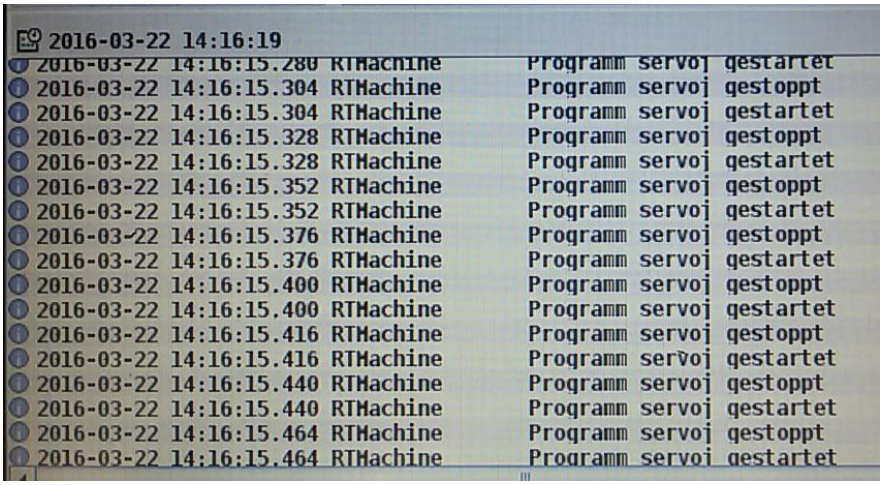



**Abbildung 4.7: Protokollmeldungen nach mehrfachem Senden des „servoj“ - Befehls**

Beim Aufruf der Bewegungsbefehle „movej“ und „movel“ sowie verwandter, ausführender Befehle werden auf dem Protokollbildschirm Meldungen selbiger Form ausgegeben:

- „Programm Befehlsname gestartet“
- „Programm Befehlsname gestoppt“

Diese Meldungen geben Aufschluss über die Zeitpunkte, an denen ein „Programm“ gestartet oder gestoppt wurde. Der Befehl „servoj“ ermöglicht laut der Dokumentation ein „online servoing“ des Roboters. Zur Implementierung und Verwendungsweise dieses Befehls existieren keine weiteren Hinweise.

Folgende Feststellungen lassen darauf schließen, dass der Roboter mit dem Befehl „servoj“ nicht über eine externe Steuerung bewegt werden kann:

- Haptische und akustische Auffälligkeiten bei der Ausführung der Bewegung
- Festgelegtes Zeitintervall (Tabelle 4.5) wird laut Protokoll (Abbildung 4.7) nicht eingehalten

**Schlussfolgerung**

Vor dem Testen wurde erwartet, dass der Roboter eine kontinuierliche, fehlerfreie Bewegung ausführt. Das Starten und Stoppen des gesendeten Befehls scheint jedoch den Controller insofern temporär zu blockieren, als die Ausführung im eingestellten Zeitintervall nicht möglich ist. Das Versenden variabler Trajektorien über die Socketschnittstelle unter Verwendung des „servoj“-Befehls ist folglich auf diese Weise nicht realisierbar.

## 4.4 Remote Senden und Starten von Skriptprogrammen

Kapitel 4.4.1 widmet sich der Umsetzung von remote Senden und Starten von Skriptprogrammen.

Eine in diesem Zusammenhang durchgeführte Recherche ergab, dass der Universal Robots Vertriebspartners Lars Skovsgaard in Singapur [10] zusätzlich zu den Robotern weitere Applikationen und Produkte anbietet, die mit dem System kompatibel verwendet werden können. Eines dieser Produkte ist die Verknüpfung einer 3D Maus (Spacenavigator von 3dconnexion) mit dem Robotersystem und das Führen des Roboters in Echtzeit [11]. Der o.g. Vertriebspartner wurde bzgl. des Problems des Versenden variabler Trajektorien kontaktiert. Als Lösungshinweis wurde daraufhin empfohlen, den „servoj“-Befehl über ein Programm auf dem Roboter selbst auszuführen und ihn nicht über die Socketschnittstelle zu senden.

### 4.4.1 Scriptinjection

In der Onlinedokumentation [13] wird beschrieben, wie über den Primary und Secondary Client auch komplette Skriptprogramm (in Textform), wie sie auch mit der Polyscope Oberfläche erstellt werden, versendet, gestartet und gestoppt werden können. Durch diese Funktionalität ist sichergestellt, dass die Steuerung der Vorgänge, weiterhin extern erfolgt.

Das Programm UrConn wurde um die Funktionalität erweitert, Skriptprogramme, die auf dem Host Rechner abgespeichert sind, einzulesen und zu versenden. Man spricht hierbei auch von einer sogenannten „Scriptinjection“. Im Automatenzustand „AS_RUNNING“ wird das Skriptprogramm Zeile für Zeile auf den Port des Secondary Clients übertragen. Nach vollständigem Empfang eines Programmes wird dieses vom Controller automatisch ausgeführt. Ein auf dem Controller laufendes Skriptprogramm kann durch das Übertragen des Befehls „stop program“ von außen terminiert werden.

Um zu gewährleisten, dass der „servoj“-Befehl wie gewünscht funktioniert, wurde ein einfaches Programm verfasst (Abbildung 4.8) und mit der neu implementieren Funktionalität auf den Controller übertragen. Das Programm „standaloneServoj“ läuft in einer Dauerschleife und inkrementiert in jedem Durchgang den ersten der 6 aktuellen Achswerte. Die angepassten Achswerte werden daraufhin mit dem Befehl „servoJ“ intern an den Controller gesendet.

```
1  def standaloneServoj():
2    WorkPos = [0, 0, 0, 0, 0, 0]
3    while (True):
4      WorkPos = get_actual_joint_positions()
5      WorkPos[0] = WorkPos[0]+0.01
6      servoj(WorkPos, a=1, v=5, t=0.05, lookahead_time=0.0079, gain=100)
7    end
8  end
9
```

**Abbildung 4.8: Skriptprogramm „standaloneServoj“**

Mithilfe obigen Lösungsansatzes führte der Roboter die Bewegung erwartungsgemäß und ohne weitere Auffälligkeiten aus.

### 4.4.2 Client-Server Kommunikation

Mit Hilfe des letzten Test wurde bestätigt, dass der „servoj“-Befehl in einem auf dem Robotercontroller kontinuierlich ausgeführten Skriptprogramm (Abbildung 4.8) die erwartete Funktion erfüllt. Für die Realisierung des Sendens variabler Trajektorien muss eine Kommunikation zwischen dem auf dem Controller laufenden Skriptprogramm und dem auf dem Host PC laufenden Programm UrConn hergestellt werden. Hierzu wurde das Programm UrConn um einen Server erweitert (Portnummer 21000), mit dem sich das auf dem Controller laufende Skriptprogramm verbinden sollte.

Ein Beispiel für die Kommunikation zwischen einem laufenden Skriptprogramm und einem Server befindet sich auf der Universal Robots Supportwebsite[13].

Unter Hinzunahme dieser Informationen wurde das Skriptprogramm „serverSendValuesSections“ (Abbildung 4.9) verfasst, das im Kern auf dem Programm „standaloneServoj“ aufbaut.

Dieses Programm verfügt über folgende zusätzliche Funktionen:

- Herstellung einer Verbindung zu einem Server, Skriptprogramm fungiert als Client
- Senden von Nachrichten an besagten Server
- Empfang von Nachrichten von dem Server

```
def serverSendValuesSections():
  Connected = False
  Running = True
  Counter=0
  ExitCounter=0
  WorkPos = [0, 0, 0, 0, 0, 0]
  TempPoint=[0, 0, 0, 0, 0, 0, 0]
  # Loop for connecting client to server
  while (Connected == False ):
    global Connected = socket_open("192.168.1.22", 21000,"socket0")
    sleep(0.1)
  end
  while (Running):
    socket_send_string("request","socket0")
    TempPoint = socket_read_ascii_float(6,"socket0")
    if (TempPoint[0]!=0):
      global Counter = 0
      while Counter < TempPoint[0]:
        WorkPos[Counter] = TempPoint[Counter+1]
        global Counter = Counter+1
      end
      servoj(WorkPos, a=0.5, v=0.2, t=0.0079, lookahead_time=0.15, gain=100)
    else:
      ExitCounter = ExitCounter+1
    end
    if (ExitCounter>10):
      global Running= False
    end
  end
end
```

**Abbildung 4.9: Skriptprogramm „serverSendValuesSections“**

Die ordnungsgemäße Kommunikation zwischen Client und Server wird durch einen sogenannten „Handshake“ sichergestellt. Nachdem das Skriptprogramm erfolgreich auf den Controller übertragen und gestartet wurde, versucht es sich mit dem Server zu verbinden. Anschließend sendet es den Text „request“(deutsch: Anfrage) an den Host PC, um ihn dahingehend zu informieren, dass eine Position von ihm gefordert wird. Wenn der Automat in dem Programm UrConn durch den Empfang eines Real Time Client Telegramms ausgelöst wurde und der „request“ auf dem Server eingegangen ist, wird eine neue Position berechnet und an den Client übertragen. Die Position muss aus 6 Werten in dem Format „(wert1, wert2, wert3, wert4, wert5, wert6)“ im Klartext bestehen. Bei einer erfolgreichen Übertragung der Positionswerte führt das Skriptprogramm den „servoj“-Befehl aus und bewegt dadurch den Roboter.

Zur Überwachung und Visualisierung des Telegramm- und Nachrichtenfluss zwischen beiden Systemen wurde die als Freeware angebotene Software Wireshark verwendet. In Wireshark muss zunächst eine Netzwerkverbindung ausgewählt und dann die Aufzeichnung gestartet werden. Das Programm loggt automatisch jedes Paket (TCP/IP) mit, welches bis zum Stoppen der Aufzeichnung über diese Verbindung übertragen wird. Abbildung 4.10 enthält einen Ausschnitt dieser Übertragung, welcher repräsentativ für den Datensatz aus ca. 3000 Werten ist.

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 854 | 1.378195 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 856 | 1.378507 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 859 | 1.386056 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 861 | 1.386426 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 864 | 1.393991 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 866 | 1.395303 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 868 | 1.401984 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 871 | 1.402577 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 885 | 1.410214 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 887 | 1.410521 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 890 | 1.418063 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 892 | 1.418548 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 895 | 1.426198 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 897 | 1.426395 | 192.168.1.22 | 192.168.1.10 | TCP | 127 | 21000 → 51731 |
| 900 | 1.434099 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 902 | 1.434420 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 905 | 1.441953 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 907 | 1.442626 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 910 | 1.449992 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 912 | 1.450284 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 915 | 1.457940 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 917 | 1.458403 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |
| 920 | 1.465951 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 922 | 1.466310 | 192.168.1.22 | 192.168.1.10 | TCP | 129 | 21000 → 51731 |
| 925 | 1.474204 | 192.168.1.10 | 192.168.1.22 | TCP | 73 | 51731 → 21000 |
| 927 | 1.474685 | 192.168.1.22 | 192.168.1.10 | TCP | 131 | 21000 → 51731 |

**Abbildung 4.10: Wireshark-Mitschnitt der Übertragung zwischen Client und Server**

**Untersuchung des Wireshark-Mitschnittes:**

Um den Übertragungsausschnitt (Abbildung 4.10) auswerten zu können, sind die in Tabelle 4.6 angegebenen Informationen nötig.

**Tabelle 4.6: Netzwerkeigenschaften Client-Server**

| | **Server** | **UR-Controller** |
|---|---|---|
| **IP-Adresse** | 192.168.1.22 | 192.168.1.10 |
| **Portnummer** | 21000 | 51731 |

Der Ausschnitt beginnt mit der Übertragung des „request“ von dem Client. Darauf folgt in der nächsten Zeile die Antwort des Servers mit den Positionen. Dies passiert nachfolgend immer im Wechsel.

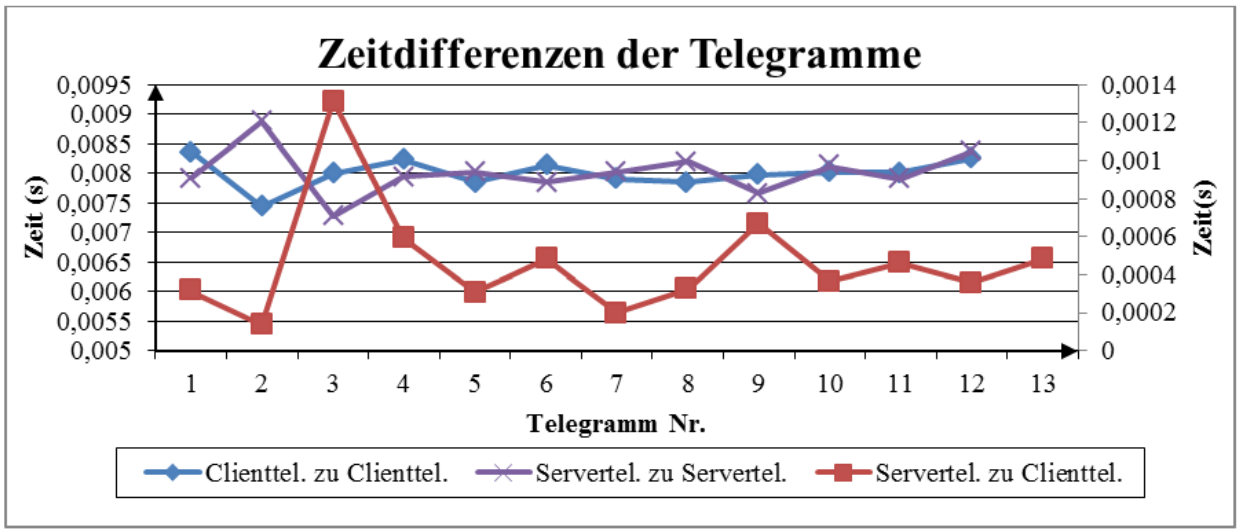



**Abbildung 4.11: Diagramm mit Telegrammzeitdifferenzen**

Zur Vereinfachung der Auswertung des Ausschnittes wurden die Werte in einem Diagramm (Abbildung 4.11) visualisiert. Die linke Achse stellt die Zeitdifferenzen von Clienttelegramm zu Clienttelegramm und von Servertelegramm zu Servertelegram dar. Die rechte Achse stellt die Antwortzeiten der Servertelegramme auf die Clienttelegramme dar. Wie an den Daten zu erkennen ist, wird der Systemtakt von 8 ms eingehalten.. Die Antwortzeit des Servers beträgt im Mittel 0,04 ms.

Damit wurde der Nachweis erbracht, dass die Sendung kontinuierlicher Bewegungsabläufen in einem Systemtakt von 8 ms möglich ist.

# 5 Entwurf der aufgabenorientierten Steuerung

Mit dem Senden von Skriptbefehlen, dem Senden und Starten von Skriptprogrammen, der Auswertung der Telegramme, welche von dem Controller gesendet werden, dem Automaten, welcher sich um Bootvorgänge als auch Abläufe kümmert, sowie der Möglichkeit kontinuierliche Bewegungsaufgaben an den Controller zu senden, sind alle Grundmodule entwickelt worden, welche für die Aufgabenorientierte Steuerung benötigt werden. Der nächste Schritt war die bisherige Programmstruktur zu modularisieren, um damit alle bisherigen Funktionen in eine klare Struktur zu bringen.

## 5.1 Modularisierung der Architektur

Auf dem folgenden Klassendiagramm (Abbildung 5.1) sind alle Module der aufgabenorientierten Steuerung zu sehen. Zur besseren Visualisierung wurden von den einzelnen Modulen nicht alle Attribute und Operationen aufgeführt.

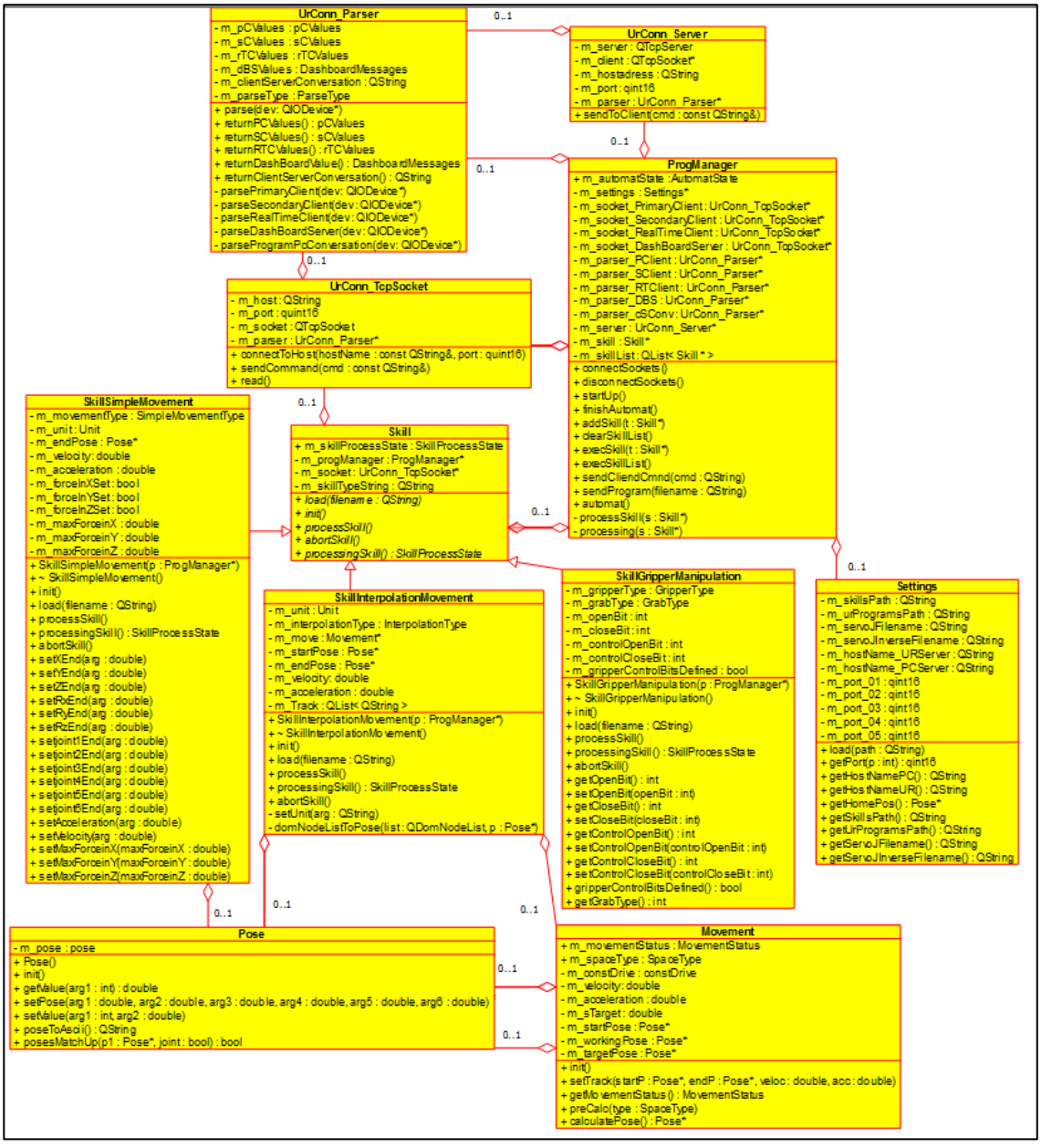



**Abbildung 5.1: Vereinfachtes Klassendiagramm der gesamten Steuerung**

Tabelle 5.1 bietet einen Überblick über die einzelnen Module, die im Klassendiagramm dargestellt sind.

**Tabelle 5.1: Beschreibung der Programmmodule**

| Modul | Beschreibung |
|---|---|
| ProgManager | Kernmodul der Steuerung |
| Settings | Einstellungsmodul für Systemparameter |
| UrConn_TcpSocket | TCP-Socket-Modul zur Socketkommunikation |
| UrConn_Parser | Parser-Modul zum Verarbeiten der Sockettelegramme |
| UrConn_Server | TCP-Server-Modul für die Client-Server-Kommunikation |
| Pose | Positionsmodul zur Verwaltung von Positionswerten |
| Skill | Abstraktes Skillinterface für die Verwaltung von Aufgaben |
| SkillSimpleMovement | Exemplarischer „Bewegungsskill“ zur Demonstration |
| SkillGripperManipulation | Exemplarischer „Greifskill“ zur Demonstration |
| SkillInterpolationMovement | Exemplarischer „Interpolationsskill“ zur Demonstration |
| Movement | Interpolationsmodul zur Berechnung von Trajektorien |

**Der Skill als abstrakte Roboteraufgabe**

Der Begriff „Skill“ (deutsch: Fähigkeit), wurde von der Definition des Manipulationsprimitivs, aus der Promotion von Herrn Prof. Dr. Jochen Maaß inspiriert. Das Manipulationsprimitiv dient als Grundlagenstruktur für die Beschreibung von Roboteraufgaben für die aufgabenorientiere Handhabung und Montage [07]. Es besteht laut Definition mindestens aus folgenden Elementen:

- Bezugssystem
- Beschreibung der Bewegungsaufgabe
- Transitionsbedingung
- Rückgabewerte

Für die Beschreibung von Roboteraufgaben wurde in dieser Arbeit der Begriff „Skill“ gewählt. Der Skill ist nicht nur eine Struktur, welche Informationen zu auszuführenden Aufgaben bereithält, sondern ist in der Lage diese Aufgaben (einfache Bewegung, Interpolation, Greifvorgänge), in Kooperation mit weiteren Modulen, auch selbst auszuführen.

## 5.2 Modul ProgManager

Der ProgManager (Abkürzung für Programmanager) ist das Kernmodul der Steuerung. Er übernimmt die Koordinierung aller anderen Module und im Steuerungskern, mit dem Automaten, die automatisierte Abarbeitung von Aufgaben. Das Modul entstand aus dem zuvor erwähnten Programm UrConn, das zusammen mit der grafischen Oberfläche verwendet worden ist.. Der Automat im Steuerungskern wurde für die Abarbeitungen der Roboteraufgaben um zwei weitere Zustände (Tabelle 5.2) erweitert.

**Tabelle 5.2: Erweiterte Zustände des Automaten**

| AutomatState | Beschreibung |
|---|---|
| AS_BOOT | Grundzustand |
| AS_POWER_ON | Spannung der Motorcontroller wurde zugeschalten |
| AS_IDLE | Motorcontroller im Leerlauf |
| AS_RUNNING | Alle Motorcontroller sind betriebsbereit, das Programm kann sich der Abarbeitung von Roboteraufgaben zuwenden falls welche vorliegen |
| AS_PROCESS | Roboteraufgabe wird gestartet |
| AS_PROCESSING | Roboteraufgabe wird abgearbeitet |
| AS_FINISH | Automat wurde vollständig beendet |
| AS_ERROR | Fehlerzustand |

Für das fertige Programm sollte auf die Verwendung einer grafischen Oberfläche verzichtet werden, daher wurde eine API (Tabelle 5.3) entwickelt, mit derer die Funktionalität der Steuerung des Automaten auch erhalten bleibt.

**Tabelle 5.3: Funktionsaufrufe der API**

| Funktionsaufruf | Beschreibung |
|---|---|
| ProgManager(Settings* stg) | Konstruktoraufruf um das Objekt ProgManager zu erstellen |
| connectSockets() | Erstellen einer TCP/IP Socketverbindung für das Parsen der Telegramme und das Versenden von Skriptbefehlen |
| disconnectSockets() | Trennen der Socketverbindungen (für Reset) |
| startUp() | Starten des Automaten zum automatischen Hochfahren der Motorcontroller bis in den Zustand „AS_RUNNING“ |
| finishAutomat() | Beenden des Automaten |
| addSkill(Skill *t) | Hinzufügen eines Skills „t“ zu einer Skillliste |
| clearSkillList() | Bereinigen der Skillliste |
| execSkillList() | Ausführung der gesamten Skillliste |
| execSkill(Skill *t) | Ausführung eines einzelnen Skills „t“ |

Der Dashboard Server, welcher bereits in Kapitel 4.2 beschrieben wurde, sendet nur Antworten an den ProgManager zurück, wenn über den Socket ein Befehl übertragen wurde. Zur Sicherung von weiteren Systemzuständen wurde der ProgManager um folgende zwei Funktionalitäten erweitert:

- Senden von „security mode“ an den Dashboard Server, um den Sicherheitszustand des Systems zu erfragen. Mit dieser Information lässt sich feststellen, wenn der Roboter durch fehlerhafte Bedienung oder durch eine Kollision in den „PROTECTIVE_STOP“ versetzt wurde. Ist der Roboter in diesem Sicherheitsmodus, lässt er sich nur händisch am Teachpanel entsperren.
- Senden von „running“ an den Dashboard Server, um festzustellen ob sich die Motorcontroller gerade in einer Bewegung befinden.

Diese Befehle wurden abwechselnd in einem Abstand von jeweils 200 ms zueinander gesendet. Zur Verarbeitung der Rückgabewerte musste der Parser des Dashboard Servers auch angepasst werden.

## 5.3 Modul Settings

Das Modul Settings dient als Informationsstruktur, in welcher Einstellparameter für den Betrieb mit dem Roboter hinterlegt sind. Um Flexibilität bei dem Wechsel des Roboters oder der Entwicklungsumgebung zu ermöglichen, oder zu verhindern, dass Einstellparameter in dem Quelltext verändert werden müssen, wurde eine XML-Konfigurationsdatei erstellt, welche die folgenden Systemparameter enthält:

- IP-Adressen des Controllers und des Servers
- Netzwerkports der Controllersockets und des Servers
- Verzeichnispfade
- Dateinamen

Um diese Konfigurationsdatei einzulesen, befindet sich in dem Settingsmodul ein XML-Parser. Das Modul sollte als erstes ausgeführt werden, um die Informationen den anderen Modulen zur Verfügung zu stellen. Dies wird gewährleistet, in dem der Aufruf des Konstruktors des ProgManagers einen Verweis auf eine Settingsstsruktur als Übergabeparameter erwartet. Der Zugriff auf die im Settingsmodul hinterlegten Datensätze erfolgt für jeden Parameter über Zugriffsfunktionen.

## 5.4 Skillmodule

Für die Umsetzung der Skills als Roboteraufgabe wurde ein abstraktes Klasseninterface Skill verfasst. Mit diesem Interface als Basis, können jegliche neue Skills generiert werden. In der Abbildung 5.2 ist das Interface, sowie drei exemplarische Skills (gekürzte Darstellung), welche aus diesem Interface geerbt haben, dargestellt.


**Abbildung 5.2: Klassendiagramm der Skillhierarchie**

Die Formulierung eines Skills erfolgte, wie auch schon bei dem Settingsmodul, außerhalb des Programmes als XML-Datei. In Abbildung 5.3 ist eine solche Skilldatei dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="SimpleMovement">
        <Poses space = "joint">
                <Endpoint unit="RAD">
                        <p1>0.0</p1>
                        <p2>-1.5707963267949</p2>
                        <p3>-1.5707963267949</p3>
                        <p4>-1.5707963267949</p4>
                        <p5>0.0</p5>
                        <p6>0.0</p6>
                </Endpoint>
        </Poses>
        <Attributes>
                <Velocity>1</Velocity>
                <Acceleration>1.5</Acceleration>
        </Attributes>
</Skill>
```

**Abbildung 5.3: Beispiel eines Bewegungsskills im XML-Format**

### 5.4.1 Modul Skill als Interface

Der Entwurf eines Interfaces hat den Vorteil, dass eine Struktur und damit auch eine Funktionalität vorgegeben werden, auf welche sich weitere Ableitungen beziehen können und teilweise auch müssen.

**Implementierung des Skillinterfaces**

In dem Modul Skill wurden virtuelle Funktionsprototypen deklariert, welche ausschließlich die Funktion haben, „Richtlinien“ für weitere Skillmodule darzustellen. Diese Funktionen sind in Tabelle 5.4 beschrieben.

**Tabelle 5.4: Funktionsprototypen**

| Funktionsprototyp | Beschreibung |
|---|---|
| virtual void load(QString filename)=0 | Einlesen der XML-Datei, welche die Informationen zu dem Skill enthält |
| virtual void init(void)=0 | Initialisieren aller Membervariablen |
| virtual void processSkill()=0 | Berechnen von Werten welche zum Ausführen der Roboteraufgabe notwendig sind, sowie Starten der Ausführung der Roboteraufgabe |
| virtual SkillProcessState processingSkill()=0 | Sicherstellung, dass die Aufgabe gemäß den Anforderungen ausgeführt und beendet wird |
| virtual void abortSkill()=0 | Abbruch der Roboteraufgabe |

Die Funktionsprototypen process() und processing() stellen das Gegenstück für die im Automaten neu hinzugefügten Zustände „AS_PROCESS“ und „AS_PROCESSING“dar.
Jede von dem Interface abgeleitete Klasse muss diese Funktionsprototypen neu definieren. Einerseits um die Abarbeitung der Programme in Verbindung mit dem Automaten zu gewährleisten, andererseits um zu verhindern, dass beim kompilieren Fehlermeldungen ausgegeben werden. Zusätzlich zu den Funktionsprototypen stellt das Interface noch weitere, schon definierte Funktionen zur Verfügung um die Kommunikation der Skills über die Sockets zu ermöglichen.

### 5.4.2 Modul SkillInterpolationsMovement

Eine der Kernaufgaben dieser Arbeit war das Versenden von variablen Trajektorien in Echtzeit. In den letzten Kapiteln wurde erläutert wie diese Funktionalität programmtechnisch umgesetzt werden konnte. Das Modul SkillInterpolationsMovement wurde explizit für diese Aufgabe entwickelt um zu zeigen wie die Realisierung dieser Aufgabe mit dem Skillinterface von statten geht. Das Modul ist bereits dazu fähig, unterschiedliche Werte zur Laufzeit zu übertragen, für die Berechnung einer Trajektorie wurde die zusätzliche Klasse „Movement“ geschrieben. Diese Klasse wurde entwickelt um eine Trajektorie zwischen zwei fest definierten Punkten zu generieren welche mit einem trapezförmigen Geschwindigkeitsprofil abgefahren werden soll.

**Implementierung der Funktionsprototypen**

Die Funktion „load()“ muss für jeden abgeleiteten Skill spezifisch geschrieben werden, da bei dem Entwurf eines Skills bis auf den äußersten Rahmen der XML-Datei, jegliche Freiheit gelassen wird. In diesem Fall wurden aus der XML-Datei folgende Werte übertragen:

1. Bezugssystem (joint/linear)
2. Einheit (Radian/m)
3. Startpunkt und Endpunkt
4. Geschwindigkeit und Beschleunigung

In der der Funktion „process()“ werden folgende Schritte durchgeführt:

1. Überprüfung der Werte und Einheiten welche aus der XML-Datei eingelesen worden sind.
2. Festlegung von Bewegungsparametern (lineare Bewegung/ Achsbewegung) und Berechnung von Hilfsparametern, welche für die Erstellung der Trajektorie notwendig sind.
3. Übertragen des Skriptprogrammes an den Controller und Anstoß zum Zustandswechsel („AS_PROCESSING“) des ProgManagers.

Die Funktion „processing()“ beschäftigt sich ausschließlich damit, dem Server bei Anfrage neue Werte zuzusenden. Tritt dieser Fall auf, werden die Werte von der Klasse Movement angefordert. Das Modul SkillInterpolationsMovement dient somit nur der Übertragung der Werte. Es kann in zukünftigen Entwicklungen auch dazu benutzt werden, Positionswerte, welche aus anderen Modulen oder auch Netzwerkports kommen, zu übertragen.

**Klasse Movement**

Die Klasse wurde entwickelt um eine Trajektorie zwischen zwei fest definierten Punkten zu generieren. Der Interpolation lag der Versuch zu Grunde, die Trajektorie mit einem trapezförmigen Geschwindigkeitsprofil abzufahren. Dafür wurde festgelegt, dass die Beschleunigung die Größe ist an der sich die Bewegung orientiert. Die Geschwindigkeit wurde somit eine variable Größe, welche bspw. beim Auftreten von zu kurzen Strecken verringert werden musste.

Die Kernfunktionen der Movementklasse sind die Functionen „preCalc()“ und „calculatePose()“. Die Funktion preCalc(), welche in der Funktion „process()“ aufgerufen wird, berechnet alle Werte, welche für die Bewegung verwendet werden. In der Funktion „calculatePose()“ wird die Trajektorie schlussendlich berechnet. Für jeden der 6 zu berechnenden Werte ist die Berechnung in drei Teile aufgeteilt. Der Beschleunigungsvorgang, die Strecke mit konstanter Geschwindigkeit und den Bremsvorgang. Fall auf Grund der Definition der Strecke es nicht möglich ist ein Trapezprofil zu fahren, wird das Profil automatisch auf ein Dreiecksprofil umgestellt.

## 5.5 Validierung

Um abschließend die entworfene aufgabenorientierte Steuerung zu validieren, wurden zwei unterschiedliche Testszenarien entworfen, die von dem Programm abgearbeitet werden sollten. Als erstes wurde das SkillInterpolationMovement Modul getestet, anschließend eine Kombination aus verschiedenen Skills in einer Liste. Während beider Tests wurde die komplette Ethernetkommunikation via Wireshark aufgenommenm, die Dateien dazu befinden sich auf der CD.

### 5.5.1 Test des Modules SkillInterpolationMovement

Das Versenden von variablen Trajektorien in Echtzeit funktionierte bereits innerhalb des Programmes UrConn in Kapitel 4.4.2. Abschließend muss noch gezeigt werden, dass das Modul SkillInterpolationMovement aus einer XML-Datei [A2] Werte einlesen kann um daraus dann eine Bewegungsmuster zu erstellen welches es anschließend abfährt. Da die Werte aus dem Programm Wireshark nicht exportiert werden können, wurde mit Hinzunahme der Debugging Funktionalität des Qt-Creators die Wertepaare während der Bewegung auf die Konsole der Entwicklungsumgebung ausgegeben. In den folgenden zwei Diagrammen Abbildung 5.4 und Abbildung 5.5 sind die Daten der Übertragung visualisiert.

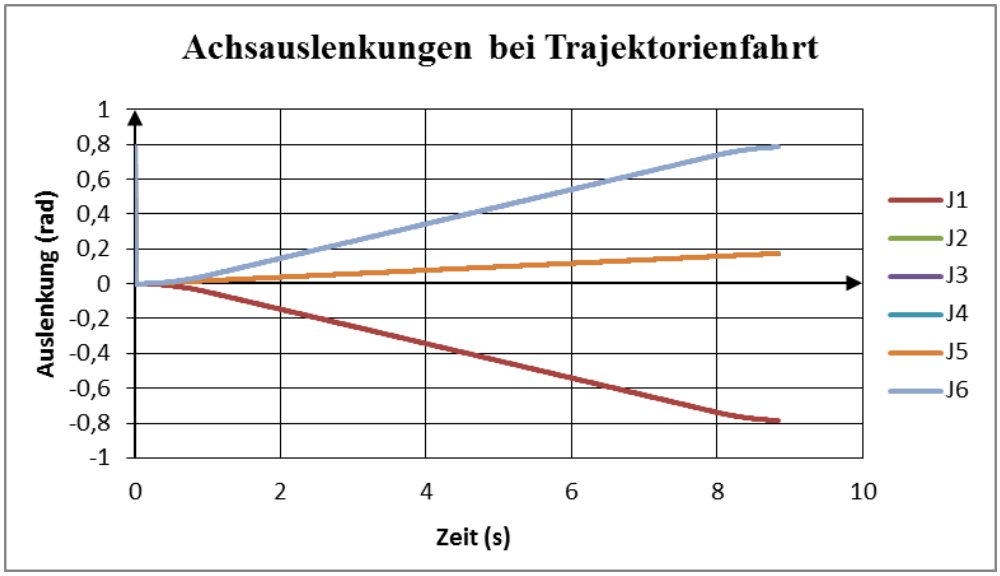



**Abbildung 5.4: Achsauslenkung bei Trajektorienfahrt**

In Abbildung 5.4 kann man das Trapezförmige Geschwindigkeitsprofil an den Achsen mit der größten Auslenkung bereits erahnen, auf Abbildung 5.5 ist es deutlich zu sehen. Dadurch, dass die Winkeldifferenzen bei 4 Achsen gleich waren, sind auf dem Diagramm nur 3 unterschiedliche Profile sichtbar.

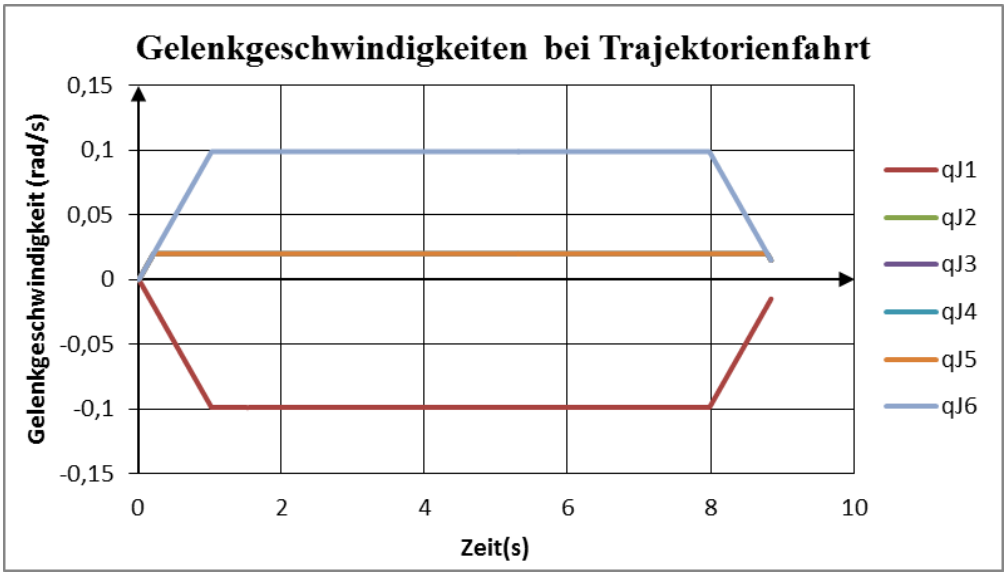



**Abbildung 5.5: Gelenkgeschwindigkeiten bei Trajektorienfahrt**

Des Weiteren lässt sich in Abbildung 5.5 sehr gut sehen, dass die Achsen J1 und J2 die eingestellte Geschwindigkeit (0,1 rad/s) vollständig ausfahren konnten. In der Tabelle 5.5 sind sich zur Darstellung der Genauigkeit der Bewegung die Soll- und Istwerte gegenübergestellt.

**Tabelle 5.5:Soll- und Istwerte der abgeschlossenen Bewegung**

| Zielposition (Grad) | Erreichte Position (Grad) |
|---|---|
| -45 | -44,975 |
| -80 | -80,054 |
| 10 | 9,945 |
| -80 | -80,054 |
| 10 | 9,945 |
| 45 | 44,975 |

### 5.5.2 Abarbeitung von mehreren Skills

Bei der letzten Validierung wurde eine Kombination aus unterschiedlichen Bewegungen (SkillSimpleMovement) und verschiedenen Greifaufgaben (SkillGripperManipulation) in folgender Reihenfolge definiert und abgearbeitet:

1. [A9] (SkillGripperManipulation)
2. [A4] (SkillSimpleMovement)
3. [A5] ( SkillSimpleMovement)
4. [A10] (SkillGripperManipulation)
5. [A3] (SkillSimpleMovement)
6. [A6] (SkillSimpleMovement)
7. [A9] (SkillGripperManipulation)

Dazu werden die einzelnen Skills mit der API-Funktion „addskill()“ in eine Listenform gebracht. Anschließend muss mit der API-Funktion „execSkillList()“ die Ausführung der „Skill-Liste“ angestoßen werden. Wenn der Automat sich im Zustand „AS_RUNNING“ befindet und die Ausführung einer Skill-Liste angefragt wurde, fängt er an jeden Skill nacheinander auszuführen, bis die Liste gelehrt wurde. In der Abbildung 5.6 ist ein Ausschnitt aus der Entwicklungskonsole zu sehen, welcher in einfacher Prototollform die Abarbeitung der Aufgaben darstellt.

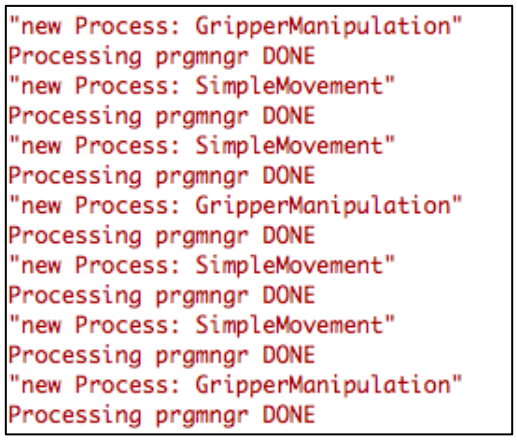



**Abbildung 5.6: Bildschirmausschnitt der Qt-Entwicklungskonsole**

# 6 Fazit und Ausblick

Die Zielsetzung der vorliegenden Arbeit bestand in der Konzeption und Entwicklung eines Steuerungsentwurfs für den CoBot UR5 Roboter der Firma Universal Robots. Schwerpunkt hierbei war die Erstellung eines modularen und erweiterbaren Steuerungskerns, der abschließend mit Hilfe exemplarischer Skills validiert werden sollte.

In der Arbeit wurde neben der Vorstellung der Grundlagen und der Definition unterschiedlicher Begrifflichkeiten und Konventionen strukturiert der Lösungsweg zu o.g. Zielsetzung vorgestellt. Die Hauptproblematik bei der versuchten Implementierung des ersten Lösungsweges bestand darin, dass die C-API Weiterentwicklung durch Universal Robots nicht weiter verfolgt wird. Die Vorteile hardwarenahen Entwickelns konnten und können in der Zukunft aufgrund dessen leider nur bedingt ausgeschöpft werden. Der zweite Lösungsansatz scheiterte letztendlich an der mangelhaften und unvollständigen UR-Script Dokumentation, bevor der dritte Lösungsansatz schließlich zum Erfolg führte. Schlüssel hierfür ist maßgeblich die Möglichkeit und Umsetzung des Remote Sendens und Startens von Skriptprogrammen mittels eines Host PCs.

Anschließend wurde mit der Definition des Skill-Interfaces die Grundlage geschaffen, abstrakte Aufgaben für den Roboter zu entwerfen. In Anbetracht der vorgestellten Lösungsansätze und dem daraus resultierenden, erfolgreichen Entwurf der aufgabenorientierten Steuerung, bildet diese Arbeit einen signifikanten Beitrag zur Weiterentwicklung bestehender Möglichkeiten der Nutzung des CoBot UR5 der Firma Universal Robots. Damit leistet die vorliegende Arbeit primär einen konkreten Beitrag als Grundlage zur weiterführenden Forschung und Entwicklung.

Hierzu zählt für die industrielle Anwendung insbesondere die Erweiterung der in dieser Arbeit entwickelten Steuerung um zusätzliche Module. Konkret kann diese beispielsweise die (1) Einbindung eines Kameramoduls als visuellen Steuerungs- und Navigationsausbau und die (2) Einbindung eines Kraft-Momenten-Sensors am Endeffektor des Roboters umfassen.

Für die Weiterentwicklung der Funktionalität bietet sich aus Sicht des Autors eine (1) Aufnahme handgeführter Trajektorien und die Möglichkeit des anschließenden Abspielens an, um die Anwendung des Roboters und dessen Handling maßgeblich auszubauen.

Softwareseitig besteht ein maßgeblicher Bedarf, die UR-Script Dokumentation zu verbessern um insbesondere die über die grafische Benutzeroberfläche hinausgehende Forschung zu vereinfachen.

# Abbildungsverzeichnis

Abbildung 2.1: Versuchsstand Roboter-Mensch-Arbeitsplatz ..... 3
Abbildung 2.2: Bosch Roboter im Käfig ..... 4
Abbildung 2.3: Kuka Roboter mit Lichtschranken ..... 4
Abbildung 2.4: Roboter Baxter der Firma „Rethink Robotics“ ..... 5
Abbildung 2.5: Controllerbox ..... 6
Abbildung 2.6: Teachpanel ..... 6
Abbildung 2.7: Roboterarm ..... 6
Abbildung 2.8: Startbildschirm ..... 8
Abbildung 2.9: Bewegungsbildschirm ..... 8
Abbildung 2.10: Progammoberfläche ..... 8
Abbildung 2.11: Übersicht von Ein-/ Ausgängen ..... 8
Abbildung 2.12: Beispiel eines einfachen Bewegungsprogrammes in UR-Script ..... 9
Abbildung 4.1: Grafische Benutzeroberfläche zu Testzwecken ..... 13
Abbildung 4.2: Startbildschirm nach erfolgreichen Hochfahren des Industrie PCs ..... 15
Abbildung 4.3: Initialisierungsbildschirm / Power Off ..... 15
Abbildung 4.4: Initialisierungsbildschirm / Leerlauf ..... 16
Abbildung 4.5: Initialisierungsbildschirm / Normal ..... 16
Abbildung 4.6: PAP zum automatisierten Starten der Motorcontroller ..... 19
Abbildung 4.7: Protokollmeldungen nach mehrfachem Senden des „servoj“ - Befehls ..... 21
Abbildung 4.8: Skriptprogramm „standaloneServoj“ ..... 22
Abbildung 4.9: Skriptprogramm „serverSendValuesSections“ ..... 23
Abbildung 4.10: Wireshark-Mitschnitt der Übertragung zwischen Client und Server ..... 24
Abbildung 4.11: Diagramm mit Telegrammzeitdifferenzen ..... 25
Abbildung 5.1: Vereinfachtes Klassendiagramm der gesamten Steuerung ..... 26
Abbildung 5.2: Klassendiagramm der Skillhierarchie ..... 30
Abbildung 5.3: Beispiel eines Bewegungsskills im XML-Format ..... 31
Abbildung 5.4: Achsauslenkung bei Trajektorienfahrt ..... 34
Abbildung 5.5: Gelenkgeschwindigkeiten bei Trajektorienfahrt ..... 35
Abbildung 5.6: Bildschirmausschnitt der Qt-Entwicklungskonsole ..... 36

# Tabellenverzeichnis

Tabelle 2.1: Controller Ports ..... 7
Tabelle 4.1: Robot Modes ..... 14
Tabelle 4.2: Erläuterung der Bewegungsbefehle „movej“ und „movel“ ..... 17
Tabelle 4.3: Zustände des Automaten ..... 18
Tabelle 4.4: Erläuterung des servoing Befehls „servoj“ ..... 20
Tabelle 4.5: servoj Parameter ..... 20
Tabelle 4.6: Netzwerkeigenschaften Client-Server ..... 25
Tabelle 5.1: Beschreibung der Programmmodule ..... 27
Tabelle 5.2: Erweiterte Zustände des Automaten ..... 28
Tabelle 5.3: Funktionsaufrufe der API ..... 28
Tabelle 5.4: Funktionsprototypen ..... 32
Tabelle 5.5:Soll- und Istwerte der abgeschlossenen Bewegung ..... 35

# Verzeichnis der verwendeten Abkürzungen und Formelzeichen

| | |
|---|---|
| API | application programming interface |
| CoBot | cooperative robot |
| PAP | Programmablaufplan |
| TCP/IP | Transmission Control Protocol / Internet Protocol |
| XML | Extensible Markup Language |
| ZIR | Zentrum für industrielle Robotik |

# Literatur- und Quellenverzeichnis

[01] Sendler, U.: Industrie 4.0: Beherrschung der industriellen Komplexität mit SysLM. Berlin – Heidelberg: Springer Vieweg, 2013

[02] UR5_User_Manual

[03] http://spectrum.ieee.org/automaton/robotics/industrial-robots/universal-robots Aktualisierungsdatum: 01.02.2016

[04] http://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/remote-control-via-tcpip-16496, Aktualisierungsdatum: 01.02.2016

[05] http://support.universal-robots.com/foswiki/pub/University/WebHome/C-API_Documentation_of_functions.pdf, Aktualisierungsdatum: 01.05.2015

[06] The URScript Programming Language, Version 3.1 July 17, 2015

[07] Maaß, J.: Ein Beitrag zur Steuerungstechnik für parallelkinematische Roboter in der Montage. 1. Auflage. Essen: Vulkan Verlag, 2009

[08] http://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/dashboard-server-port-29999-15690/, Aktualisierungsdatum: 15.01.2016

[09] Christiansen, M.: Secondary client communications interface

[10] http://www.zacobria.com, Aktualisierungsdatum: 01.07.2015

[11] http://www.zacobria.com/3dconnexion-zacobria-universal-robots-spacenavigator.html, Aktualisierungsdatum: 01.07.2015

[12] Oestergaard, E.: RemoteAccess, Aktualisierungsdatum: 01.07.2015

[13] Flemming, K.: EthernetCommunicationFromScript, Aktualisierungsdatum: 01.07.2015

# Anhang

## A Skills

[A1] Skill1.xml – Interpolation

[A2] Skill2.xml - Interpolation

[A3] Skill3.xml - SimpleMovement

[A4] Skill4.xml - SimpleMovement

[A5] Skill5.xml - SimpleMovement

[A6] Skill6.xml - SimpleMovement

[A7] Skill7.xml - GripperManipulation

[A8] Skill8.xml - GripperManipulation

[A9] Skill9.xml - GripperManipulation

[A10] Skill10.xml – GripperManipulation

## [A1] Skill1.xml - Interpolation

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="InterpolationMovement">
	<Poses space = "linear">
		<Startpoint unit="MM">
			<p1>100.0</p1>
			<p2>0.0</p2>
			<p3>400.0</p3>
			<p4>0.0</p4>
			<p5>0.0</p5>
			<p6>0.0</p6>
		</Startpoint>
		<Endpoint>
			<p1>200.0</p1>
			<p2>0.0</p2>
			<p3>500.0</p3>
			<p4>0.0</p4>
			<p5>0.0</p5>
			<p6>0.0</p6>
		</Endpoint>
	</Poses>
	<Attributes>
		<Velocity>0.5</Velocity>
		<Acceleration>0.5</Acceleration>
	</Attributes>
</Skill>
```

## [A2] Skill2 .xml - Interpolation

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="InterpolationMovement">
	<Poses space = "joint">
		<Startpoint unit="DEG">
			<p1>0.0</p1>
			<p2>-90.0</p2>
			<p3>0.0</p3>
			<p4>-90.0</p4>
			<p5>0.0</p5>
			<p6>0.0</p6>
		</Startpoint>
		<Endpoint>
			<p1>-45.0</p1>
			<p2>-80.0</p2>
			<p3>10.0</p3>
			<p4>-80.0</p4>
			<p5>10.0</p5>
			<p6>45.0</p6>
		</Endpoint>
	</Poses>
	<Attributes>
		<Velocity>0.1</Velocity>
		<Acceleration>0.1</Acceleration>
	</Attributes>
</Skill>
```

## [A3] Skill3.xml – SimpleMovement

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="SimpleMovement">
	<Poses space = "linear">
		<Endpoint unit="M">
			<p1>0.03</p1>
			<p2>-0.60</p2>
			<p3>0.6</p3>
			<p4>0.65</p4>
			<p5>2.19</p5>
			<p6>-1.5</p6>
		</Endpoint>
	</Poses>
	<Attributes>
		<Velocity>0.05</Velocity>
		<Acceleration>0.05</Acceleration>
	</Attributes>
</Skill>
```

## [A4] Skill4.xml – SimpleMovement

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="SimpleMovement">
	<Poses space = "joint">
		<Endpoint unit="RAD">
			<p1>0.0</p1>
			<p2>-1.5707963267949</p2>
			<p3>-1.5707963267949</p3>
			<p4>-1.5707963267949</p4>
			<p5>0.0</p5>
			<p6>0.0</p6>
		</Endpoint>
	</Poses>
	<Attributes>
		<Velocity>1</Velocity>
		<Acceleration>1.5</Acceleration>
	</Attributes>
</Skill>
```

## [A5] Skill5.xml – SimpleMovement

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="SimpleMovement">
	<Poses space = "joint">
		<Endpoint unit="DEG">
			<p1>-71.49</p1>
			<p2>-86.18</p2>
			<p3>-98.190</p3>
			<p4>-16.04</p4>
			<p5>74.91</p5>
			<p6>40.86</p6>
		</Endpoint>
	</Poses>
	<Attributes>
		<Velocity>1</Velocity>
		<Acceleration>1.5</Acceleration>
	</Attributes>
</Skill>
```

## [A6] Skill6.xml – SimpleMovement

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="SimpleMovement">
	<Poses space = "joint">
		<Endpoint unit="DEG">
			<p1>0.0</p1>
			<p2>-90</p2>
			<p3>0.0</p3>
			<p4>-90</p4>
			<p5>0.0</p5>
			<p6>0.0</p6>
		</Endpoint>
	</Poses>
	<Attributes>
		<Velocity>1</Velocity>
		<Acceleration>1.5</Acceleration>
	</Attributes>
</Skill>
```

## [A7] Skill7.xml - GripperManipulation

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="GripperManipulation">
	<Grabtype grippertype="1">open</Grabtype>
	<OperationBits>
		<close>0</close>
	</OperationBits>
	<ControlBits function= "OFF">
		<open>3</open>
		<closed>4</closed>
	</ControlBits>
</Skill>
```

## [A8] Skill8.xml - GripperManipulation

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="GripperManipulation">
	<Grabtype grippertype="1">close</Grabtype>
	<OperationBits>
		<close>0</close>
	</OperationBits>
	<ControlBits function= "OFF">
		<open>3</open>
		<closed>4</closed>
	</ControlBits>
</Skill>
```

## [A9] Skill9.xml - GripperManipulation

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="GripperManipulation">
	<Grabtype grippertype="2">open</Grabtype>
	<OperationBits>
		<open>0</open>
		<close>1</close>
	</OperationBits>
	<ControlBits function= "OFF">
		<open>3</open>
		<closed>4</closed>
	</ControlBits>
</Skill>
```

## [A10] Skill10.xml - GripperManipulation

```
<?xml version="1.0" encoding="UTF-8"?>
<Skill type="GripperManipulation">
	<Grabtype grippertype="2">close</Grabtype>
	<OperationBits>
		<open>0</open>
		<close>1</close>
	</OperationBits>
	<ControlBits function= "OFF">
		<open>3</open>
		<closed>4</closed>
	</ControlBits>
</Skill>
```

# Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 29.03.2016

_____

Frank Hoch