Hochschule Darmstadt

- Fachbereich Informatik -

Untersuchung der Offenen Schnittstellen des UR5 Roboters anhand eines Anwendungsbeispiels

Abschlussarbeit zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

vorgelegt von

Andreas Collmann

Referent: Prof. Dr. Horsch
Koreferent: Prof. Dr. Akelbein

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 28.03.2014

Abstract

Um die Vorteile des kollaborativen Arbeitens von Menschen und Robotern anzuwenden, wird im Zuge dieser Arbeit der für die Kollaboration zugelassene Roboter UR5 der Firma “Universal Robots” untersucht. Es wird Untersucht, welche Möglichkeiten diesen Roboter zu programmieren möglich sind. Die Untersuchung erfolgt aufgrund einiger Kriterien, die auf den Einsatz mit kollaboration zielen. Die Schnittstellen des UR5 Roboters werden untersucht und dokumentiert. Am Ende dieser Arbeit wird eine Entscheidungsfindung zusammengefasst, welche Schnittstelle zu welchem Anwendungsfall am besten zu wählen ist. Um dies zu evaluieren, wurde eine Beispielanwendung in jeder Schnittstelle entwickelt. Die Ergebnisse sind am Ende knapp zusammengefasst.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

1.1. Fachliche Umgebung

Hauptaugenmerk dieser Arbeit liegt bei der überprüfung der Möglichkeiten der Roboter-Mensch-Kollaboration, in der Industrie, Medizin, oder Schule. Inwiefern der Mensch mit einem heutigen Roboter mit entsprechenden Richtlinen programmiert werden kann, um in entsprechenden Feldern mit dem Menschen zusammenzuarbeiten.

1.2. Motivation und Ziel des Projekts

In der Industrie werden Roboter in den Fertigungsanlagen eingesetzt. Dies geschieht meist nur in Koordination mit anderen Robotern, jedoch nie kollaborativ mit Menschen. In der Nähe dieser Roboter darf sich kein Mensch aufhalten. Die Roboter sind umhaust, sprich in einem speziellen Bereich abgesichert, damit keine Unfälle passieren können. Auf diese Weise kann man sehr effizient über automatisierte Fließbandstraßen Produkte herstellen.

Wenn jedoch eine sehr filigrane Arbeit gefragt ist, muss das Werkstück von einem Menschen bearbeitet werden, da der Mensch wesentlich bessere Fähigkeiten hat, auf Probleme zu reagieren oder Korrekturen vorzunehmen. In diesem Fall wird die Fließbandstraße unterbrochen. Das Produkt muss aus dem umhausten Bereich gebracht werden, wo es von einem Menschen bearbeitet werden kann.
Für die Prodution wäre es viel sinnvoller und zeitsparender, wenn Roboter für den Menschen so sicher sind, dass keine Trennung zwischen Mensch und Robotern existiert.

In dem Bereich Pflege und Medizin, müssen oft Hebe-Arbeiten ausgeführt werden. Dies führt dazu, dass die Menschen in solchen Berufen im späteren Alltag mit Rückenproblemen oder ähnlichem Leiden leben müssen. Roboter, die eingesetzt werden, um diese Lasten abzunehmen, würden die Arbeit erleichtern und Verletzungen vorbeugen.
Es soll untersucht werden, inwiefern es möglich ist, den UR5 Roboter zu programmieren, um mit Menschen zu kollaborieren.

1.3. Aufgabenstellung

Es soll ein Anwendungsprogramm für alle möglichen Programmierschnittstellen, des UR5 Roboters von Universal Robots entwickelt werden. Dieses Anwendungsprogramm soll als eine Beispielanwendung einer Roboter-Mensch Kollaboration dienen. Diese verschiedenen Programme werden miteinander verglichen. Es soll eine Entscheidungshilfe gegeben werden, für welchen Anwendungsfall welche Schnittstelle am besten geeignet ist.

Die Programmierschnittellen sollen möglichst gut dokumentiert werden.

1.4. Einordnung in die Themenfelder der Informatik

Die Schnittstellen werden mit den etablierten Programmiersprachen C/C++ und Python programmiert. Hinzu kommt noch die eigens von Universal Robots entwickelte URScript Sprache. Da versucht wird, den Roboter von einem anderen Rechner zu steuern, wird auch Netzwerkprogrammierung benötigt. Es muss ein kleines Protokoll entwickelt werden, mit dem der Roboter kommunizieren kann, um Anwenderdaten zu speichern.

2. Grundlagen

2.1. Roboter-Mensch-Kollaboration

Man unterscheidet die Arbeiten mit einem Roboter in mehreren Arten. Wenn Roboter mit anderen Robotern gleichzeitig arbeiten, bezeichnet das als Kooperation zwischen Robotern. Der Mensch ist in diesem Arbeitsumfeld nicht dabei und kann nur von außen Einfluss nehmen.

Darüber hinauss gibt es die Kollaboration zwischen Roboter und Mensch. Hier wird auch eine Unterteilung vorgenommen die unterschiedliche Richtlinien erfordert.

- Sicherheitshalt, wenn der Mensch den Kollaborationsraum betritt
- Dauerhafte Überwachung des Abstands zwischen Mensch und Roboter, der mit reduzierter Geschwindigkeit arbeitet
- Verminderte Geschwindigkeit bei der Führung des Roboters durch den Mensch. Sensoren erfassen die Kräfte, die vom Menschen ausgeführt werden und übertragen sie auf den Roboter
- Beschränkung der im Roboter ausgeführten Energie & Überwachung des Roboters auf Kollision und sofortiges Stoppen ein

2.1.1. Richtlinien

In so gut wie allen Fällen sind Roboter in der Industrie in einem extra abgesicherten Bereich, damit kein Arbeiter sich verletzen kann. Es ist nicht möglich in einem gemeinsamen Arbeitsbereich zu kollaborieren. Damit Menschen im Arbeitsbereich vom Robotern arbeiten dürfen, müssen diese Roboter bestimmte Sicherheitsrichtlinien entsprechen. Der Roboter darf unter keinen Umständen eine lebensbedrohliche Gefahr darstellen.
Die DIN ISO Normen 10218-1 und 10218-2 sind in der Industrie einzuhalten, wenn Roboter mit Menschen kollaborieren.
“Die Norm ISO 10218-1 legt Anforderungen und Anleitungen für die inhärent sichere Konstruktion, für Schutzmaßnahmen und die Benutzerinformation für Industrieroboter fest. Sie beschreibt grundlegende Gefährdungen in Verbindung mit Robotern und stellt Anforderungen,

um die mit diesen Gefährdungen verbundenen Risiken zu beseitigen oder hinreichend zu verringern. ” [DINISO-2012]

2.2. UR5 Roboter

Die dänische Firma Universal Robots hat den leichten UR5 und mittelgroßen UR10 Roboter mit den erfüllbaren Normen hergestellt, um mit diesem Roboter zu kollaborieren. Man kann sich im laufenden Betrieb in der Nähe aufhalten, um Wegpunkte zu teachen oder auch gleichzeitig an einem Werkstück zu arbeiten. Im Folgenden Kapitel werden die Eigenschaften des UR5 Roboters erörtert.

2.2.1. Kinematik

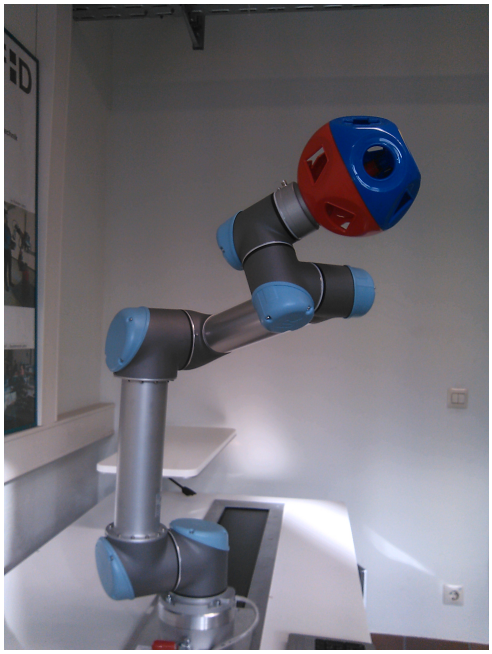

Abbildung 2.1.: Abbildung zeigt den UR5 Roboter von Universal Robots

Der Roboter besitzt sechs Gelenke die ihm ermöglichen einen 360° Arbeitsbereich mit einem Radius von ca. 85 cm zu ermöglichen. Der Roboterarm hat eine Tragfähigkeit von 5 kg. In den Motoren der einzelnen Gelenke sitzt auch gleichzeitig die Steuerung des Gelenks. Die Hardware des Roboters wird von einem Linux Rechner, der sich in der Nähe befindet, angesprochen. Die Festplatte für das System ist eine Speicherkarte, die leicht ausgetauscht werden kann.
Der Linux Rechner besitzt zehn digitale Eingänge, zehn digitale Ausgänge, vier analoge Eingänge, zwei analoge Ausgänge, die zum Steuern des Roboters benutzt werden können. Des weiteren besitzt der Roboter einen Netzwerk Anschluss, über dem eine Verbindung zu einem

oder mehreren Rechnern möglich ist.

Um den Rechner anzusprechen, existiert bei Lieferung ein Touch Tablet(siehe Abbildung C.6), das für das Linux System den visuellen Output gibt. Es ist möglich über USB eine Tastatur anzuschließen, um nicht text mit dem Tocuh Tablet auskommen zu müssen.
Beim Starten des Systems wird auch automatisch die Software für den Roboter gestartet. Die Software nennt sich Polyscope und wurde in Java geschrieben. Diese Software verbindet sich per Transmission Control Protocol / Internet Protocol (TCP/IP) auf den URController(2.4). Ein Server Programm das die Schnittstelle von dem Linux System zu dem Roboter Controller auf dem Rechner herstellt.

2.2.2. Grundlegendes

Die Polyscope Software läuft im normalen Modus und dem administrativen Modus. Der normale Modus ermöglicht, es Programme zu erstellen, laufen zu lassen und Grundeinstellungen vorzunehmen. Außerdem kann die Polyscope Software aktualisiert werden.

System aktualisieren
Zwei Arten von Updates sind hier zu unterscheiden. Zum einen kann das Linux System aktualisiert werden. Dies ist über den Paketmanager des Systems möglich oder wenn man das neueste Image von Universal Robots herunterlädt und das System neu überspielt. Zum Updaten gibt es eine Dokumentation, beiliegend auf der CD.

2.3. Programmierschnittstellen vom UR5

Der UR5 Roboter kann auf drei Ebenen angesprochen werden.

- Polyscope
- URScript
- C-Api

In dieser Arbeit wird versucht über alle diese Ebenen den Roboter Roboter anzusprechen. Es wird außerdem, aufbauend auf URScript, ein eigener Adapter entwickelt, um eine neue Möglichkeit zu untersuchen, den Roboter anzusteuern. Der Adapter wird für die Programmiersprache Python entwickelt. Gründe hierfür werden im Kapitel für diese Schnittstelle erörtert(siehe 2.6)

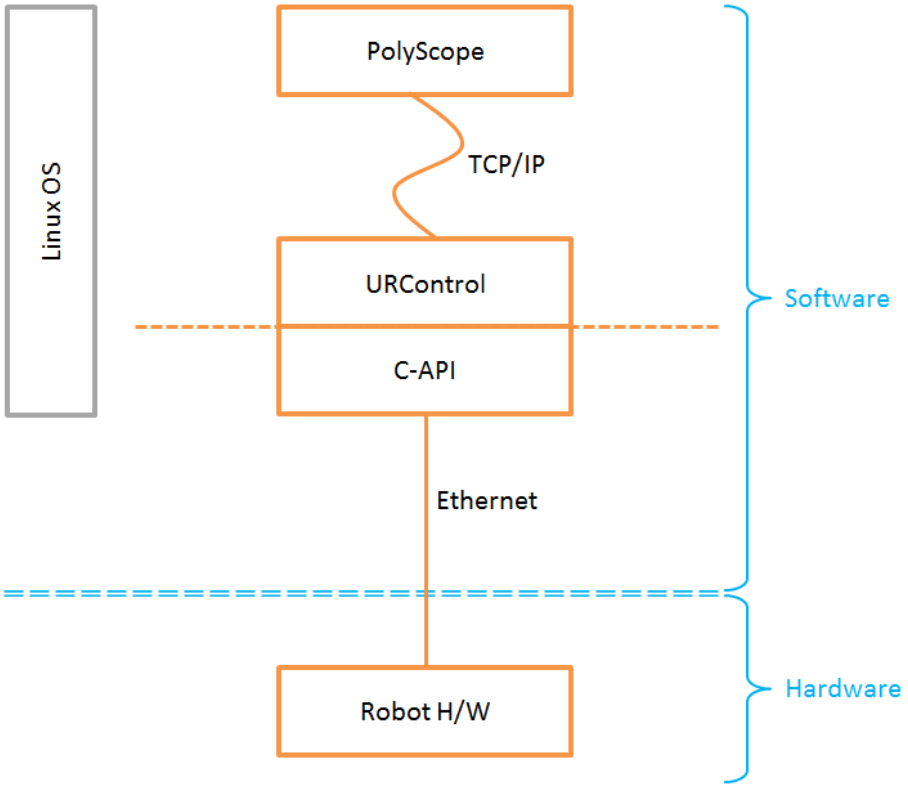

Abbildung 2.2.: Übersicht über die Schichten der bestehenden Software Schnittstellen des UR5 Roboters

2.3.1. Kriterien für die Bewertung der Schnittstellen

Die Schnittstellen werden wie folgt bewertet:

- Programmierbarkeit
- Interaktion mit Programm,
- Möglichkeit zu Debuggen und Testbarkeit
- Aufwendung

Wie schwer ist es, ein Programm für die einzelnen Schnittstellen zu entwickeln? Kann der Mensch das Programm intuitiv bedienen? Wichtig hierbei ist, dass der Mensch mit dem Roboter kommunizieren kann. Dies geschieht am besten, wenn der Mensch nichts Kryptisches was eingeben muss. Er braucht Anwenderfreundliche Programme.

Beim Entwickeln von Programmen ist es wichtig, dass der Entwickler Fehler im Programm

entdeckt, um diese schnell zu beheben. Je größer und komplexer das Programm wird, desto schwieriger wird es, Fehler zu entdecken.

2.4. URController

Der URController ist eine Server Anwendung die auf dem Rechner des Roboters läuft. Dieser Controller dient als Schnittstelle, zu der der Roboter Hardware und der Software, die das Roboterprogramm liefert.

2.4.1. Konfiguration des URControllers

Den URController kann man konfigurieren. Die Konfigurationsdatei ist abgelegt im folgenden Verzeichniss:

```
\root\.urcontrol\urcontrol.conf
```

Beim Starten des Controllers wird diese Konfigurationsdatei eingelesen. Hier werden wichtige Einstellungen vorgenommen, die zu den jeweiligen Modellen der UR5 oder UR10 Serie gehören. Hier ist ein Ausschnitt der Konfigurationsdatei zu sehen

```
[DH]
a = [0.00000, -0.42500, -0.39243,  0.00000,  0.00000,  0.0000]
d = [0.08920,  0.00000,  0.00000,  0.10900,  0.09300,  0.0820]

[Link]
mass = [3.7000, 8.3930, 2.2750, 1.2190, 1.2190, 0.1879]     # Series 3 with tool
gravity = [0, 0, 9.82]          # upright mounting
[Config]
# masterboard_version, 0 = Zero-series, 1 = One-series, 3 = Pause function enabled, 4 =
    first cb2 version, 5 = ur10 support added
masterboard_version = 4

[Hardware]
controller_box_type = 2 # 1=CB1, 2=CB2UR5, 3=CB2UR10
robot_type = 1  # 1=UR5, 2=UR10
robot_sub_type = 1
```

Listing 2.1: Ausschnitt aus der Datei urcontrol.conf zur Vorkonfigurierung des UR5 Roboters

2.4.2. Echtzeit-Schnittstelle

Die Echtzeit-Schnittstelle ist eine TCP/IP Schnittstelle, die im 125Hz Takt Datenpakete an die verbundenen Clients sendet. Diese Schnittstelle kann keine Daten von den Clients empfangen. Wenn man diese Datenpakete auslesen will, muss man die einzelnen Datentypen in dem Paket

parsen. Eine Besonderheit ist noch, dass die Byte-Reihenfolge der Datenpakete im Big-Endian[1] über das Netzwerk übertragen werden. Da der Unix Rechner und der Client Rechner die Byte-Reihenfolge Little-Endian[2] benutzen, muss diese für die einzelnen Datentypen umgewandelt werden. Hierfür wurde in C ein Struct[3] geschrieben und eine Funktion, die das Datenpaket für das Struct in die richtige Byte-Reihenfolge umwandelt.

```
1  struct ur5_data_rci * parse_ur5_realtime_ci(struct ur5_realtime_ci *ur5_rci, char *buf){
2      // ur5_rci points to the struct that will contain the data of the package
3      // buf is the recieved Package from the Real Time Interface
4      ur5_rci = (struct ur5_realtime_ci*) buf;
5
6      // the first 4 Byte are the message length of the package. the rest of the packages are
           8 Bytes long so we can just iterate over all variables in the package
7      ur5_rci->length = ntohl(ur5_rci->length);
8      int i;
9      for (i = 0; (i < (sizeof(ur5_rci->data_union.data_packed)/sizeof(double))); i++){
10         ur5_rci->data_union.data_packed[i] = htobe64(ur5_rci->data_union.data_packed[i]);
11     }
12     return &ur5_rci->data_union.data;
13 }
```

Listing 2.2: Umwandlung der Byte-Order für Packet über die Echtzeit-Schnittstellen

Die in der CD beiliegende Dokumentation beinhaltet ist eine die Beschreibung, wie die Schnittstelle angesprochen wird und wie die Daten benutzt werden, um den Roboter zu analysieren. Im Anhang(C.1) sind Beispiele von Bewegungsprofilen, die von der Echtzeit-Schnittstelle ausgelesen wurden, um zu erfahren wie der URController im Gegensatz zu der Software mit der C-API die Bewegungsprofile berechnet.

2.4.3. Secondary und Primary Schnittstelle

Die Secondary Schnittstelle ist eine TCP/IP Schnittstelle, die in einem 60Hz Takt Nachrichten über den Roboter an verbundene Clients sendet. Die Nachrichten beinhalten Informationen wie z. B. den Roboter Status oder die Positionen der einzelnen Joints.

Zusätzlich kann die Secondary Schnittstelle Befehle von verbundenen Rechnern empfangen. Diese Befehle können URScript Befehle sein. Ein ganzes Programm in URScript geschrieben oder spezielle zugelassene Befehle, die den Roboter Status verändern.

[1] Das Big Engian Format ist die Festlegung der Byte-Reihenfolge, wie das Computersystem Speicherbereiche interpretieren und beschreiben soll. Dieses Format legt fest, dass das höchstwertige Bit an der kleinsten Speicheradresse liegt.

[2] Wie bei Big-Endian Format legt das Little-Endian Format die Byte-Reihenfolge fest. Mit Little-Endian jedoch wird das niedrigstwertenste Bit an die kleinste Speicheraddresse gesetzt.

[3] Ein Struct ist in C/C++ ein Datentyp, der als Container mehrerer variablen verschiender Datentypen dient

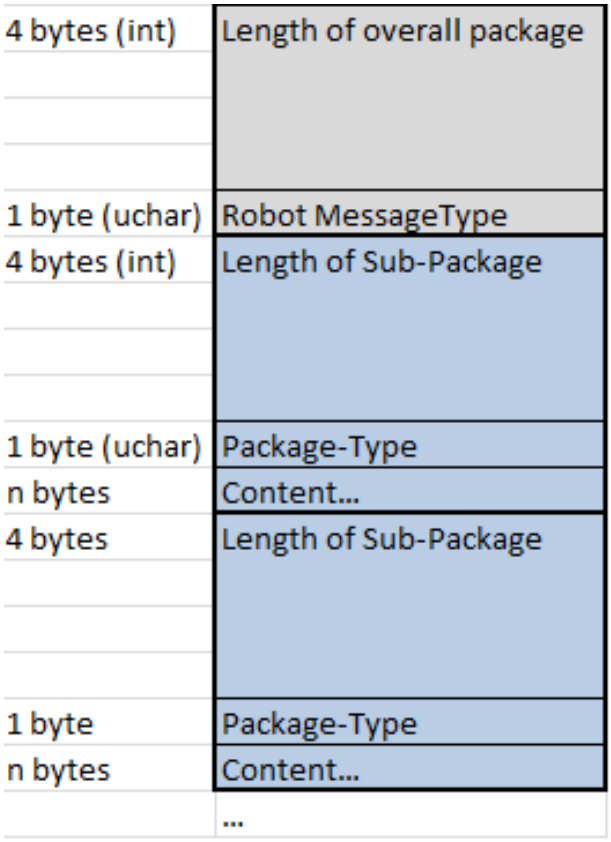

Abbildung 2.3.: Grobe Darstellung wie die Datenpakete von der Secondary/Primary Schnittstelle gesendet werden.

2.4.4. Polyscope

Polyscope ist eine Anwendung, die auf dem Roboter-Rechner läuft. Die Anwendung verbindet sich per TCP/IP auf den URController(2.4) und sendet URScript Befehle an den Roboter um diesen zu steuern. Diese Anwendung wird auf dem Tablet angezeigt. Hierüber kann man per Toucheingabe ein neues URP Programm erstellen. Dieses Programm wird zur Laufzeit in ein Script umgewandelt. Die Polyscope Software schickt nun in Schritten die einzelnen Script-Befehle an den URControl, der diese ausführt. Im Programmbaum kann eingesehen werden, an welchem Schritt sicht das Programm befindet.

2.5. C-API

Die C-API ist von dem Hersteller UR eine zur Verfügung gestellte C Library mit einer Header Datei, die etwaige Funktionen der Library erklärt. Die Header Datei enthält nicht alle Funktionen, somit sind nicht alle zugänglich. Die C-API erlaubt es einen eigenen Controller für den

Roboter zu entwickeln. Der für den Roboter zur Verfügung gestellte Controller mit der Polyscope Software und die Anwendung, die C-API benutzt, können aber nicht gleichzeitig laufen. Es schließen sich also die Programmiersprache URScript und ein eigener Controller zunächst aus. Es könnte ein eigener Controller entwickelt werden, der die Befehle in URScript selbst interpretiert und diese wie bei dem URController ausführt. So könnte man die vorhandene Sprache nehmen und diese sogar erweitern.

2.5.1. Kontrollstruktur

Die C-API ermöglicht es, eine Verbindung zum Roboter zu öffnen und über eigene Funktionen Befehle abzuschicken. Dies erfolgt in einem streng festgelegten Muster.

```
1  while(!endcondition) { // At ROBOT_CONTROLLER_FREQUENCY times per second
2    robotinterface_read_robot_state_blocking();
3    robotinterface_get_actual_positions(&positions);
4     // >>> various calculations <<<
5    robotinterface_command_position_velocity_acceleration( xxx, yyy, zzz);
6    robotinterface_send_robot_command();
7  }
```

Listing 2.3: Beispiel der Kontroll-Struktur

Die Funktion *robotinterface_read_state_blocking()* startet den Bereich, in dem Datenabfragen an den Roboter gestellt werden können. Daten wie z. B. Temperatur der Motoren, der Stand der Gelenke, die Geschwindigkeit der Gelenke etc. In der Dokumentation beiliegend zu dieser Arbeit sind alle Daten noch einmal aufgelistet. Nachdem die Daten abgefragt wurden, kann mit C-API Funktionen Position, Geschwindigkeit und Beschleunigungswerte übermittelt werden, die der Roboter durch seinen Regler auszuführen versucht.
Es können jedoch keine Wegpunkte festgelegt werden, die dann automatisch vom Roboter angefahren werden. Dies muss der Entwickler selbst berechnen. Es gibt mehrere Verfahren, in dieser Arbeit sind Point to Point (PTP)-Verfahren und Linear Verfahren(siehe 2.5.2) getestet worden.

Zum Abschluss wird die Funktion *robotinterface_send()* aufgerufen, die dafür sorgt, dass der Acht-Millisekundentakt eingehalten wird und die Befehle an den Roboter weitergeleitet werden. Falls die acht Millisekunden überschritten werden, wird der Roboter in einen Sicherheitsmodus gesetzt und angehalten.
Wenn so etwas im URController passiert, kann der Anwender diesen wieder abschalten sobald alles in Ordnung ist. Dies muss mit der C-API selbst geschrieben werden. Die C-API liefert hierfür auch Funktionen. Dass die richtigen Richtlinien aber auch eingehalten werden, muss von dem Wechsel des Sicherheitsmodus in den normalen Modus eine Benutzerabfrage verlangt werden.

2.5.2. Bewegungsprofile

In der Robotik gibt es drei Verfahren, wie man den Roboter zwischen zwei Punkten bewegen kann.

- PTP
- Linear
- Circular

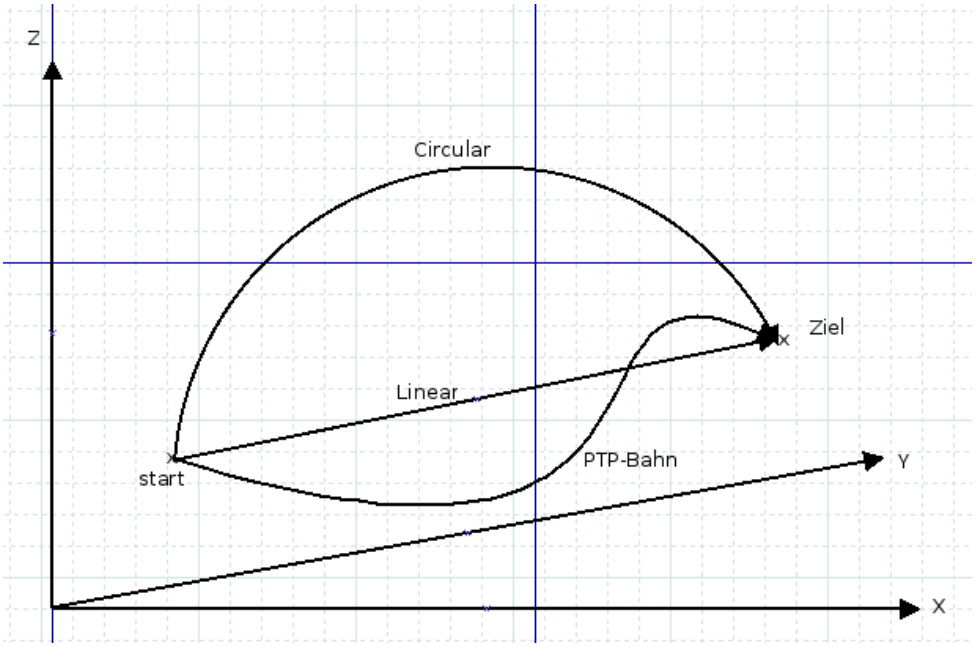

Abbildung 2.4.:]
Übersicht über drei verschiedenen Bewegungsarten in der Robotik

Für die C-API wurde das PTP und das Linear-Verfahren umgesetzt. Die Informationen für die Berechungen sind entnommen aus der Lektüre *Industrieroboter von Wolfgang Weber* [WW-2013].

PTP-Verfahren

Um den Roboter bestimmten Wegpunkten abfahren zu lassen, muss man die Bewegungsprofile selbst berechnen und über die C-API an den Roboter im 125Hz Takt übergeben. Das PTP-Verfahren setzt dabei voraus, dass die einzelnen Positionen der Gelenke bekannt sind. Die Positionen sind die Achs-Werte. Der Wert ist angegeben in radiant. Auch die Zielposition ist in Achs-Werten anzugeben.

Linear-Verfahren

Das Linear-Verfahren bedeutet den Roboter von dem Tool Center Point (TCP) Punkt aus zu bewegen. Die Bewegung des Roboters wird so berechnet, dass der TCP sich linear zum Zielpunkt bewegt(siehe Abbildung 2.4). Um die Berechnung durchzuführen, muss die Position des TCP im Raum(kartesische Koordinaten) bekannt sein, um eine Strecke zu einem Zielpunkt abfahren zu können. Der UR5 Roboter kann aber nur Positionen in Achs-Ebene verarbeiten. Deswegen muss zuerst eine Berechnung von Achs-Ebene in kartesische Koordinaten und nach der Berechnung der Strecke wieder zurück auf Achs-Ebene erfolgen.

2.6. Eigene Adapter-Schnittstelle aufbauend auf URScript

Die Secondary Schnittstelle(2.4.3) kann benutzt werden, um einzelne Scriptbefehle an den Roboter zu senden. Auf diesem Prinzip aufbauend, kann ein Adapter für jede Programmiersprache entwickelt werden, der die Befehle an den Roboter sendet. Dadurch kann nun ein Anwendungsprogramm in dieser Sprache mit all seinen Vorteilen entwickelt werden.

In dieser Arbeit wurde dafür Python gewählt. Gründe hierfür sind:

- Weit verbreitete Programmiersprache
- Ein vorhandener Parser für die Secondary Schnittstelle
- Viele vorhandene Software Bibliotheken
- Höhere Sprache als z.B C und somit etwas leichter zu Programmieren

Da aufbauend auf dieser Arbeit eventuell mit dem Roboter weitergearbeitet wird, wurde eine Sprache genommen die weit verbreitet ist. Python ist eine Sprache die nicht so Hardware nah ist, dass man sich um Speicherbelegung kümmern muss, aber den Code schnell ausführt. Für den UR5 wurde eine Robot Operating System (ROS)[4] Schnittstelle entwickelt, bei dem schon die Pakete der Secondary Schnittstelle geparsed(Parser) werden. Das Projekt ist öffentlich, so konnte dieser Code Abschnitt in die Arbeit übernommen werden.

[4] ROS ist eine Große Bibliothek um Robotersteuerung zu abstrahieren und das ansprechen von Robotern zu vereinfachen

3. Evaluierungskonzept

3.1. Anwendungsbeispiel

Das Anwendungsbeispiel ist ein Kinderspiel. Dieses Spiel soll die motorischen Fähigkeiten bei Kindern verbessern. Gegeben ist ein Kugel mit löschern aus Verschiedenen Formen(Kreis, Oval, Viereck, Trapez, etc.). Zu diesen Formen existieren die Entsprechenden Klötzchen, die entsprechend Groß sind un die Form der Löscher besitzen. Die Aufgabe des Spiel ist es alle Klötzchen in die entsprechende Form zu drücken, bis alle in der Kugel sind.

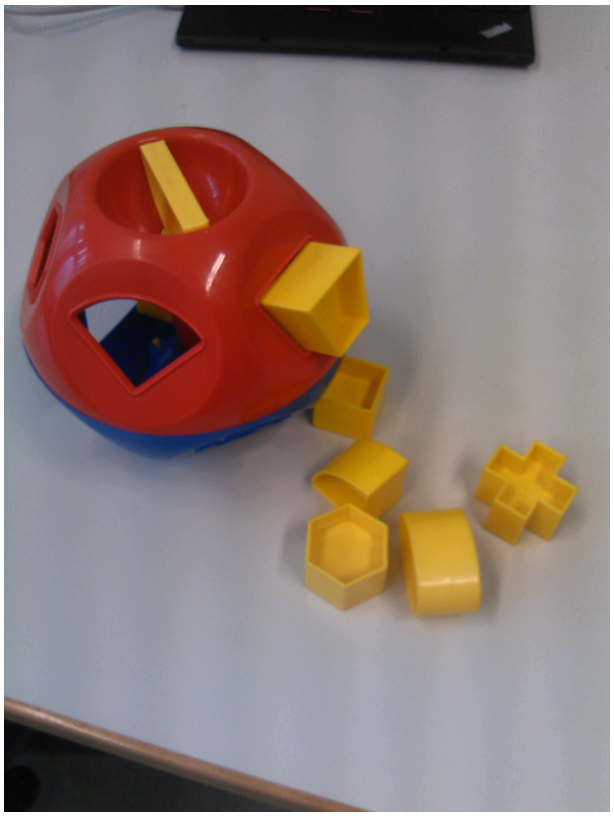

Abbildung 3.1.: Kinderspiel zur Evaluierung der Software Schnittstellen

Die Kugel wird an den Kopf des Roboterarms befestigt. Es soll eine Anwendung entwickelt werden, die für den entsprechenden Spieler die Höhe des Roboters einstellt. Der Spieler soll die Möglichkeit haben, die Startposition zu verstellen und für sich zu speichern. Bei einem bestimmten Knopf druck soll der Roboter das Loch für die jeweils nächste Form so ausrichten, damit der Mensch das Klötzchen nur noch einwerfen braucht.

3.2. Speichern der Anwendungsdaten

Um auf bestimmte Menschen zugeschnittene Bewegungsabläufe zu machen muss der Roboter Daten über den Anwender kennen. Diese sollten persistent gespeichert werden, damit bei einem Wechsel des Anwenders die Daten nicht verloren gehen. Daten der Anwender sind z.B. Name, Alter, bestimmte Positionen im Roboter Programm, etc.

Speichern über Polyscope und URScript

In der Polyscope Software oder in einem URScript Programm, können Daten die von den Benutzern erstellt oder erhoben werden nicht persistent gespeichert werden. Hierzu muss eine zweite Anwendung entwickelt werden, auf die sich das URScript oder Univeral Robot Program (URP) Programm verbindet und die Daten zum persistenten speichern versendet. In Polyscope und URScript muss sehr aufwendig mit den vorhandenen Script Befehlen eine Socket Verbindung aufgebaut werden. Damit diese zwei Programme miteinander Kommunizieren können muss ein gemeinsames Protokoll mit bestimmten Befehlen festgelegt werden. Es ist möglich Text, Zahlen oder Dezimalzahlen zu versenden und zu empfangen. Es kann nur eins dieser drei Typen versendet, von dem aber beliebig viele.

Speichern über Eigene API

Mit der Eigenen API muss keine zweite Software entwickelt werden, da die API auf einem Client Rechner läuft und dort die Daten persistent gespeichert werden können. Es muss im Anwendungsprogramm eine Verbindung zu einer Datenbank aufgebaut werden und dort können die Daten gespeichert werden.

4. Realisierung

4.1. C-API

In folgendem Kapitel wird beschrieben wie die C-API genutzt werden konnte, um dem Roboter bestimmte Wegpunkte abzufahren.

4.1.1. Beispielanwendung

Es konnte eine Anwendung erstellt werden, dass den Roboter Initialisiert und dann in einer Schleife die Positions, Geschwindigkeits und Beschleunigungsdaten sendet. Desweiteren konnte ein Bewegungsprofil errechnet werden, dem der Roboter gefolgt ist.

Bevor man Daten vom Roboter Abfragen kann, muss eine Verbindung zum Roboter hergestellt werden, diesem einen hochfahren und initialisieren lassen. Folgend werden diese Vorgänge beschrieben.
Mit dem Befehl *robotinterface_open()* kann die Verbindung zum Roboter hergestellt werden.

Um sicher zu gehen das die Verbindung offen ist, wird in einer Schleife eine bestimmte Zeit, immer wieder abgefragt ob der Roboter verbunden ist. Falls dies nicht funktioniert, wird der Vorgang abgebrochen und das Programm sollte beendet werden. Es kommt vor, dass der Roboter beim Starten noch in einem Sicherheitsmodus ist. Wenn dies der Fall ist, muss der Modus abgestellt werden. Dies geht mit der Funktion *robotinterface_unlock_security_stop();* .

Auch hier wird zur Sicherheit eine bestimmte Zeitschleife der Befehl wiederholt an den Roboter gesandt. Wenn der Roboter dennoch in dem Sicherheitsmodus ist, ist es möglich, dass der Notausschalter am Touch Tablet aktiviert ist. Nachdem die Verbindung offen ist, muss der Roboter mit Strom versorgt werden. Mit dem Befehl *robotinterface_power_on_robot()* kann das Bewerkstelligt werden. Auch hier wird eine bestimmte Zeitschleife abgewartet, bis der Roboter hochgefahren ist. Abgefragt werden kann dies mit der Funktion *robotinterface_is_power_on_robot()* .

Nun wird der Roboter Initialisiert. Der Roboter geht nach dem Starten automatisch in den Initialisierungs Modus. Jeder einzelne Joint muss nun solang in eine Richtung bewegt werden, bis der Joint in den normalen Modus übergeht. Um die Gelenke zu bewegen, wird eine Geschwindigkeitsvorgabe an den Roboter gesandt.(siehe Listing 4.1)

```
puts("Initializing robot");
/// Set zero velocity and acceleration as guard
int j;
for (j=0; j<6; ++j) {
  pva_packet.velocity[j] = 0.0;
  pva_packet.acceleration[j] = 0.0;
}
do {
  ++i;
  robotinterface_read_state_blocking();
  int j;
  for (j=0; j<6; ++j) {
    // initialize_direction is 1 or -1. it determines in which direction die Joint is
        moving during the initialization
    pva_packet.velocity[j] = ((robotinterface_get_joint_mode(j) ==
        JOINT_INITIALISATION_MODE)) ? (initialize_direction)* 0.1 : 0.0;
   }
  robotinterface_command_velocity(pva_packet.velocity);
  robotinterface_send();
} while (robotinterface_get_robot_mode() == ROBOT_INITIALIZING_MODE && exit_flag == false);
puts(" Done!");
```

Listing 4.1: Initialisierung der einzelnen Gelenke

Nachdem die Initialisierung abgeschlossen ist, muss wie in Listing 2.3 eine Schleife mit der vorgegebenen Struktur durchlaufen werden, bis das Programm beendet, oder die Verbindung zum Roboter geschlossen werden soll. Wenn dies nicht so gemacht wird, geht der Roboter automatisch in den Sicherheitsstopp, da nicht innerhalb von 8 Millisekunden Nachrichten an den Roboter gesendet wurden.

Innerhalb der beiden Befehlen *robotinterface_read_state_blocking()* und *robotinterface_send()* kann nun eine Interpolation berechnet werden und die Vorgaben für Position, Geschwindigkeit und Beschleunigung an den Roboter gesandt werden(siehe Listing 4.2).

```
// loop through interpolation length
for(i=0; i < move_pva_packet.interpolations+1; i++){
  robotinterface_read_state_blocking();

  // abort interpolation if Robot is in securitystop mode
  if(robotinterface_is_security_stopped()) {
      robotinterface_get_actual_current(currents_actual);
      robotinterface_command_empty_command();
      robotinterface_send();
      break;
  }

  // get current time of interpolation
```

```
14    move_pva_packet.point_in_time= (double) i * T_IPO;

16    // interpolate with sinoide profile and write result in variable move_pva_packet
17    interpolation_sin_ptp(&move_pva_packet);

19    // write the triple to robot
20    robotinterface_command_position_velocity_acceleration(move_pva_packet.pva.position,
21                                                           move_pva_packet.pva.velocity,
22                                                           move_pva_packet.pva.acceleration);
23    // send command to robot
24    robotinterface_send();
25  }
```

Listing 4.2: Interpolation eines berechneten Wegs

4.1.2. Interpolation

TODO interpolation !!

4.1.3. Aufgetretene Probleme

Der Roboter geht ab einem bestimmten Winkel in den Sicherheitsstopp. Die Abweichung der Position wird zu groß. Dies kann analysiert werden, wenn man sich den Durchschnitt der Soll-Werte und der Ist-Werte der Position ansieht(siehe Abbildung 4.1).

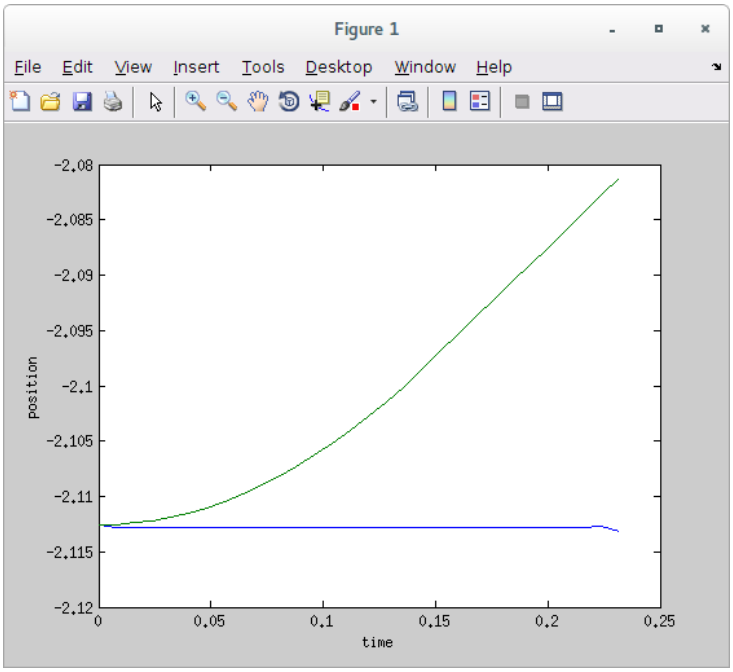

Abbildung 4.1.: Abbildung zeigt die Soll und Ist Werte der Position, bis zum Sicherheitsstop des Roboters

Die Abweichung steigt beim 2. Gelenk hoch. Die Motorleistung reicht anscheinend nicht aus um über die Schwelle der Reibungskräfte und die Erdanziehung zu kommen. Zu sehen ist in Abbildung **??**, dass für die Stromstärke ein Offset mitberechnet wird. Vergleicht man die Wert, die bei der selben Bewegung von Polyscope berechnet werden(siehe Abbildung **??**), sieht man eine höhere Stromstärke bei der Polyscope Software, bei der die Bewegung ohne Probleme Funktioniert.

Mit der C-API ist es nicht möglich selbst den wert der Stromstärke vorzugeben, deswegen sind die Soll-Werte bei der Polyscope Software und der eigenen Anwendung mit der C-API etwas verschieden.

4.2. Polyscope

4.2.1. Programmierung

Die Programmierung findet meist nur auf dem Touch Tablet statt. Ein neues Programm fängt mit einem leeren Ereignisbaum an. Es kann per Toucheingabe alle möglichen Funktionen, die die Script Sprache bietet dem Baum hinzugefügt werden. Wenn das Programm abläuft, werden von der Wurzel an die Befehle abgearbeitet. Wie in Abbildung 4.2 zu sehen ist, ist die Ansicht des Programmbaumes sehr unübersichtlich.

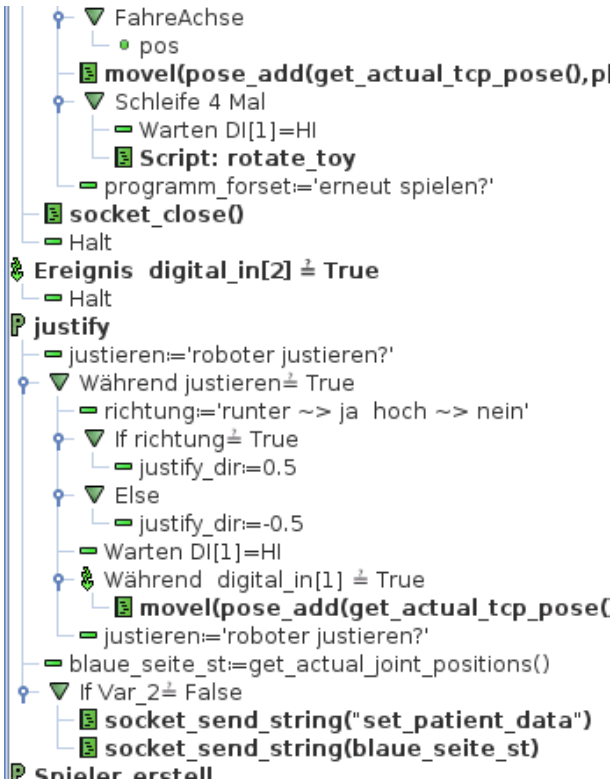

Abbildung 4.2.: Ein Ausschnitt aus einem Programm Baum in Polyscope

Es ist möglich andere Script Dateien in das Programm einzufügen. Dazu gibt es ein Feld *Script* . Das Script Programm muss sich auf dem Linux Rechner befinden. Es ist also nötig das Script Programm auf einem anderen Rechner zu programmieren und bei jeder Änderung auf den Linux Rechner des Roboters zu senden. Mit dieser Möglichtkeit könnten Programmabschnitte ausgelagert werden. Es ist aber nicht Übersichtlich welche Script Dateien benutzt werden.
Alternativ zu dem Touch Tablet, könnte der X-Server von dem Linux Rechner auf ein anderen Rechner umzuleiten um dort mit dem Programm per Maus und Tastatur zu arbeiten. Dies wurde aber noch nicht getestet und könnte zu verzögerungen beim ausführen kommen.
Eine andere Möglichkeit ein Programm unter Polyscope zu programmieren gibt es nicht.

4.2.2. Benutzer Interaktion

Die Möglichkeiten zur Interaktion mit dem Benutzer sind sehr begrenzt. Die Software und die URScript Sprache lassen es zu, dass auf dem Touch Tablet (Popup) Nachrichten auftauchen. Wenn man mit dem Benutzer Interagieren will, gibt mehrere Arten dieser Popups. Als Nachricht, ja/nein Fragen, oder Text abfragen. Der Benutzer kann dann mit einen Text oder wählen zweier ja/nein Buttons antworten. In Abbildung **??** ist als Beispiel eine Nachricht und eine Ja/Nein Popup zu sehen.

Abbildung 4.3.: Abbildung zeigt Zwei verschiedene Arten von Popups in Polyscope

Kompliziertere Menüs sind mit dieser Methode nicht möglich. Wenn ein Programm erstellt werden soll, bei dem der Benutzer viele eingaben machen muss, ist es mit Polyscope sehr schwer das zu realisieren.

4.2.3. Test und Fehlersuche im Programm

Bevor Polyscope ein Programm ablaufen lässt wird das Script auf die richtige Syntax geprüft. Sollte ein Fehler vorhanden sein wird dies beim Start als Popup angezeigt. Fehler die in Abschnitten mit Touch hinzugefügt wurden, können jedoch nicht lokalisiert werden. Nur in extra eingefügtem Script Code kann grob lokalisiert werden, welcher Fehler aufgetreten ist, weil dieser Teil extra geprüft wird.

Da das Programm mit dem Touch Tablet ausgeführt werden kann, ist es möglich während der Programmierung das Programm ablaufen zu lassen. Es kann sehr schnell getestet werden ob die gewünschten Einstellungen dem Ergebnis entsprechen. Bei Großen Programmen mit vielen Benutzeranfragen, kann dies jedoch viel Zeit in Anspruch nehmen. Es muss von einem Benutzer bei jeder Anfrage eines Popups von Hand geantwortet werden.

4.2.4. Aufwand der Programmierung

Kleine Programme in Polyscope sind sehr schnell geschrieben. Mit dem Touch Tablet kann sehr schnell eine kleine Kontrollstruktur aufgebaut werden. Das Tablet hat jedoch große Nachteile. Wie in Abbildung **??** zu sehen ist der Bereich für den Ereignisbaum sehr klein. Wenn ein Programm nun z. B. 500 Befehle enthält, ist es nicht möglich ein Überblick zu halten. Auch ist es sehr aufwändig zwischen bestimmten Bereichen hin und her zu Wechseln, da das Touch Tablet nicht genau ist. Möglich ist es geschriebene Bereiche auf Script Dateien zu verschieben, die dann im Polyscope Programm verwendet werden.

4.2.5. TCP Server mit Datenbank zum dauerhaften speichern der Daten

Um mit Polyscope und URScript erhobene Daten zu Speichern wurde ein kleiner TCP/IP Server geschrieben, der eine Verbindung zulässt und Daten in einer Datenbank speichert. Die Daten sind Objektorientiert, und werden von dem Server erstellt. In Polyscope und URScript gibt es keine Objektorientierung, deshalb muss dort alles nacheinander angefragt werden. Ein Beispiel einer TCP/IP Servers liegt im Anhang(siehe D.1)

4.3. URScript

Die URScript Sprache ist sehr stark an Python gelehnt. Das Manual von Univeral Robots umfasst alle nötigen Funktionen um Komplexe Aufgaben zu Erfüllen. Um Daten persistent zu

speichern, muss wie in Polyscope eine (Socket) Verbindung zu einer Zweiten Anwendung aufgebaut werden, die die Daten speichert(siehe 4.2.5)

4.3.1. Laden des Scripts auf den Controller

Das Script kann nicht direkt auf dem Rechner über die Polyscope Software ausgeführt werden. Um ein selbst geschriebenes Programm in URScript auszuführen ist es nötig sich mit der Secondary Schnittstelle des URControllers 2.4.3 per TCP/IP zu verbinden und dann die Einzelnen Zeilen der Script Datei an den Controller zu senden.

Es ist möglich einzelne Befehle oder ein großes Programm auszuführen. Um einzelne Befehle auszuführen, werden diese nacheinander versendet. Ein ganzes Programm wird versendet, indem wie in Listing 4.5 gezeigt eine Funktion die ganzen Befehle umschließt. Der Controller führt diese Funktion aus, sobald diese mit dem “end” der Funktion abgeschlossen ist. Zu beachten ist noch, dass der URcontroller am Ende jeden Befehls oder Programm’s ein Zeilenumbruch erwartet.

```
1 def myProg():
2   popup("hello world","test", False, False)
3   set_digital_out(1, True)
4   movej([0.23,1.23,0.343,0.34.0.0,0.0],a=0.5,v=0.3)
5 end
```

Listing 4.3: Kleines Beispielprogramm in URScript

4.3.2. Programmierung

Programmiert werden kann das Script mit allen vorhandenen Textverarbeitungsprogrammen. Vorteilhaft ist es, wenn das Programm Syntax Highligting für Python beherrscht Da URScript sehr stark an Python angelehnt ist, hilft dies ein wenig den Überblick zu behalten.
Die Sprache bietet keine möglichkeiten Kommentare zu nutzen, was aber sehr wichtig ist, wenn ein Programm wächst und mehrere Programmierer am Projekt beteiligt sind. Es ist wichtig für die verständlichkeit. Deshalb wurde dafür in dem Programm, welches das URScript liest und an die Schnittstelle sendet, ein Pre-Prozessor eingebaut. Die Script Datei wird nach Kommentaren durchsucht und schneidet diese vor dem senden raus.(siehe 4.4)

```
1   def testprog():
2     # mit # wird ein Kommentar angefangen
3     movej([0.23,1.23,0.343,0.34.0.0,0.0],a=0.5,v=0.3) # Bewegung auf Achs Ebene zur
          Startposition
4   end

6   wird zu

8   def testprog():
```

```
9     movej([0.23,1.23,0.343,0.34.0.0,0.0],a=0.5,v=0.3)
10  end
```

Listing 4.4: Beispiel-Kommentare vor und nach dem Pre-Prozessor

4.3.3. Test und Fehlersuche im Programm

Nach dem Senden des Programms an den URController, ist die einzige Möglichkeit zu sehen ob das Programm Fehler enthält, wenn der Controller ein entsprechendes Bit setzt, das in den Datenpaketen von der Secondary Schnittstelle gesendet wird. Über dieses *Programm läuft* Bit, kann man sehen ob ein Programm läuft oder nicht läuft.
Man erhält keine Nachrichten was nicht in Ordnung ist, falls das Script Programm nicht abläuft. Um Fehler auszuschließen muss also der Bereich isoliert werden, indem der Fehler vorkommt. Das geht meist nur in einem aufwändigem ausschluss Verfahren, bei dem immer wieder Scriptcode entfernt wird.

4.3.4. Benutzer Interaktion

Das Manual für URScript nennt nur die Popup Funktion um dem Anwender eine Nachricht zu geben. Andere Möglichkeiten zur Interaktion ist im Manual nicht angegeben. Jedoch bietet Polyscope auch über verschiedene Popup Arten möglichkeiten zur Interaktion. Diese Popups gibt es auch für URScript. Die Befehle können aus dem von Polyscope erzeugtem URScript Code von URP Programmen eingesehen werden. Somit bestehen genau die gleichen Möglichkeiten wie bei Polyscope.

4.3.5. Aufwand der Programmierung

Im Gegensatz zur Polyscope Software, kann mit einem Textverarbeitungsprogramm sehr schnell mit guter Übersicht ein größeres komplexeres Programm erstellt werden. Es können auch leicht Kommentare eingefügt werden und der Code ist im späteren Fall leichter verständlich für neue Programmierer. Da schwer Fehler zu entdecken sind und deswegen häufig das Programm manuell getestet werden muss, ist dennoch bei großen Anwendungen ein größerer zeitlicher Aufwand von Nöten

4.4. Anwendung mit Adapter zu URScript

Im folgenden Kapitel wird das Anwendungsbeispiel in Python entwickelt. Um zu zeigen, wie die Benutzerinteraktion mit einem Eigenen Adapter gestaltet werden kann, wurde hierfür die

Software Bibliothek *TKinter* [1], mit der man sehr schnell eine Graphical User Interface (GUI) entwickeln kann. Die Interaktion erfolgt durch Buttons, die dann über den Adapter Befehle an den Roboter sendet.

4.4.1. Adapter zur Secondary Schnittstelle

Die Script befehle zur Secondary Schnittstelle, werden als Text übergeben. Der Adapter, wird in Form einer Klasse geschrieben, die die einzelnen Script Befehle in Funktionen mitliefert(siehe Listing **??**).

```
2  # moveJ moves the Robot with joint coordinates
3  # positions should include the target joint positions
4  def movej(self, positions=None, a_max=None, v_max=None):
5      if positions is None:
6          positions= self.get_joint_positions()
7      if a_max is None:
8          a_max=math.radians(40)
9      if v_max is None:
10         v_max=math.radians(60)
11     message="""movej(%s,a=%f,v=%f)
12     """%(positions,a_max,v_max)
13     print message
14     self.start_program(message)

16 # movel moves the Robot Linear in kartesian coordinates
17 # positions should contain the target tcp positions
18 def movel(self, positions=None, a_max=None, v_max=None):
19     if positions is None:
20         positions= self.get_tcp_positions()
21     if a_max is None:
22         a_max=math.radians(40)
23     if v_max is None:
24         v_max=math.radians(60)
25     message="""movel(p%s,a=%f,v=%f)
26     """%(positions, a_max, v_max)
27     print message
28     self.start_program(message)
```

Listing 4.5: Ausschnitt zeigt Funktionen, die Scriptbefehle in der Adapter Klasse umgesetzten

Diese Klasse öffnet zwei Verbindungen zur Secondary Schnittstelle. Eine zum Empfangen der Datenpakete und eine zum Senden der URScript Befehle. Der Adapter besitzt eine (Queue) um die Befehle, nacheinander zur Secondary Schnittstelle zu senden. Da Befehle eventuell viel Zeit benötigen, um ausgeführt zu werden, wird gewartet bis der Befehl abgearbeitet oder abgebrochen wurde. Erst dann wird in der Queue der nächste Befehl an die Schnittstelle gesendet.

```
1  while self.__run_flag:
2      DequePrograms.lock.acquire()
```

[1] Mehr Informationen über TKinter unter folgender Website: https://wiki.python.org/moin/TkInter

```
3     if(len(self.s_interface.program_queue) > 0):
4         # print("message queue contains messsages %d" % len(self.s_interface.program_queue)
              )
5         message=self.s_interface.program_queue.popleft()
6     else:
7         message=None
8     DequePrograms.lock.release()
9     if(message is not None):
10        SecondSendInterface.send_lock.acquire()
11        self.s_interface.send_messages_queue.append(message)
12        SecondSendInterface.send_lock.release()

14        //blocks the thread until URScript finishes
15        self.s_interface.block_program()
16    time.sleep(0.2)
17 return 0
```

Listing 4.6: Ausschnitt zeigt die Abarbeitung der Queue

Schnittstelle und arbeitet nacheinander eine Queue ab, die Befehle an die Schnittstelle beinhaltet. In einer Anwendung kann nun diese Klasse benutzt werden um den Roboter zu steuern. Der Aufwand für ein Programm ist mit einer Eigenen API anfangs deutlich höher als die URScript Sprache direkt zu nutzen. Es muss erst ein Adapter geschrieben und getestet werden, bevor die eigentliche Anwendung geschrieben werden kann.

4.4.2. Programmierung mit Adapter

Sobald der Adapter in einer etablierten Programmiersprache programmiert ist, und Fehler in diesem so gut wie ausgeschlossen sind, kann nun mit normalen Softwareentwicklungstechniken leicht ein Programm geschrieben werden. In dieser Arbeit wurde Python gewählt. Python bietet viele Bibliotheken und Design Patterns die das Programmieren vereinfacht. Es können Entwicklungswerkzeuge benutzt werden um einen leichten Überblick über das Programm zu behalten.

4.4.3. Benutzer Interaktion

Eine etablierte Programmiersprache bietet natürlich auch alle möglichen Bibliotheken und Möglichkeiten ein interaktives und leicht verständliches Interface zu erstellen. Es ist möglich übersichtliche Formulare zu erstellen mit denen Informationen vom Anwender zu erfassen. Abbildung 4.4 zeigt ein Interface, mit dem der Roboter primitiv in alle Richtungen gesteuert werden kann. Für eine Umsetzung von allen Möglichkeiten wurde wegen Zeitmangels verzichtet.

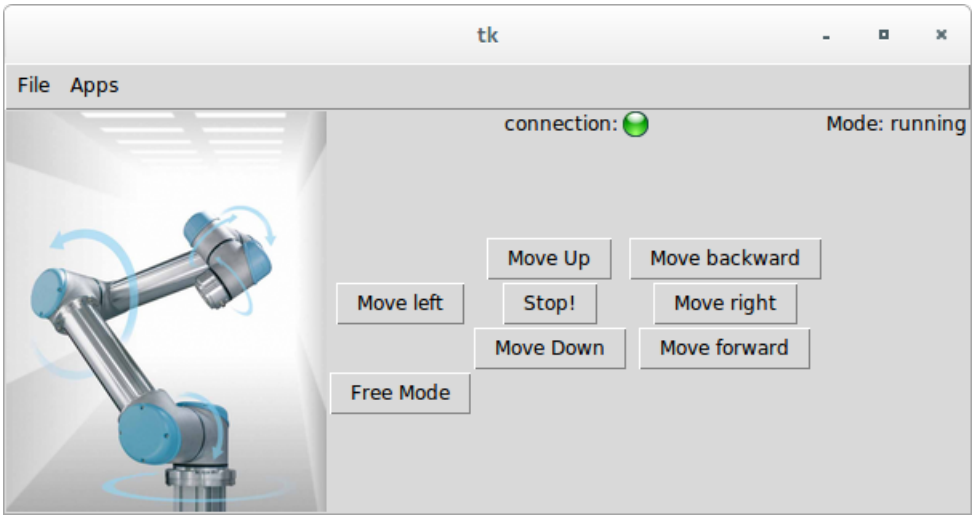

Abbildung 4.4.: Startfenster des selbst erstellten GUI's zur Steuerung des UR5 Roboters. Es ist möglich den Roboter in alle Richtungen Linear zu bewegen.

4.4.4. Test und Fehlersuche im Programm

Nach Ausschluss der Fehler in dem Adapter, kann in der Programmiersprache vorhandene (Unittest) oder andere automatische Tests für das Programm verwendet werden. Die Schnittstelle zum Roboter wird hier durch einen sogenannten (Mock) erstetzt. Dadurch kann auch offline getestet werden. Fehler werden in den etablierten Programmiersprachen so leichten gefunden und lokalisiert. (Interpreter) Programmiersprachen analysieren den Softwarecode auf Fehler in der Syntax, bevor sie ihn ausführen. Auch Sprachen die keine Interpreter benutzen und den Softwarecode Kompilieren, testen den Code auf Syntaxfehler und zeigen Fehler frühzeitig an.

4.4.5. Aufwand der Programmierung

Der Aufwand für ein Programm ist mit einer Eigenen API anfangs deutlich höher, gegenüber den anderen Methoden. Es muss erst ein Adapter geschrieben und getestet werden, bevor die eigentliche Anwendung geschrieben werden kann. Nach dieser Hürde, ist es aber sehr leicht Programme zu erstellen die auf den Adapter zugreifen um den Roboter zu steuern.

5. Ergebnis

Im Folgenden Kapitel werden die Schnittstellen werden gegenübergestellt und verglichen. Desweiterem werden die nicht erreichten Ziele erörtert.

Kriterium	**C-API**	**Polyscope**
Programmierbarkeit	Schwer einfache Roboterprogramme zu Realisieren.	Leichter Einstieg zum Programmieren für Anfänger.
Benutzerinteraktion	Es ist möglich ein Übersichtliches & intuitives Interface zu entwickeln	Keine Komplexen Menüs möglich. Nur schwache Interaktion möglich.
Testen	Test sind möglich, aber nur mit Simulation des Roboters	Keine eigenen Tests möglich. Getestet wird immer Live an Roboter.
Debuggen	Compiler findet Syntax Fehler & Debugging ist nur mit Simuliertem Roboter möglich	Bedingt möglich beim Testen Live am Roboter.
Aufwand	Sehr großer Aufwand von nöten. Es muss alles selbst Entwickelt werden.	Bei kleinen Programmen kaum Aufwand. Aufwand steigt enorm bei mehr Anforderungen.

Tabelle 5.1.: Zusammenfassung der Evaluierungskriterien für C-API und Polyscope

Kriterium	URScript	Eigener Adapter
Programmierung	verständnissvolle Dokumentation ermöglicht es einen schnellen Einstieg. Entwicklerwerkzeuge und **Syntax Highlighting!** (**Syntax Highlighting!**) erleichtern die Übersicht und Vereinfachen die Programmierung	Eine Etablierte Programmiersprache erleichtert das Programmieren deutlich. Vorhandene **Librarys!** erleichtern das Programmieren.
Benutzerinteraktion	Die möglichkeiten bleiben wie bei Polyscope mit Popups beschränkt(siehe 4.2.2)	Wie bei C-API können GUI's erstellt werden, die komplexe Formulare und Menüs bieten.
Testen	Mit gegebenen Mitteln sind automatische Tests nicht möglich. Es kann nur wie mit Polyscope Live getestet werden.	Automatische Test sind möglich.
Debuggen	Es gibt keine Möglichkeit zu Debuggen. URController liefert bei Fehler in der Syntax keine information	Fehler werden leicht gefunden, da die Etablierten Programmiersprachen den Programmcode nach Syntaxfehlern durchsuchen und anzeigen.
Aufwand	Ähnlich wie bei Polyscope, jedoch etwas besser durch mehr Übersicht des Projektes	Anfangs ein Großer Aufwand von nöten. Bei mehreren Anwendungen für den Roboter ist Aufwand jedoch geringer als bei den anderen Schnittstellen.

Tabelle 5.2.: Zusammenfassung der Evaluierungskriterien für URScript und Eigener Adapter

5.1. Vergleich der Schnittstellen

Die C-API ist eine sehr Hardware nahe Schnittstelle zum Roboter. Der Aufwand der Betrieben werden muss ist sehr hoch. Diese Schnittstelle sollte nur in seltenen Fällen eingesetzt werden. Nur Spezielle Anwendungen die zur Laufzeit anpassungen an Bewegungssteuerung geben sollten diese Schnittstelle nutzen.

Die Polyscope Software ist sehr gut geeignet für wenige komplexe Anwendungen, die keine bzw. kaum Benutzerinteraktion erfordern. Für eine Kollaboration die auf Interaktion angewiesen ist, ist diese Schnittstelle nicht zu empfehlen. Diese Schnittstelle kann nur sehr aufwändig auf persistente Daten zugreifen und speichern.

URScript bietet in sachen Benutzerinteraktion nur die gleichen möglichkeiten, wie die Polyscope Software(siehe 4.3.4). Es ist Übersichtlicher und verständlicher gegenüber Polyscope Anwendungen zu entwickeln, jedoch bietet die eigens entwickelte Scriptsprache nur wenig möglichkeiten, wirklich komplexe Anwendungen zu entwickeln.

Ein eigener Adapter zum URController vereint die Vorteile einer etablierten Programmiersprache, nämlich vorhandene Entwicklerwerkzeuge und Software Bibliotheken zu nutzen. Der Roboter wird über URScriptbefehle gesteuert wird, deshalb muss nicht tief in die Robotersteuerung eingreifen werden wie bei der C-API. Wenn viel Benutzerinteraktion von nöten ist, ist diese Schnittstelle zu empfehlen.

5.2. Nicht erreichte Ziele

Mit der C-API konnte keine Anwendung geschrieben werden, die für die Evaluierungskriterien vollständige Daten liefern.

6. Fazit

6.1. Zusammenfassung

Der Beste weg für die Kollaboration ist einen eigenen Adapter in einer Programmiersprache zu schreiben um Programme mit richtigen GUI's zu entwickeln.

Polyscope
Die Polyscope Schnittstelle bietet keine Möglichkeiten, diese zu erweitern. Die möglichkeiten für die Kollaboration sind hier auch sehr gering. Diese Schnittstelle kann nur benutzt werden um kleine und wenig Komplexe Programme zu schreiben.

Die URScript Sprache ist ein wenig angenehmer zu Programmieren als die Polyscope Software, jedoch bleiben die möglichkeiten der Kollaboration genauso beschränkt.

Die C-API bietet nur wenig Möglichkeiten zur Steuerung des Roboters. Wenn die hürden überwunden sind, ist es jedoch möglich wie mit dem eigenen Adapter über GUI's mit dem Roboter zu Kollaborieren.

6.2. Ausblick

URScript
Das Programm, dass für die Beispielanwendung geschrieben wurde um URscript Programm an den Roboter zu senden, kann erweitert werden. Es ist möglich den Pre-Processor so zu erweitern, dass die URScript Sprache mit Funktionen erweitert wird, in dem Textstellen ersetzt werden. Zusätzlich kann man die Syntax überprüfen, um Fehler in der Syntax früh zu erkennen.

Eigener Adapter
Der Adapter kann un unzähliche möglichkeiten erweitert werden. Komplexe Strukturen in der URScriptsprache können leicht abstrahiert werden und mit dem Adapter umgesetzt werden. Es ist möglich auch andere Programmiersprachen zu nutzen. Als Vorbild könnte der geschriebene Adapter dienen. Am Besten geeignet, sind aber schnelle Programmiersprachen. Der Ro-

boter sendet im 60hz Takt die Daten an den Adapter, da wäre es nicht sinnvoll, wenn das Parser (parsen) der Daten zu lange dauert.

C-API

Mit der Beispielanwendung ist ein kleiner Schritt getan, um über die C-API den Roboter zu steuern, jedoch müssen erst die aufgetretenen Fehler behoben werden.

Literaturverzeichnis

[WW-2013] : Weber, Wolfgang: Industrieroboter: Methoden der Steuerung und Regelung. 2.Auflage, Carl Hanser Verlag GmbH & Co. KG, München, 2007.

[ROSPR-2013] : ROS.org/PascalRey: ROS Wiki: Documentation. http://wiki.ros.org/, 2013-12-17 14:34:30, zuletzt besucht am 25.03.2014

[DINISO-2012] : DIN EN ISO 10218-1:2012-01 : Industrieroboter: Sicherheitsanforderungen - Teil 1 Deutsche Fassung http://www.beuth.de/de/norm/din-en-iso-10218-1/136373717, 2012-01, zuletzt besucht am 25.03.2014

B. Glossar

Syntax Highligting Zur Verbesserung der Lesbarkeit und der Übersicht, wird in einem Textverarbeitungsprogramm der Programmcode unterschiedlich dargestellt. Meist mit unterschiedlichen Farbwerten. Der Entwickler sieht mit einem Blick ob er es mit Textvariablen, Zahlenwerten zu tun hat.

ISO International Organization for Standardization: Die ISO ist eine Internationale Vereinigung um Standartisierte Normen in der Industrie zu erarbeiten und festzulegen. Jedes Land, dass Mitglied ist, muss sich an diese Normen halten. Es gibt fast kein Land, dass nicht Mitglied ist.

PTP Point to Point: PTP in Deutsch auch Punktsteuerung genannt, ist die einfachste Methode um einen Roboter auf einen anderen Zielpunkt zu fahren.

API Application Programming Interface: Eine Schnittstelle um eine Software mit einer anderen Software zu verbinden. Die Schnittstelle in Form eines Programmteils wird öffentlich gemacht und gut Dokumentiert. Die Externe Software benutzt diesen Programmteil um die Software mit der Schnittstelle zu nutzen.

URP Univeral Robot Program: URP ist eine Dateiendung für ein Programm geschrieben über die Polyscope Software.

UR Universal Robots: UR ist eine Dänische Firma die den UR5 Roboter Herstellt.

Library Software Bibliotheken: Eine Software Bibliothel, oft auch Modul oder Library genannt, ist eine Kapselung von Programmcode der wiederverwendet werden kann.

TCP/IP Transmission Control Protocol / Internet Protocol: TCP/IP ist beinhalten mehrere Netzwerkprotokolle, die es ermöglichen, dass man mehrere Rechner Vernetzen und Nachrichten austauschen lassen.

Popup Ein Fenster oder anderes Visuelles Element um einem Benutzer einer Anwendung Nachrichten zukommen zu lassen.

parsen Parser: Parser: Informationen zerlegen und entsprechend interpretieren.

Mock Ein Platzhalter für Software Objekte. Wird benutzt um Software zu testen, bei dem ein Teil der Software noch nicht existiert oder ausgeschlossen werden soll.

Interpreter In der Softwareentwicklung sind Interpreter die Kernpunkte von Programmiersprachen die den Code nicht in Machinensprache Kompilieren. Interpreter lesen den Textcode analysieren ihn auf fehler und führen ihn nach der Analyse aus.

Big-Endian Big-Endian Format: Big Engian Format ist die Festlegung der Byte-Reihenfolge, wie das Computersystem Speicherbereiche interpretieren und beschreiben soll. Dieses Format legt fest, dass das höchstwertigste Bit an der kleinsten Speicheradresse liegt.

Little-Endian Little-Endian Format: Wie bei **Big-Endian Format!**, legt das Little-Endian Format die Byte-Reihenfolge fest. Mit Little-Endian jedoch wird das niedrigwertenste Bit an die kleinste Speicheraddresse gesetzt.

ROS Robot Operating System: “Provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.” [ROSPR-2013]

Queue Eine Warteschlange, ähnlich wie bei einem Supermarkt. Die Elemente in einer Queue werden Nacheinander abgearbeitet.

TCP Tool Center Point: Der TCP beschreibt den Punkt des Werkstücks, der auf den Roboter Montiert ist. In der Regel ist dieser Punkt an der Spitze des Werkzeugs angeben.

teachen : In der Robotik ist teachen, das Anlernen von Wegpunkten, durch handliches führen am Roboter.

SCP Secure Copy: Ein Linux Programm zum Sicheren austauschen von Daten zwischen zwei verschiedenen Host-Systemen

GUI Graphical User Interface: Ein GUI erlaubt es einen Benutzer mit einem Programm über graphische Symbole zu Interagieren

Unittest : Unittests erlauben es einzelne Komponenten/Module in einem Programm zu Testen

Socket : Ein Socket ermöglichen ein Datenaustausch zwischen Programmen im Rechner oder auch zwischen 2 verschiedenen Host Systemen

C. Bilder

C.1. Bewegungsprofile geloggt über die Echtzeitschnittstelle & geplottet in Matlab

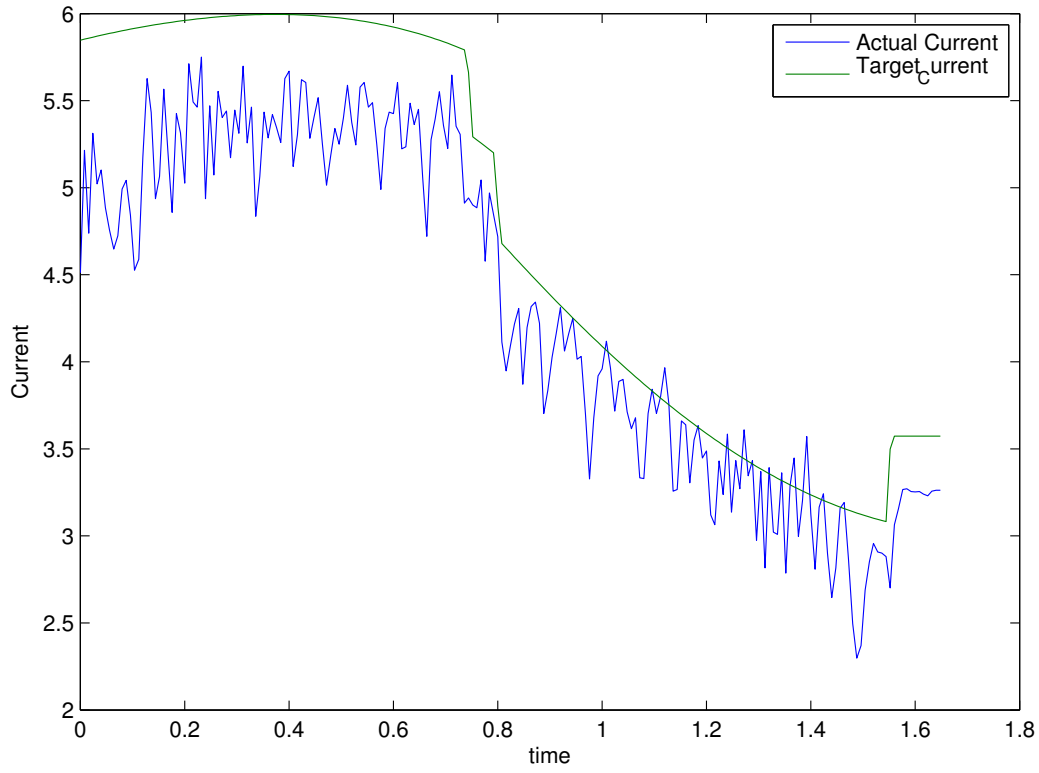

Abbildung C.1.: Abbildung zeigt eindeutig den Berechneten Offset des Ist-Wertes.

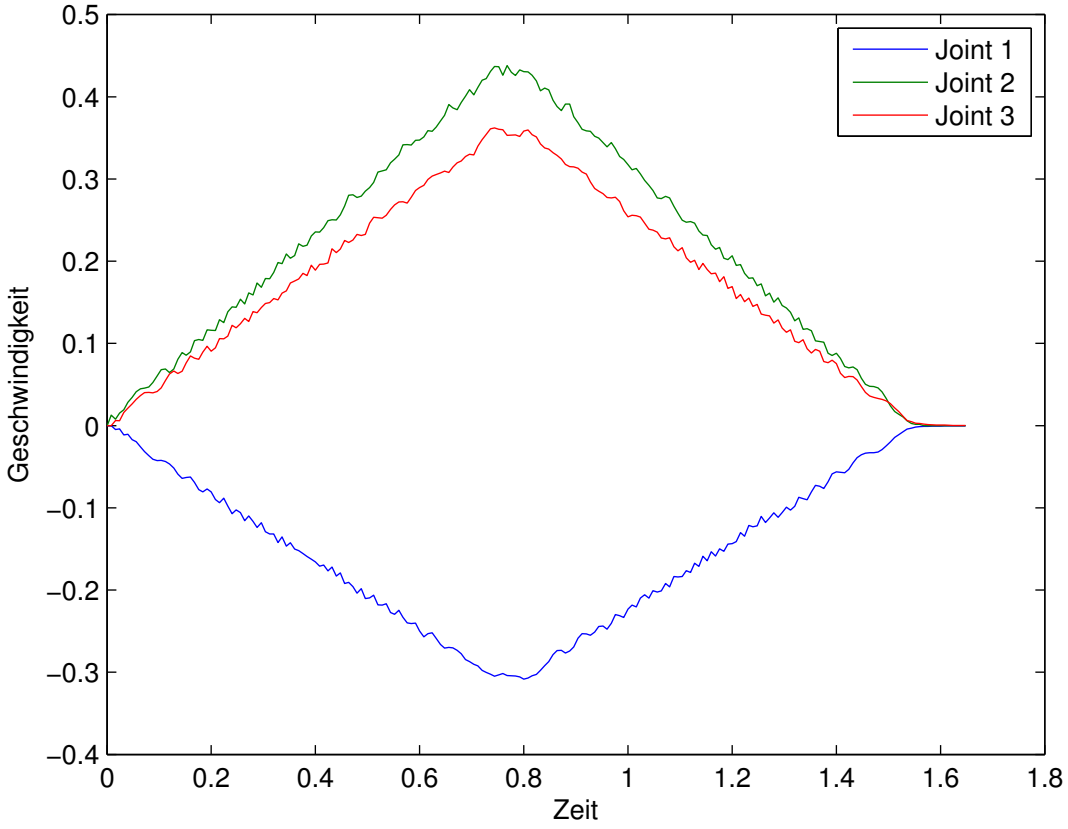

Abbildung C.2.: Abbildung zeigt das Geschwindigkeitsprofil der Gelenke 1-3 wärend eines Bewegungsprofils mit Polyscope. Profil wurde mit der Echtzeitschnittstelle geloggt

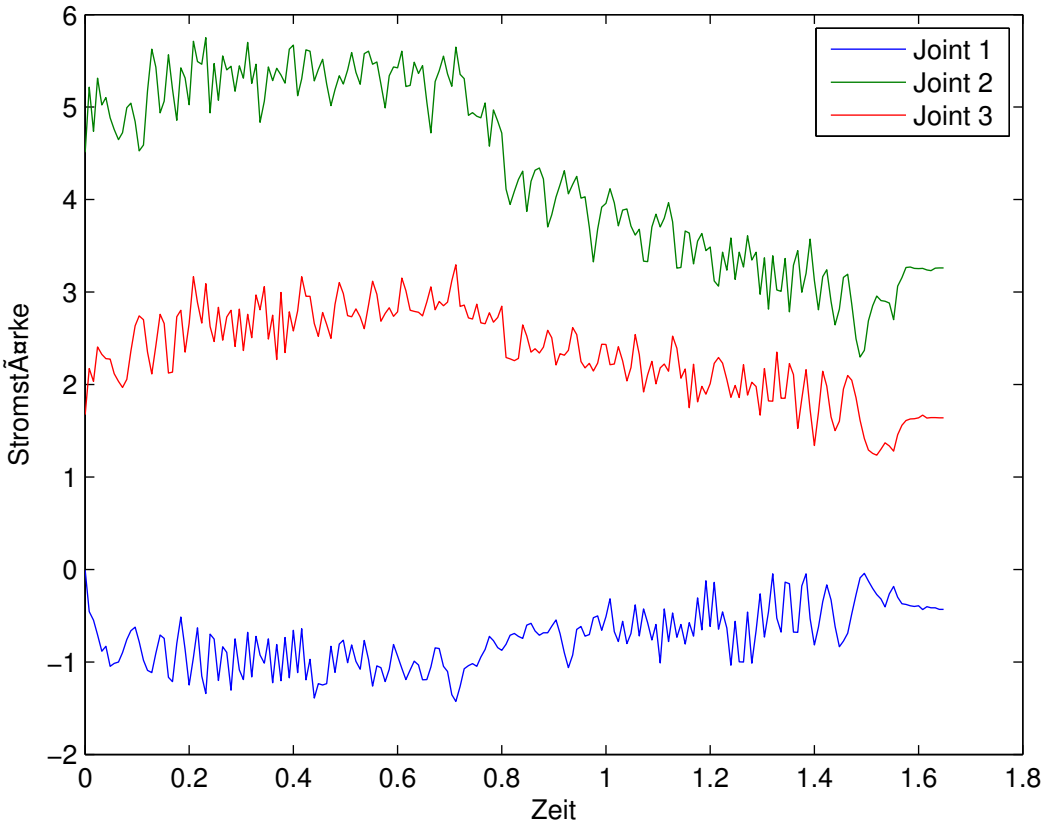

Abbildung C.3.: Abbildung zeigt die Stromstärke der drei Gelenke 1,2 und 3 wärend eines Bewegungsprofils. Profil wurde mit der Echtzeitschnittstelle geloggt

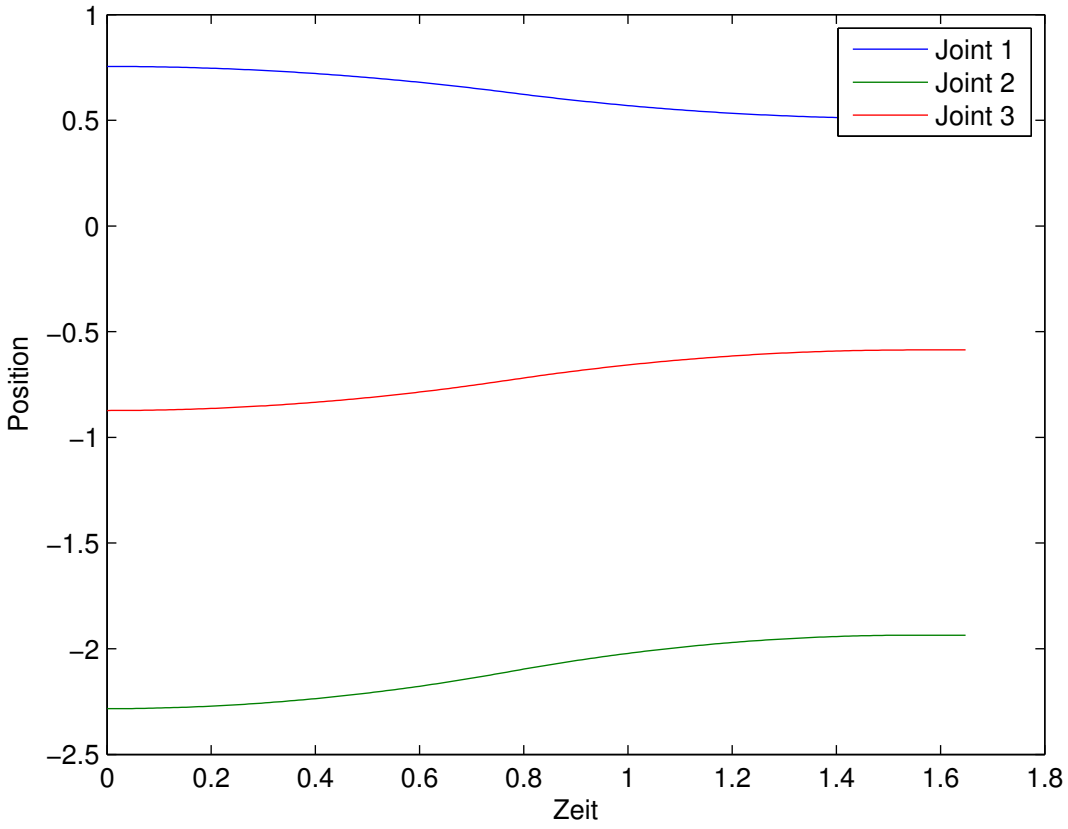

Abbildung C.4.: Abbildung zeigt die Position der Gelenke 1-3 wärend eines Bewegungsprofils mit Polyscope. Profil wurde mit der Echtzeitschnittstelle geloggt

C.2. Bewegungsprofile geloggt in der C-API

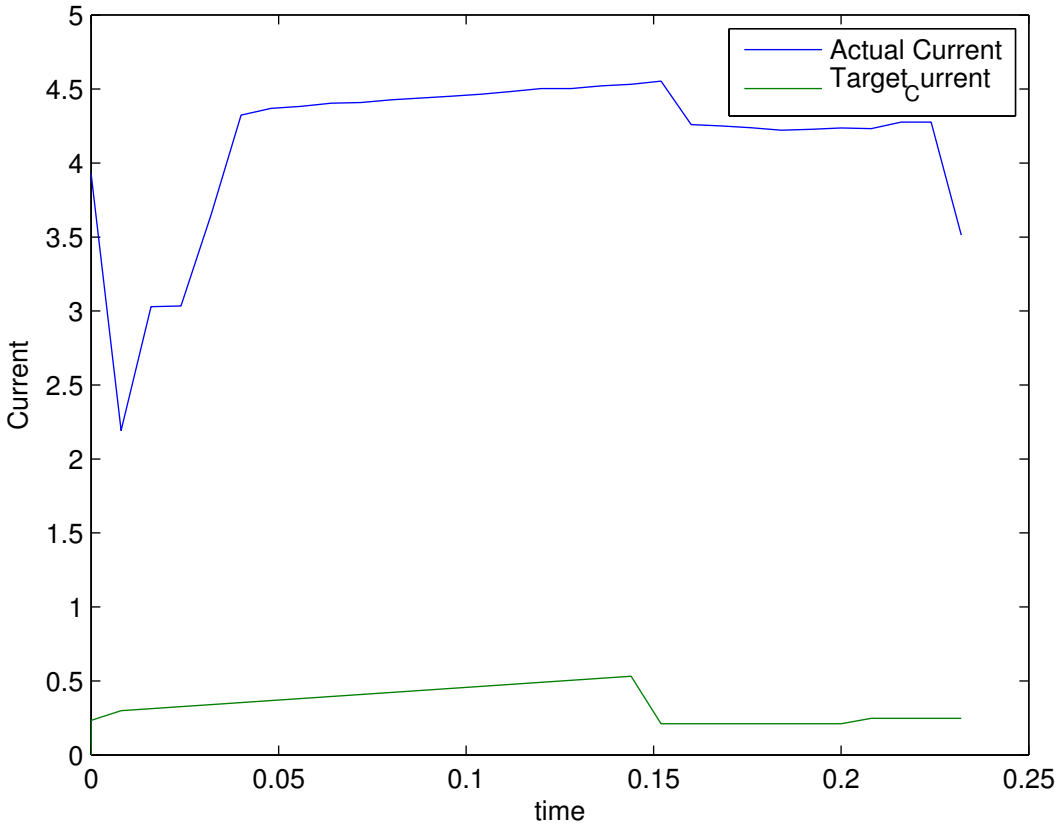

Abbildung C.5.: Abbildung zeigt die Soll-Werte und Ist-Werte der Stromstärke des 2.Gelenks, kurz vor dem Sicherheitsstopp

C.3. Bilder vom Roboter

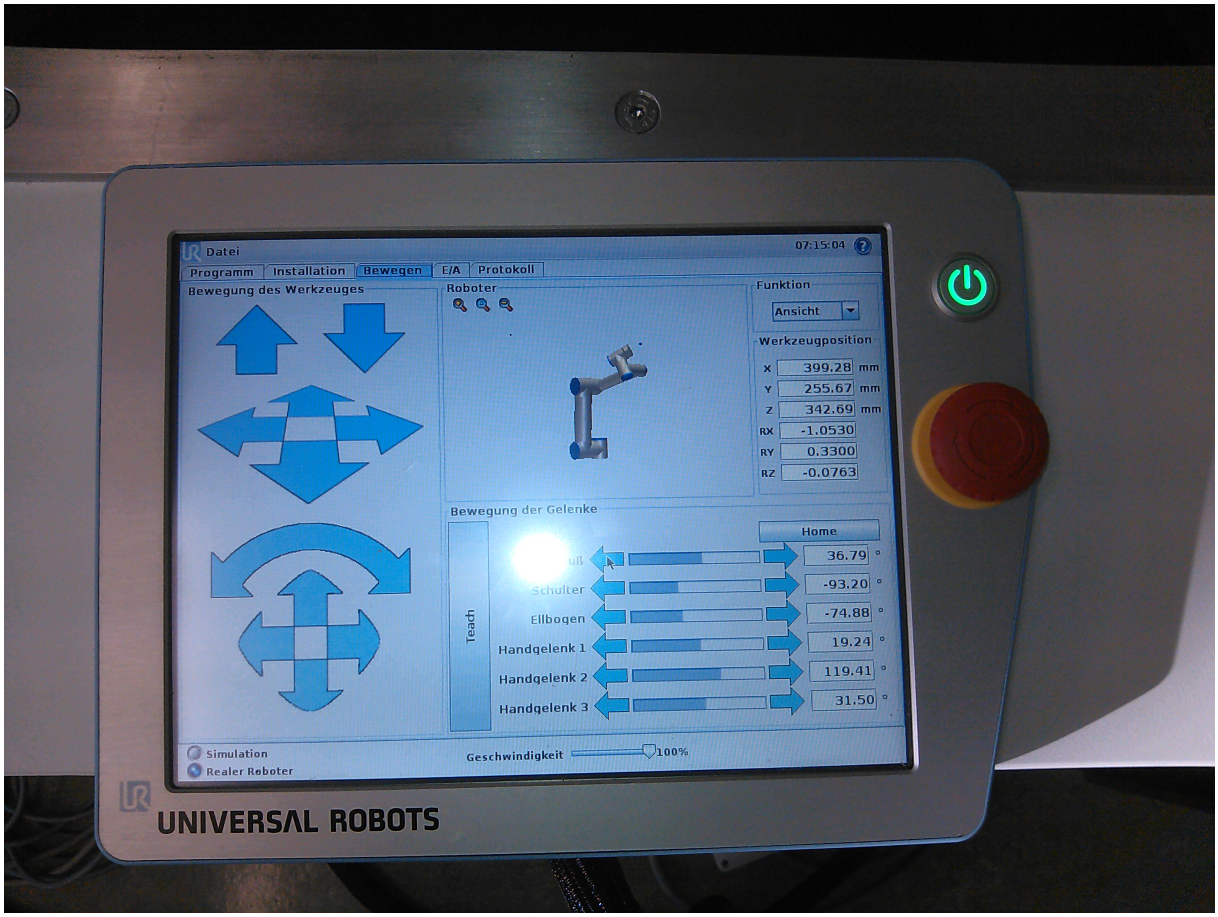

Abbildung C.6.:

D. Quellcode

D.1. Speichern der Daten über TCP in der Datenbank

```
#!/usr/bin/env python2.7

from my_utils import connection, Player, get_ip
import socket
import signal
import sys
import os

class SaveDataInterface():
    def __init__(self, interface="localhost"):
        self.__run_flag=False
        self.interface = interface
        self.socket=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ip = get_ip(interface)
        if ip is None:
            ip="127.0.0.1"
        else:
            ip = get_ip(interface)
        port = 8000
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        try:
            self.socket.bind((ip, port))
            print "server is listen on %s:%d" %(ip, port)
            self.socket.listen(1)
        except socket.error, e:
            print(e[0])
            self.socket.close()
            self.socket=None

    def run(self):
        self.__run_flag=True
        # print("starting send interface waiting for queue")
        while self.__run_flag:
            if self.socket is not None:
                conn, addr = self.socket.accept()
                conn.settimeout(2)
                print "client connected: {0}".format(addr)
                player=None
                while self.__run_flag:
                    msg = self.read_from_socket(conn)
                    if msg is not None:
                        if msg == "new_patient":
                            # print("wait for new patient name")
```

```
                            name = self.read_from_socket(conn)
                            player = connection.Player()
                            player.name = name.rstrip()
                            player.save()
                        elif msg == "load_patient":
                            print("wait for patient name")
                            name = self.read_from_socket(conn)
                            player = connection.Player.find_one({'name': name.rstrip()})
                            if player is not None:
                                print("send %s"%"1".encode('ascii'))
                                print("positions %s"%("(%s)"%player.start_pos[1:-1]).encode
                                    ('ascii'))
                                conn.send("(1)".encode('ascii'))
                                conn.send(("(%s)"%player.start_pos[1:-1]).encode('ascii'))
                            else:
                                conn.send("(0)".encode('ascii'))
                        elif msg == "set_patient_data":
                            # print("wait for player data")
                            start_position = self.read_from_socket(conn)
                            player.start_pos=start_position.rstrip()
                        else:
                            print("recieved unknown command")
                    else:
                        break
                conn.close()

        self.socket.close()
        return 0

    # This Function reads from the Socket conn the next msg checks for errors
    def read_from_socket(self, conn):
        msg = None
        while self.__run_flag:
            try:
                msg = conn.recv(1024)
            except socket.timeout, e:
                err = e.args[0]
                # this next if/else is a bit redundant, but illustrates how the
                # timeout exception is setup
                if err == 'timed out':
                    # print 'recv timed out, retry later'
                    continue
                else:
                    print e
                    msg=None
            except socket.error, e:
                print e
                msg=None
            else:
                if len(msg) == 0:
                    msg=None
                    break
                else:
                    print msg
                    break
        return msg
```

```
101 sdi = SaveDataInterface(interface="eth0")

103 sdi.run()
```