UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Konzeption und Entwicklung einer Applikation zur robotergestützten Ultraschallbildgebung

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.) im Studiengang
Medizinische Informatik

vorgelegt von

Peter Karl Seitz

23. September 2015

Referent: Prof. Dr. Rolf Bendl

Korreferent: Dr. Christoph Maier

Betreuer: Prof. Dr. Rolf Bendl

Abstract

In dieser Bachelor-Thesis wird ein Ansatz zur robotergestützten Ultraschall-Bildgebung entwickelt. Es wird hierbei hauptsächlich die Kommunikation zwischen Roboter und Computer, sowie die Steuerung des Roboters, betrachtet. Die Bilderfassung und Darstellung ist nicht Teil dieser Arbeit.

Ergebnis dieser Arbeit sind eine detaillierte Anforderungsanalyse, eine GUI zum Steuern der erstellten Roboterteilprogramme und das Roboterprogramm an sich. Auf Seiten des Roboters sind Programme für das Beibringen von Positionen, das Anfahren dieser und das Verfahren des Ultraschallkopfes an diesen (manuelle Steuerung mit einem Joystick) implementiert. Für eine Atembewegungskompensation sind Beispielprogramme erstellt worden. Die Kommunikation zwischen Roboter und Computer baut auf OpenIGTLink auf. Ein weiteres Ergebnis der Thesis ist die Architektur des Programms, die es ermöglicht, dem Roboter beliebig viele neue Befehle beizubringen.

Danksagung

Ich möchte allen danken, die mich unterstützt haben.

Insbesondere meiner Familie, die mich immer unterstützt hat und mir das Studium ermöglicht hat.

Dank gilt auch Prof. Dr. Rolf Bendl und Dr. Christoph Maier, für das Betreuen und Helfen während der Bachelor-Thesis.

Außerdem danke ich Ursula und Beatrice Baumann, für das Korrekturlesen.

Inhaltsverzeichnis

List of Listings

Abbildungsverzeichnis

Abkürzungsverzeichnis

CIRC	Kurvenbewegung
CT	Computer Tomographie
FRI	Fast Roboter Interface
GUI	grafische Benutzeroberfläche (engl. graphical user interface)
JAXB	Java Architecture for XML Binding
LIN	Linearbewegung
LWJGL	Lightweight Java Game Library
MVC	Model-View-Controller
OpenIGTLink	Open Network Interface for Image- Guided Therapy
PTP	Point to Point
RA	Roboter-Applikation
RIGTS	RobotOpenIGTLinkServer
RMI	Remote Method Invocation
TCP	Tool Center Point(z.B. Spitze des Ultraschallgerätes)
UKK	Ultraschall-Kommando-Klasse
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation

Bei der fraktionierten Bestrahlung von Tumoren werden zur Lagekontrolle Röntgenaufnahmen oder Kontroll-CTs verwendet. Dadurch ergibt sich eine zusätzliche Strahlenbelastung für den Patienten. Außerdem lässt sich durch Atembewegungen die Position des Tumors während der Bestrahlung nicht immer fehlerfrei vorhersagen. Um dieses Problem zu verbessern, könnte man die Lage des Tumors durch Ultraschall kontrollieren. Dafür wäre es notwendig einen Ultraschallkopf während der Bestrahlung an den Patienten zu halten. Der Ultraschallkopf müsste sich in einer Position befinden, in der er den Tumor erkennen und die Bewegungen des Tumors erfassen kann. Da während der Bestrahlung keine andere Person in dem Raum sein darf, ist es notwendig, den Ultraschallkopf über eine Vorrichtung festzuhalten. Starre Apparaturen eignen sich jedoch nicht, da diese die Atembewegung des Patienten nicht kompensieren können. Die Anbindung des Ultraschallkopfes an den Patient ist jedoch für die Qualität des Ultraschallbildes unerlässlich. Ein Roboter mit geeigneter Sensorik könnte dieses Problem lösen. Er könnte über Kraftmessungen die Atembewegungen des Patienten indirekt erfassen und durch anpassen des Drucks der Ultraschallkopfspitze die Atembewegungen kompensieren. Die Bewegungen des Tumors können so verfolgt und die Bestrahlung entsprechend angepasst werden.

Ein weiteres Anwendungsszenario einer robotergestützten Ultraschalldiagnostik ist die Telemedizin. So könnte ein Spezialist über das Internet eine Ferndiagnose erstellen und könnte, wie vor Ort, den Ultraschallkopf führen.

1.2 Ziele

Die Robotik in der Medizin befindet sich momentan größtenteils noch in der Forschung. Daher gibt es noch keine konkrete Lösung für dieses Problem. Es gibt jedoch schon vereinzelte Ansätze [1] [2]. Mit dem im Biosignallabor der Hochschule Heilbronn verfügbaren KUKA lbr-iiwa-7-800 soll in dieser Bachelorarbeit nun folgendes Szenario (Abschnitt 2.1) erforscht werden: Um unterschiedliche Tumorarten behandeln zu können, werden dem Roboter zu Beginn einer Bestrahlung Ausgangspositionen beigebracht, an denen er später Schallen kann. Während der Therapie soll der Roboter sich an diesen Positionen bewegen lassen. Es sollte möglich sein andere Positionen einzunehmen und die Atmung zu kompensieren. Dafür müssen folgende Ziele erreicht werden:

1. Entwerfen einer geeigneten Gesamtarchitektur
2. Erstellung einer Roboter-Applikation (RA) zur Steuerung von:
    - Anfahren von Positionen
    - Führen des Roboters von Hand

- Entferntes manuelles Steuern des Roboters
- vorgegebene Abläufe abrufen (z.B. Schwenkbewegung des Ultraschallkopfes)
- Druckkontrolle des Ultraschallgerätes mit verfolgen des Patienten bei Bewegung
- Sicherheitsüberwachung

3. Daten- und Befehlsaustausch zwischen Roboter und Steuerungscomputer mit:
   - Einrichten des Datenaustausches des Roboters mit einem externen Computer
   - Überwachung der Verbindung zwischen Roboter und Steuerungscomputer
4. Entwicklung einer Computerapplikation mit:
   - Erstellung einer GUI zur entfernten Steuerung des Roboters
   - Ermöglichen einer Eingabe zum Schwenken des Ultraschallkopfes

1.3 Vorgehensweise und Struktur der Arbeit

Am Anfang der Arbeit wurde sich mit den Grundlagen der Robotersteuerung auseinandergesetzt (2.2). Als nächster Schritt wurde die Programmierung des Roboters erlernt (2.3). Anschließend wurde die Verbindung zwischen Roboter und externem Computer eingerichtet (2.4). Im Hauptteil wurden dann eine Anforderungsanalyse und Beispielprogramme der einzelnen Teilziele erstellt (Methodik Kapitel 3). In Kapitel 4 (Ergebnisse) wurden diese Teilziele in einer Anwendung zusammengefügt.

2 Grundlagen

In diesem Kapitel wird das Anwendungsszenario vorgestellt. Daraus leiten sich verschiedene Teilbereiche ab. Dazu gehören die Robotersteuerung, die Roboterprogrammierung, sowie die Kommunikation des Roboters mit einem externen Computer, als auch dessen Programm und Eingabemöglichkeit. Im Folgenden werden daher die notwendigen Grundlagen zu diesen Themenfeldern vermittelt.

2.1 Anwendungsszenario

In diesem Abschnitt werden das allgemeine Anwendungsszenario und die daraus resultierenden Teilbereiche beschrieben. Ein späterer klinischer Ablauf könnte wie folgt funktionieren:

2.1.1 Ablauf des Anwendungsszenarios

1. Der Patient legt sich auf den Behandlungstisch. Der Roboter befindet sich in seiner Ausgangslage.
2. Ein Arzt oder Assistent montiert ggf. einen speziellen Ultraschallkopf oder eine andere Halterung des Ultraschallkopfes am Roboter.
3. Er führt dann den montierten Ultraschallkopf von Hand zu dem Patienten.
4. Nacheinander kann er so den Ultraschallkopf an verschiedenen Stellen des Patienten positionieren und deren Position abspeichern. Der Roboter übermittelt die Positionen einem externen Computer im Kontrollraum.
5. Der Arzt oder Assistent verlässt den Raum, der Patient ist nun allein im Raum. Der Roboter geht zurück in seine Ausgangsstellung.
6. Von einem Kontrollpult außerhalb wird dem Roboter nun gesagt zu welcher Position er gehen soll.
7. Der Roboter bewegt sich zu seiner neuen Position und drückt anschließend den Ultraschallkopf gegen die Oberfläche des Patienten. Er überwacht die Kraft am Ultraschallkopf und kompensiert fortan die Atembewegung.
8. An der Position wird geschallt und über eine Eingabe am externen Computer lässt sich der Ultraschallkopf bewegen.
9. Der Tumor kann nun in den Ultraschallbildern markiert werden. Damit später nicht durch den Roboter gestrahlt wird, kann es notwendig sein den Tumor aus verschiedenen Richtungen lokalisieren zu können. Daher werden die drei vorhergehenden Schritte für ggf. alle gesetzten Positionen wiederholt.

10. Anschließend wird anhand der Einstahlrichtung festgelegt, zu welchem Zeitpunkt der Roboter von welcher Position aus schallen kann. Ferner müssen die Übergänge bei einer dynamischen Bestrahlung und Roboterbewegungen koordiniert werden.
11. Zuletzt folgt die Bestrahlung. Der Roboter durchläuft die definierten Abläufe. Das heißt er kompensiert die Atmung und wechselt ggf. die Position.
12. In Ausnahmefällen muss der Roboter auf Reaktionen des Patienten reagieren können. Damit ist gemeint, dass dieser zum Beispiel in Paniksituationen nicht einfach stehen bleibt und den Patienten festhält, sondern dynamisch ausweicht und auch keine anderen Personen in dem Raum verletzt.

2.1.2 Teilbereiche der Arbeit

Der Ablauf lässt erkennen, dass sich die Arbeit in folgende größere Bereiche unterteilt: Der erste Bereich ist der Roboter und dessen Interaktion mit dem Menschen. Wichtig hierbei ist das Führen von Hand und die spätere entfernte Steuerung des Roboters. Der nächste Bereich ist der Austausch des Roboters mit einem externen System bzw. Computer. Dieser ist notwendig um dem Roboter Befehle zu übergeben und von diesem Daten zu erhalten. Der letzte Bereich ist der externe Computer. Dieser verwaltet den Roboter und ermöglicht eine Steuerung des Roboters aus dem Kontrollzimmer heraus. Ferner ist dieser notwendig um die Ultraschallbilder darzustellen und den Tumor zu markieren.

2.2 Robotergrundlagen

Hier werden die allgemeinen Grundlagen und Besonderheiten des verwendeten Roboters erklärt. Die Grundlagen der Roboterprogrammierung stehen in Abschnitt 2.3. Die von KUKA bereitgestellte Dokumentation ist im Literaturverzeichnis gelistet: [3, 4, 5, 6, 7, 8, 9, 10]

2.2.1 KUKA lbr-iiwa-7-800

Bei dem verwendeten Roboter handelt es sich um den Leichtbauroboter KUKA lbr-iiwa-7-800. Dieser besitzt 7 Achsen. Daraus folgt, dass dieser Roboter 7 Freiheitsgrade hat. Darunter fallen die Kartesischen Koordinaten x, y, z sowie die Ausrichtung in Winkelmaßen a, b, c als auch eine alternativ Stellung des Roboters r. Diese erlaubt es ihm ohne Bewegung des Flansches eine Positionsveränderung vorzunehmen. Die üblicherweise in der Industrie verwendeten 6 Achsroboter bieten diese Funktion nicht. Jedoch ist diese Alternativstellung wichtig um später den Ultraschallkopf an der gleichen Position halten und gleichzeitig dem Bestrahlungsstrahl ausweichen zu

können. Die einzelnen Achsen werden von unten nach oben durchnummeriert, beginnend bei eins. Zusätzlich bietet dieser Roboter eine Kraftsensorik. Damit kann er auf einwirkende Kräfte reagieren und später so z.B. die Atembewegung kompensieren. Der verwendete Roboter ist in Abbildung 2.1 dargestellt.

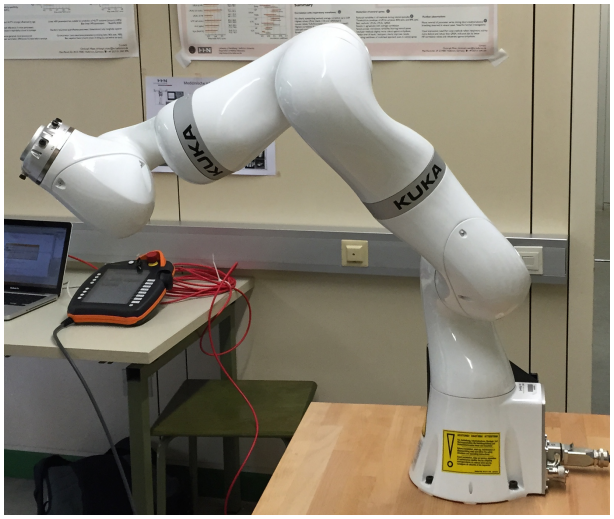


Abbildung 2.1: KUKA lbr-iiwa-7-800

2.2.2 Bewegungsarten

Der Roboter unterstützt mehrere Bewegungsarten:

- **Linearbewegung (LIN):** Hierbei fährt der Roboter auf einer Geraden von einem Punkt zum Nächsten.
- **Point to Point (PTP):** Ein Punkt wird direkt angefahren. Direkt bedeutet, der Roboter sucht sich den für ihn schnellsten Weg aus. Dieser kann auch geschwungen sein und muss nicht auf einer Ebene liegen.
- **Kurvenbewegung (CIRC):** Dabei werden über drei Punkte die Werte einer Kreisbahn beschrieben, die der Roboter verfolgen soll.

Wichtig ist hier, später die richtige Bewegungsart zu verwenden. Gerade bei der PTP Bewegung lässt sich die Roboterbewegung nur schwer vorhersehen. Damit der Roboter sich später nicht durch den Patienten bewegt, müssen diese Bewegungsarten z.B. kombiniert nacheinander ausgeführt werden.

2.2.3 Impedanz

Unter Impedanz versteht man, dass der Roboter nachgiebig in einer oder mehreren Achsen agiert. Das bedeutet, dass der Roboter ab einer bestimmten Kraft seine vorgegebene Bewegung verlässt und der Kraft nachgibt. Hierfür muss der Roboter die Kraft, die auf ihn einwirkt messen können. KUKA bietet hierfür eine Kartesische-Impedanz an [3, S. 335ff]. Diese erlaubt eine Kartesische-Impedanz sowohl in dem Tool Center Point(z.B. Spitze des Ultraschallgerätes) (TCP) des Werkzeugs, der Roboterbasis oder dem Flansch einzustellen. Mit Hilfe von weiteren Parametern lässt sich die Nachgiebigkeit des Roboters verfeinern. So kann zum Beispiel die maximal erlaubte Kraft, oder die Federsteifigkeit und Amplitude der virtuellen Feder, bei Auslenkung mit einer externen Kraft gesteuert werden.
Die Impedanz wird später verwendet um auf die Bewegungen des Patienten reagieren zu können. Durch die Nachgiebigkeit während der Roboterbewegung wird zusätzlich die Sicherheit erhöht.

2.2.4 Singularität

Bei einer Singularität kann der Roboter seine aktuelle Bewegung nicht weiter ausführen, da seine Achsstellung das nicht zulässt oder der Roboter sich mit unendlicher Geschwindigkeit in einer oder mehreren Achsen drehen müsste, um die Zielvorgabe zu erfüllen. Vermeiden lässt sich die Singularität durch definierte Übergänge oder Stoppunkte, in denen der Roboter seine Achsstellung für die nächste Bewegung vorbereitet. Da sich Singularitäten nicht grundsätzlich vermeiden lassen, braucht es in der späteren Anwendung Algorithmen, die eine Singularität erkennen und den Roboter aus diesen lösen kann.

2.2.5 SmartPad

Mit Hilfe des SmartPad lässt sich der Roboter bedienen, dazu gehört:

- Manuelles verfahren des Roboters
  - Einstellen des Koordinatenursprungs
  - Kartesisch oder achsspezifisch verfahren
  - Auswählen eines eingespeicherten Werkzeugs
- Starten von Programmen
- Benutzerinteraktion während der Programmausführung (2.3.5)
- Ansicht des aktuellen Status des Roboters
- Notaus

- Justieren von einzelnen Achsen
- Anzeigen von Log-Meldungen und Fehlerberichten

Weitere Funktionalitäten des SmartPads stehen im Benutzerhandbuch [3, S. 59ff] Das SmartPad ist eine der wichtigsten Schnittstellen des Roboters. In der späteren Anwendung wird es gebraucht um direkt mit dem Roboter zu kommunizieren, z.B. Abspeichern von Positionen über Dialoge. Außerdem wird es benötigt um das Roboterprogramm zu starten. Ferner ist wichtig, dass bei Fehlverhalten des Roboters, z.B. Hardwaregrenzen wurden beim Verfahren überschritten, durch Einstellen des KRF-Modus und sicheres Verfahren der Achsen am SmartPad beheben lassen.

2.3 Roboterprogrammierung

Die Programmierung des Roboters erfolgt in Java (Version 1.6). Dazu bietet KUKA eine eigene Entwicklungsumgebung (2.3.1) an. Zum Nachlesen und Ergänzen der hier beschriebenen Grundlagen stellt KUKA eine noch detailliertere Betriebsanleitung für den Roboter bereit [3]. Da die Programmierung eines Roboters in Java eher speziell und für diese Art von Robotern konzipiert ist, werden in diesem Kapitel die notwendigen Grundlagen der Roboterprogrammierung gezeigt. Es wird im Folgenden jedoch nur auf die für die spätere Programmierung relevanten Grundlagen der Roboterprogrammierung eingegangen.

2.3.1 Sunrise OS

Sunrise OS ist die Entwicklungsumgebung für den Roboter. Sie setzt auf Eclipse auf und ermöglicht die von KUKA bereitgestellten Roboterklassen zu nutzen. Weiterhin enthält sie roboterspezifische Ergänzungen. Dazu gehören Einstellmöglichkeiten im Bereich Sicherheit, Werkzeugdefinition und Erstellen von Frames(2.3.3). Ein Screenshot von Sunrise OS befindet sich im Anhang (A.1). Um Roboterprogramme zu schreiben wird Sunrise OS benötigt. Ferner überträgt es das Projekt vom lokalen Rechner auf die Robotersteuerung.

2.3.2 RA-Klasse

Jede lauffähige Roboter-Applikation (RA) benötigt eine RA-Klasse. Standardmäßig lässt sich diese über Sunrise OS erzeugen. Eine neu generierte RA-Klasse ist im Anhang: B.1 dargestellt. Die RA-Klasse erbt von RoboticsAPIApplication. Auf dem Roboter können nur Programme mit dieser Hauptklasse gestartet werden. Die wichtigsten Methoden sind:

- **initialize():** setzt die Felder mit den für den späteren Verlauf wichtigen Roboterklassen. Erfüllt die Aufgabe des Konstruktors.

- **run():** Diese Methode ist der Kern der RA. Sie wird nach dem initialisieren gestartet und wenn diese fertig ist, wird die RA beendet. Es bietet sich an, hier eine bedingte Endlosschleife zu erstellen, da Befehle wie system.exit(0) zu Fehlern führen. In Listing B.1(Anhang) bewegt sich der Roboter zu seiner Ausgangsposition.
- **main(String[] args):** klassische java main-Methode startet die Applikation. Hierzu werden intern erst die initialize() Methode und zuletzt die run() Methode ausgeführt.

2.3.3 Frame

Unter einem Frame versteht man eine Position des Roboters oder eine relative Position zu einem anderen Frame. Mithilfe von Sunrise OS (2.3.1) lässt sich ein ganzer Frame-Baum erstellen. Die untergeordneten Frames beziehen sich dabei auf die Vorhergehenden. Solche abgespeicherten Frames lassen sich wie in Listing 2.1 aufrufen. Für die spätere Anwendung ergibt sich daraus, dass allgemeine Positionen, wie z.B. Wartepositionen oder Ausgangsstellungen, direkt in Sunrise OS als Frame oder gar Frame-Baum definiert werden können.

```
1     robot.move(ptp(getApplicationData().getFrame("/Position1")));
```

Listing 2.1: Beispiel für das Aufrufen eines abgespeicherten Frames

2.3.4 Werkzeuge einstellen

Damit der Roboter später ein Werkstück oder Werkzeug bewegen und steuern kann, ist es notwendig ihm dieses beizubringen. Mit Hilfe von Sunrise OS (2.3.1) können Werkzeuge eingestellt werden. Ist ein Werkzeug eingestellt kann es zur Bewegung verwendet werden. In Abbildung 2.2 ist dargestellt, wie ein Frame (TCP) des Werkstückes definiert werden kann. In dem Beispiel handelt es sich um einen Ultraschallkopf, der rechtwinklig zum Roboterflansch mit einem Abstand in x, y und auch in z Richtung montiert ist. Wählt man das Werkstück direkt aus, so kann man dessen Masseschwerpunkt und dessen Gewicht angeben. Das ist wichtig, damit der Roboter seine Bahnplanung korrekt umsetzen kann. Ist das Gewicht größer als 1 kg, kann der Roboter den Schwerpunkt der Masse des Werkzeugs selber vermessen. Es lassen sich beliebig viele TCPs festlegen. In Abbildung 2.2 markiert der Punkt neben dem Koordinatensymbol den default TCP. Das heißt eine Bewegung geht standardmäßig von diesem aus. In Abbildung 2.2 ist so das Werkzeug Ultraschallkopf1 mit drei unterschiedlichen TCPs erstellt worden. Wobei das Frame UltraschallkopfSpitze die Transformation vom Flanschzentrum zur Spitze des Ultraschallkopfes beschreibt.

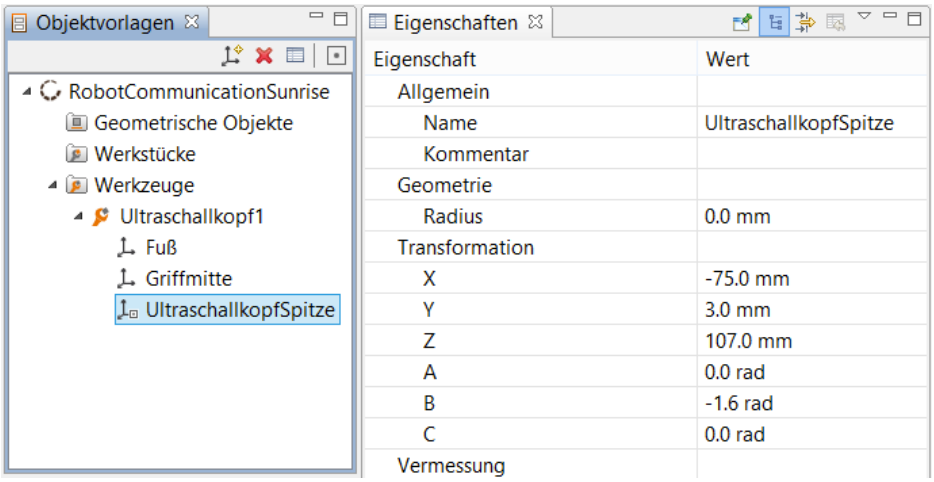


Abbildung 2.2: Mit Hilfe von Sunrise OS können Werkzeuge erstellt werden. In einem untergeordneten Frame kann man die Transformation zu diesem angeben, hier die Ultraschallspitze

Im Programmcode kann das eingegebene Werkzeug wie in Listing 2.2 angegeben abgerufen werden. Erst durch den attach()-Befehl werden Werkzeuge oder Werkstücke mit dem Flansch oder anderen Objekten verbunden. Dadurch ist es jetzt möglich Bewegungsbefehle von der Ultraschallkopfspitze zu definieren. In der späteren Anwendung lässt sich so z.B. eine Drehbewegung um die Spitze des Ultraschallkopfes ohne großen Aufwand realisieren.

```
1    ultrasoundHead = getApplicationData().createFromTemplate(
2        "Ultraschallkopf1");
3    ultrasoundHead.attachTo(robot.getFlange());
```

Listing 2.2: Beispiel im Code zum Anwenden und Anbringen eines Werkzeugs

2.3.5 Benutzerinteraktion mit SmartPad

Benötigt man die direkte Kommunikation mit dem Roboter, so eignet sich das SmartPad als Ein- und Ausgabegerät. Es lassen sich die nicht verwendeten Buttons sowie eigene Benutzerdialoge programmieren. In Listing 2.3 ist dargestellt wie ein Fragedialog mit drei Antwortmöglichkeiten aussehen kann. Als Antwort wird ein Integer geliefert, der nun mit einer switch-case-Anweisung koordiniert werden kann.

Die Programmierung eines Eingabebuttons ist genauso einfach. Ein einfaches Codebeispiel für einen Button befindet sich im Anhang: B.2. Die Funktionsweise ist dieselbe, wie z.B. ein JButton und dessen Listener. Die Gestaltungsmöglichkeiten des Buttons

```
1     int answer = getApplicationUI().displayModalDialog(
2         ApplicationDialogType.QUESTION, "Soll ich mich bewegen?",
              "Ja",
3         "Nein", "Ein bisschen");
4     switch (answer) {
```

Listing 2.3: Beispiel für einen Benutzerdialog

sind etwas komplexer und können dem KUKA-Benutzerhandbuch [3, S. 313ff] entnommen werden.

Für die Anwendung ergibt sich daraus, dass für den Teil, bei dem der Assistent oder Arzt im Raum des Patienten ist, dieser direkt über das SmartPad den Roboter steuern kann. Es wird damit kein weiteres Eingabedevice (z.B. Computer) im Behandlungszimmer benötigt. Das vermindert die Komplexität und erleichtert die Bedienung.

2.3.6 Bewegungsbefehle

Um den Roboter zu Bewegen, gibt es verschiedene Bewegungsbefehle. Später werden die hier verwendeten Bewegungsbefehle zum Vorpositionieren verwendet. Eine dynamische und zyklische Bewegung wird über SmartServo realisiert (Abschnitt 2.3.7). Ein Bewegungsbefehl setzt sich aus folgenden Komponenten zusammen:

- **Art der Ausführung**
  - **synchrone Ausführung mit move(...):** Hierbei wird die RA solange unterbrochen, bis der Befehl ausgeführt wurde. In Listing 2.4 wird Zeile 2 erst aufgerufen, wenn die Bewegung in Zeile 1 abgearbeitet ist.
  - **Asynchrone Ausführung mit moveAsync(...):** Bei der asynchronen Bewegung wird sofort die nächste Zeile ausgeführt. in Listing 2.4 Zeile 3 wird nach Aufrufen der Bewegungsmethode sofort die Methode doAwesomeThings() ausgeführt. Werden hier neue Bewegungsziele aufgerufen, so wird die aktuelle Bahn des Roboters überschliffen.
- **Objekt:** Vor dem eigentlichen Bewegungsbefehl wird angegeben, von welchem Objekt diese Bewegung ausgeht. In Listing 2.4 ist zu erkennen, wie in Zeile 1 die Bewegung vom Roboter-Welt-Koordinatensystem ausgeht, da hier direkt nach dem Roboterobjekt der Bewegungsbefehl kommt, wohingegen in Zeile 3 der Flansch das Zentrum der Bewegung ist. Hier ist es auch möglich angebrachte Werkzeuge und deren Frames als Objekte in der Bewegungsplanung zu verwenden.
- **Bewegungsart** Wie in Abschnitt 2.2.2 vorgestellt kann die Art der Bewegung festgelegt werden. In Listing 2.4 Zeile 2 steht das ptp(...) für eine PTP-Bewegung.

```
1     robot.move(ptpHome());
2     robot.move(ptp(getApplicationData().getFrame("/Position1")));
3     robot.getFlange().moveAsync(linRel(x, y, z));
4     doAwesomeThings();
5     robot.move(ptp(new JointPosition(0.5, 0.8, 0.2, 1.0, -0.5,
          -0.5, -1.5))
6         .setJointVelocityRel(10));
```

Listing 2.4: Beispiel für die Verwendung von Frames und Bewegungsbefehlen

2.3.7 SmartServo

SmartServo beschreibt eine eigene Bewegungsumgebung für den Roboter. Sie bietet eine Möglichkeit zyklische Zielvorgaben zu setzen, die der Roboter durch Überschleifen der aktuellen Bahn anfahren kann. Sie erlaubt somit dem Roboter in 25 ms Zyklen neue Zielvorgaben zu geben. Über die Klasse SmartServo lassen sich später dynamische Schwenkbewegungen des Ultraschallkopfes in Echtzeit realisieren. Das bedeutet, dass die Werte des Eingabegerätes sofort die Bewegung des Ultraschallkopfes verändern können.
Eine detaillierte Anleitung für SmartServo wird von KUKA bereitgestellt [4]. KUKA bietet auch drei Beispielprogramme, von denen leider nur eins funktioniert hat. Daher hier der Hinweis: SmartServo braucht eine lange Einarbeitungszeit um richtig verstanden zu werden und hat noch seine Tücken.
Die Grundidee von SmartServo lässt sich in [4][S.25ff] nachlesen:

1. Eine neue Instanz der SmartServo-Klasse erzeugen
2. Ausführen einer asynchronen Bewegung mit einem Werkzeug
3. Die Laufzeitumgebung von SmartServo abfragen
4. Zyklische Zielvorgaben an den Roboter übergeben
5. Die Bewegung beenden

2.4 Datenaustausch

Um den Roboter wie in dem Anwendungsszenario (Abschnitt 2.1) beschrieben von einem externen Computer aus steuern zu können, wurde ein Verbindungsprotokoll benötigt. Es wären mehrere Verbindungsprotokolle möglich gewesen. Die Abwägung und Analyse der alternativen Möglichkeiten ist in Abschnitt 3.7 zu finden. Es wurde sich hier für Open Network Interface for Image- Guided Therapy (OpenIGTLink) entschieden. Es bietet die beste Möglichkeit verschiedene Befehle und Daten an den Roboter bzw. den externen Computer zu senden. Daher wird im Folgenden nur dieses erläutert.

2.4.1 OpenIGTLink

Bei Open Network Interface for Image- Guided Therapy (OpenIGTLink) handelt es sich um eine Open Source Library. Diese bietet eine Schnittstelle zum Schicken von Nachrichten im medizinischen Bereich. Eine Ergänzung für Roboternachrichten ist bereits angedeutet und in einem Projekt umgesetzt worden [11]. Im Weiteren wird der Aufbau von OpenIGTLink in Java erklärt. Downloadlink und eine ausführliche Dokumentation befindet sich auf der OpenIGTLink-Seite[12]. Da jedoch speziell die Java Version nur spärlich dokumentiert ist sind hier die wichtigsten Klassen im Überblick:

- **OpenIGTMessage:** Ist die Hauptnachrichtenklasse, von ihr müssen alle anderen Nachrichtenklassen abgeleitet werden, um später die Nachrichten verschicken zu können. Diese Klasse enthält zwei wichtige Methoden:
  - PackBody()
  - UnpackBody()

  Die Methoden sind dafür zuständig, dass aus den Datenfeldern Bytecode bzw. aus dem Bytecode Datenfelder erstellt werden. Die Klasse enthält außerdem den Header der Nachricht. Er enhält den Typ der Nachricht, welcher für die Interpretation der abgeleiteten Nachricht genutzt wird. Einen Gerätenamen, an wen die Nachricht geht, sowie einen Zeitstempel und die Größe des Nachrichtenkörpers.
- **OpenIGTClient:** Client für die Verbindung. Benötigt einen ResponseHandler, der die Nachrichten interpretiert. Dafür wird mit jeder eingehenden Nachricht ein neuer ResponseHandler erstellt, der in eine Warteschlange gesteckt wird.
- **ResponseHandler:** Interpretiert die Nachrichten, die vom Server geschickt werden. Erzeugt dafür die Nachrichteninstanzen aus dem übergebenen Bytecode.
- **OpenIGTServer:** Server-Klasse, die ähnlich aufgebaut ist wie der OpenIGT-Client.
- **MessageHandler:** Synonym zum ResponseHandler auf Seiten des Servers.

Da die Verbindung von OpenIGTLink bidirektional ist, lassen sich Server und Client gleich verwenden. In der späteren Anwendung ist der externe Computer der Server. Das erlaubt später z.B. weitere Roboter an das System anzubinden, die über den Computer zentral verwaltet werden. Außerdem kann sich der Roboter so beim Start nur bei einer festen IP Adresse anmelden. Damit ist es später nicht möglich den Roboter fälschlicherweise von einem anderen Computer zu steuern. Ferner ist die Firewall des Roboters darauf ausgelegt Verbindungen von außen grundsätzlich zu blockieren. Eine Anwendung mit Server auf dem Roboter ist damit schwer zu konstruieren. Zusätzlich besteht hier das Problem, dass laufende Serverthreads bei einem Programmabsturz nicht beendet werden und ein Neustart des Systems, durch blockierte Ports, notwendig ist. Daher ist der Roboter der Client.

2.5 Externe Roboterverwaltung und Steuerung

Das Anwendungsszenario (Abschnitt 2.1) beschreibt, dass der Roboter von einem externen Computer gesteuert werden soll. Das bedeutet, der externe Computer im Kontrollzentrum muss später das Ultraschallbild darstellen und eine Oberfläche zum Steuern des Roboters bieten. Außerdem benötigt es Funktionen um unterschiedliche Ultraschallköpfe oder Einstellungen zu speichern. Zum Schwenken des Ultraschallkopfes, wäre eine Steuerung über Maus und Tastatur umständlich gewesen, deshalb wurde hier ein Joystick verwendet.

2.5.1 Joystick

Um die Eingabe eines Joysticks zu verwerten, wurde die Lightweight Java Game Library (LWJGL) 2.9.1 verwendet. Die Dokumentation zu den Klassen befindet sich auf der LWJGL-Homepage [13]. Da hier nur die Steuerung des Joysticks gebraucht wurde, wurden nur folgende drei Klassen verwendet:

- net.java.games.input.ControllerEnvironment: Liefert die angeschlossenen Joysticks
- net.java.games.input.Controller: Eine Klasse die den Controller repräsentiert (Joystick)
- net.java.games.input.Component: Die Buttons oder Achsen des Joysticks

Um den Joystick verwenden zu können, wird nur ein Thread benötigt, der die Knöpfe und Achsen pollt und einem Action-Handler übergibt.

2.5.2 Einstellungs- und Datenspeicherung.

Zum Speichern von Einstellungen wurde Extensible Markup Language (XML) verwendet. Zum Parsen und Erstellen der XML-Dokumente wurde Java Architecture for XML Binding (JAXB) verwendet. Die Dokumentation von JAXB kann der Oracle Seite [14] entnommen werden. In der späteren Anwendung werden darüber die Einstellungen des Joysticks und die zur Verfügung stehenden Tools gespeichert.

2.6 Stand der Technik

Zum Steuern eines Leichtbauroboters, aus einem medizinischen Bilddarstellungsprogramm heraus, gibt es einen Ansatz und Umsetzung mit 3D Slicer. 3D Slicer ist ein Visualisierungsprogramm für medizinische Bilddaten [15]. Bei dieser Arbeit wurde 3D Slicer mit Hilfe von OpenIGTLink und einem KUKA-Leichtbauroboter um eine Robotersteuerungs- und Darstellungskomponente ergänzt. Der Roboter kann in 3D Slicer in Echtzeit dargestellt werden, außerdem ist es möglich, Positionen anzufahren. [11]

3 Methodik

In diesem Kapitel werden die für das Ergebnis notwendigen Teilschritte und deren Lösungen präsentiert. Das zugrunde gelegte Anwendungsszenario wird in Abschnitt 2.1 detailliert beschrieben. Zusammengefasst stellte sich folgendes Problem:
Ultraschall beruht auf dem Prinzip der Schallausbreitung. Hierzu wird gemessen wie lange der Schall unterwegs ist. Unterschiedliche Gewebestrukturen lassen den Schall unterschiedlich schnell hindurch. An Organgrenzen kommt es zu Reflektionen des Ultraschalls. Befindet sich Luft z.B. im Darm wird der Schall komplett reflektiert. Es lässt sich somit nicht durch Luft schallen. Für die Ultraschalldiagnostik bedeutet es, dass ein dynamischer Anpressdruck notwendig ist, um keine Luft zwischen Schallkopf und Patient zu haben. Beim konventionellen Ultraschall, positioniert der Arzt den Ultraschallkopf an der richtigen Position auf dem Patienten mit genügend Anpressdruck. Will man somit den Arzt durch einen Roboter ersetzen um bei der Bestrahlung weiterhin ein Ultraschallbild zu erhalten, muss eine Apparatur diese Fähigkeiten besitzen. Eine starre Halterung scheidet somit aus. Ein Roboter kommt in Frage, benötigt jedoch eine Sensorik um auf einwirkende Kräfte dynamisch reagieren zu können. Der verwendete Roboter besitzt diese und soll damit auf eine mögliche Anwendung getestet werden. Ferner spielt auch die Positionierung des Ultraschallkopfes an den Roboter eine Rolle. Je nachdem wie dieser montiert ist, kann der Roboter seine Achsen nutzen oder ist sich selber im Weg.
Stimmt die Ankopplung des Ultraschallkopfes durch den Roboter braucht es eine Kommunikationsmöglichkeit um vom Kontrollraum den Roboter steuern zu können. Dieses muss sowohl schnelle Steurungsbefehle z.B. zum Schwenken des Ultraschallkopfes, als auch Nachrichten zum Aufrufen von verschiedenen Abläufen unterstützen.
Im Kontrollraum selber braucht es nun einen Computer, der den Roboter steuert. Darunter fällt auch die Darstellung des Ultraschallbildes. Da ein Arzt durch kleine Bewegungen die Position des Ultraschallkopfes verändern kann um eine andere Schnittebene zu sehen, ist es notwendig eine Möglichkeit zu bieten den Ultraschallkopf von einem Computer aus zu steuern. Hierfür ist eine mögliche Eingabevariante zu suchen, die es ermöglicht diese Schwenkbewegungen in allen Richtungen zu unterstützen.

3.1 Einteilung in Teilschritte

Da das Anwendungsszenario sehr komplex ist, war es notwendig die Gesamtanwendung erst durch kleine Teilschritte und Beispiele anzunähern und später diese zu einem System zusammenzufügen. Für ein generelles Verständnis ist eine Einarbeitung in den Roboter als erster Schritt hilfreich. Da dieser Schritt jedoch nur indirekt zur Lösung beitrug wird er hier nicht weiter erklärt. Anzumerken ist nur, dass die Einarbeitungszeit für den Roboter nicht unterschätzt werden darf!
Aus dem Anwendungsszenario lässt sich in einem ersten Schritt eine **Anforderungsanalyse** erstellen. Weiter lässt sich der zeitliche Ablauf des Anwendungsszenarios (siehe Abschnitt 2.1) in Teilkomponenten des späteren Systems einteilen. Bevor der Roboter einen Ultraschallkopf bewegen kann, benötigt es eine Montagemöglichkeit für den

Ultraschallkopf. Dazu gehört auch dessen Entwicklung (**Ultraschallkopfhalterung**). Am Anfang der Anwendung muss sich der Roboter von Hand führen lassen und Positionen abspeichern können (allgemein **Positionen beibringen**). Später muss der Roboter diese **Posititionen anfahren** können. Weiter muss er dort selbständig die **Atemkompensation** durchführen. Um ihn von außerhalb steuern zu können benötigt es eine **Kommunikationsschnittstelle**. Zum Nachjustieren der vorher angefahrenen Positionen braucht es ein **Eingabegerät**, welches die Bewegung des Ultraschallkopfes von einem externen Computer aus ermöglicht. Zusätzlich benötigt es ein Beispiel für das Zusammenwirken des Eingabegerätes, der Kommunikationsschnittstelle und der Steuerung des Roboters (**Zusammenwirken der einzelnen Teilschritte**). Die bisherigen Teilschritte müssen in einem **Entwurf** zusammengefasst werden. Aus den dick gedruckten Wörtern ergeben sich die Teilschritte die in den folgenden Abschnitten behandelt werden.
Anzumerken ist hier, dass diese Teilschritte sich später in drei größere Komponenten unterteilen lassen. Das eine ist die Anwendung auf dem Roboter. Dazwischen die Kommunikationsschnittstelle und zuletzt der Steuerungscomputer.

3.2 Anforderungsanlayse

Die Anforderungsanalyse teilt sich in die Bereiche Roboter, Kommunikationsschnittstelle und externer Computer auf. Sie ergibt sich aus dem in Abschnitt 2.1 vorgestellten Anwendungsszenario.

3.2.1 Anforderungen an den Roboter

Die Anforderungen an den Roboter beziehen sich sowohl auf die RA als auch auf den Roboter selbst.

- **Positionen beibringen und speichern:** Es muss möglich sein dem Roboter Positionen beizubringen und diese zu speichern. Darunter fällt auch:
  - **Bewegen von Hand:** Um einzelne Punkte schnell angeben zu können ist es notwendig den Roboter direkt mit der Hand zur gewünschten Position zu führen. Eine Steuerung mit Hilfe der Achsen oder des kartesischen Arbeitsraumes wäre hier zu umständlich. Die Bewegung von Hand sollte nicht gestartet werden, wenn bereits zu viel Druck auf den Roboter ausgeübt wird.
  - **Speichern der Positionen:** Die angesteuerten Positionen müssen für die spätere Verwendung abgespeichert werden und dem Steuerungscomputer übermittelt werden.

- **Selbständigkeit:** Der Roboter soll möglichst selbständig sein. Gemeint ist, das der Roboter z.B. Programmabläufe und Kraftmessungen selbständig ausführt und nur im Ausnahmefall Rückmeldung an den Steuerungscomputer gibt.
- **Direkte Benutzerinteraktion:** Für den Fall, dass es notwendig ist direkt mit dem Roboter zu kommunizieren, sollen geeignete Dialoge für das SmartPad entwickelt werden.
- **Verwendung von unterschiedlichen Werkzeugen:** Es soll möglich sein, dem Roboter unterschiedliche Aufsätze z.B. verschiedene Ultraschallköpfe an den Flansch zu montieren. Wichtig hier ist z.B. das Festlegen des TCP und des Schwerpunktes[1] des Werkzeugs.
- **Positionen anfahren:** Es muss möglich sein, zuvor gespeicherte Positionen anzufahren. Dabei muss es ein Konzept geben, wie der Patient bei dem Anfahren verschiedener Positionen berücksichtigt wird. Damit ist gemeint, das der Roboter sich nicht durch den Patienten bewegen darf.
- **Komplexe Bedienbarkeit:** Es muss möglich sein unterschiedlichste Befehle ausführen zu können ohne ein neues Roboterprogramm manuell starten zu müssen. Daraus folgt: Der Roboter muss unterschiedliche Kommandos interpretieren und ausführen können. Er muss diese verwalten und kontrollieren. Damit später ergänzende Befehle auch interpretiert werden können muss die RA erweiterbar sein.
- **Sicherheit** Die wohl wichtigste Anforderung an den Roboter, bei Verwendung im medizinischen Umfeld ist die Sicherheit. Darunter fallen:
  - **Kraftüberwachung beim Ultraschallen:** Damit das Ultraschall einwandfrei funktionieren kann ist es notwendig für genügend Anpressdruck zu sorgen. Es muss darauf geachtet werden, dass der Roboter nicht zu viel Kraft auf den Körper ausübt.
  - **Panik des Patienten:** Beim Bewegen des Roboters ist dieser nicht immer nachgiebig. Damit in einer Panikreaktion des Patienten dieser nicht zwischen Roboter und Liege gefangen ist, braucht es einen Mechanismus um auf diese Gefahr zu reagieren.
  - **Positionsreferenzierung:** Es muss gewährleistet sein, dass angegebene Positionen auch erreicht und korrekt angefahren werden.[2]
  - Der Roboter darf sich nicht durch den Patienten bewegen.

[1] Wichtig für die korrekte Bahnplanung des Roboters
[2] Das Referenzieren mit Hilfe von Trackingsystemen ist nicht Teil dieser Arbeit.

3.2.2 Anforderungen an den Datenaustausch

- **Erweiterbarkeit:** Es muss möglich sein neue Befehls- oder Daten-Nachrichten zu erstellen und einzubinden. Notwendig, da später von außerhalb der Roboter gesteuert werden muss. Dieser sollte viele seiner bereits vorhandenen Befehle und Möglichkeiten nutzen können, daher ist eine gut erweiterbare Kommunikationsschnittstelle unerlässlich.
- **Sprachunabhängigkeit:** Da die RA in Java und Bildverarbeitungsprogramme eher in C++ geschrieben sind, wäre eine Sprachunabhängigkeit wünschenswert. Ansonsten ist Java Voraussetzung.

3.2.3 Anforderungen an den Steuerungscomputer

- **Eingabegerät:** Es muss eine Eingabemöglichkeit gesucht werden, die:
  - Schwenkbewegungen des Ultraschallkopfes ermöglichen.
  - Die sich auf die späteren Achsen des Ultraschallkopfes kalibrieren und einstellen lässt.
  - Eine Möglichkeit besitzt diese Einstellungen zu speichern.
- **Anforderungen an die grafische Benutzeroberfläche (engl. graphical user interface) (GUI):** Damit einfaches Arbeiten mit dem Roboter möglich ist benötigt es eine GUI mit folgenden Eigenschaften:
  - Es muss eine einfache GUI zum Steuern des Roboters geben.
  - Nach Möglichkeit sollte der Roboter in seiner Position oder in seinen aktuellen Koordinaten dargestellt werden.
  - Über die GUI muss es möglich sein, die Einstellungen der Eingabe für die manuelle Steuerung zu verändern.
  - Die Befehle zum Ausführen auf dem Roboter müssen passend in der GUI repräsentiert werden.
  - Durch eine Eingabe auf der GUI darf es nicht zu Fehlern beim Roboter kommen. Rückmeldungen des Roboters müssen interpretiert werden.
  - Später muss es eine Echtzeitdarstellung des Ultraschallbildes geben.

3.3 Ultraschallkopfhalterung

Zu Beginn der Arbeit wurde eine Halterung für den Ultraschallkopf hergestellt. Diese erlaubte es den Ultraschallkopf direkt rechtwinklig an den Roboterflansch zu montieren. Diese Variation stellte sich jedoch als unbrauchbar dar, da der Roboter zu nahe an der Spitze des Ultraschallkopfes war und somit keine großen Schwenkbewegungen möglich waren. Da durch diese Ausrichtung nur wenige Positionen möglich waren, musste die Halterung neu entwickelt werden.

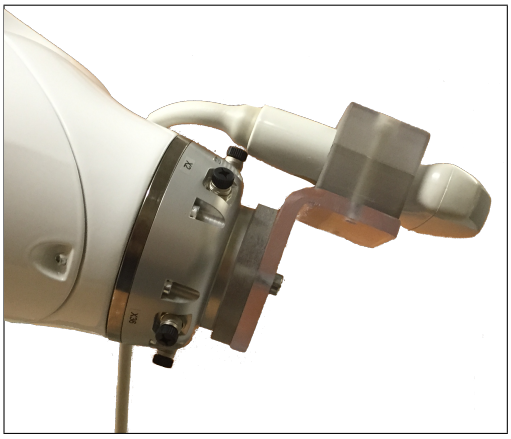
3.3.1 Entwicklung

Im Anwendungsszenario wurde bereits beschrieben, dass unterschiedliche Ultraschallköpfe und Ausrichtungen verwendet werden können. Diese Tatsache ergab sich aus der Fragestellung wie die ideale Halterung für den Ultraschallkopf aussieht. Durch die bereits erstellte Variante ist klar geworden, dass die Halterung großen Einfluss auf die Funktionalität des Roboters hat. So lässt sich die erste Halterung weder groß schwenken noch drehen. Durch die Ausrichtung des Ultraschallkopfes war der Roboter schlicht zu klein um den Ultraschallkopf an allen Positionen um mehr als 90° um die Griffachse zu drehen. Damit der Roboter fix an einer Position im Bestrahlungszimmer stehen kann, benötigt dieser je nach Anwendung eine andere Ausrichtung vom Flansch aus betrachtet. Ideal ist es, wenn die Ultraschallkopfspitze nah am Flansch-Zentrum ist, damit der Roboter seine Achsen nutzen kann und kleine Drehbewegungen des Ultraschallkopfes nicht große Bewegungen des Roboters zur Folge haben. Jedoch darf der Ultraschallkopf nicht so nah am Flansch montiert sein, dass der Roboter bei Schwenkbewegungen durch den Patienten muss. Um die bereits erstellten Teile trotzdem nutzen zu können, bot es sich als Lösung an, Zwischenstücke zu konstruieren, die es ermöglichen den Ultraschallkopf in verschiedenen Winkeln an den Flansch zu befestigen.

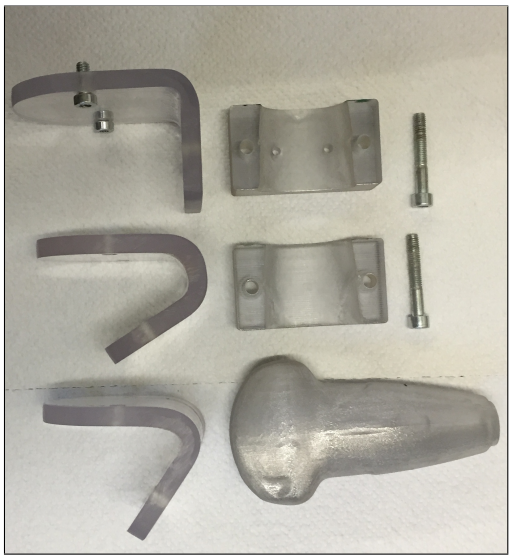
3.3.2 Umsetzung

Die Halterung des Ultraschallkopfes und die Montageplatte am Flansch wurden additiv gefertigt (ausgedruckt). Die Zwischenstücke hätten auch mit diesem Verfahren hergestellt werden können. Einfacher und vor allem schneller war jedoch die Herstellung mit einer 8 mm dicken Acrylglasplatte. Diese wurden passend zu gesägt und mit den notwendigen Bohrungen und Gewinden versehen. Anschließend wurden diese an einer Geraden entlang erhitzt und um den gewünschten Winkel gebogen. In Abbildung 3.1 sind die aktuellen Montagemöglichkeiten für den Ultraschallkopf gezeigt. Die aktuelle Halterung (Abbildung 3.1a) erlaubt ein großes Schwenkfeld und ermöglicht es alle Positionen zu erreichen. Mit diesem Verfahren wurden insgesamt drei Winkel mit 90°,45° und 30° gefertigt. Zu beachten bei dieser Vorgehensweise ist, dass die Schrauben sich noch in den Flansch schrauben lassen und nicht durch den Winkel

blockiert werden. Weiter bieten diese Winkel die Möglichkeit einen Griff rechtwinklig zum Ultraschallkopf zu montieren. Dieser enthält zusätzlich den Taster für die Positionsbestätigung und einen Dreiwege-Sicherheitsschalter für das manuelle Führen. Da dieser Griff jedoch aufgrund der kurzen Zeit nicht sonderlich schick aussieht, wird hier auf eine Abbildung verzichtet.



(a) aktuell verwendeter Montageaufsatz



(b) Ultraschallkopfdummy und alle Montageteile: links die selbstgebauten Zwischenstücke, rechts die Ausgangsbauteile

Abbildung 3.1: Ultraschallkopf Montagemöglichkeiten auf Roboter

3.3.3 Einrichten des Werkzeugs

Um das Werkzeug verwenden zu können, muss es wie in Abschnitt 2.3.4 dem Programm übergeben werden. Dafür braucht es die Transformationswerte x, y, z, und Rotationswerte a, b, c. In diesem Fall waren diese aufgrund der Eigenkonstruktion nicht vorhanden bzw. nicht ausreichend genau bekannt. Um diese zu ermitteln wurde wie folgt vorgegangen. Rotationen (a, b, c) wurden gemessen (Drehbewegung um den Ultraschallkopf. Eine Abbildung für die Bezeichneten Achsen befindet sich in Abschnitt 3.8). Es bleibt somit nur noch die Transformation:

1. Grobes Vermessen der Koordinaten von Ultraschallkopfspitze bis Flanschmitte
2. Hochladen dieser Einstellungen
3. Auswählen des Werkzeugs im SmartPad
4. Schwenken um eine der Achsen mit den Winkeln a, b, c an einem Referenzpunkt

- Liegt der eingegebene TCP vor der Ultraschallkopfspitze wird sich der Ultraschallkopf um den Referenzpunkt bewegen. Ggf. lässt sich dieser Radius feststellen und kann bei der x, y oder z Achse addiert werden.
- Ist der eingegebene TCP in dem Ultraschallkopf, schwenkt sich der Kopf vor dem Referenzpunkt. x-, y- oder z-Achse muss verlängert werden.

5. Korrektur der Eingaben:
   - Beim Schwenken um a: Liegt der TCP mittig auf der y-Achse des Flansches muss nur in x-Richtung wie oben beschrieben korrigiert werden.
   - Beim Schwenken um b: z-Richtung
   - Beim Schwenken um c: y-Richtung
6. Wiederholen der vorhergehenden Schritte

Die genaue Einstellung des Werkzeugs ist schwierig und zeitaufwändig. Auch ist ein sinnvoller TCP schwer zu finden. So ist der Ultraschallkopf auch nicht komplett symmetrisch aufgebaut. Zum anschließenden Schwenken ist es jedoch unabdingbar die Werkzeuge genau zu vermessen.

3.4 Positionen beibringen

Zum Beibringen von Positionen gehören zwei Hauptaufgaben: Das Bewegen von Hand und das Speichern von Positionen. Der Roboter bietet ein Teaching Konzept, das Positionen von verschiedenen TCPs aus speichern kann. Diese stehen später auch in Sunrise OS zur Verfügung und können für die Bahnplanung verwendet werden. Für dieses Konzept wird der Roboter über die Achsen oder über den kartesischen Arbeitsraum gesteuert. Damit eignet sich dieses Konzept nicht um den Roboter schnell von Hand zu einer beliebigen Position zu führen und diese zu speichern. Zusätzlich unterscheiden sich die Patienten und die Positionen, weshalb eine generelle Speicherung der Position nicht notwendig ist. Damit ist es sinnvoller den Roboter von Hand direkt an die gewünschte Position zu führen und diese zu speichern als vorher das Teaching Konzept zu verwenden.
Um den Roboter von Hand bewegen zu können, kann man eine Methode (handguiding) von KUKA [3][S. 223ff] nutzen. Diese ermöglicht es den Roboter von Hand zu führen. Diese Methode braucht jedoch einen Sicherheitsschalter (Dreiwege Schalter, der nur in der Mittelstellung ein Signal liefert) und einen definierten Sicherheitszustand. Sicherheitszustände lassen sich in Sunrise OS einrichten. Dazu gehören z.B. Schutzräume die der Roboter nicht erreichen darf oder ein Taster, der gedrückt werden muss. Da zu Beginn der Arbeit dieser Schalter noch nicht zur Verfügung stand und gegen Ende dieser Arbeit zu wenig Zeit war, wurde eine kleine eigene Methode geschrieben.

Um einen vollständig nachgiebigen Roboter zu erhalten, bot sich an die bereits zur Verfügung stehende Impedanz-Klasse von KUKA zu nutzen. Diese realisiert das Konzept der Impedanz (Abschnitt 2.2.3) Sie erlaubt somit indirekt die Festigkeit des Roboters einzustellen. Momentan gibt es hier nur eine kartesische Impedanz-Klasse. Das bedeutet die Steifigkeit des Roboters wird auf ein Koordinatensystem bezogen. Eine achsspezifische Impedanz würde sich hierfür besser eignen, da hier die Festigkeit jeder einzelnen Achse eingestellt werden könnte. Solange diese von KUKA noch nicht bereitgestellt ist und kein Taster verfügbar ist, bietet sich nur die kartesische Impedanz-Klasse an.

In Listing 3.1 wird die konkrete Implementierung gezeigt. Zuerst muss ein neues Objekt der CartesianImpedanceControlMode-Klasse erstellt werden. Diese Klasse erlaubt die Parameter für die Impedanz zu setzen. In diesem Fall ist die Festigkeit in den translatorischen Achsen auf 10 N/m und auf 0 Nm/rad in den rotatorischen Achsen festgelegt. Über den PositionHoldContainer kann die aktuelle Position mit diesen Steifigkeitswerten gehalten werden. In diesem Fall wird jedoch nichts gehalten, sondern der Roboter lässt sich führen.

```
1 private void moveManualy() {
2     impedanceControlMode = new CartesianImpedanceControlMode();
3     impedanceControlMode.parametrize(CartDOF.TRANSL).setStiffness
          (10);
4     impedanceControlMode.parametrize(CartDOF.ROT).setStiffness(0);
5     impedanceControlMode.setMaxCartesianVelocity(1000,1000, 1000,
          2, 2, 2);
6     impedanceControlMode.setNullSpaceStiffness(0);
7     positionHoldContainer = rA.getTool().moveAsync((new
          PositionHold(
8                   impedanceControlMode, -1, null)));
9 }
```

Listing 3.1: Beispiel für eine eigene handguiding Methode

Das Abspeichern einer Position kann nun ein Listener (einfaches Java Interface mit der Methode setPoint(Frame f)) übernehmen. Aktuell wird mit Hilfe eines Dialoges gefragt ob eine Position gespeichert werden soll. Später soll hierzu ein Knopf am Handführgerät genutzt werden. Die Abfrage der aktuellen Position ist wie in Listing 3.2 zu realisieren. Aufpassen muss man hier, dass später die Position immer vom Werkzeug, Flansch oder anderen TCP abhängt. Damit dort keine Fehler entstehen sollte immer der gleiche TCP zum Bewegen und Speichern ausgewählt sein. Alternativ lässt sich auch die Jointposition (achsspezifische Position) des Roboters abfragen, diese ist eindeutig.

```
1     Frame f = new Frame();
2     f = robot.getCurrentCartesianPosition(rA.getTool()
3         .getDefaultMotionFrame());
4     listener.savePoint(f);
```

Listing 3.2: Methode zum Abfragen der Istposition

3.5 Positionen anfahren

Nachdem in Abschnitt 3.4 das Beibringen von Positionen das Ziel war, sollen diese nun angefahren werden können. Einfachste Variante hierfür wäre eine PTP-Bewegung mit der Zielvorgabe des neuen Ziels. Problematisch ist dieses Vorgehen jedoch, wenn Menschen oder später der Patient in der Nähe des Roboters ist. Der Verlauf einer PTP-Bewegung lässt sich nicht vorhersehen. Dadurch können Probleme entstehen:

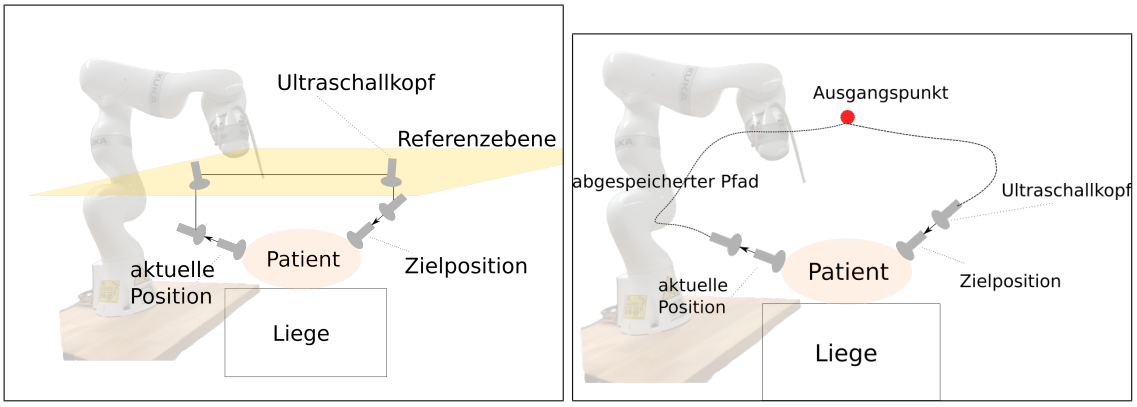
- Roboter drückt gegen oder durch einen Patienten um zu einer neuen Position zu gelangen.
- Der Patient hat sich bewegt und der Roboter drückt beim Anfahren zu stark um eine nun falsche Position zu erreichen.

Um den Problemen entgegen zu wirken, braucht es einen intelligenteren Algorithmus um Positionen an einem Patienten anfahren zu können. Mögliche Varianten:

- Oberfläche des Patienten wird durch eine **Time of Flight Kamera** erkannt und dem Roboter als verbotener Bereich übergeben.
  → Möglich, jedoch zu komplex um in kurzer Zeit zu lösen.
- Der Roboter probiert langsam eine neue Position einzunehmen und versucht **Hindernissen bei Berührung auszuweichen**.
  → Mit Hilfe der Impedanz oder Kraftsensorik zwar machbar. Jedoch voraussichtlich keine Ideallösung für den Patienten. Da dieser im Zweifel mehrmalig unkoordiniert sanft getroffen werden könnte.
- Anstelle einer PTP-Bewegung werden **fünf LIN-Bewegungen ausgeführt**. In Abbildung 3.2a ist der Verlauf grafisch dargestellt. Zuerst bewegt sich der Ultraschallkopf in seiner z-Achse vom Patienten weg. Anschließend wird er zu einer Referenzebene angehoben. Zum Anfahren einer Position wird nun auf dieser Referenzebene bis zur Zielposition, mit einem Offset in dessen z-Achse gefahren. Anschließend wird der Ultraschallkopf zur Zielposition mit Offset in z-Achse abgelassen. Zuletzt kann nun der Ultraschallkopf nur noch in z-Richtung gegen den Patienten gedrückt werden. Zwischen den einzelnen Punkten kann auch die Geschwindigkeit des Roboters angepasst werden. So kann dieser langsam vom Patienten weg und hin, auf Referenzebene jedoch schneller verfahren. Ergänzt

man den Impedanzmodus bei dieser Bewegung reagiert der Roboter noch nachgiebig auf Kräfte und kann Hindernissen ausweichen oder diese treffen ohne sie zu verletzen.
→ relativ einfacher Ansatz, der in Teilen auch schon in der späteren Anwendung implementiert ist. Jedoch wird der Bewegungsbereich eingeschränkt, da nicht alle Positionen über eine einheitliche Referenzebene angefahren werden können.

- Eine alternative Variante wäre, den **Pfad des Roboters** beim Setzen der Positionen zu speichern. Der Pfad könnte jeweils an einem bestimmten Ausganspunkt beginnen und bis kurz vor die Oberfläche des Patienten gehen. Das Andrücken des Ultraschallkopfes erfolgt nun mit einer LIN-Bewegung. In Abbildung 3.2b ist das Vorgehen grafisch dargestellt.
→ relativ komplexer Ansatz, der jedoch eine gute Lösung für das Problem bietet.



(a) vorgegebener Ablauf (b) beigebrachter Ablauf

Abbildung 3.2: Mögliche Arten einen Ultraschallkopf an den Patienten zu drücken

Als Beispielprogramm wurde hier eine vereinfachte Form der fünf LIN-Bewegung realisiert. Anstelle einer Referenzebene fährt der Roboter 10 cm von der Oberfläche des Patienten weg und anschließend direkt zu einem Zwischenpunkt. Von diesem fährt er nun zum Zielpunkt mit Offset in z-Richtung. Zuletzt bleibt die Bewegung in z-Richtung zum Andrücken des Ultraschallkopfes. Hierbei wurde auch die Impedanz aktiviert, damit der Roboter bei Krafteinwirkung etwas nachgiebiger ist. Wird der Ultraschallkopf jedoch von unten gegen den Patienten gedrückt, könnte dieser Algorithmus den Ultraschallkopf beim Lösen gegen die Liege des Patienten drücken. Somit muss dieser bei späterer Verwendung noch verbessert werden.

3.6 Atemkompensation

Für das Ultraschallen ist eine dauerhafte Anbindung des Schallkopfes an den Patienten unerlässlich. Daher soll hier die Atemkompensation realisiert werden. In Frage kommen

hierzu die Kraftsensorik an sich anzusprechen oder die vorhandenen Impedanz-Klassen zu verwenden. Daher ergaben sich für die Atem- und Bewegungskompensation folgende Möglichkeiten:

3.6.1 Impedanz zur Atemkompensation

Die Impedanzsteuerung ist ein wesentliches Konzept dieses Leichtbauroboters. Es lässt sich bei jeder Bewegung verwenden. Schwierigkeiten bei der Verwendung sind die richtigen Werte für die Impedanz und deren Rahmenparameter zu finden. Darunter fallen z.B. die Federsteifigkeit, Amplitude und die Nullraumbewegung. Man muss darauf achten, wie die Impedanz bei der gegebenen Bewegungsart zu verwenden ist. Gerade bei SmartServo ist die Impedanzsteuerung schwierig, da falsch gewählte Parameter Fehler verursachen: Ist z.B. die Grundsteifigkeit der virtuellen Feder zu niedrig, kann der Roboter sich nicht bewegen, da allein die Schwerkraft zu groß ist. Ein möglicher Algorithmus der sich über die Impedanz-Klassen einstellen lässt wäre:

1. Bewege dich dauerhaft in z-Richtung (Ultraschallkopfspitze in Richtung Patient)
2. Stelle die Impedanz in z-Richtung auf die gewünschte Kraft ein
3. Aktiviere die Impedanz und Bewegung

→ Roboter versucht in z-Richtung zu drücken. Ist der Druck groß genug gibt er nach, hält damit den Druck konstant. Funktioniert im Ansatz, hat jedoch zu unerwarteten Fehlern geführt: Die Softwaregrenzen des Roboters wurden bei der permanenten Bewegung in eine Richtung überschritten. Vermutlich wird durch den eigentlichen Bewegungsbefehl die aktuelle Position des Robotes neu berechnet. Wird nun diese durch die Impedanz verändert, kann es sein, dass die virtuelle Position des Roboters sich außerhalb dessen Grenzen befindet und er deshalb einen Sicherheitshalt ausführt.

3.6.2 Kraftmessung und aktive Steuerung zur Atemkompensation

Ein anderer Ansatz ist das Abfragen der aktuellen Kraft am Ultraschallkopf und das aktive Drücken oder Entfernen des Ultraschallkopfes. Der Roboter bietet hierzu verschiedene Schnittstellen [3, S. 266 ff]. Mögliche Algorithmen zur Atemkompensation die sich daraus ergeben sind:

- **naiver Ansatz:** Fehlt die Kraft bewegt er sich um einen definierten Betrag nach vorne. Ist die Kraft zu groß bewegt er sich um einen definierten Betrag zurück. → Funktioniert nur bei kleinen Bewegungen. Ist die Bewegung zu groß oder zu schnell, braucht der Roboter zu lange. Wählt man den Betrag zu groß, ist die Kraft zu hoch.

- **komplexerer Ansatz:** Je größer die einwirkende oder fehlende Kraft, desto schneller und weiter versucht sich der Roboter zu bewegen. Ansonsten das gleiche Vorgehen wie beim naiven Ansatz.
  → Funktioniert, ist jedoch nicht sonderlich besser, als der naive Ansatz.
- **Condition:** KUKA bietet eine eigene Bedingungsklasse [3, S. 285ff] an, die es erlaubt eine Kraftbedingung zu definieren.
  → Wurde noch nicht ausprobiert. Die KUKA-Methoden scheinen eher für Abbruchbedingungen verwendet zu werden, als zum Steuern neuer Positionen.

3.6.3 Umgesetzte Variante in der späteren Anwendung

Da keine der hier angedeuteten Varianten zufriedenstellend und zuverlässig funktioniert hat, wurde auf die Kraftüberwachung in der späteren Anwendung vorerst verzichtet. In der Hauptapplikation wird stattdessen eine Impedanzeinstellung verwendet, die es erlaubt, den Roboter, um bis zu 3 cm, bei jeder Bewegung wegzudrücken. Die Gefahr erdrückt zu werden ist somit minimiert. Jedoch ist eine Aufsichtsperson unabdingbar.

3.7 Kommunikationsschnittstelle

Mit Hilfe der Kommunikationsschnittelle soll eine Verbindung zwischen Roboter und dem externen Computer geschaffen werden. Diese ist notwendig um Befehle oder Daten auf den Roboter schicken zu können. Zusätzlich soll der Roboter seine Position und Meldungen an den Computer weiterleiten. Daraus folgt, dass ein Nachrichtenprotokoll benötigt wird, dass sowohl Steuerbefehle, als auch Datennachrichten schicken und empfangen kann.

3.7.1 Auswertung möglicher Nachrichtenprotokolle

Für die Realisierung der Nachrichtenschnittstelle kamen 3 Protokolle in Frage:

- **Remote Method Invocation (RMI):** Ein von Java bereitgestelltes Mittel um entfernte Methoden aufzurufen. [16]
  → Wurde verworfen, da die Firewall des Roboters Verbindungen von außen grundsätzlich ablehnt. Es wurde versucht einen neuen Port freizugeben um dieses Problem zu umgehen. Erstes Hindernis hierbei war, dass KUKA keine Systempasswörter preisgibt und diese erst gefunden werden müssen. Nachdem dieses gefunden und die Einstellung in der Windowsumgebung geändert worden war, blieb das Problem jedoch bestehen. Daher wurde RMI verworfen.

- **Fast Roboter Interface (FRI):** Schnittstelle von KUKA. Bietet einen C++ Client und ein Interface zum Ausführen von Bewegungsbefehlen. Es bietet Abfrageoptionen für die Verbindung des Roboters, seine Istposition in kartesischen Koordinaten oder für die achsspezifische Istposition. Es bietet eine hohe zyklische Zugriffsrate, die Zugriffe auf den Roboter im Millisekunden-Bereich erlaubt [4]. Vorteile daraus:
  - hohe zyklische Zugriffsrate zum Übergeben von Befehlen und Abfragen von Istpositionen
  - bereits Entwickelter C++ Client und Roboterapplikation
  - vorhandene Verbindungskontrolle und Abbruchbedingungen.

  → Wäre in Frage gekommen. Es bietet eine Schnittstelle um dem Roboter Bewegungsbefehle direkt zu übermitteln. Dabei wäre ein Client in C++ verfügbar gewesen. Da später der Roboter seine Bewegungen größtmöglich selber steuern sollte, kam es nicht in Frage im ggf. med. Bildverarbeitungsprogramm die Steuerungsbefehle bis hin zu Bewegungsbefehlen zu implementieren.
- **OpenIGTLink mit selbst geschriebener Erweiterung** Wie bereits in Abschnitt 2.6 erwähnt, existiert eine Erweiterung für 3D Slicer, die einen Leichtbauroboter mit Hilfe von OpenIGTLink und SmartServo steuert. Anfangs wurde versucht die dort geschriebene Applikation zu erweitern und auszubauen. Der Quellcode ist unter `https://github.com/tauscherSw/LWROpenIGTIF.git` verfügbar. Ein kurzes Tutorial [17] ist ebenfalls verfügbar. Leider war die Dokumentation des Programms nicht ausreichend genug um es direkt zu verwenden und zu erweitern. Die Vorteile von OpenIGTLink mit Verwendung von SmartServo (Abschnitt 2.3.7) sind:
  - Opensourceprotokoll, das erweitert werden kann
  - Eine OpenIGTLink Version ist in C++ und Java verfügbar
  - Es existiert bereits ein Projekt, in dem es funktioniert hat.
  - SmartServo (Abschnitt 2.3.7) unterstützt die zyklische Befehlsübermittlung und kann alle 25 ms eine neue Position als Zielposition setzen.
  - Möglichkeit auf dem Roboter geschriebene Abläufe und Programme abzurufen.

Die Vorteile von OpenIGTLink mit SmartServo schien die beste Vorlage für das Problem zu sein. Somit wurde diese Variante im weiteren Verlauf ausgebaut. Die Vorlage mit 3D Slicer wurde jedoch nicht verwendet. Auch die existierenden Roboter Klassen in OpenIGTLink waren für das Problem nicht ausreichend oder noch nicht vorhanden.

3.7.2 Einrichten von OpenIGTLink

Für OpenIGTLink gibt es eine Programmierung in Java. Diese kann unter `https://code.google.com/p/igtlink4j/` heruntergeladen werden. Die Grundlagen von OpenIGTLink lassen sich in Abschnitt 2.4.1 nachlesen. Für den Roboter wurde ein OpenIGTLink Client benötigt, der Befehlsnachrichten interpretieren und ausführen kann. Auf Seiten des Externen Computers wird ein Server benötigt, der Daten- oder Statusnachrichten des Roboters interpretieren kann. Da OpenIGTLink selbstgeschriebene Klassen nicht automatisch in Nachrichten verpacken kann, mussten zuerst folgende Probleme gelöst werden:

- Umwandlung von Datenfeldern in Bytecode und zurück
  → Erstellung eines Interfaces, das Methoden vorgibt, die Bytecode liefern oder aus Bytecode Datenfelder erstellen. Somit haben eigene Klassen, wie z.B. eine Klasse zum Speichern kartesischer Positionen, dieses Interface implementiert. In den Nachrichtenklassen können die Datenfelder nun einfach über deren Methoden in Bytecode verwandelt werden. Später lässt sich so einfach ein Frame serialisieren und in einer Nachricht verschicken.
- Fehlende Methoden zum Erzeugen von Bytecode für Integer und selbst erstellten Enumerationen.
  → Eigene Methode erstellt, die Integer in einen 4 Byte langen Bytecode verwandelt. Enumerationsfelder werden in Integern hinterlegt und können so zwischen Enumeration und Integer gewechselt werden. In der eigentlichen Nachrichtenklasse kann so ein Enum als Integer gewandelt über die Leitung geschickt werden.
- Unzuverlässige Methode zum Parsen von Strings. Es wird zwar eine Methode angeboten, diese funktionierte jedoch nicht einwandfrei. So wurden als String deklarierte Kommandos nicht erkannt, da der String nach dem zurück wandeln nicht der Gleiche war.
  → Strings wurden komplett vermieden. Dadurch wurden mehrere Nachrichtenklassen und Enumerationen gebraucht.

3.7.3 Nachrichtenaustausch

Im Anhang befinden sich zwei Sequenzdiagramme, die den Ablauf der Kommunikation darstellen (C.1 und C.2). Generelles Konzept ist, das auf jede Nachricht eine Antwort erfolgt. Zu Beginn meldet sich der Roboter und erhält einen Befehl als Antwort. Der Roboter liefert ab hier immer seinen Status als Antwort. Der externe Computer kann so kontrollieren ob der Roboter noch auf dem richtigen und aktuellen Pfad ist. Ferner könnte eine Animation des Roboters in Echtzeit auf dem externen Computer erzielt werden. (KUKA bietet hierfür ggf. schon eine Schnittstelle namens SmartServoRendering an). Um zusätzlich die Verbindung zu überprüfen soll der Server beim Schicken

einer Nachricht angeben, wann die nächste Nachricht kommt. Damit ist es dem Roboter möglich zu erkennen ob er noch das aktuelle Kommando ausführt, oder ob keine Verbindung mit dem Server besteht.

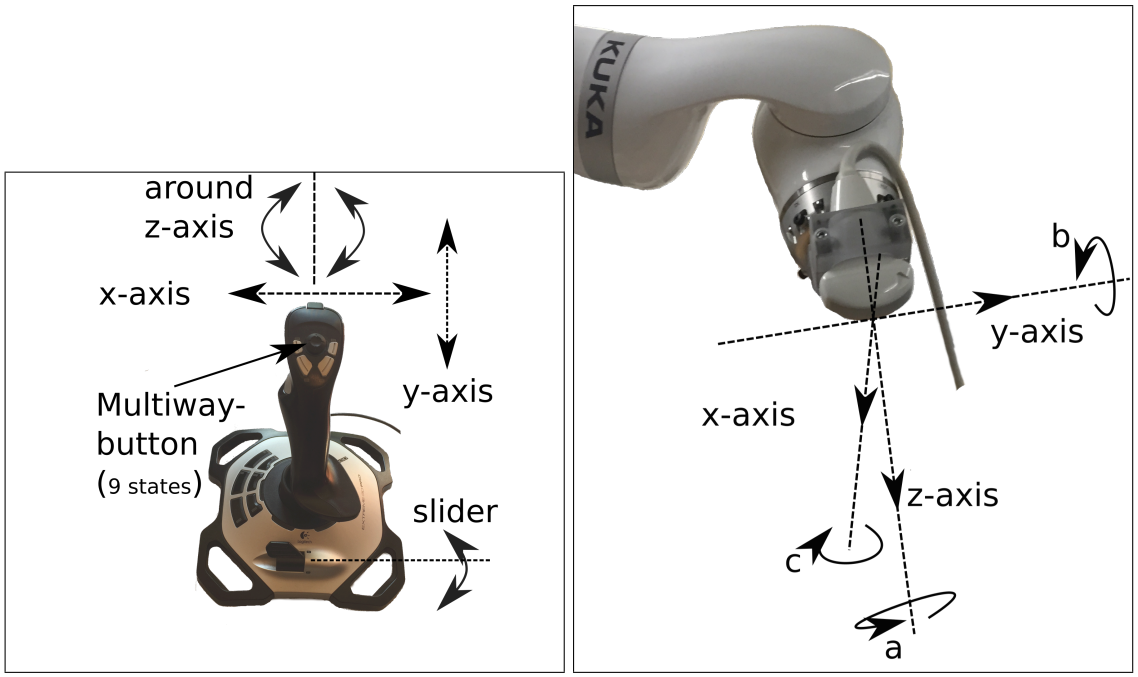
3.8 Eingabegerät

Für die entfernte Steuerung des Ultraschallkopfes kamen mehrere Eingabegeräte in Frage. Es lässt sich vorstellen, das über die Maus und Tastatur einzelne Achsen des Ultraschallkopfes angesprochen werden können. Alternativ auch indirekt über Schieberegler in der GUI. Diese Varianten haben jedoch einen Nachteil: Es ist unheimlich schwierig die 3 Achsen in denen sich der Ultraschallkopf schwenken lässt, gleichzeitig über Tasten oder Schieberegler zu steuern. Die Lösung bietet ein handelsüblicher Joystick. Dieser besitzt standardmäßig drei oder mehrere Achsen. Der verwendete Joystick ist ein Logitech Extreme 3D pro Joystick.
Durch die Verwendung ergeben sich weitere Möglichkeiten/Probleme, die betrachtet werden müssen. Ist es sinnvoll die Achsen des Joysticks immer den gleichen Achsen des Ultraschallkopfes zu zuweisen? Welche Werte liefert der Joystick, welche Werte brauchen die Roboterachsen? Wie werden die Werte des Joysticks erfasst?

3.8.1 Achszuweisung Joystick

Damit später eine Achszuweisung erfolgen kann, müssen die vom Joystick gelieferten Werte interpretiert werden. Um eine leicht verständliche Bezeichnung zu erhalten wurden die Hauptachsen wie in Abbildung 3.3a bezeichnet. Die Tasten des Joysticks wurden entsprechend ihrer Nummer zugeordnet (1-12). Weiter wurden die Achsen des Ultraschallkopfes eingeteilt. Sie ergeben sich aus dem Roboterkoordinatensystem des Werkzeugs (Ultraschallkopf). In Abbildung 3.3b sind diese dargestellt. Um den Ultraschallkopf schwenken zu können, werden die Winkel a,b,c gebraucht. Erste statische Zuweisung der Achsen war folgendermaßen:

- x-Achse Joystick → Drehung um y-Achse des Ultraschallkopfes (c)
- y-Achse Joystick → Drehung um x-Achse des Ultraschallkopfes (b)
- um-z-Achse Joystick → Drehung um z-Achse des Ultraschallkopfes (a)



(a) Logitech Extreme 3D PRO (b) Ultraschallkopf

Abbildung 3.3: Darstellung der Achsen

Wie man jedoch in Abbildung 3.3b weiter erkennen kann, besitzt der Ultraschallkopf mehr Freiheitsgrade als der Joystick Achsen bietet. Auch lässt sich schnell nachvollziehen, dass es sinnvoller ist die Achsen des Joysticks z.B. in einer GUI den Achsen des Roboters zuzuweisen. Weiterhin würde sich eine Speicherung der Einstellungen empfehlen. Somit wäre es möglich für unterschiedliche Aufgaben (z.B. Positionieren oder Schwenken des Ultraschallkopfes) unterschiedliche Einstellungen zu verwenden. In Kapitel 4 wird darauf eingegangen und eine Lösung präsentiert. Für dieses Teilergebnis reicht eine statische Zuweisung. In Abschnitt 3.9 wird dieser Teilschritt ausgebaut und die Bewegung des Roboters realisiert.

3.8.2 Anforderungen an die Kommunikationsschnittstelle

Durch die Verwendung eines Joysticks ist es möglich mehrere Achsen gleichzeitig zu bewegen. Daraus ergibt sich die Notwendigkeit einer Nachrichtenklasse die diese Werte übermitteln kann. Nach Möglichkeit sollten sich die Werte in dieser Nachrichtenklasse schon auf die Werte der Roboter bzw. Ultraschallkopf-Achsen beziehen. Weiter muss geklärt werden in welchem Zeitintervall die Werte des Joysticks abgefragt werden. In einem ersten Beispiel wurden die Werte des Joysticks in einem 25 ms Intervall abgefragt und über eine Nachrichtenklasse an ein anderes Programm geschickt, welches die Werte des Joysticks in einer GUI darstellt. Dadurch konnte subjektiv die Latenzzeit bestimmt

werden. Es reicht somit, wenn die Werte in 25-100 ms Intervallen abgefragt werden. Die Kommunikationsschnittstelle muss somit diese Zyklusrate unterstützen. Was sie in diesen ersten Tests auch getan hat.

3.8.3 Implementierung

Wie in Abschnitt 2.5.1 bereits erwähnt, werden die Knöpfe und Achsen des Joysticks in einem Thread abgefragt. LWJGL liefert dabei nur ein Array von Komponenten. Die Hauptachsen werden wie oben beschrieben zugewiesen und die Knöpfe erhalten jeweils ein Datenfeld mit ihrer Bezeichnung.
Erster Schritt zur Nutzung des Joysticks in Java war einen Listener zu schreiben, der die Werte des Joysticks interpretiert und dem Hauptprogramm übergibt. Werden die Werte in einem dauerhaften Zyklus von 25 ms oder weniger abgefragt, macht es Sinn nur bei Veränderungen des Zustands eine Nachricht zu schicken. Dieses lässt sich ganz leicht mit einem Vergleich der vorherigen Werte realisieren. Sind diese identisch wird der Listener nicht aufgerufen. Damit die Leistung des Computers bei einer solchen Dauerschleife nicht belastet wird, wurde die Priorität des Threads auf 3 gesetzt.

3.9 Zusammenwirken der einzelnen Teilschritte

In diesem Abschnitt werden die aus Abschnitt 3.9, 3.7.2 und 3.8 gewonnenen Beispiele benötigt. In Abbildung 3.4 ist der Ablauf dargestellt. Die vom Joystick bereitgestellten Werte, werden von einem Listener entgegengenommen und dem OpenIGTServer übermittelt. Dieser erstellt eine Nachricht mit allen Werten des Joysticks und schickt diese dem Roboter. Die Nachricht wird vom OpenIGTClient empfangen und von einem Messageinterpreter interpretiert. Die Werte werden zuletzt einem Executer übergeben, der nun mit Hilfe von SmartServo die Werte in die Zielpositionsplanung einbindet.
Es wurden hierbei die Achsen wie in Abschnitt 3.8.1 zugewiesen. Durchdrücken des Joysticks (Wert 1 oder -1) wurde hierbei auf eine Veränderung der Istposition des Roboters um Wert Joystick * 0.1 rad definiert. Die eigentliche RA-Klasse enthält eine Endlosschleife, die alle 25 ms ausgeführt wird. In der Schleife wird zuerst die aktuelle Position des Roboters erfragt. Anschließend werden auf die aktuelle Position die Werte des Joysticks addiert. Zuletzt wird noch diese neue Position als Zielposition gesetzt.
Mit diesem Setup ergeben sich einige **Probleme:** Wird immer die aktuelle Position abgefragt, kann der Roboter sich von seinem aktuellen Referenzpunkt weg bewegen. Es wurde versucht den Roboter immer auf einen veränderten Referenzpunkt anfahren zu lassen. Das bedeutet, nur einmal die aktuelle Position abfragen und danach immer die Werte auf diesen zu addieren. Leider war die Verwendung von SmartServo hier zu komplex und warf Interpolationerrors. Gleiches gilt für das Übersteuern von Werten außerhalb der Grenzen des Roboters. Angedacht war hier ein try-catch-Block zu verwenden. Es wurde aber nicht erreicht diesen Fehler abzufangen. Weiter wird die Steuerung mittels SmartServo erschwert, da eine Zieleingabe immer mit dem

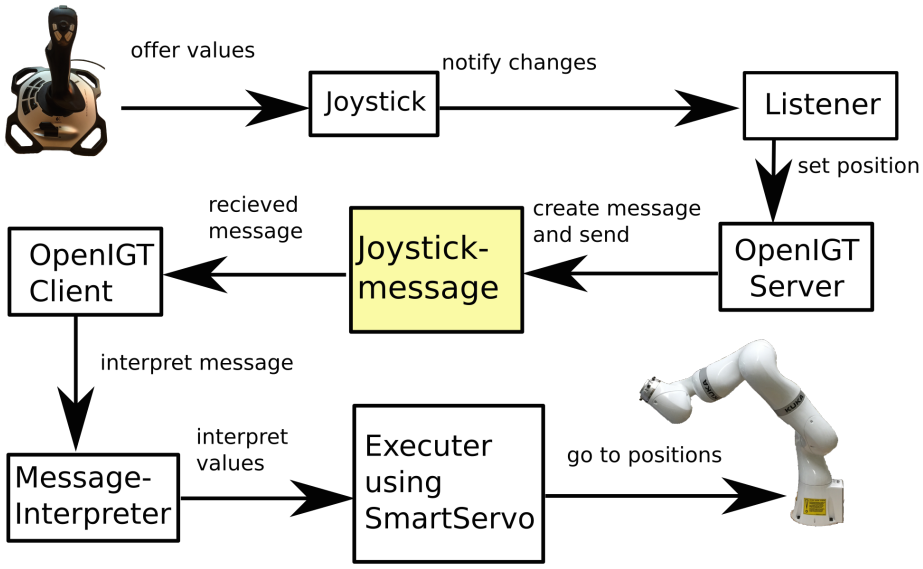


Abbildung 3.4: Ablauf des SmartServo-Prinzips mit Eingabe eines Joysticks, übermittelt mit OpenIGTLink

Weltkoordinatensystem des Roboters erfolgt. Es ist somit nicht immer nachvollziehbar welche Koordinaten als Zielkoordinaten eingegeben werden müssen. Daher lässt sich das Beispielprogramm nur in gewissen Grenzen nutzen:

- Werte des Joysticks müssen in den Grenzen des Erreichbaren sein.
- Muss der Roboter sich viel bewegen, kann die aktuelle Position von der ausgehenden Position abweichen.

3.10 Entwurf

Nach dem Programmieren der Teilschritte, konnte ein erster Entwurf entwickelt werden. In Abbildung 3.5 ist dieser dargestellt. Die Haupteinteilung erfolgt in drei Module: Externer Computer, Datenaustausch und Roboterapplikation.

3.10.1 Externer Computer

Dieses Modul übernimmt alle Aufgaben, die auf dem externen Computer ausgeführt werden sollten. Dazu gehören:

- **GUI:** Übernimmt die Darstellung zum Einstellen der Befehlsparameter.
- **Server:** Schickt die Nachrichten, die von den Kommandoklassen benötigt werden

- **Steuerbefehle verwalten:** Verwaltet alle programmierten Kommandos und deren Parameter
- **Joystick:** Stellt die Werte des Joysticks für Kommandos bereit.

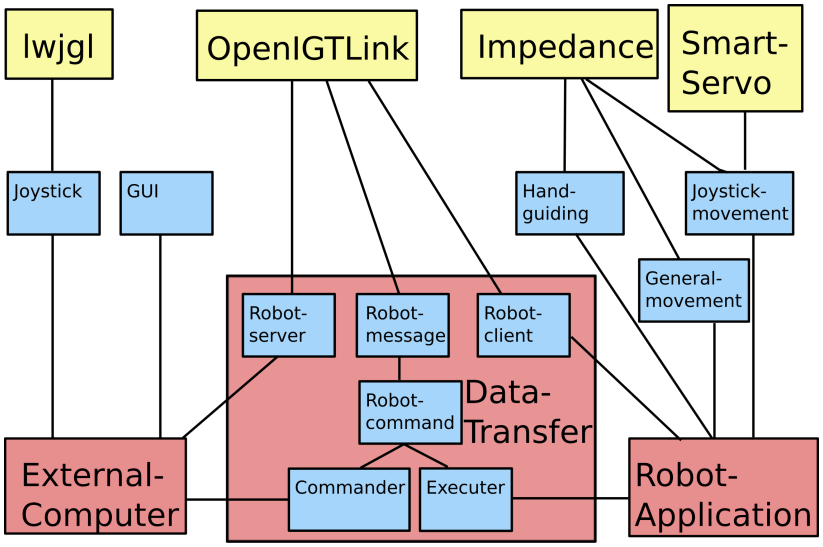


Abbildung 3.5: Erster Entwurf: Gelb sind verwendete Bibliotheken, rot sind die Hauptmodule und blau die Untermodule. Eine Linie bedeutet, dass die Komponente die Verbundene nutzt.

3.10.2 Datenaustausch

Dieses Modul übernimmt alle Aufgaben, die den Datenaustausch betreffen. Dazu gehören die Nachrichtenklassen und der Server, sowie Client. Es leitet sich direkt aus der OpenIGTLink Bibliothek ab. Im wesentlichen muss hier die in Abschnitt 3.7 entwickelte Kommunikationsschnittstelle weiterentwickelt werden, z.B. mit weiteren Nachrichtenklassen für spezielle Befehle.

3.10.3 Robotikapplikation

Alles was Teil des Roboters und dessen Bewegungskoordination ist, gehört zur Robotikapplikation. Um Bewegungen nachgiebig zu machen soll bei jeder Bewegung die Impedanz aktiv sein. SmartServo sollte auf Grund der Komplexität nur bei zyklischen Bewegungsbefehlen im Millisekunden-Bereich verwendet werden. Zum Anfahren von Positionen sollen die Standard-Bewegungsarten genutzt werden. Weiterhin soll vorläufig erst einmal mit der Impedanz das Handführen ermöglicht werden. Sobald der Sicherheitstaster funktioniert, sollte in Erwägung gezogen werden, die KUKA eigene Methode zu verwenden.

4 Ergebnisse

Zu den Ergebnissen dieser Arbeit gehören vor allem die in Kapitel 3 gelösten Teilschritte: Anforderungsanalyse, das grundlegende Konzept und die gelösten Teilschritte: Ultraschallkopfhalterung, Positionen beibringen und anfahren, Konzepte zur Atemkompensation, Kommunikationsschnittstelle und Eingabegerät . Hauptergebnis dieser Arbeit ist nun die Zusammenführung aller Teilschritte und Konzepte zu einem Ganzen.
Das in Abschnitt 2.1 vorgestellte Anwendungsszenario wurde bis zum Schwenken des Ultraschallkopfes implementiert. Damit lässt sich der Roboter zuerst durch Führen von Hand beliebige Positionen beibringen. Diese werden auf den externen Computer (z.B. im Kontrollraum) übertragen und dargestellt. Am Externen Computer lassen sich die gespeicherten Positionen abrufen und der Roboter fährt zu diesen. An der jeweiligen Position kann nun mit Hilfe des Joysticks der Ultraschallkopf geschwenkt werden.
Um die Teilschritte aus der Methodik zusammenzuführen, musste der dort vorgestellte Entwurf (Abschnitt 3.10) in eine konkrete Architektur überführt werden. Zusätzlich müssen die dort entdeckten Probleme oder Möglichkeiten noch gelöst oder erweitert werden. Daher werden im Folgenden zuerst die Architektur und dann die jeweiligen Programmteile auf dem externen Computer bzw. dem Roboter vorgestellt.

4.1 Architektur

4.1.1 Generelles Konzept

Das generelle Konzept ist in Abbildung 4.1 dargestellt. Wie sich aus der Abbildung erkennen lässt, ist das Programm in zwei Hauptmodule aufgeteilt. Die Anwendung auf dem Roboter und dem externen Computer. Der externe Computer übernimmt dabei die Aufgabe den Roboter von dem Kontrollraum aus zu verwalten und zu steuern. Dazu gehört die Darstellung von Befehlen und eine geeignete Eingabemaske für Einstellungen. Weiter gehört die Verwaltung und Einstellung des Joysticks zu den Aufgaben des externen Computers. Die Anwendung auf dem Roboter übernimmt die Steuerung des Roboters. Hierzu interpretiert sie die Nachrichten des externen Computers. Die Einteilung der Anwendung in Pakete und deren Komponente befindet sich im Anhang B.1

4.1.2 Aufbau externer Computer

Auf Seiten des externen Computers wurde das **Model-View-Controller (MVC)-Prinzip** erweitert. Das Model ist dabei das Kommando. Für jedes Kommando kann eine eigene GUI und deren Controller erstellt werden. Alle Controller werden über den Main-Controller verwaltet. Wird auf der GUI ein neues Kommando gestartet, führt der Weg über die GUI zu deren Controller, zum Main-Controller, der dem Robot Server die Anfrage weitergibt. Wird kein anderes Kommando ausgeführt, lädt der RobotServer

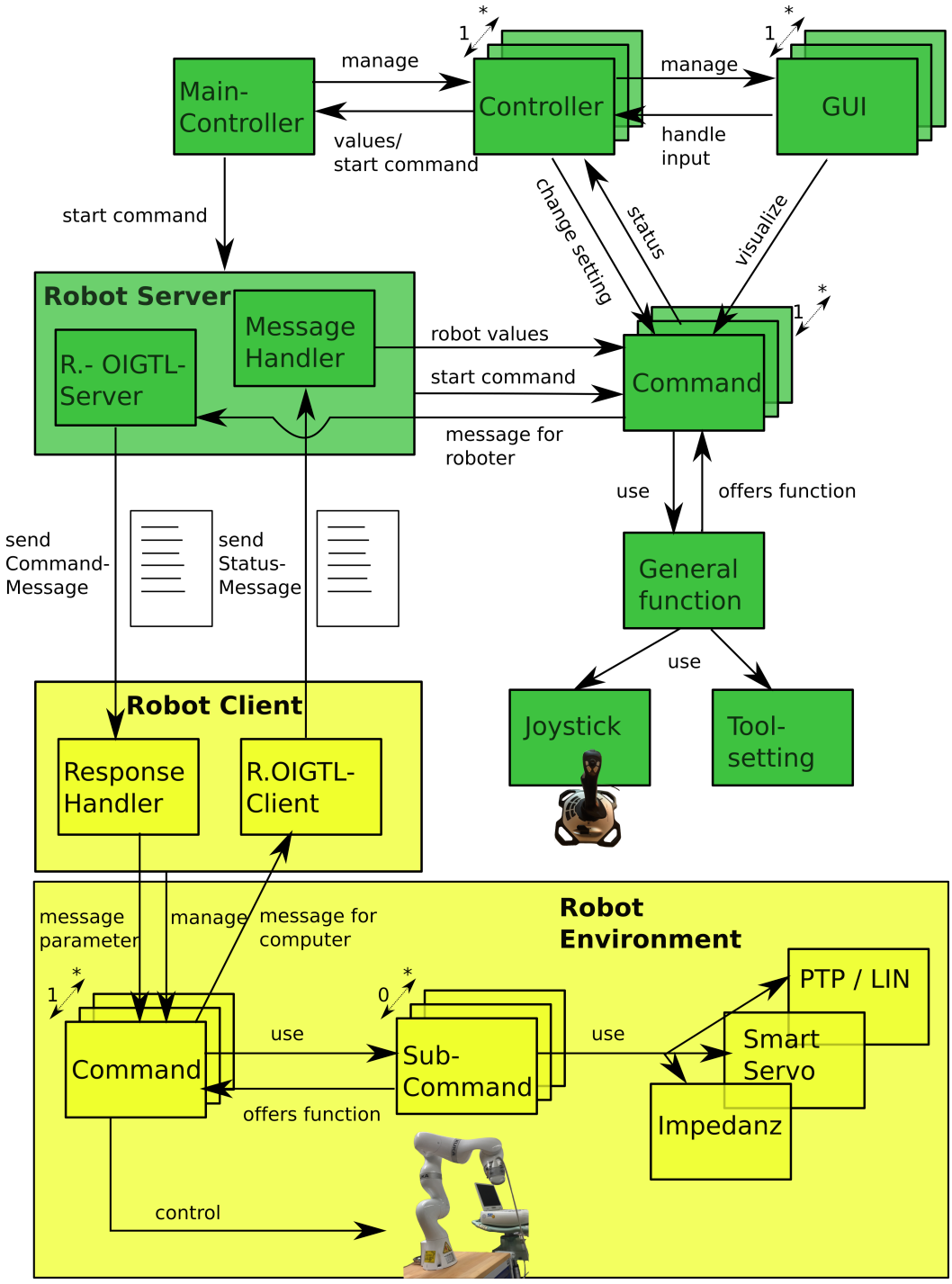


Abbildung 4.1: Das generelle Konzept. Alle grünen Objekte sind Komponenten des externen Computers, die gelben gehören zum Roboter.

das Kommando als aktuelles Kommando. Dabei wird gleich eine Start-Nachricht an den RobotClient geschickt. Ab hier funktioniert die Interaktion anders: Eingaben auf der GUI gehen sofort über den Controller an das Kommando. Das Kommando verwaltet diese Eingaben selbständig und übergibt dem RobotOpenIGTLinkServer (RIGTS) diese Nachricht. Dieser sendet dem Roboter diese Nachricht. Rückmeldungen werden von dem MessageHandler zu den Kommandos weitergeleitet, die mit dieser Information etwas anfangen können. Ein Kommando kann als Thread implementiert sein, muss es jedoch nicht. Ein Kommando kann eine generelle Funktion wie den Joystick nutzen.

4.1.3 Aufbau Roboter

Das Modul auf dem Roboter ist ähnlich aufgebaut. Natürlich gibt es hier keine GUI-Klassen. Eingaben über das SmartPad (Dialog oder Button) werden direkt in der Kommandoklasse verwaltet. Der ResponseHandler nimmt die Nachrichten auf, wandelt sie in die eigentliche Nachrichtenklasse zurück und übergibt sie dem jeweiligen Kommando. Zu Beginn werden die Start-Nachrichten mit Hilfe eines **Command-Design-Pattern** in ein Kommando umgewandelt. Hierbei wird das auszuführende Kommando als aktives Kommando in dem RobotClient gesetzt. Bei einer Befehlsnachricht wird diese immer nur dem aktuellen Kommando übergeben. Das bedeutet, es gibt immer nur ein aktiv auszuführendes Kommando. Das Kommando befindet sich in der Roboter-Umgebung, das bedeutet, es hat Zugriff auf den Roboter und dessen Komponenten (SmartPad, usw.). Ein Kommando kann in Subkommandos aufgeteilt werden, muss es jedoch nicht. Für die Bewegung werden zuletzt Klassen wie die Impedanz, SmartServo oder andere verwendet. Ein Kommando kann auch hier als ein Thread aufgebaut sein. Antworten oder der generelle Status werden über den Robot OpenIGTLinkClient als Nachricht verpackt dem externen Computer gesendet. Die Kommandoklasse steuert zuletzt den Roboter.

4.2 Externer Computer

Der externe Computer ermöglicht es den Roboter aus dem Kontrollraum heraus steuern zu können. Dazu benötigt es eine Darstellung der Kommandos und deren Parameter, sowie deren Verwendung und interner Ablauf. Für das Steuern mit Hilfe des Joysticks, benötigt es noch eine Einstellungsmöglichkeit, sowie deren Darstellung, da diese in den Teilschritten der Methodik noch nicht gelöst worden sind.

4.2.1 Verwendung, Darstellung und Einstellungen von Kommandos

In diesem Abschnitt wird die Verwendung und der Ablauf eines Kommandos im Programm beschrieben. GUIs, die noch nicht verwendet werden, die jedoch schon erstellt wurden, befinden sich im Anhang (A.2). Die Beschreibung des Ablaufs geht nur bis

zur Nachricht an den Roboter. Die Roboterapplikation wird im Abschnitt 4.3 erklärt. Für den Benutzer stehen am Anfang die verschiedenen Kommandos über ein JTabPanel zur Verfügung. In Abbildung 4.2 ist das Ultraschall-Kommando ausgewählt. Um dieses wird es im Folgenden gehen. Dieses setzt im wesentlichen den in Abschnitt 2.1 dargestellten Ablauf um.

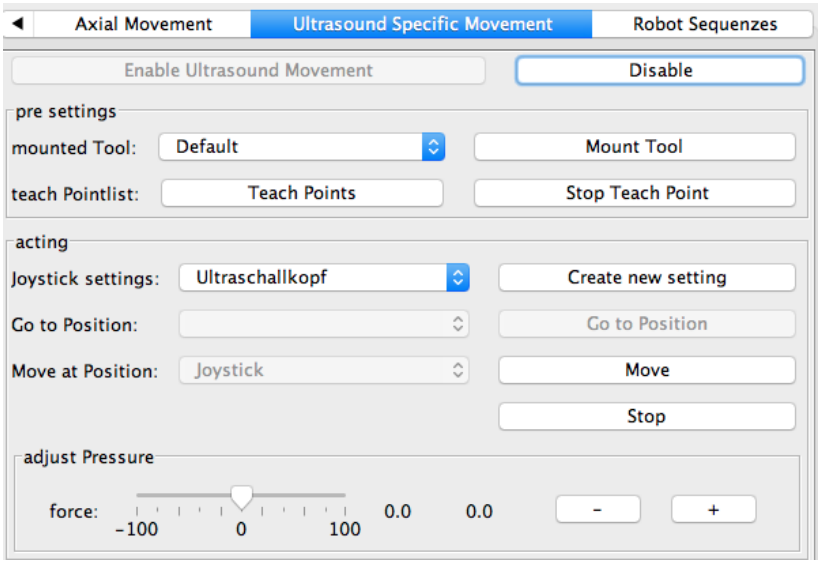


Abbildung 4.2: Ultraschallkommando GUI

Ein möglicher Programmablauf sieht wie folgt aus:

1. **Starten eines Kommandos:** Über den Enable_Ultrasound_Button wird das Kommando ausgewählt. Beim Klick auf den Button, ruft der Controller den RobotServer auf. Ist noch kein Kommando gestartet, wird das Kommando als aktives Kommando gesetzt und initialisiert. Der RobotServer ruft nun die Initialisierungsmethode der Ultraschall-Kommando-Klasse (UKK) auf. Hierbei erstellt das Kommando alles, was es später zum Ausführen braucht. Mit einer Start-Nachricht wird der Roboter informiert, welches Kommando kommt. Dieser richtet sich dann entsprechend ein. Weiter ruft der RobotServer die execute-Methode der UKK auf. Sie startet die eigentliche Aufgabe des Kommandos.

2. **Werkzeug festlegen:** Auf der GUI kann nun der Werkzeugdialog am SmartPad oder ein Werkzeug direkt aufgerufen werden. Aktuell existiert nur ein Werkzeug. Daher schickt dieser Button nur eine Nachricht, die am Roboter nicht berücksichtigt wird. Später lassen sich hier jedoch verschiedene Werkzeuge auswählen. Die Werkzeuge können aktuell in XML gespeichert und geladen werden.

3. **Positionen beibringen:** Der Teach_Points_Button ruft über den Controller die startTeachingPoints()-Methode in der UKK auf. Diese schickt eine TeachPoint-

Nachricht an den Roboter. Während dem Führen von Hand schickt der Roboter abgespeicherte Positionen an den RobotServer. Der MessageHandler reicht diese dabei an die UKK weiter. Diese ruft den Controller der GUI auf und übergibt ihm die neue Position. Der Controller setzt die Auswahl der möglichen Positionen in der GUI neu. Über den Stop_Teach_Point-Button lässt sich das Führen von Hand beenden. Die UKK schickt dazu eine weitere Teach-Point-Nachricht, die die Bewegung am Roboter beendet.

4. **Joystickeinstellung verändern:** In die Combobox (Joystick settings:) kann nun die Einstellung für den Joystick gesetzt werden (siehe Abschnitt: 4.2.2).
5. **Positionen anfahren:** Über eine JComboBox kann jetzt die gewünschte Position ausgewählt werden. Über den Button Go To Position wird diese angefahren. Dazu schickt die UKK über den RobotOpenIGTLinkServer (RIGTS) eine Nachricht an den Roboter. Ist die Position erreicht, meldet der Roboter dieses mit einer Nachricht. Über den MessageHandler wird die UKK informiert. Diese meldet es dem Controller weiter.
6. **Schwenkbewegung des Ultraschallkopfes durch den Joystick:** Aktuell gibt es nur eine Joystick Bewegung. Später kann hier eine Schwenkbewegung oder ähnliches ausgewählt werden. Mit dem Move-Button wird diese Bewegung ausgeführt. Hierbei ruft der Controller der UKK die startMoveAtPoint()-Methode auf. Sie setzt die UKK in die Liste der JoystickListener und schickt eine MoveManually-Nachricht an den Roboter. Wird der Joystick bewegt, ruft dieser in der UKK eine valueChange()-Methode auf. Diese schickt dem Roboter eine Nachricht mit den Werten des Joysticks (jeweils der Joystickeinstellung entsprechend). Mit dem Stop-Button kann die Bewegung beendet werden.
7. **Atemkompensation und Anpressdruck:** Als letztes war angedacht, über einen Schieberegler den Anpressdruck einzustellen, was jedoch aufgrund der Komplexität von SmartServo nicht funktionierte.

4.2.2 Joystick Werte- und Achsenzuweisung

Für den Joystick ergab sich bei der Lösung des Teilschrittes in Abschnitt 3.8, dass dieser eine Möglichkeit braucht, seine Achsen beliebigen Achsen des Ultraschallkopfes zuweisen zu können.Außerdem ergab sich, dass der Joystick sich später für andere Kommandos verwenden lassen sollte. Daher wurde der Joystick als Singleton definiert. Das bedeutet, eine Einstellung gilt für das gesamte Programm. Zum Einstellen der Achsen und Abstände wurde eine GUI erstellt. In Abbildung 4.3 ist diese dargestellt. Es kann eine existierende Einstellung geladen und editiert werden. Um eine neue Einstellung zu erstellen, muss ein neuer Name eingegeben werden. Für jede Achse des verwendeten Werkzeugs lässt sich die Distanz in mm oder Winkelgrad eingeben, außerdem die Achse des Joysticks und ob diese invertiert ist oder nicht. Um mehr

Achsen zur Verfügung zu haben, wurden jeweils zwei Buttons zu einer Achse definiert. Später könnte man sich hier auch vorstellen, dass einzelne Buttons spezielle definierte Funktionen des Roboters auslösen (z.B. Schwenkbewegung, Aufnahmen, usw.) .

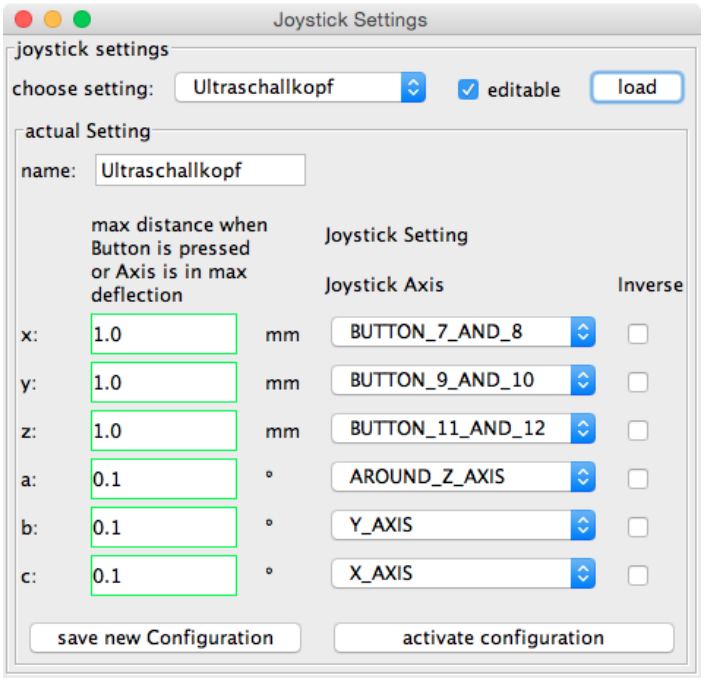


Abbildung 4.3: Joystick Einstellungs-GUI

Ein Klassendiagramm des Joysticks befindet sich im Anhang (C.2). Die Hauptklasse ist RobotJoystick. Sie bietet nach außen hin alle Funktionen, die im Paket joystick für die RA verfügbar sind. Ziele beim Joystick waren:

- **Einstellmöglichkeiten:** Um die Achsen des Joysticks mit beliebigen Funktionen zu belegen, wurde ein Interface mit Namen IJoystickAction definiert. In der Klasse RobotJoystick hat jeder Knopf und jede Achse ein entsprechendes Datenfeld mit IJoystickAction. Die Klasse JoystickOrigin, welche die Werte des Joysticks pollt, ruft nun in der Klasse RobotJoystick die Methoden des Joystick-Listener auf. Diese Methoden rufen ihrerseits wiederum die in den Datenfeldern abgelegten IJoystickAction-Methoden auf. Über Setter kann das Datenfeld verändert werden. Damit ist es möglich neue Aktionen beim Drücken von Knöpfen zu definieren.
- **Invertierung und Distanzregelung:** Für die Distanzregelung wurde einfach ein Multiplikator verwendet. Dieser multipliziert den Wert des Joysticks. Um den Wert zu invertieren, kann einfach ein negativer Multiplikator gewählt werden.

- **Speicherung der Einstellungen:** Zur Speicherung der Einstellungen wurde eine Klasse JAxisToRAxisSetting implementiert. Sie speichert die Zuweisungen für die Roboterachsen (x, y, z, a, b, c). Dabei werden hierfür jeweils die Joystickachse (enum JoystickAxis), deren eingestellte Distanz und ob invertiert wurde den Roboterachsen zugewiesen. Zum persistieren der Daten wurde eine Liste von diesen JAxisToRAxisSetting erstellt, die mit JAXB über die Klasse FileManager gespeichert werden kann.

4.3 Roboterapplikation

Die Roboterapplikation ist die Anwendung, die auf dem Roboter läuft. Sie verwaltet die programmierten Kommandos und setzt diese als Bewegung um. Dazu nimmt sie die Kommandos des externen Computers entgegen und interpretiert diese. Umgesetzte Kommandos sind hier das Führen von Hand mit Positionen abspeichern, das Anfahren von Positionen und das Schwenken des Ultraschallkopfes über den Joystick. Diese Teilkommandos, wurden in Kapitel 3 (Methodik) bereits implementiert. Diese Teillösungen mussten für das Ergebnis noch zusammengeführt und um die entfernte Steuerung erweitert werden. Dafür war eine Architektur auf Seiten des Roboters nötig, welche die Kommandos und Einstellungen verwaltet. Eine allgemeine grafische Darstellung der Architektur befindet sich im Abschnitt 4.1.1 unter Abbildung 4.1.

4.3.1 Aufbau / Architektur

Der Aufbau der RA ist in Abbildung 4.4 dargestellt. Ein Kommando muss das Interface IExecutor implementieren und in der Kommandoliste der RobotApplication-Klasse eingefügt sein. Die RobotApplication-Klasse, bietet die Umgebung des Roboters. Sie ist gleichzeitig der Client für OpenIGTLink. Das Kommando Ultraschall wurde bereits auf Seiten des externen Computers erläutert. In den folgenden Abschnitten wird nun das Gegenstück auf Seiten des Roboters erklärt. Die Klasse UltrasoundCommand ist hierbei die Umsetzung des Ultraschallkommandos. Es teilt sich in drei Teilkommandos auf:

- **JoystickMove:** Klasse, die eine Joystick-Bewegung ermöglicht.
- **GoToPoint:** Steuert einen angegebenen Punkt an. Ist der Punkt erreicht, wird dem Listener Bescheid gegeben.
- **TeachingPoints:** Schaltet die Handführung ein. Beim Speichern von Positionen wird die savePoint()-Methode des Listeners ausgeführt.

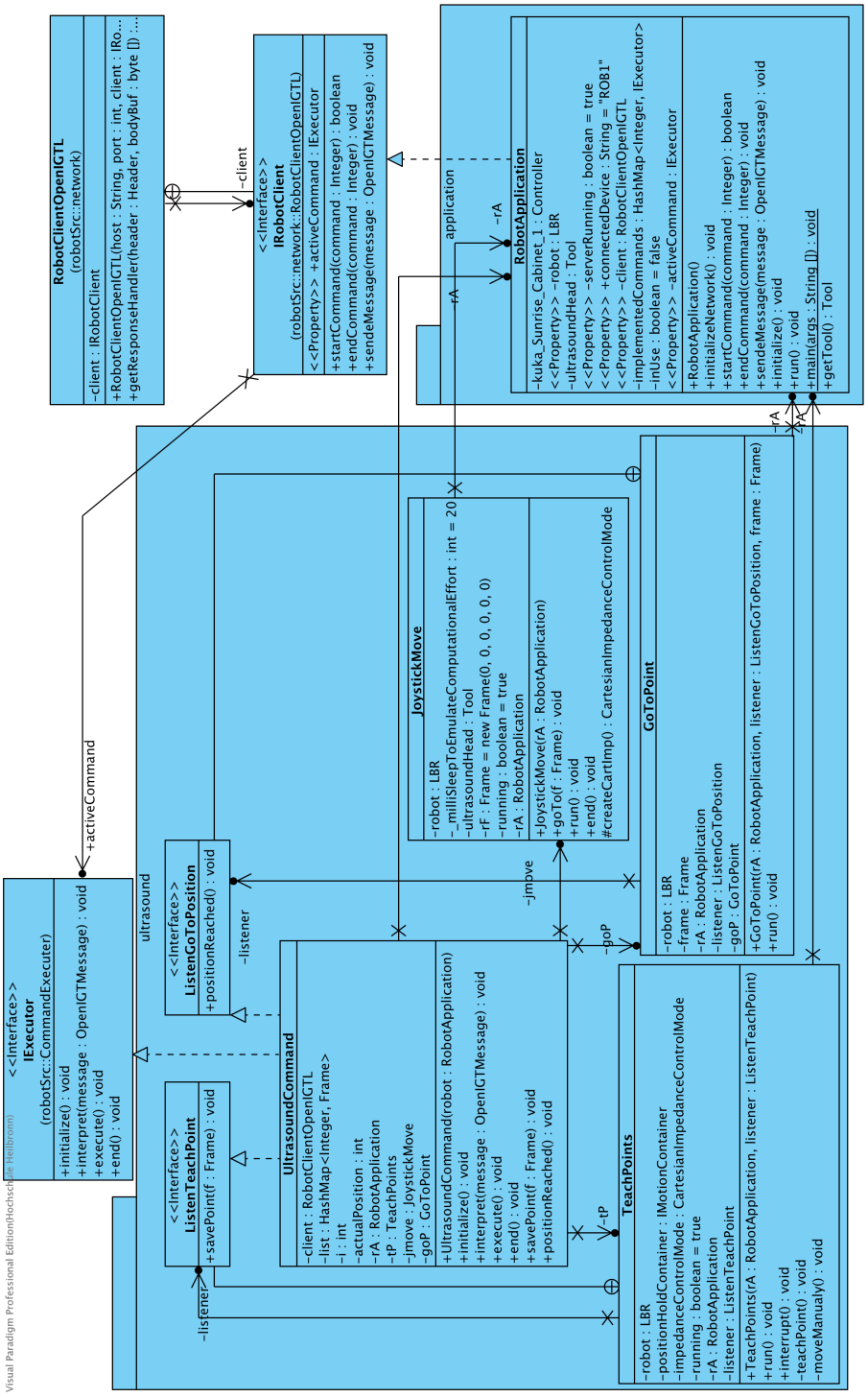


Abbildung 4.4: Klassendiagramm RobotApplication

4.3.2 Befehlserkennung

Eingehende Nachrichten werden über die Klasse RobotResponseHandler verwaltet. Ein Auszug des Quellcodes befindet sich im Anhang (B.3). Die Zuweisung der Kommandos erfolgt in der RobotApplication-Klasse. In Listing 4.1 ist dargestellt, wie ein Kommando verwendet wird. Im Konstruktor wird eine neue Instanz der Kommandoklasse in eine HashMap gelegt. Über den eindeutigen Index kann diese später abgerufen werden. Der RobotResponseHandler ruft bei einer StartCommand-Nachricht die Methode startCommand der RobotApplication-Klasse auf. Wird noch ein aktuelles Kommando ausgeführt, wird false zurückgegeben. Kann jedoch das Kommando ausgeführt werden, wird das Kommando aus der HashMap geladen und als aktives Kommando gesetzt. Alle Nachrichten, die ein Kommando ausführen, werden nun diesem Kommando übergeben.

```
1  public RobotApplication() {
2    super();
3    implementedCommands = new HashMap<Integer, IExecutor>();
4    initializeNetwork();
5    implementedCommands.put(AxialMovementCommand.name,
6        new AxialMovementExecutor(this));
7    implementedCommands.put(UltrasoundMovementCommand.name,new
         UltrasoundCommand(this));
8  }
9
10 public boolean startCommand(Integer command) {
11   if (inUse) {
12     return false;
13   } else {
14     activeCommand = implementedCommands.get(command);
15     activeCommand.initialize();
16     activeCommand.execute();
17     inUse = true;
18     return true;
19   }
```

Listing 4.1: RobotApplication-Klasse-Befehlszuweisung

4.3.3 Befehlsinitialisierung und Ausführung

Nach dem Starten des Kommandos werden alle Nachrichten des externen Computers an das aktuell auszuführende Kommando übergeben. Der RobotResponseHandler ruft dabei die interpret()-Methode auf. Über ein switch-case-Statement und Abfragen des Nachrichtentyps, werden die Datenfelder der Nachrichten interpretiert und den ggf. vorhandenen Subkommandos übergeben. In Listing 4.2 sieht man, wie die Methode interpret funktioniert. Es wird jeweils der Typ der Nachricht erfragt und ggf. anschließend eine Methode ausgefürt.

```
1     @Override
2     public void interpret(OpenIGTMessage message) {
3         if (message instanceof UltrasoundInitializeMessage) {
4             //TODO: hier nichts machen aendern
5         } else if (message instanceof TeachPointMessage) {
```

Listing 4.2: Ultraschall-Kommando-Klasse Roboter interpret() Beginn

4.3.4 Beispiel: schwenken des Ultraschallkopfes

In Listing 4.3 ist die Koordination der manuellen Bewegung mit dem Joystick implementiert. Über ein switch-case-Statement wird die Unterkommandoklasse koordiniert. Wichtiger Teil hier ist der Zustand EXECUTE: Hier wird ein Frame aus der übergebenen Position erstellt und die Methode goTo(Frame f) des Unterkommandos ausgeführt.

```
1         } else if (message instanceof MoveManuallyMessage) {
2             switch (((MoveManuallyMessage) message).
                  getExecutionState()) {
3                 case NONE:
4                     break;
5                 case START:
6                     jmove = new JoystickMove(rA);
7                     jmove.start();
8                     break;
9                 case END:
10                  jmove.end();
11                  jmove = null;
12                  rA.getLogger().info("BeendeManuelle Bewegung");
13                    break;
14                case EXECUTE:
15                  CartesianPositionModel p =((MoveManuallyMessage)
                        message).getDestinationPosition();
16                    Frame f = new Frame(p.getX(),p.getY(),p.getZ()
                          ,p.getA(),p.getB(),p.getC());
17                    jmove.goTo(f);
18                    break;
19            }
20        }
```

Listing 4.3: Ultraschall-Kommando-Klasse Roboter interpret() Abschnitt Joystickbewegung

Diese Umsetzung funktioniert, ist jedoch noch anfällig für Fehler. Wird der Ultraschallkopf zu stark geschwenkt, hält der Roboter die Spitze des Ultraschallkopfes nicht genau an ihrer Position. Bei der Bewegung wurde die Impedanz verwendet um den Roboter nachgiebig auf Bewegungen reagieren zu lassen. Ist deren Nachgiebigkeit zu

hoch eingestellt, klappt der Roboter beim Schallen nach unten und verliert wie ein schlaffer Arm den Kontakt zum Patienten. Ist die Einstellung zu fest, gibt er bei Atembewegungen nur wenig nach und drückt zu stark auf den Patienten. Dieses Problem (Atemkompensation) konnte somit noch nicht ausreichend umgesetzt werden. Erschwerend hierzu war die SmartServo-Klasse. Diese erlauben nur Koordinaten im Roboterfuß-Koordinatensystem einzugeben. Für eine notwendige Umrechnung war jedoch am Ende zu wenig Zeit, weshalb diese noch später ergänzt werden sollte.

5 Diskussion

Das Ausgangsszenario (Abschnitt 2.1) beschreibt, wie sich ein Ultraschallkopf zur Lagekontrolle während der Bestrahlung einsetzen lässt. Da Ultraschall eine kontinuierliche Anbindung an die Oberfläche des Patienten fordert, reicht eine starre Halterung nicht aus. In dieser Arbeit, sollte deswegen ein Roboter mit geeigneter Sensorik auf diese Problematik hin getestet werden. Dieser sollte sich in einen Ablauf einbinden lassen, in dem er später das Ultraschallgerät während der Bestrahlung an den Patienten drücken kann und damit eine Lagekontrolle des Tumors ermöglicht. Hierzu sollten die dafür notwendigen Schritte, Konzepte und Anwendungen entwickelt werden. Basis der Arbeit bildet das Anwendungsszenario (Abschnitt 2.1). Es beschreibt, wie der Arzt zuerst den Roboter von Hand zu gewünschten Schall-Positionen führt. Diese Positionen werden gespeichert. Anschließend lassen sich die Positionen an einem externen Computer im Konrtollraum auswählen und der Roboter fährt diese an. Zuletzt kann er an diesen Positionen die Ausrichtung des Schallkopfes über einen Joystick verändern. Im wesentlichen beschreiben diese Teilschritte Teilprobleme, die in dieser Arbeit gelöst wurden.

5.1 Ausgangslage

Zu Beginn der Arbeit gab es nur die Grundidee, dass es möglich sein könnte, eine Lagekontrolle mittels Ultraschall durchzuführen und den Gedanke dieses mit dem im Biosignallabor verfügbaren Leichtbauroboter zu realisieren. Erschwerend kam hier dazu, dass bisher noch keinerlei Erfahrungen mit diesem Roboter gesammelt wurden. Der erste Schritt musste somit sein, ein grundsätzliches Verständnis für den verwendeten Roboter zu erarbeiten. Wichtig hierbei ist vor allem das Austesten und Herausfinden der Grenzen des Ganzen.

5.1.1 Einarbeitung Roboter

Die Einarbeitung in den Roboter war trotz Verwendung von Java schwierig und zeitaufwändig. Die mitgelieferte Dokumentation von KUKA bietet jedoch eine gute Grundlage, um die Programmierung und Funktionsweise des Roboters zu verstehen. Das Benutzerhandbuch [3] bietet eine schrittweise Herangehensweise an den Roboter. Es eignet sich auch für jemanden, der noch keine Erfahrungen mit Robotern gesammelt hat. Etwas umständlich ist jedoch, dass zusätzlich zum Benutzerhandbuch noch andere Dokumente existieren, z.B. SmartServo. Dadurch ist manchmal nicht ganz nachzuvollziehen, was wo nachzulesen ist. Einige Funktionen bleiben so auch unentdeckt. Die Dokumentation von SmartServo ist zwar detailliert, da jedoch die referenzierten Beispielprogramme nicht alle funktioniert haben, schwierig nachzuvollziehen. Zusammenfassend lässt sich sagen, dass KUKA genügend Dokumente bereitstellt, um alle Möglichkeiten des Roboters zu verwenden. Voraussetzung ist allerdings genügend Zeit für die Einarbeitung.

5.1.2 Vorarbeiten

Die Recherche ergab nur wenige Artikel, die ähnlichen Szenarien beschrieben. Im Bereich **Roboterkommunikation** [17, 11] wurde eine gute Quelle für ein ähnliches Problem gefunden. Es wird hierbei eine Möglichkeit beschrieben eine Fräße über einen Leichtbauroboter unter Verwendung eines medizinischen Bildverarbeitungsprogramms zu steuern. Nach der Onlinerecherche war es die einzige Quelle, die einen Ansatz, bzw. eine Lösung für das Problem bot. Die allgemeine Architektur des Projekts ermöglichte es, OpenIGTLink in Betracht zu ziehen, was letztendlich auch zur Lösung dieses Problems beitrug. Durch die nicht ausreichende und fehlende Dokumentation der bereits erstellten Klassen, war jedoch eine direkte Erweiterung des vorhandenen Projekts nicht möglich. Erst nach eigenem Entwickeln konnte deren Lösungsweg nachvollzogen werden. Zusammenfassend lässt sich hier sagen, dass dieses Projekt einen guten Einstieg in die Problematik bietet. Gerade dadurch, dass es als einziges der gefundenen Projekte seinen Quellcode angibt, lassen sich einige Ansätze (z.B. Robotervisualisierung) noch aus diesem Projekt entnehmen und ggf. in diesem erweitern.

5.2 Anforderungen

Nachdem keine konkreten Umsetzungen und Ansätze für die Aufgabe gefunden wurden, musste zuerst ein eigenes Szenario mit Ablauf erstellt werden (Abschnitt 2.1). Aus diesem ergaben sich folgende Hauptanforderungen:

- **Positionen beibringen:** Um später den Ultraschallkopf an verschiedenen Positionen schnell positionieren zu können, ist es notwendig den Roboter direkt von Hand führen zu könnnen.
- **Positionen speichern und anfahren:** Die beigebrachten Positionen müssen sich speichern und wieder anfahren lassen. Dazu braucht es einen Algorithmus, der beim Anfahren niemanden verletzt.
- **Atemkompensation:** Befindet sich der Ultraschallkopf an einer Position müssen die Atembewegungen kompensiert werden können.
- **Entfernte Steuerung:** Der Roboter muss sich während der Bestrahlung von außerhalb steuern lassen. Damit später der Ultraschallkopf nachträglich ausgerichtet werden kann, braucht es ein Eingabegerät, dass Schwenkbewegungen ermöglicht (Joystick).
- **Darstellung der Kommandos:** Es muss eine Möglichkeit gefunden werden Kommandos und deren Parameter in einer GUI darzustellen.
- **Ausrichtung des Ultraschallkopfes:** Am Anfang war nur eine einfache Halterung des Ultraschallkopfes verfügbar. Da diese keine großen Schwenkbewegungen zuließ, musste eine Alternative konstruiert werden.

Hier ist klar zu sagen, das für eine spätere Realanwendung noch viele weitere Anforderungen zu klären sind. Dazu gehören z.B. die Darstellung und Erfassung des Ultraschallbildes. Somit lassen sich diese Anforderungen nur als Grundlage für die Robotersteuerung einer robotergestützten Ultraschalldiagnostik nehmen.

5.3 Ziele

Aus den Anforderungen können folgende Hauptziele abgeleitet werden:

- Beschreiben eines möglichen Anwendungsszenarios und dessen Ablauf
- Erstellen einer Roboter-Applikation zur Steuerung von: Anfahren von Positionen, Führen des Roboters von Hand, Entfernte Steuerung des Roboters, Atemkompensation und zur Sicherheitsüberwachung.
- Entwickeln eines externen Steuerungscomputers: Dieser muss später den Roboter von außerhalb steuern können und Schwenkbewegungen eines Joysticks auf den Ultraschallkopf übertragen können.
- Insgesamt muss somit ein Konzept und eine Architektur entwickelt werden, welche die Ziele zuvor ermöglicht.

5.4 Lösungsschritte

Zur Lösung der Ziele, wurden als erstes Beispielprogramme für den Roboter geschrieben. Diese lösten die in den Anforderungen beschriebenen Probleme. Später wurden diese Beispielprogramme in Kommandos umgewandelt und in der eigentlichen Anwendung implementiert. Der Aufwand die Beispielprogramme nachträglich in eine Anwendung einzubetten, war zwar groß, hat sich jedoch bewährt. Im Umgang mit dem Roboter ist es einfacher kleine Abschnitte zu testen und zu debuggen, als ein ganzes Programm. Daher konnten über die Beispielprogramme erstmals die Funktionalität getestet werden und die Parameter, z.B. für die Impedanz, ermittelt werden.

5.5 Zusammenfassung und Bewertung der Ergebnisse

Für das Hauptziel der Arbeit (Konzeption und Erstellung einer Applikation zur robotergestützten Ultraschalldiagnostik) wurde ein Ansatz entwickelt. Der ausgedachte Ablauf ist in einer Anwendung bereits mit folgenden Hauptfunktionen umgesetzt:

- **Halterung des Ultraschallkopfes:** Um den Roboter möglichst vielseitig einsetzen zu können, wurde eine Möglichkeit gefunden, die Ultraschallköpfe in unterschiedlichen Winkeln am Roboter zu befestigen.

- **Bewegen des Roboters von Hand, Abspeichern von Positionen:** Es ist möglich den Ultraschallkopf von Hand zu der gewünschten Position zu ziehen. Die Positionen lassen sich über einen Dialog am SmartPad einfach abspeichern und an den externen Computer schicken. An die Grenzen stößt die Anwendung jedoch, wenn zu schnell an dem Ultraschallkopf gezogen wird. Der Roboter bleibt aus Sicherheitsgründen dabei stehen. Auch ist der Roboter bei der bisherigen Implementierung zu leicht nachgiebig. Dadurch ist es notwendig ihn mit beiden Armen zu heben und an die gewünschte Position zu ziehen. Schwierig wird das Ganze, da gerade gleichzeitig noch der Sicherungstaster des SmartPads gedrückt werden muss. Besser zum Abspeichern der Positionen, wäre ein Taster, der auf Knopfdruck die Position sichert. Mit etwas mehr Zeit leicht noch zu ergänzen und bereits angefangen.
- **Anfahren von Positionen:** Es wurde ein einfacher Algorithmus geschrieben um Positionen anzufahren, ohne den Patienten vorläufig zu berühren. Es können so die zuvor gespeicherten Positionen von einem externen Computer aus aufgerufen und mit dem Roboter angefahren werden.
- **Schwenken des Ultraschallkopfes mit dem Joystick:** Ein entferntes Schwenken des Ultraschalkopfes über den Joystick wurde realisiert. Die Schwenkbewegung ist jedoch noch nicht fehlerfrei. Falsch eingestellte Parameter, z.B. Wert Joystick entspricht Wert Roboterachse, können noch zu Fehlern führen. Auch fehlt hier noch die Atemkompensation.
- **Einrichten einer Steuerungs-GUI:** Über eine GUI lassen sich verschiedene Kommandos verwalten und Parameter für Joystick oder Kommandos einstellen. Für eine spätere Anwendung fehlt hier noch das Darstellen des Ultraschallbildes.

Es ist verständlich, dass im Zuge der Komplexität der Aufgabe nicht alle Ziele vollständig umgesetzt wurden. Für eine spätere Anwendung außerhalb der Laborbedingungen müssen noch viele weitere Sicherheitsaspekte erfüllt werden. Diese Arbeit lässt sich somit als Grundlage nehmen, um das Gesamtkonzept einer robotergestützten Ultraschalldiagnostik zu verstehen und zu erweitern.
Die Arbeit ist sehr darauf ausgelegt, das Problem als Ganzes zu lösen. Daher sind einige Ansätze zwar vorhanden, nicht jedoch bis in die Tiefe programmiert. Vor allem im Bereich der Fehlerbehandlung traten einige Schwierigkeiten auf, die bis zuletzt noch nicht richtig gelöst werden konnten. Beispiel hierfür ist die Steuerung des Roboters mit dem Joystick. Bei falschen oder zu großen Werten muss das Teilkommando neu gestartet werden. Auch erwies sich die Verwendung von SmartServo als schwieriger, als erwartet. Die Verwendung von OpenIGTLink ist zurückblickend richtig gewesen. Gerade durch die Möglichkeit neue Nachrichtenklassen definieren zu können ist die Erweiterbarkeit gesichert. Zu beobachten ist hier, wie sich dieses Protokoll weiterentwickelt und ob die Robotersteuerung weiter ausgebaut wird.

Abschließend lässt sich sagen, dass die entwickelte Roboterapplikation eine gute Basis bietet, den Bereich der medizinischen Robotik weiter auszubauen. Es lassen sich ohne Probleme weitere Befehle ergänzen. Das entwickelte Konzept zeigt außerdem viele Facetten der von KUKA bereitgestellten Methoden und ihrer Anwendung.

5.6 Ausblick

Das Themenfeld der med. Robotik ist gerade erst wieder am Aufleben. Bevor es in der großen Masse verwendet werden kann, braucht es noch viele weitere Projekte. Gerade die Sicherheit ist ein großes Feld, welches gelöst werden muss. Die Thesis zeigt, dass es möglich ist, einen Roboter zur Ultraschalldiagnostik verwenden zu können. Das Programm könnte nun auch für das Erstellen eines 3D Ultraschalls erweitert werden. Durch eine Schwenkbewegung und die Verwendung der Koordinaten, könnte das realisiert werden. Ferner wäre ein Ziel, in einem CT-Bild, einen Punkt auszuwählen und den Roboter selbständig diesen Punkt im Ultraschallbild sichtbar machen zu lassen. Somit bleibt es ein spannender Themenbereich, der noch viel Raum für Innovationen lässt.

6 Literaturverzeichnis

[1] Conti F, Park J, Khatib O, "Interface design and control strategies for a robot assisted ultrasonic examination system. proc int symposium on experimental robotics.," 2010.

[2] Beyl T, Brennecke T, Raczkowsky J, Wörn H , "Ultrasound tomography using a light weight robot. proc. of the 25th int. congress computer assisted radiology and surgery cars.," 2011.

[3] KUKA, *KUKA Sunrise.OS 1.3 KUKA Sunrise.Workebench 1.3*, 31.7.2014.

[4] KUKA, *KUKA Sunrise.Connectivity Servoing 1.5 Sunrise.Connectivity Rendering 1.5*, 11.11.2014.

[5] KUKA, *KUKA Sunrise.Connectivity FRI 1.5*, 31.7.2014.

[6] KUKA, *KUKA Sunrise Cabinet*, 29.7.2014.

[7] KUKA, *LBR iiwa*, 31.7.2014.

[8] KUKA, *WorkVisual 3.0*, 25.11.2013.

[9] KUKA, *LBR iiwa Quick Start*, 29.7.2014.

[10] KUKA, *Sicherheit LBR iwa*, 29.7.2014.

[11] Sebastian Tauscher, Junichi Tokuda, et. al , "Openigtlink interface for state control and visualisation of a robot for image-guided therapy systems," 13.6.2014. DOI 10.1007/s11548-014-1081-1.

[12] The OpenIGTLink Community and Brigham and Women's Hospital, "Openigtlink." `http://openigtlink.org`, 22.8.2015.

[13] Lightweight Java Game Library Project, "Lwjgl." `http://www.lwjgl.org`, 22.8.2015.

[14] B. Ed Ort, "Jaxb." `http://www.oracle.com/technetwork/articles/javase/index-140168.html`, 22.8.2015.

[15] "3d slicer." `http://www.slicer.org`, 21.9.2015.

[16] Oracle, "Rmi." `https://docs.oracle.com/javase/tutorial/rmi/`, 22.8.2015.

[17] S. Tauscher, "Lightweightrobotigt- getting started." `http://wiki.slicer.org/slicerWiki/images/1/17/Tutorial_LightWeightRobotIGT_Gettingstarted.pdf`, 22.8.2015.

Anhang

A Screenshots

A.1 Sunrise OS

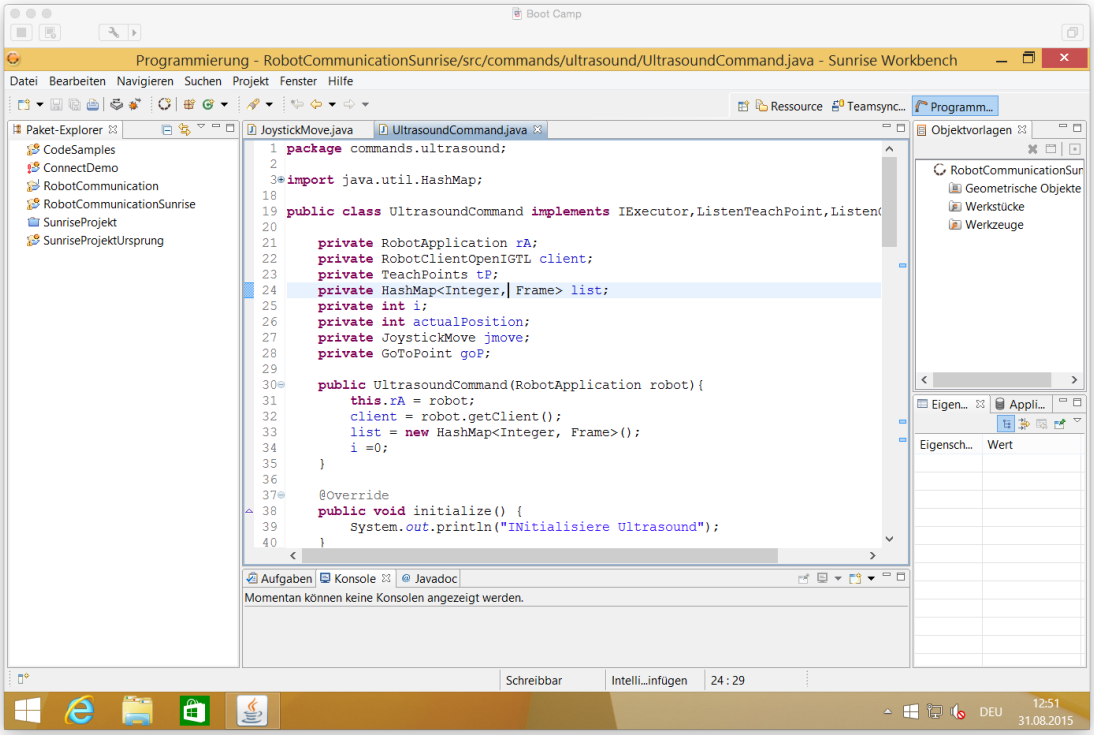


Abbildung A.1: Sunrise OS

A.2 GUIs ohne Funktion

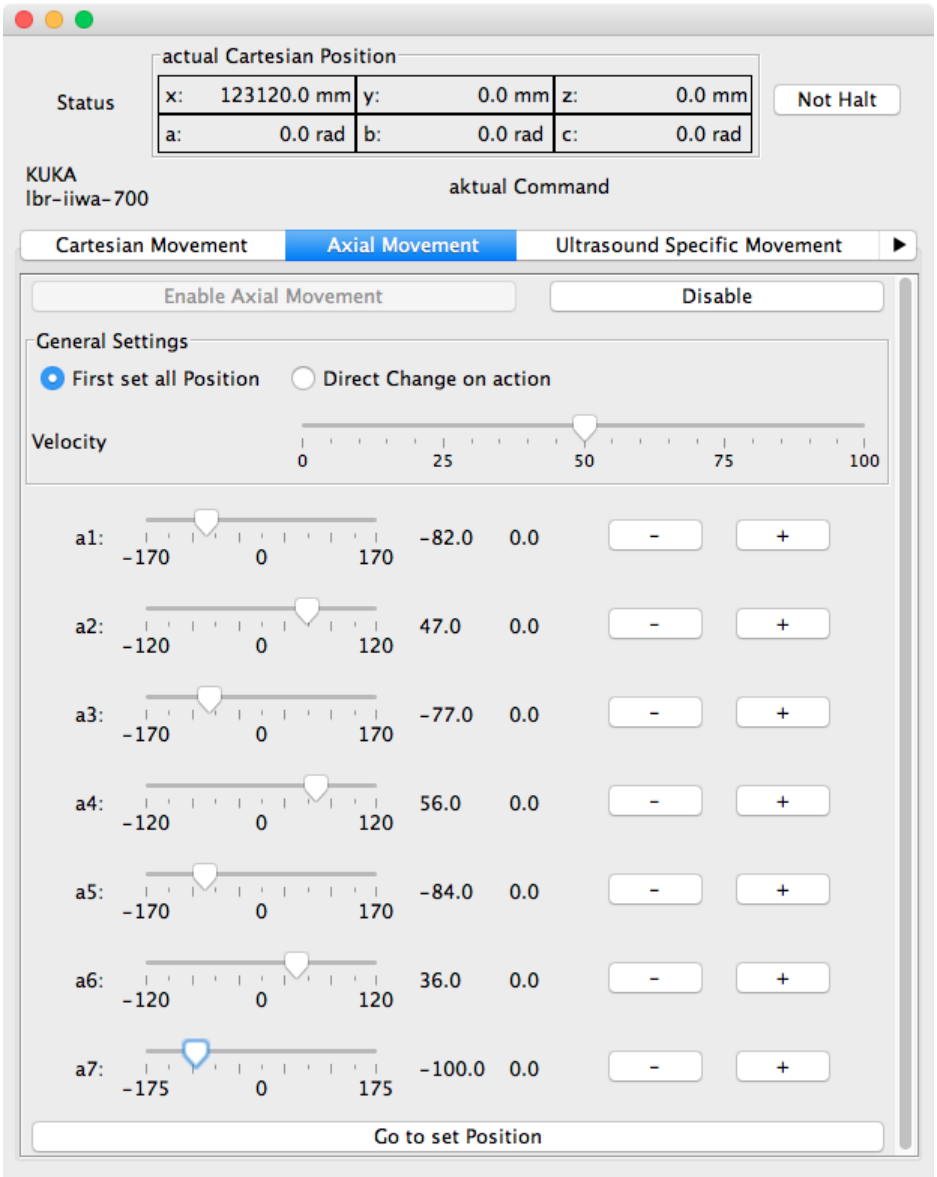


Abbildung A.2: Achsiale Bewegungs-GUI: Angedacht ist, dass entweder die Position über die Schieberegler eingestellt wird, oder die Achsen bei Bewegung der Regler angesteuert werden. Auszuwählen wäre das über den Radio-Button First set all Position bzw. Direct Change on action. Neben dem Zielwinkel ist außerdem der aktuelle Winkel des Roboters dargestellt.

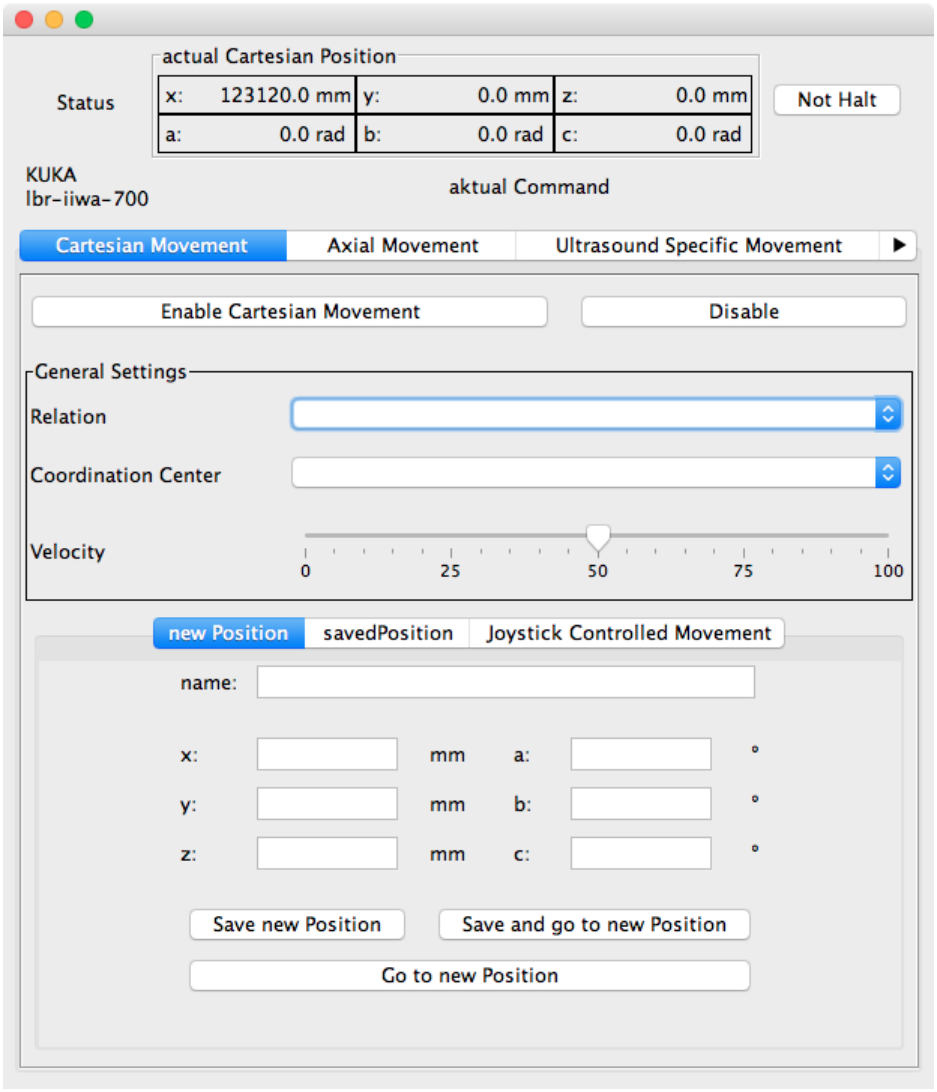


Abbildung A.3: Kartesische Bewegungs-GUI: Bei der Kartesischen Bewegung muss zuerst die Relation (absolut oder relativ) und der Koordinatenursprung(Roboterfuß, Flansch,...) eingestellt werden. Anschließend lassen sich neue Koordinaten eingeben. Gespeicherte Koordinaten sollen auswählbar sein und eine Joystickbewegung im Kartesischen Bereich ist auch angedacht.

B Anwendung

B.1 Einteilung in Pakete

Die Einteilung der Pakete erfolgt in 4 Hauptmodule:

- **communicationAndGeneralData:** Beinhaltet Klassen, die sowohl vom Roboter, als auch von dem externen Computer verwendet werden:
  - **communication.message:** Beinhaltet alle Nachrichtenklassen, die Kommandos oder Einstellungen repräsentieren. Unterpakete sind dabei die unterschiedlichen Kommandos.
  - **communication.network:**
    - **client:** Enthält den RobotOpenIGTLinkClient und den RobotResponseHandler.
    - **server:** Beinhaltet alle Klassen des RobotServer.
  - **exception:** Eigene Fehlerklassen.
  - **robot:** Allgemeine Klassen, die den Roboter beschreiben.
    - **model:** Enthält Klassen zur Speicherung von Positionen, des Status oder allgemeiner Definitionen des Roboters.
    - **tool:** Verwaltet und speichert Werkzeuge.
- **robotClient:** Beispielclient mit System.out.println() zum Testen des Servers.
- **externerComputer:** Bildet die Applikation auf dem externen Computer.
  - **commands:** Enthält die programmierten Kommandos.
  - **gui:** Beinhaltet die GUIs und deren Controller.
  - **joystick:** Die Klassen für den Joystick:
    - **controllers:** Enthält die Joystick-Klassen.
    - **data:** Verwaltet und speichert Joystickeinstellungen.
    - **samples:** Bietet Beispielprogramme um die Funktionalität des Joysticks darzustellen.
  - **startApplikation:** Enthält die Klasse mit der main-Methode.
- **robotCommunicationSunrise:** Projektordner in Sunrise. Enthält sowohl die Pakete, als auch die Sicherheitseinstellungen und das StationsSetup.
  - **application:** Enthält die Roboterapplikation.

- **commands:** Paket mit allen verwendeten Kommandos. Kommandos werden in Unterpaketen verwaltet.
- **testing:** Testprogramme zum Probieren von Funktionalität.

B.2 Quellcode

B.2.1 Standard Roboter-Applikation

```
1  import com.kuka.roboticsAPI.applicationModel.
       RoboticsAPIApplication;
2  import static com.kuka.roboticsAPI.motionModel.BasicMotions.*;
3  import com.kuka.roboticsAPI.controllerModel.Controller;
4  import com.kuka.roboticsAPI.deviceModel.LBR;
5
6  public class RobotApplication extends RoboticsAPIApplication {
7    private Controller kuka_Sunrise_Cabinet_1;
8    private LBR robot;
9
10   public void initialize() {
11     kuka_Sunrise_Cabinet_1 = getController("KUKA_Sunrise_Cabinet_1
         ");
12     robot = (LBR) getRobot(kuka_Sunrise_Cabinet_1,
13         "LBR_iiwa_7_R800_1");
14   }
15
16   public void run() {
17         robot.move(ptpHome());
18   }
19
20   public static void main(String[] args) {
21     RobotApplication app = new RobotApplication();
22     app.runApplication();
23   }
24 }
```

Listing B.1: Eine Standard Roboter Applikation wie sie später erweitert werden kann

B.2.2 Eingabemöglichkeiten Smartpad

```
1     IUserKeyBar serverBar = getApplicationUI().createUserKeyBar(
2         "ServerOIGT");
3     IUserKeyListener closeConnection = new IUserKeyListener() {
4       @Override
5       public void onKeyEvent(IUserKey key, UserKeyEvent event) {
6         //:TODO Action on Event
7       }
8     };
9     IUserKey openKey = serverBar.addUserKey(0, closeConnection,
          true);
10    openKey.setText(UserKeyAlignment.Middle, "Beschriftung");
11    getApplicationUI().getUserKeyBar("ServerOIGT").publish();
```

Listing B.2: Beispiel für die Buttonprogrammierung

B.2.3 Eigene Anwendung

```
1     public RobotResponseHandler(Header header, byte[] body,
          OpenIGTClient openIGTClient, IRobotClient client) {
2         super(header, body, openIGTClient);
3         getCapabilityList().add(AxialMovementMessage.name);
4         getCapabilityList().add(StartCommandMessage.name);
5         getCapabilityList().add(EndCommandMessage.name);
6         getCapabilityList().add(UltrasoundInitializeMessage.name);
7         getCapabilityList().add(TeachPointMessage.name);
8         getCapabilityList().add(GoToPointMessage.name);
9         getCapabilityList().add(MoveManuallyMessage.name);
10        this.client = client;
11    }
12
13    @Override
14    public boolean perform(String messageType) throws Exception {
15        if (messageType.equals(StartCommandMessage.name)) {
16            StartCommandMessage m = new StartCommandMessage(
                  getHeader(), getBody());
17            client.startCommand(m.getCommand());
18        } else if (messageType.equals(EndCommandMessage.name)) {
19            EndCommandMessage message = new EndCommandMessage(
                  getHeader(), getBody());
20            client.endCommand(message.getCommand());
21        } else if (messageType.equals(AxialMovementMessage.name))
              {
22            AxialMovementMessage am = new AxialMovementMessage(
                  getHeader(), getBody());
23            client.getActiveCommand().interpret(am);
24        } else if (messageType.equals(UltrasoundInitializeMessage.
              name)) {
25            client.getActiveCommand().interpret(new
                  UltrasoundInitializeMessage(getHeader(), getBody
                  ()));
26        } else if (messageType.equals(TeachPointMessage.name)) {
27            client.getActiveCommand().interpret(new
                  TeachPointMessage(getHeader(), getBody()));
28        } else if (messageType.equals(GoToPointMessage.name)) {
29            client.getActiveCommand().interpret(new
                  GoToPointMessage(getHeader(), getBody()));
30        } else if (messageType.equals(MoveManuallyMessage.name)) {
31            client.getActiveCommand().interpret(new
                  MoveManuallyMessage(getHeader(), getBody()));
32        }
33        return true;
34    }
```

Listing B.3: RobotResponseHandler: Im Konstruktor werden alle Nachrichtenklassen hinzugefügt, die später interpretiert werden können. in der perform()-Methode werden die Nachrichten vom Bytecode zurück gewandelt und ggf. ein Methode aufgerufen.

C Diagramme

C.1 Sequenzdiagramme Nachrichtenaustausch

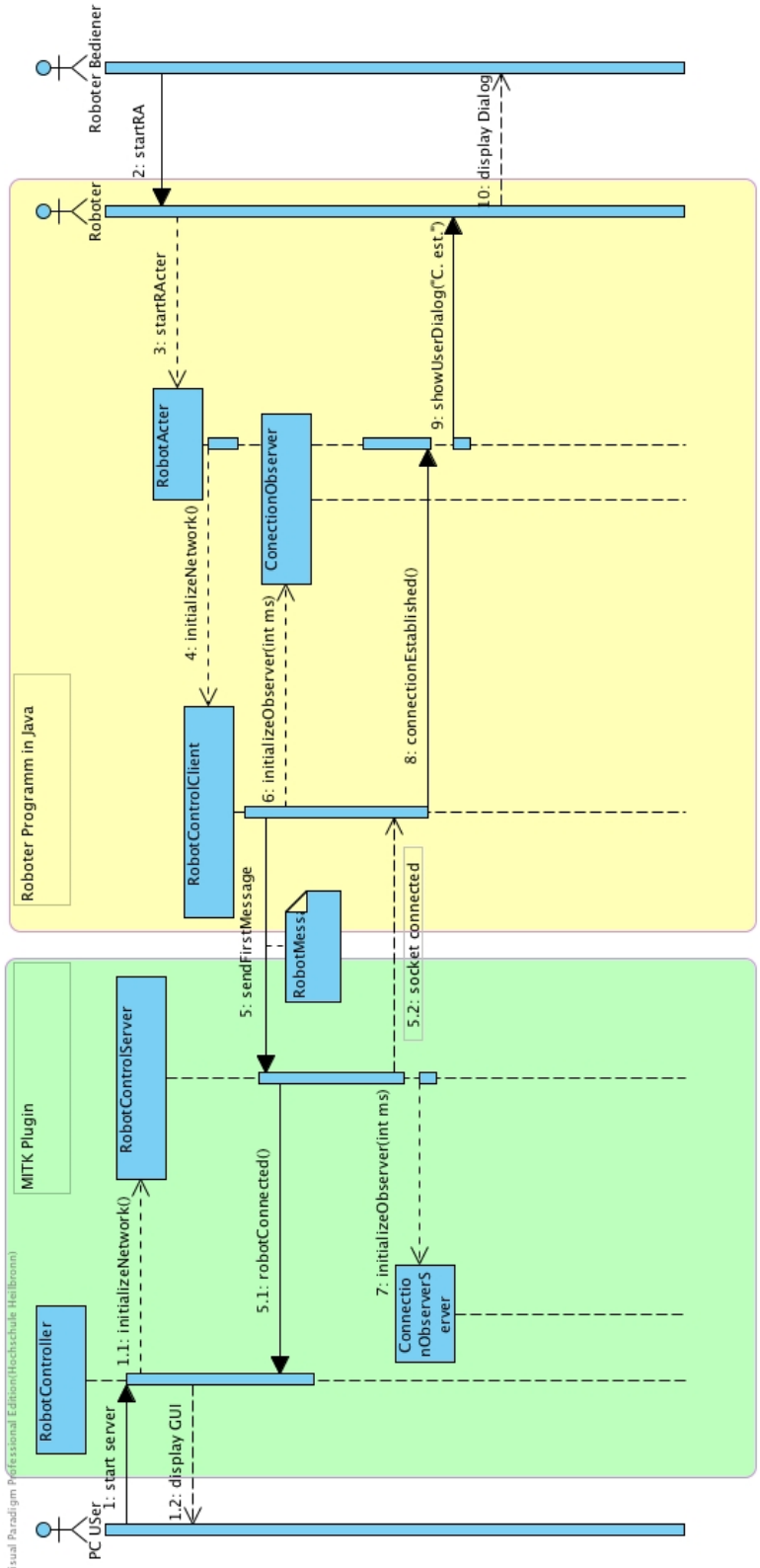


Abbildung C.1: Beim ersten Verbinden, meldet sich der Roboter beim Server und übergibt ihm eine Initialisierungsnachricht. Auf dem SmartPad erfolgt die Rückmeldung, wenn der Server verbunden ist.

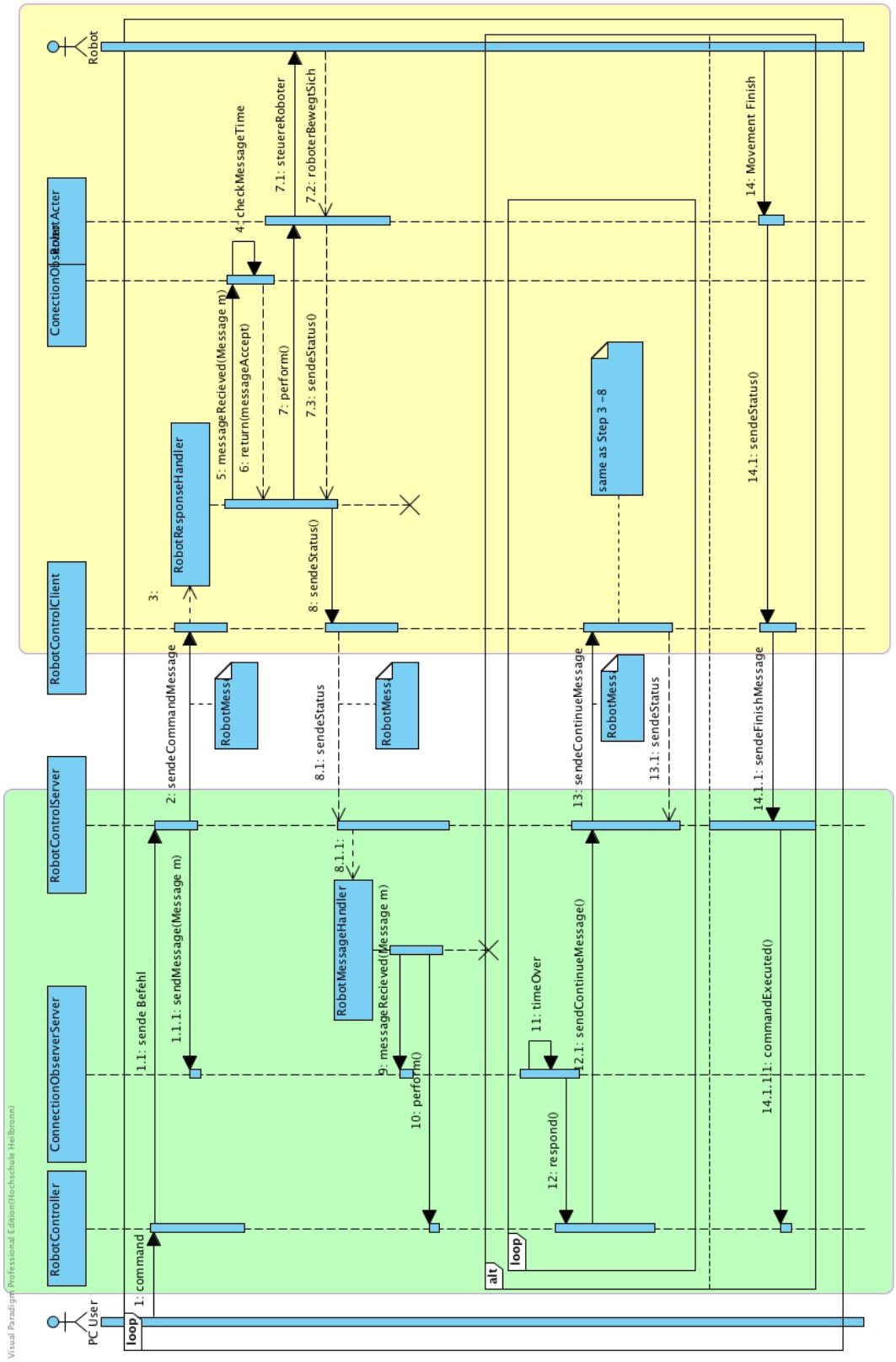
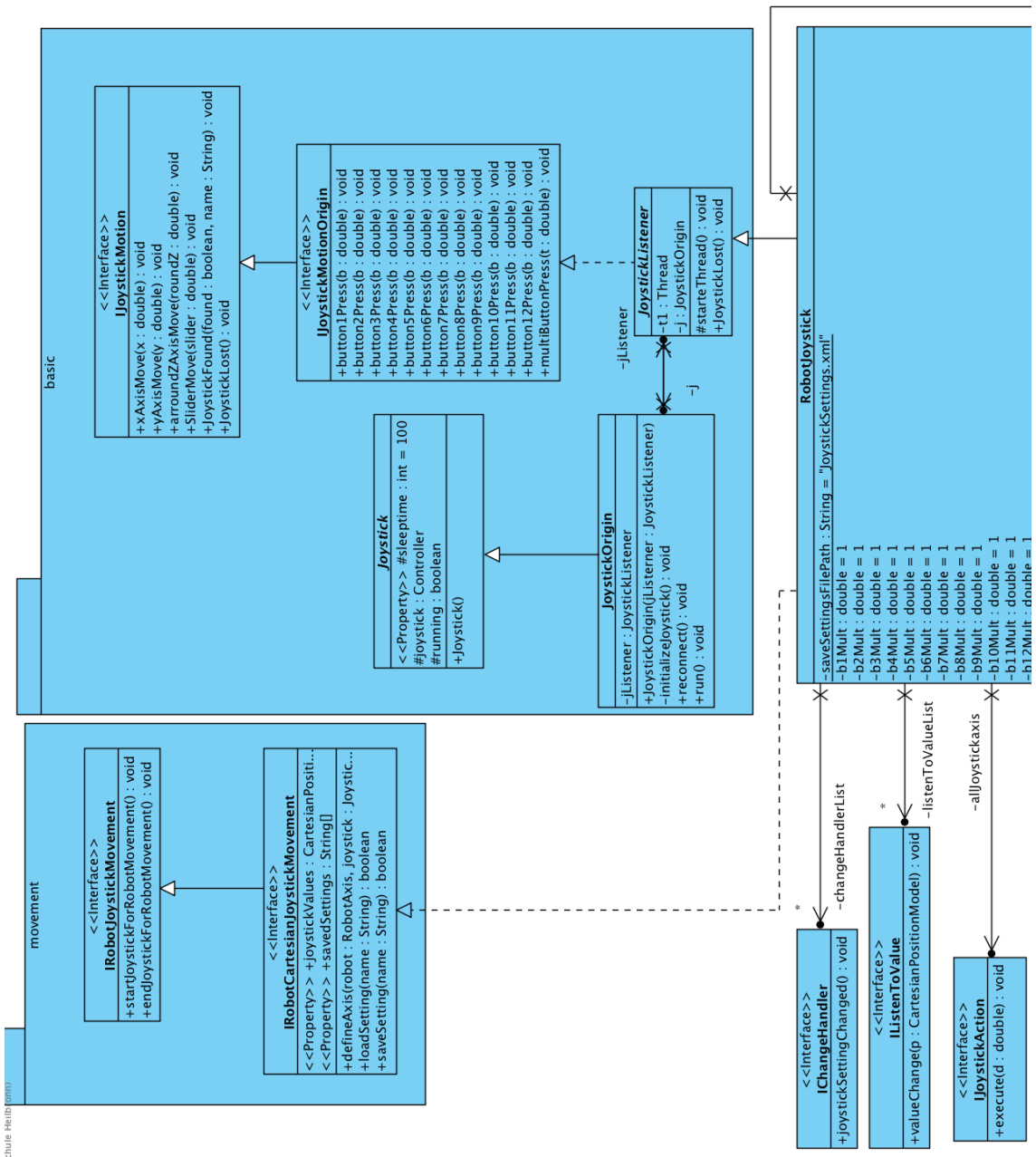
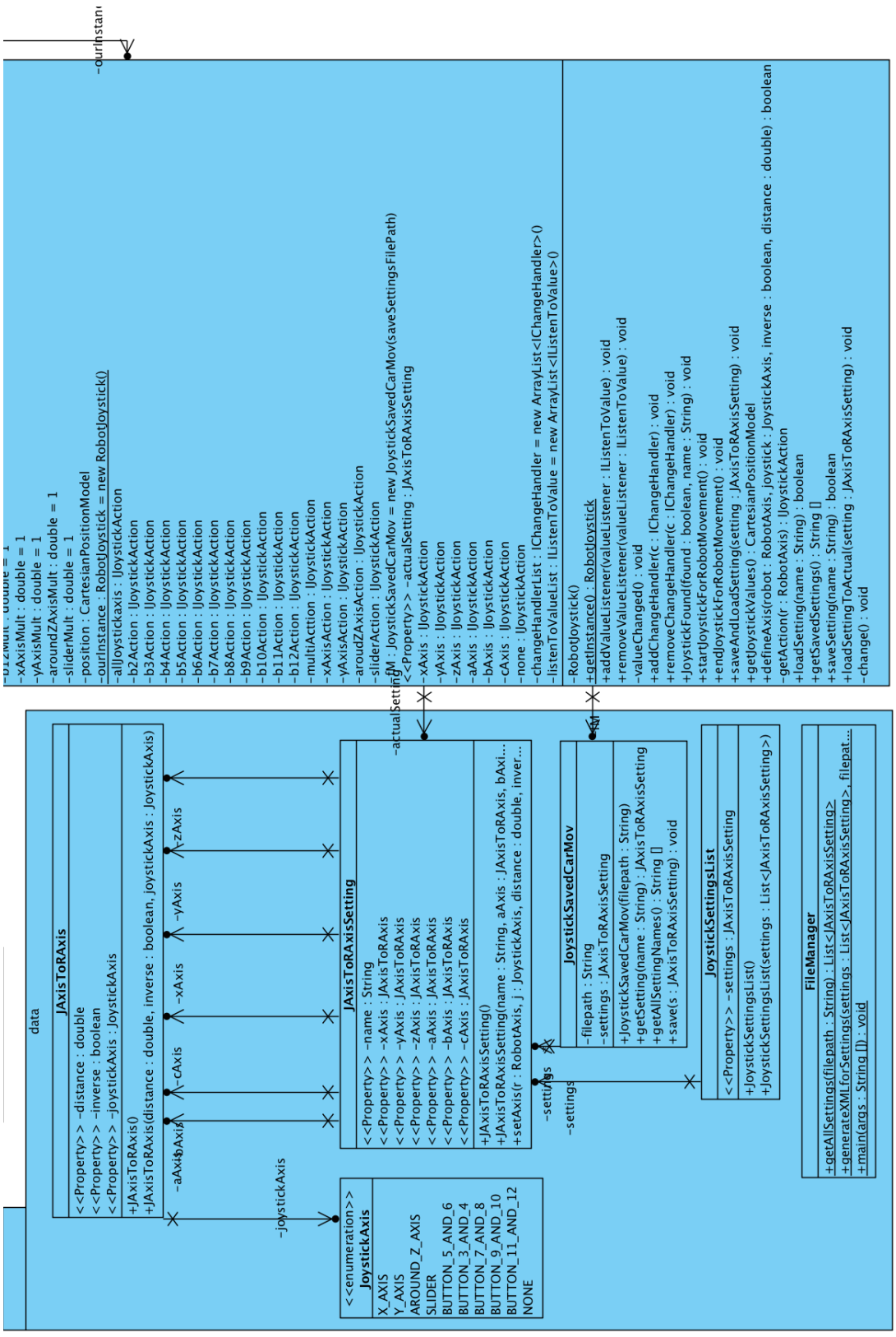


Abbildung C.2: Ausführen eines Kommandos: Beim Ausführen von Kommandos geht eine Initialisierungsnachricht voraus. Sie übergibt Parameter und Rahmenbedingungen für den Befehl. In einer Schleife kommen die aktuellen Werte des Befehls und werden vom Roboter ausgeführt. Ist dieser fertig oder der Befehl, wird eine Endnachricht geschickt.

C.2 Joystick Klassendiagramm


basic
<<Interface>>
IJoystickMotion
+xAxisMove(x : double) : void
+yAxisMove(y : double) : void
+arroundZAxisMove(roundZ : double) : void
+SliderMove(slider : double) : void
+JoystickFound(found : boolean, name : String) : void
+JoystickLost() : void
<<Interface>>
IJoystickMotionOrigin
+button1Press(b : double) : void
+button2Press(b : double) : void
+button3Press(b : double) : void
+button4Press(b : double) : void
+button5Press(b : double) : void
+button6Press(b : double) : void
+button7Press(b : double) : void
+button8Press(b : double) : void
+button9Press(b : double) : void
+button10Press(b : double) : void
+button11Press(b : double) : void
+button12Press(b : double) : void
+multiButtonPress(t : double) : void
Joystick
<<Property>> #sleeptime : int = 100
#joystick : Controller
#running : boolean
+Joystick()
JoystickOrigin
-jListener : JoystickListener
+JoystickOrigin(jListerner : JoystickListener)
-initializeJoystick() : void
+reconnect() : void
+run() : void
-jListener
-j
JoystickListener
-t1 : Thread
-j : JoystickOrigin
#starteThread() : void
+JoystickLost() : void
RobotJoystick
-saveSettingsFilePath : String = "JoystickSettings.xml"
-b1Mult : double = 1
-b2Mult : double = 1
-b3Mult : double = 1
-b4Mult : double = 1
-b5Mult : double = 1
-b6Mult : double = 1
-b7Mult : double = 1
-b8Mult : double = 1
-b9Mult : double = 1
-b10Mult : double = 1
-b11Mult : double = 1
movement
<<Interface>>
IRobotJoystickMovement
+startJoystickForRobotMovement() : void
+endJoystickForRobotMovement() : void
<<Interface>>
IRobotCartesianJoystickMovement
<<Property>> +joystickValues : CartesianPositi...
<<Property>> +savedSettings : String[]
+defineAxis(robot : RobotAxis, joystick : Joystic...
+loadSetting(name : String) : boolean
+saveSetting(name : String) : boolean
<<Interface>>
IChangeHandler
+joystickSettingChanged() : void
-changeHandlerList
<<Interface>>
IListenToValue
+valueChange(p : CartesianPositionModel) : void
-listenToValueList
<<Interface>>
IJoystickAction
+execute(d : double) : void
-allJoystickaxis


data
JAxisToRAxis
<<Property>> -distance : double
<<Property>> -inverse : boolean
<<Property>> -joystickAxis : JoystickAxis
+JAxisToRAxis()
+JAxisToRAxis(distance : double, inverse : boolean, joystickAxis : JoystickAxis)
<<enumeration>>
JoystickAxis
X_AXIS
Y_AXIS
AROUND_Z_AXIS
SLIDER
BUTTON_5_AND_6
BUTTON_3_AND_4
BUTTON_7_AND_8
BUTTON_9_AND_10
BUTTON_11_AND_12
NONE
JAxisToRAxisSetting
<<Property>> -name : String
<<Property>> -xAxis : JAxisToRAxis
<<Property>> -yAxis : JAxisToRAxis
<<Property>> -zAxis : JAxisToRAxis
<<Property>> -aAxis : JAxisToRAxis
<<Property>> -bAxis : JAxisToRAxis
<<Property>> -cAxis : JAxisToRAxis
+JAxisToRAxisSetting()
+JAxisToRAxisSetting(name : String, aAxis : JAxisToRAxis, bAxi...
+setAxis(r : RobotAxis, j : JoystickAxis, distance : double, inver...
JoystickSavedCarMov
-filepath : String
-settings : JAxisToRAxisSetting
+JoystickSavedCarMov(filepath : String)
+getSetting(name : String) : JAxisToRAxisSetting
+getAllSettingNames() : String []
+save(s : JAxisToRAxisSetting) : void
JoystickSettingsList
<<Property>> -settings : JAxisToRAxisSetting
+JoystickSettingsList()
+JoystickSettingsList(settings : List<JAxisToRAxisSetting>)
FileManager
+getAllSettings(filepath : String) : List<JAxisToRAxisSetting>
+generateXMLforSettings(settings : List<JAxisToRAxisSetting>, filepat...
+main(args : String []) : void
-ourInstan
-xAxisMult : double = 1
-yAxisMult : double = 1
-aroundZAxisMult : double = 1
-sliderMult : double = 1
-position : CartesianPositionModel
-ourInstance : RobotJoystick = new RobotJoystick()
-allJoystickAxis : IJoystickAction
-b2Action : IJoystickAction
-b3Action : IJoystickAction
-b4Action : IJoystickAction
-b5Action : IJoystickAction
-b6Action : IJoystickAction
-b7Action : IJoystickAction
-b8Action : IJoystickAction
-b9Action : IJoystickAction
-b10Action : IJoystickAction
-b11Action : IJoystickAction
-b12Action : IJoystickAction
-multiAction : IJoystickAction
-xAxisAction : IJoystickAction
-yAxisAction : IJoystickAction
-aroudZAxisAction : IJoystickAction
-sliderAction : IJoystickAction
-FM : JoystickSavedCarMov = new JoystickSavedCarMov(saveSettingsFilePath)
<<Property>> -actualSetting : JAxisToRAxisSetting
-xAxis : IJoystickAction
-yAxis : IJoystickAction
-zAxis : IJoystickAction
-aAxis : IJoystickAction
-bAxis : IJoystickAction
-cAxis : IJoystickAction
-none : IJoystickAction
-changeHandlerList : IChangeHandler = new ArrayList<IChangeHandler>()
-listenToValueList : IListenToValue = new ArrayList<IListenToValue>()
-RobotJoystick()
+getInstance() : RobotJoystick
+addValueListener(valueListener : IListenToValue) : void
+removeValueListener(valueListener : IListenToValue) : void
-valueChanged() : void
+addChangeHandler(c : IChangeHandler) : void
+removeChangeHandler(c : IChangeHandler) : void
+JoystickFound(found : boolean, name : String) : void
+startJoystickForRobotMovement() : void
+endJoystickForRobotMovement() : void
+saveAndLoadSetting(setting : JAxisToRAxisSetting) : void
+getJoystickValues() : CartesianPositionModel
+defineAxis(robot : RobotAxis, joystick : JoystickAxis, inverse : boolean, distance : double) : boolean
-getAction(r : RobotAxis) : IJoystickAction
+loadSetting(name : String) : boolean
+getSavedSettings() : String []
+saveSetting(name : String) : boolean
+loadSettingToActual(setting : JAxisToRAxisSetting) : void
-change() : void
-joystickAxis
-aAxis
-bAxis
-cAxis
-xAxis
-yAxis
-zAxis
-actualSetting
-settings
-settings

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

(Ort, Datum) (Unterschrift)