Bachelorarbeit:

Entwurf und Implementierung eines Remote-User-Interfaces zur Robotersteuerung auf einem Tablet-PC

Ulrike Schäfer
(7schaefe@informatik.uni-hamburg.de)

Technische Aspekte Multimodaler Systeme
Department Informatik
Universität Hamburg

Erstbetreuer: Prof. Dr. Jianwei Zhang
Universität Hamburg
Department Informatik, TAMS

Zweitbetreuer: Dr. Norman Hendrich
Universität Hamburg
Department Informatik, TAMS

26. November 2012

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

Ein lange existierendes Feld der Robotik ist die Kräfteverteilung und Steuerung von robotischen Händen [Nar88]. Die Planung von Algorithmen zum Greifen [PP93] von Objekten, vom Erkennen von Objekten die gegriffen werden sollen beschäftigt viele wissenschaftliche Arbeiten. Auch ein Teil der Forschung auch am Fachbereich TAMS, der Informatik der Universität Hamburg, ist mit diesen Problemen der Robotik konfrontiert. In dieser Arbeit soll die Nutzung eines Tablet-PCs zur Steuerung eines Service-Roboters möglich gemacht werden indem ein Client-/Server-basiertes Werkzeug entworfen und implementiert wird. Die Klasse der Tablet-PCs verspricht eine gute Eignung zur Steuerung von Service-Robotern, da sie einen hohen Grad an Mobilität bietet. Sie sind portabler und verbrauchen weniger Leistung und können daher örtlich flexibler eingesetzt werden als Desktop-PCs. Außerdem bieten sie den wichtigen Aspekt der Multitouchfähigkeit der im Einsatz mit Robotern die viele Freiheitsgrade haben besonders sinnvoll sein könnte.

1.1 Problemstellung

Die Vor- und Nachteile eines Tablet-PCs werden bei dem Entwurf und der Implementierung des Remote-User-Interfaces zur Robotersteuerung berücksichtigt und beleuchtet. Da es sich um ein iPad und eine iOS-Anwendung [iSLA12a] handelt wird speziell auf die Beschränkungen in diesem Fall eingegangen und die Anwendung nach dem objektorientierten Softwareparadigma designt. Die Eingabe mit Multitouch-Gesten wird recherchiert und erprobt daraufhin eine konkrete Bedienung für die Shadow-Hand C5/C6 [Com08] mit ihren 20 Freiheitsgraden entworfen und implementiert.

1.2 Zielsetzung

Um eine alternative Steuerung einer menschenähnlichen Roboterhand zu bieten, soll ein multitouchfähiges Tablet-PC genutzt werden. Auf diesem mobilen Gerät soll ein Remote-User-Interface für die vorhandenen Schnittstellen entwickelt werden. Es soll für den konkreten Service-Roboter eine Steuerung mit geeignete Interfacetypen entwickelt werden. Die Anwendung soll an eventuelle

Änderungen der Roboterumgebung anpassbar sein und in der Forschung an der Shadow-Hand ein Werkzeug zu deren Steuerung darstellen.

Kapitel 2

Planung der Software

Das Werkzeug dient zur Bedienung einer Shadow C6 Hand [Com08] von der Shadow Robot Company auf einem iPad und soll die Verwendung eines touchfähigen mobilen Endgerätes zur Robotersteuerung implementieren und untersuchen. Die Hand kann in dem Simulator GraspIt [CSD12] durch ein Roboter-Model simuliert werden. In diesem Kapitel wird der Umfang des Werkzeuges eingegrenzt und die Anforderungen der Anwendung in Form von Szenarien dargestellt.

2.1 Anforderung an die Software

Die Anwendung umfasst soweit fünf voneinander abgekoppelte Arbeitsabläufe, die unten als die Benutzungsszenarien aufgelistet sind. Insbesondere soll die ShadowApp in der Lage sein alle Parameter der Roboterhand zur Laufzeit zu steuern. Das User-Interface auf dem Touchscreen und die Bedienung soll sich von dem User-Interface des mausbasierten Programms absetzen.

2.1.1 Beispiel einer mausbasierten Steuerung: hdbt-gui.jar

Zur Steuerung der Shadow Dextrous Hand (C5 air-muscle type/C6 motor type) gibt es bisher bereits ein Werkzeug namens Shadow Dextrous Hand C5/C6 GUI von Dr. Norman Hendrich. Bei der Betrachtung der Oberfläche fällt eine ausschließliche Benutzung von Sliderelementen ins Auge.

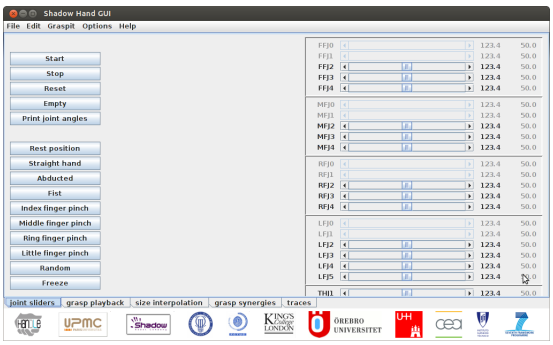

Abbildung 2.1: Fenster zur Gelenksteuerung in der Desktop-Anwendung

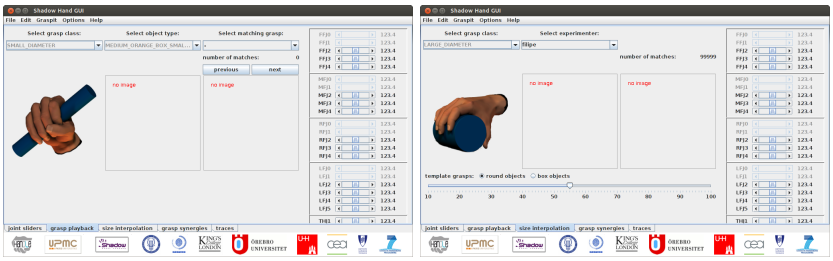

Abbildung 2.2: Fenster: 1.) Auswahl von Griffen und 2.) Griffklassen

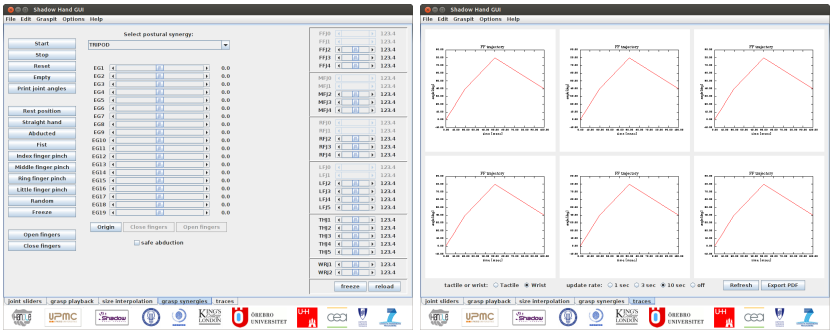

Abbildung 2.3: Fenster: 1.) Auswahl von Synergien mit Parametern und 2.) Anzeige der Traces

2.2 Benutzungsszenarien

Es folgt eine Darstellung der Benutzungsszenarien wie sie als Benutzer während der Steuerung der Shadow-Hand mit der Anwendung ablaufen. Die Szenarien sind alle darauf ausgelegt eine Shadow-Hand mit 20 Freiheitsgraden zu Steuern, die sich vor dem Benutzer in dessen Sichtfeld befindet, da es sich bei der Software um ein Remote-User-Interface zur Robotersteuerung handelt.

2.2.1 Verbindung herstellen

Der Benutzer startet die Anwendung und möchte sich mit einem Server verbinden, und wissen ob er sich erfolgreich verbunden hat. Diese Ansicht ermöglicht das Verbinden mit einer Shadow-Hand und das Prüfen der aktuellen Verbindung. Dazu muss man Servername und Serverport eingeben und Verbindung mit dem Handserver herstellen lassen. Die letzten Einträge der Serverliste werden gespeichert und verfügbar dargestellt.

2.2.2 Vordefinierte Posen einnehmen

Ermöglicht das Einnehmen von Posen die bereits gespeichert sind. Es gibt eine Auswahl von gespeicherten Posen die auswählbar sind. Zusätzlich gibt es eine Möglichkeit einzelne oder alle Fingergelenke auf einen 90° oder 0°-Winkel zu setzen, oder die Hand graduell zu öffnen.

2.2.3 Roboter direkt steuern

Ein Benutzer steuert den Roboter direkt über Eingabefelder. Es können die relativen Gelenkwinkel der verschiedenen Finger direkt gesteuert werden. Es sollen zu jeder Zeit die absoluten Gelenkwinkel einzelner Gelenke sichtbar sein. Es soll eine Warnung beim Erreichen des Maximum- oder Minimumwerts eines Gelenks abgegeben werden um den Benutzer auf eine Grenzposition hinzuweisen.

2.2.4 Griffe aus einer Datenbank abspielen

Der Benutzer kann einen Grifftyp und einen Objekttyp auswählen und erhält die passenden Testversuche angezeigt und kann sich einen. Es können Griffe abgespielt werden die als Daten aus Testversuchen vorliegen.

Kapitel 3

Planung und Konzeption des User-Interface

Sich abwendend von mauseingabe-basierten Programmen, vollzieht sich ein Paradigmenwechsel hin zur Gestensteuerung als Form der Eingabe. Die Gestensteuerung des iPads bietet den Vorteil, dass sie bedeutend weniger visuelle Aufmerksamkeit erfordert [Lum03] und dadurch eine Hürde beseitigt, die dem Benutzer das Interagieren mit der Anwendung erschwert. Zusätzlich ist die Bereitschaft höher eine Geste auf einem Touchscreen auszuführen als eine Mausbewegung, da das Zielen mit der Maus auf ein Objekt und dem darauf folgendem Klick mehr Präzision und Aufmerksamkeit erfordert [Bre]. Auch die Verringerung der Größe der Plattform muss beachtet werden. Die höhere Mobilität der Geräte schränkt den Bildschirmplatz erheblich ein und muss auch beim Entwickeln der Oberfläche in Betracht gezogen werden. [Bre02]

3.1 Multitouch-Architektur

Die Verwendung von unseren Gesten wird mit einer Multitouch-Software-Architektur [EKM] umgesetzt, die die rohen Daten interpretiert und zu Events verarbeitet, die dann ausgewertet werden. Bei uns ist diese Architektur das iOS-Betriebssystem welches in einem Kernelement, der UIView als Teil des UIKits [Inc12b] das Handling von Gesten unterstützt.

Innerhalb der Anwendung, kann man in der Mischpult-View durch die Daumensteuerung, die Fingersteuerung und die Fingersteuerung mit Feinjustierung mit Slidern wechseln, indem man in der MischpultAnsicht am oberen Bildrand einen Button antippt, dies wird in der Beschreibung der Planung des Mischpult-Fensters weiter erläutert.

3.2 Verwendete Gesten

Wir benutzen zur Steuerung der Winkel der Finger nur Gesten die zur feinen Präzision geeignet sind "Fine-Action-Group-Gestures" [VSMS]. Das sind Ein-Finger- und Zwei-Finger-Schiebebewegungen mit konstantem Fingerabstand in eine beliebige Richtung oder eine Drehbewegung zweier Finger mit konstantem

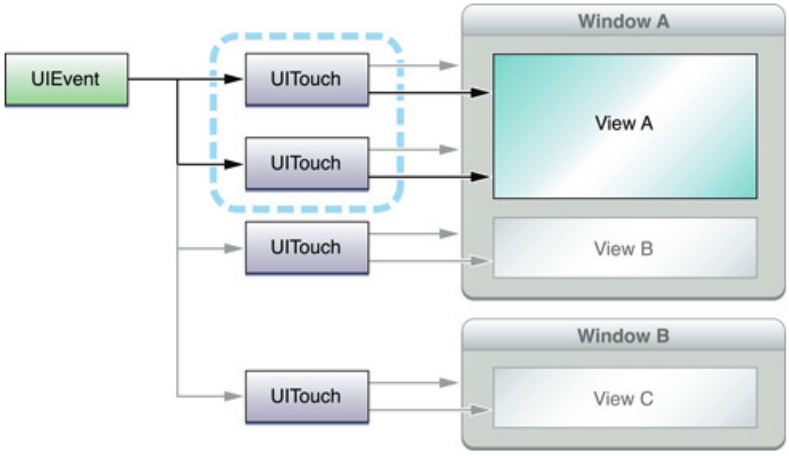

Abbildung 3.1: Schema der Multitouch-Architektur im verwendeten System [Inc12a]

Abstand. In der Daumen-Steuerung werden auch horizontale drei-Finger-Gesten verwendet um die Rotation des Gelenks J5 zu realisieren.

3.3 Design des Haupt-User-Interface

Aufgrund der verschiedenen Abläufe die keine gemeinsamen Aufgaben enthalten bietet sich keine hierarchische Struktur innerhalb der Anwendung am, sondern eine parallele Aufteilung, wie sie auch in der oben abgebildeten Shadow-Hand-GUI eine Verwendung findet. Bei iOS Anwendungen hat man die Wahl zwischen vielen Elementen für diverse Anwendungszwecke, allerdings wird in den iOS Human Interface Guidelines explizit für unsere flache Hierarchie eine Tabbar empfohlen.

“In general, use a tab bar to organize information at the app level. A tab bar is well-suited for use in the main app view because it’s a good way to flatten your information hierarchy and provide access to several peer information categories or modes at one time.“ [Inc12a] S.61

Da diese parallele Aufteilung im Rahmen von iOS Anwendungen durch eine ”tabbed application” realisiert werden, habe ich diese Option gewählt. Eine ”tabbed application” ermöglicht es jederzeit von überall in der Anwendung zu jedem anderen Prozess zu wechseln, was für die unterschiedlichen Steuerungen eine Roboterhand sehr vorteilhaft ist. Durch Antippen eines Feldes in der Tabbar unten am Bildschirmrand gelangt man zu einem der vier Benutzungsszenarien. Im folgenden wird diskutiert welche Elemente in der Anwendung eingesetzt werden um die Funktionalität der einzelnen Szenarien zu gewährleisten.

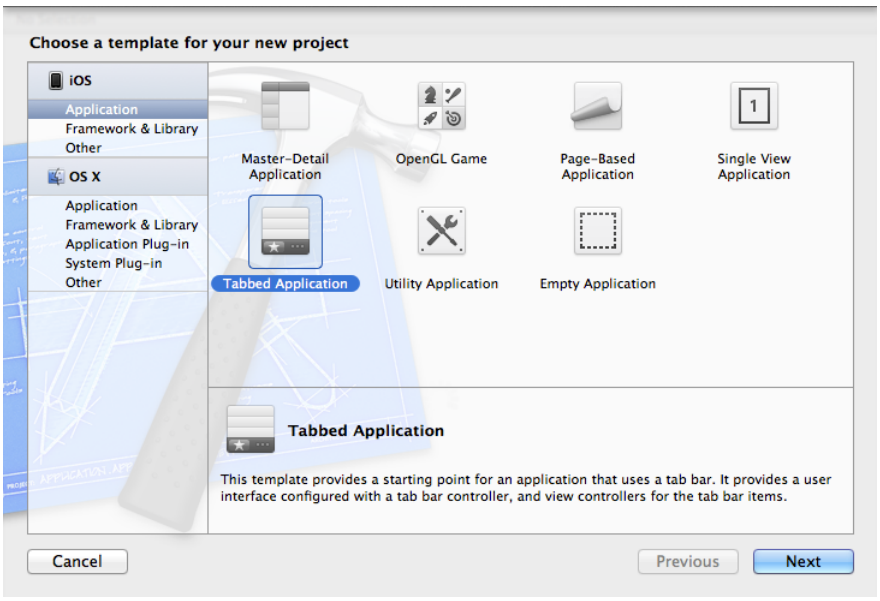

Abbildung 3.2: XCode – Templateauswahl

Abbildung 3.3: Beispiel Tabbar [Inc12a] (Usage of UI-Elements)

3.4 Design der einzelnen Anwendungsabschnitte

Für die fünf verschiedenen Szenarien wird jeweils eine View erstellt, welche über die Tabbar zu erreichen ist. Aufgrund der abweichenden Aufgaben ist es notwendig für jede View eigene zweckmäßige Elemente zu diskutieren, auszuwählen und gegebenfalls zu implementieren. Wobei die Standard-Elemente der Objekt-Bibliothek von iOS genutzt werden und die Empfehlungen des Human-Interface-Guides [Inc12a] in Betracht gezogen werden, wenn es um die Benutzung von Standardelementen geht.

3.4.1 Server

Die Serverliste ermöglicht durch Antippen des Verbinden-Buttons eine Verbindung zu einem Server aufzubauen. Eine aufgebaute Verbindung wird als ”Verbunden” in grün dargestellt. Die Server werden automatisch in einer Liste gespeichert. Durch Antippen eines Servers in der Liste kann man sich zu ihm

verbinden und durch eine Wischgeste von rechts nach links kann man einen Eintrag aus der Liste entfernen.

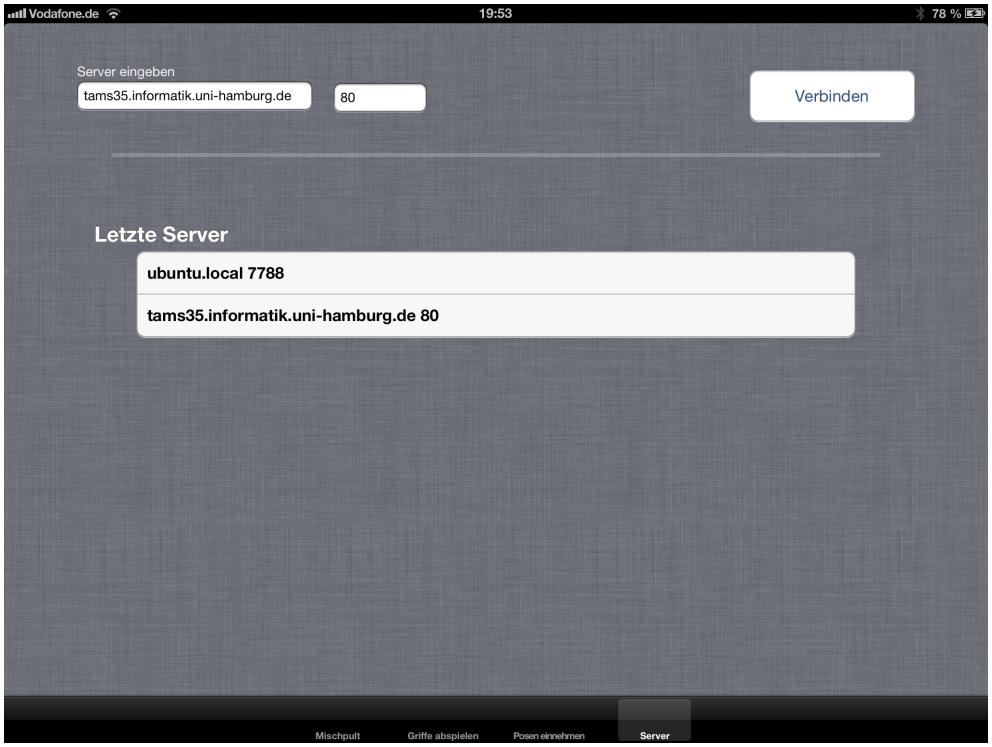

Abbildung 3.4: Posenauswahl

3.4.2 Griffauswahl-Ansicht

Die View dient zur Auswahl eines Griffes. Um einen Griff auszuwählen muss man ein Objekt und eine Griffklasse auswählen und sich dann zwischen den möglichen Alternativen entscheiden, die Auswahl findet in Tabellen statt. Die Kombinationen werden aus drei Dateien geladen und ein Vorschaubild für den Griff wird in einer ImageView angezeigt.

3.4.3 Posen einnehmen

In dem Posenfenster gibt es eine Auswahl der Handpositionen. Da es keine Kategorien gibt und nur eine begrenzte Anzahl von Gesten die eingenommen werden können, ist das Layout der View sehr minimal gehalten. Für jede Pose gibt es einen Button der gedrückt werden muss damit die Hand ihn ausführt. Für Buttons gilt das sie ab 40 Pixel in jeder Dimension als Touchfläche für den Benutzer eingesetzt werden sollen. [Inc12a]. Die Buttons sind in Form einer 4×3-Matrix angeordnet und haben eine leicht treffbare Größe von ca. 100×100 Pixeln.

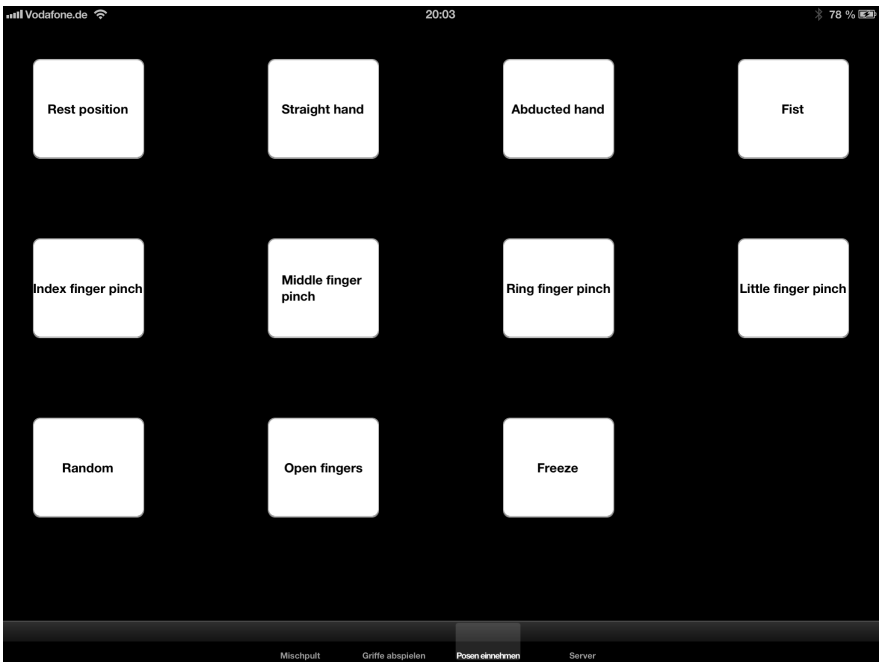

Abbildung 3.5: Serverliste

3.4.4 Mischpult-View

Das Mischpult setzt sich im Verwendungszweck deutlich von den anderen Szenarien ab, denn es wird die Roboterhand mit ihren 20 Freiheitsgraden in Echtzeit gesteuert. Die Roboterhand soll nicht mit Standard-Elementen bedient werden, da dies schon in der Desktopanwendung erprobt wurde und nicht ideal ist. Das Design und Verhalten der Bedienoberfläche wird in eigens implementierten Klassen umgesetzt. Die Bedienung der einzelnen Finger wird in den folgenden Abschnitten diskutiert und geplant. Das Mischpult, als in sich gekapseltes Szenario, hat drei verschiedene Kategorien. Dabei handelt es sich um das relative Steuern das Daumens in Verbindung mit der relativen Steuerung von Zeigefinger/Mittelfinger, das relative Steuern aller vier Finger und das Steuern per Präzisionsregler von Daumen/Zeigefinger/Mittelfinger/Ringfinger und kleiner Finger. Die dritte View für das Steuern der Gelenkwinkel per Slider. Diese drei Kategorien des Mischpults sind als Zellen am oberen Bildschirmrand wählbar und können. Da der Platz auf dem mobilen Gerät begrenzt ist werden die Winkel der aktuellen Ziehbewegung in der View die die Gesten erkennt auch gleichzeitig auf derselben Fläche dargestellt.

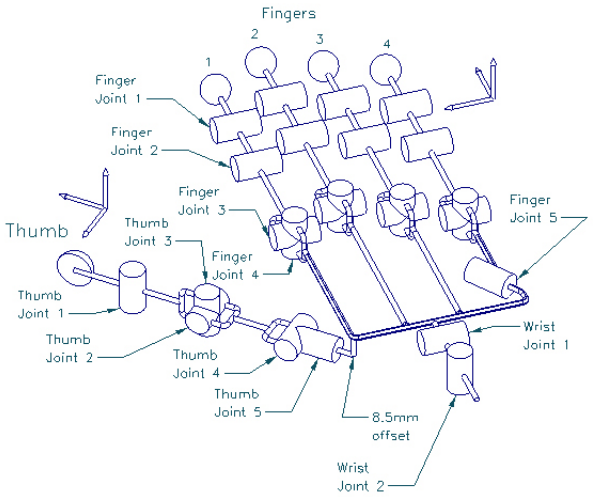

Abbildung 3.6: Schematic: Shadow C6 Hand [Com08]

3.5 Design der Bedienung

3.5.1 Verwendung von anderen Elementen

Damit der Benutzer eine Rückmeldung beim Erreichen von maximalen oder minimalen Winkeln erhält, würden wir idealerweise ein haptisches Feedback [LMBS07] verwenden, welches als Unterstützung zur non-visuellen Gestensteuerung eine non-visuelle Warnung ausgibt, dies ist aber nicht möglich, da ein iPad keine Vibration hat. Deshalb bedienen wir uns der akustischen Warnung mit einem Hinweiston und dem Einfärben der Winkel der entsprechenden View in roter Signalfarbe.

3.5.2 Steuerung eines Fingers

Die Steuerung eines Fingers findet auf einer Fläche auf dem Bildschirm statt, da dies die einzige Möglichkeit ist mit der Anwendung zu interagieren. Ein einzelner Finger hat nur drei Gelenke und braucht deshalb nur den Platz für eine begrenzte Anzahl von Bediengesten. Die drei mittleren Finger der Shadow Dextrous Hand haben vier Gelenke und haben dasselbe Eingabefeld. Die View für die Steuerung eines Fingers erstreckt sich über den ganzen vertikal verfügbaren Bildschirmplatz, welcher 600 Pixel beträgt. Das erste und zweite Gelenk in dem Schema "'Shadow C6 Hand" bezeichnet als "joint 1", "joint 2" agieren zusammen. Man steuert nur das zweite Gelenk und das erste Gelenk nimmt automatisch den Winkel des Zweiten ein, was als Nachbildung einer menschlichen Hand durchaus realistisch ist, wie man ganz einfach prüfen kann, wenn man zum Beispiel den Knöchel der mittleren Fingers in eine 90°-Position bringt. Im Abschnitt über die verwendeten Gesten wurde schon einmal darauf verwiesen dass es eine Kategorie der "Fine-Action-Group-Gestures" [VSMS] gibt, die für präzise Eingaben empfohlen werden. Ein Finger wird jeweils in einer View mit drei Gesten gesteuert, eine Geste mit einer Berührung des Bildschirms und zwei

Gesten mit zwei Berührungen. Die Minimal-und Maximalwinkel der drei mittleren Finger der Hand sind ebenfalls identisch und haben als Spanne: joint 1, joint 2, joint 3 =90° Winkel und joint 4= 50° Winkel. Da nur joint 2, joint 3 und joint 4 gesteuert werden müssen haben wir zwei identische Winkel die eine 90° Spanne haben, die auch gleich skaliert sein müssen. Da ein Nutzer ein Schiebegeste von sich weg vollführt wenn er sein zweites/drittes Gelenk des Zeigefinger streckt, während der Streckung des Fingers bringt er den Winkel des Gelenks in unserem Model auf 0°, ist in unserer Implementation der Steuerung des Fingers das Hochschieben eine Geste zur Streckung der Roboterhand. Da die beiden Gelenke so ähnliche Funktionen erfüllen sind die Gesten zur Steuerung auch fast identisch, mit einem Finger schiebt man Gelenk 2 und das daran gekoppelte Gelenk 1 und wenn man dieselbe Schiebegeste mit zwei Fingern auf dem Bildschirm ausführt, dann bearbeitet man das Gelenk. Da beide Gelenke die gleiche Winkelspanne haben ist auch die Übersetzung von geschobenen Pixeln auf dem Bildschirm zum veränderten Winkel an der Hand identisch. Als Übersetzung hat sich 20 Pixel für einen Grad als günstig in Hinblick auf Feinheit und Platzbedarf erwiesen. Am oberen Steuerungsfeld eines Fingers sind die aktuellen Winkel des Gelenks zur schnellen Prüfung angezeigt. Beim erreichten minimalen oder maximalen Winkel wird die Anzeige rot und es ertönt ein Warngeräusch.

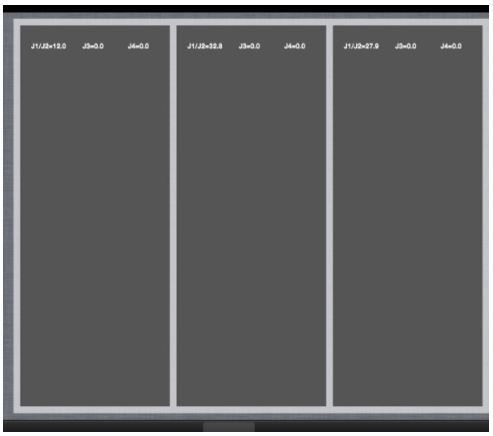

Abbildung 3.7: Bedienfeld für die Steuerung von Zeigefinger, Mittelfinger und Ringfinger

3.5.3 Bedienungsgesten

Die konkreten Gesten zu Steuerung von Zeigefinger, Mittelfinger und Ringfinger sind hier noch einmal als Schema dargestellt. Jeder Pfeil entspricht einem Finger eines Benutzers, der eine Bildschirmberührung ausführt. Die folgenden Grafiken geben eine visuelle Beschreibung der Bediengesten für Zeigefinger Gelenk 2 (45°), Zeigefinger Gelenk 3 (45°) und Zeigefinger Gelenk 4 (0°) .

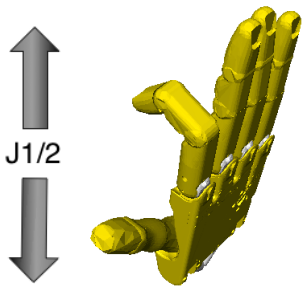

Abbildung 3.8: Bediengeste für das Gelenk 1 und Auswirkung

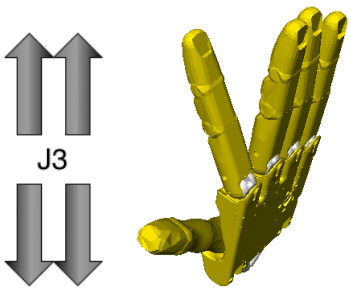

Abbildung 3.9: Bediengeste für das Gelenk 3 und Auswirkung

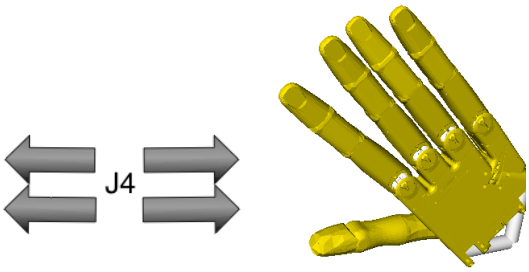

Abbildung 3.10: Bediengeste für das Gelenk 4 und Auswirkung

3.5.4 Steuerung des kleinen Fingers

Die Steuerung des kleinen Fingers der Shadow-Hand unterscheidet sich zu der der drei mittleren Finger darin das zusätzlich das fünfte Gelenk in dem Handschema ”joint 5” vorhanden ist. Die mögliche Winkeländerung beträgt 40° und das Gelenk dreht den kleinen Finger zur Handinnenfläche herum. Es würde sich zum Eindrehen des kleinen Fingers eine Rotationsbewegung anbieten, eine Rotationsbewegung erfordert aber immer zwei Finger. Die Rotationsbewegung mit zwei Fingern ist aber aus praktischen Gründen nicht möglich, da der Benutzer sie nicht korrekt ausführen kann, sodass oft die seitliche Schiebebewegung die das vierte Gelenk steuert unabsichtlich betätigt wird. Als Lösung, die die

Konsistenz mit der Bedienung mit den anderen Fingern gewährleistet, lässt sich der kleine Finger steuern wie die anderen drei Finger und hat eine zusätzliche hellgraue Fläche zur horizontalen Schiebebewegung um das fünfte Gelenk zu drehen. Die Steuerung eines Fingers wird als um das Gelenk 5 erweitert. Jeder Pfeil beschreibt eine Berührung des Bildschirms durch Benutzer.

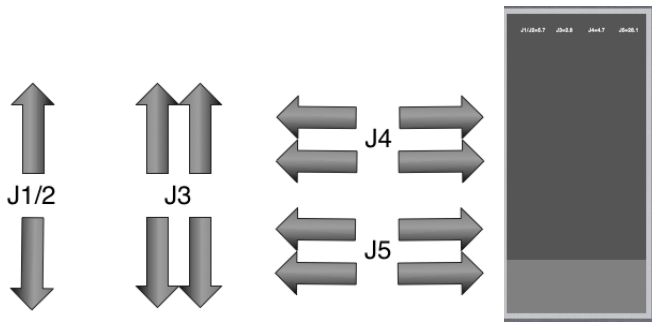

Abbildung 3.11: Gesten und Eingabefeld für den kleinen Finger

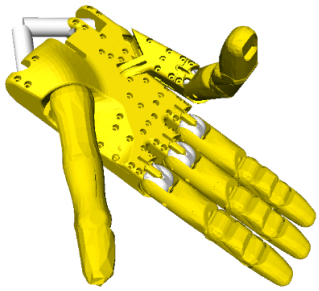

Abbildung 3.12: Kleiner Finger im GraspIt!-Simulator

3.5.5 Steuerung des Daumens

Die Daumensteuerung ist komplexer, da der Daumen fünf Gelenke also fünf Freiheitsgrade aufweist. Die Daumensteuerung benötigt mehr Bildschirmraum als eine normale Fingersteuerung, sie nimmt mindestens das halbe Fenster ein. Rechts neben der Daumensteuerung sind zusätzliche Bedienfelder für Zeigefinger und Mittelfinger untergebracht. Als zusätzliche Steuermöglichkeit sollen sie es dem Benutzer ersparen zwischen den Views wechseln zu müssen, da viele Griffe sowohl Daumen als auch andere Finger benötigen. Zur konkreten Steuerung des Daumens: Die Anwinkelung von Gelenk 1 und Gelenk 2 ist wie bei der Steuerung der mittleren drei Finger der Hand, wie sie oben bereits erläutert wurde. Das Anwinkeln erfolgt durch eine vertikale Schiebebewegung eines Fingers für Gelenk 1 und eine Schiebe-Geste mit zwei Fingern für Gelenk 2. Mit einem Finger und der horizontalen Geste bewegt man das Gelenk 3 welches die obere Hälfte des Daumen seitlich um jeweils 15° knickt. Es folgen Gelenk 4 und

5, die bei der intuitiven Bedienung für den Benutzer ein Problem darstellen da sie die Position des gesamten Daumen sehr verändern, es hat sich herausgestellt dass es sich zur Bedienung des Daumens als sehr vorteilhaft erweist, wenn die Gelenke in der umgedrehten Reihenfolge ihrer Nummerierung gestellt werden. Also zuerst die horizontalen Gesten mit drei und zwei Fingern um den Daumen grob zu positionieren, dann die vertikalen Gesten mit zwei und dann einem Finger und als letztes die seitliche Verschiebung mit Gelenk 3, indem man mit einem Finger eine horizontale Geste ausführt. Im Folgenden sind alle Gesten mit Wirkung gezeigt sowie eine Darstellung der Bedienoberfläche für den Daumen mit den beiden zusätzlichen Fingerbedienfeldern.

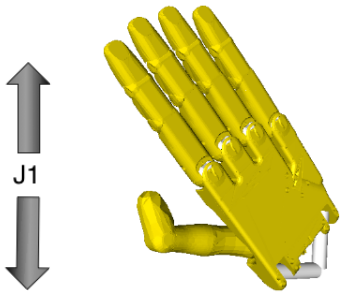

Abbildung 3.13: Bediengeste für das Gelenk 1 und Effekt im Simulator

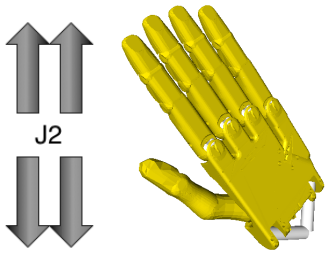

Abbildung 3.14: Bediengeste für das Gelenk 2 und Effekt im Simulator

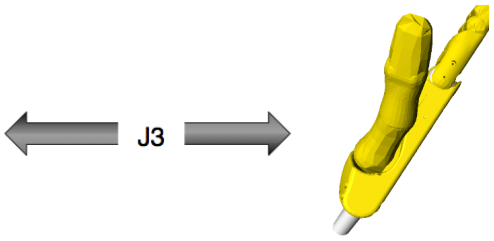

Abbildung 3.15: Bediengeste für das Gelenk 3 und Effekt im Simulator

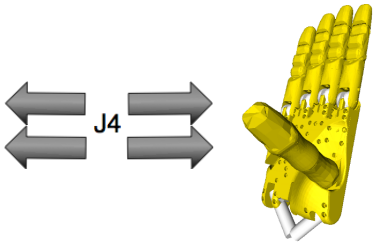

Abbildung 3.16: Bediengeste für das Gelenk 4 und Effekt im Simulator

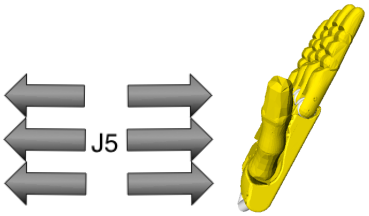

Abbildung 3.17: Bediengeste für das Gelenk 5 und Effekt im Simulator

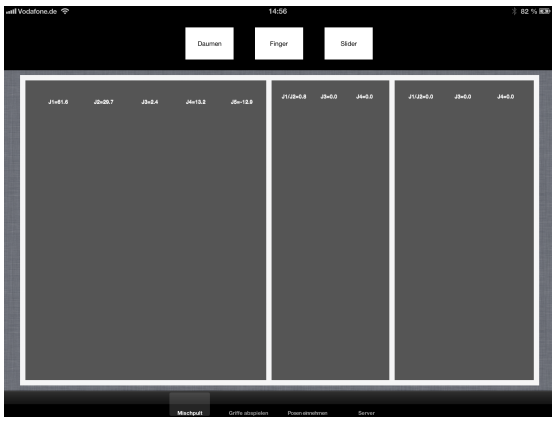

Abbildung 3.18: Bedienoberfläche des Daumens, mit zusätzlicher View für Zeigefinger und Mittelfinger

Kapitel 4

Architektur der Software

Die Architektur ist nach dem Model der ”Model-View-Controller”-Architektur [Vli94a] entworfen. Die MVC-Architektur trennt drei Aspekte einer Anwendung: Das Model, also den aktuellen Zustand der Anwendung, die View (Oberfläche) und die Controller, welche die Interaktion mit den Views vorschreiben. Die Anwendung umfasst voneinander abgekoppelte Views, die die Oberfläche für die Benutzungsszenarien enthalten. Im iOS Betriebssystem ist das MVC-Model als Weg zur Applikationsentwicklung vorgesehen und wird dementsprechend durch zahlreiche Code-Templates und Sprachfeatures unterstützt. Es gibt eine UIView als Pendant zur ”View” aus dem MVC-Model. Eine UIView ist immer auch ein UIResponder-Objekt, das auf andere Events reagiert und sie behandelt. Die implementierten UIView-Unterklassen die in der Anwendung verwendet werden sind ebenfalls architektonisch gesehen Teil der View-Perspektive, und verhalten sich wie es im Model vorgesehen ist. Die Views stellen die Oberfläche dar, nehmen die Eingaben der Benutzer an, verarbeiten Eingaben und senden Daten an das Model. Das Model ist die Abstraktion von der Gesamtheit der konkreten Objekte der Anwendung, und stellt hier die Instanz der Anwendung dar. Das Model ist auch für die Datenhaltung zuständig, beispielsweise die Speicherung von XML-Dateien. In der Shadow-App werden Daten in XML-Dateien gespeichert und Daten über die TCP-Verbindung abgerufen. Die Controller sind Klassen die von UIViewController erben, sie werden wie die Views nach den den Anforderungen entworfen. Die ViewController sind für die Steuerung ihrer konkreten View berechtigt, und sorgen dafür dass die Views dynamisch geändert werden können oder sich aufgrund von geänderten Daten ändern.

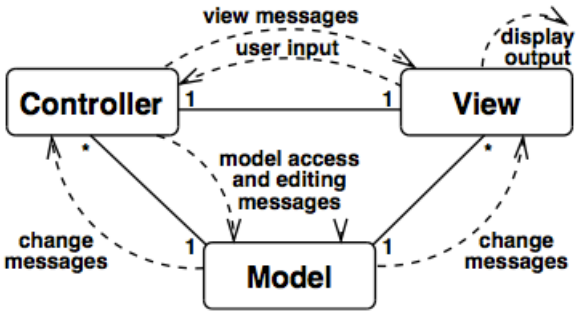

Abbildung 4.1: ”Model-View-Controller”-Architektur [VH03]

4.1 Modellierung - Klassendiagramm

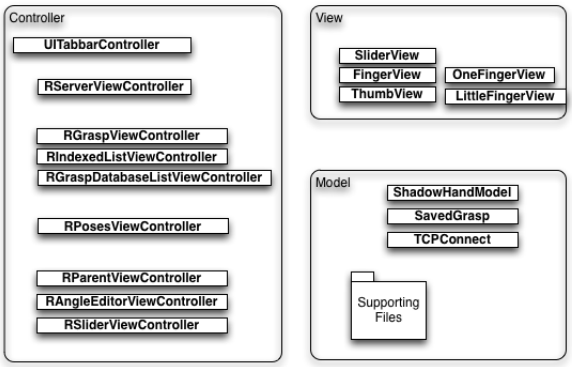

Abbildung 4.2: Zuordnung der Klassen nach dem MVC-Modell

4.2 Kommunikation mit der Hand

Die Anwendung kommuniziert mit der Hand über Netzwerk indem es ein TCP-Protokoll verwendet. Die TCPConnect-Klasse der Anwendung verwendet ein Framework für einen fehlerfreien asynchronen Datenverkehr, weitere Details in Kapitel Vier. Das benutzte TCP-Protokoll wird vom ”graspit-proxy-server.jar” implementiert und dieser ist mit dem Port 7788 anzusprechen. Es kann immer nur eine TCP-Verbindung zu einem Port aufgebaut werden, deswegen muss die Anwendung als einziges mit dem Port verbunden sein. Die Besonderheit bei dieser Form der Kommunikation in dieser Anwendung, besteht darin, dass sich Sprünge bei der Hand entwickeln können, wenn ein TCP-Paket sehr viele Befehle zur Winkeländerung eines Gelenks enthält. Es gehen dann die Anfragen zwischen der ersten und der letzten Winkelposition verloren, wenn der Simulator mit Kollisionserkennung arbeitet. Dieses Problem wird in Implementierung der

Anwendung weiter erläutert. Ebenfalls ist die Kommunikation über das TCP-Protokoll nicht in der Lage zwischen den kontinuierlichen Befehlen zur Winkeländerung eines Gelenks durch den Anwender zusätzlich häufig den Status der Hand abzufragen, wenn die Kollisionserkennung nicht abgeschaltet wird.

4.3 Datenstruktur

Es gibt eine datei-basierte Datenspeicherung innerhalb der Anwendung, aber da dieses Werkzeug eine steuernde Funktion ausübt liegt die Priorität eher in der Kommunikation als in der Datenhaltung. Die konkrete interne Datenstruktur ist wandelbar, denn beim Initialisieren der Verbindung wird eine Abfrage mit "get-joint-angles"gesendet, die Hand liefert ein Array von Strings der Gelenke der aktuellen Roboterhand zurück. Anhand dieser Antwort legt das ShadowHand-Model eine Gelenkanzahl und Gelenkbenamung fest. Das ShadowHandModel ist als ein Singleton implementiert, sodass es von dieser Klasse nur eine Instanz geben kann [Vli94b]. In der Posenauswahl werden Daten aus einer xml-Datei gelesen und zur Erzeugung der Buttons verwendet und die Griffauswahl bedient sich einer Bibliothek von Bildern und Griffen die als plists gespeichert sind.

Kapitel 5

Implementierung

Konkret wird ein Client-/Server-basiertes Werkzeug entworfen, und auf der mobilen Plattform iOS 6/5 für einen Tablet-PC implementiert. Die Software wird als Clientanwendung über eine TCP-Verbindung für den Proxy-Handserver in Verbindung mit dem Simulator GraspIt oder dem Roboter Shadow C6 Hand konzipiert. Das Design wie es bereits beschrieben wurde lässt sich im Interface-Builder noch einmal gut betrachten. Der Interface-Builder ist ein Tool innerhalb der Entwicklungsumgebung XCode, in dem Objekte wie zum Beispiel TableViewController, ImageViews, Textfields etc. in das Anwendungsfenster platziert werden können, wenn sie keine zusätzliche Funktion erfüllen sollen. In der Abbildung ist die grafische Repräsentation innerhalb des MainStoryboard.storyboard zu sehen, mithilfe von Relationships wird in unserer HauptView dem Tabbar-Controller zwischen den vier Unter-Haupt-Views innerhalb der App gewechselt. Denn der TabbarController, kann die Views anhand ihrer Anmeldung im Interfacebuilder erkennen.

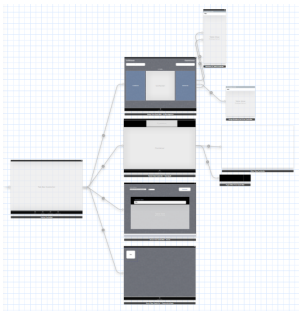

Abbildung 5.1: Interfacebuilder-Ausschnitt

5.1 Einschränkungen der Entwicklungsumgebung

Die Shadow-App ist ein Anwendung für das Betriebssystem iOS 6, und somit an die iPads der Firma Apple Inc. gebunden, welche den "Hardware & Software Agreements" [iSLA12a] unterliegen. Die Wahl das gewünschten Betriebssystems

ist innerhalb der dedizierten Entwicklungsumgebung XCode [iSLA12b] unter dem Punkt Project - Deployment Target wählbar.

Das iPad als Hardware begrenzt die konkrete Eingabefläche und den Ausgabeplatz auf eine Größe von 9.7 in (250 mm) und die Pixel bei der ersten und zweiten Generation auf 1024×768 Pixel, iPads der dritten und vierten Generation haben 2048×1536 Pixel. Da das iPad zumindest bis zur aktuellen vierten Generation keine Vibrationsaussender enthält, kann als nonvisuelles Feedback nur ein akustisches Signal verwendet werden.

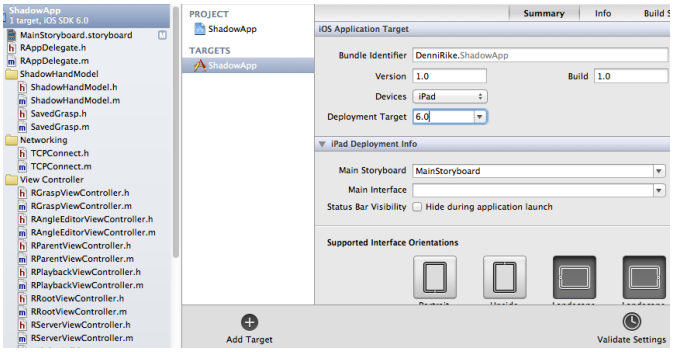

Abbildung 5.2: Auswahl des Deployment-Targets in den Projekteinstellungen

5.2 Testumgebung

Der Benutzer der Anwendung befindet sich mit einem iPad im Netzwerk und muss berechtigt sein eine TCP-Verbindung zum Port 7788 aufzubauen, was durch das VPN der Universität Hamburg möglich ist, wenn der VPN-Benutzer die Rechte hat auf diesen Port zuzugreifen. Zum Testen der Anwendung stehen zwei Alternativen zur Verfügung. Die Simulation der Roboterhand oder der Einsatz einer Shadow-Hand [Com08], welche zum Beispiel im Fachbereich TAMS der Informatik der Universität Hamburg vorhanden ist.

5.2.1 Simulator - GraspIt!

Als Simulator dient "GraspIt!" [CSD12] in der Version 2.2. entwickelt von dem Computer Science Department der Columbia University. Der Simulator ist unter der GNU General Public License [FSF12] veröffentlicht worden und ist unter den Betriebssystemen Windows und Ubuntu Linux installierbar. Im GraspIt!-Simulator muss ein ShadowHand.xml-File als Robotermodel geladen werden, welches als Server dient. Der Simulator hat eine Einstellung Kollisionserkennung die ggf. deaktiviert werden um die Responsivität zu erhöhen. Zusätzlich wird ein graspit-proxy-server.jar von Dr. Norman Hendrich ,Fachbereich Informatik Universität Hamburg, gestartet. Dieser Proxy-Server realisiert das beschriebene Protokoll zur TCP-Kommunikation mit dem Simulator.

5.3 Implementierung des Mischpult und seiner Komponenten

Die MischpultView sie wird von einem `ParentViewController` dargestellt. Dieser ViewController beinhaltet wiederum zwei ViewController, einen für die Auswahl der aktuellen Steuerungsansicht den RAngleViewController und einen für die Anzeige der Steuerung RSliderViewController. Im `RAngleViewController` einer Subklasse vom UICollectionViewController wird festgelegt welche View beim Antippen eines Buttons geladen wird. Das Laden passiert indem dem SliderViewController eine View zur Anzeige übergeben wird, die zuvor vom AngleViewController erstellt wurde. Entweder es wird eine `ThumbView`, `FingerView` oder eine `SliderView` erstellt. Um eine zusätzliche Unterview für das Mischpult einzufügen muss in der RSliderViewController-Klasse eine weitere View erstellt werden. Innerhalb der verschiedenen Bedienfenster werden immer korrekte Winkel von dem einen ShadowHandModel geholt, wie es in der MVC-Architektur vorgesehen ist.

```
-(void)reloadWithFinger:(NSString*)finger
{

    [self.theSubViewSlider removeFromSuperview];
    [self.theSubViewFinger removeFromSuperview];
    [self.theSubViewThumb removeFromSuperview];

    if ([finger isEqual:@"Daumen"]) {
        ThumbView* tview = [[ThumbView alloc] init];
        self.theSubViewThumb = tview;
        NSLog(@"%@ selected", finger);
        [self.view addSubview:tview];
    }
    else if ([finger isEqual:@"Finger"]){

        FingerView* wview =[[FingerView alloc] init];
        self.theSubViewFinger = wview;
        NSLog(@"%@ selected", finger);
        [self.view addSubview:wview];
    }
    else if ([finger isEqual:@"Slider"]){

        SliderView* sview = [[SliderView alloc] init];
        self.theSubViewSlider = sview;
        NSLog(@"%@ selected", finger);
        [self.view addSubview:sview];
    }
}
```

Abbildung 5.3: Erstellung der Mischpult-Unterviews im RSliderViewController in der ”reloadWithFinger:”-Methode

5.3.1 FingerView

Die FingerView erstellt drei Instanzen von der `OneFingerView` und eine von der `LittleFingerView`. Das Layout der vier Views zur Darstellung der Finger ist in der FingerView in der initWithFrame:-Methode festgelegt. Es sind einfache

UIViews mit grauem Hintergrund. Die drei OneFingerViews werden mit einer property ”fingername” erstellt und bekommen jeweils eine InfoView am oberen Rand mit den aktuellen Winkeln die von der ShadowHandModel-Klasse mit Informationen gefüllt wird. Sowohl der OneFingerView als auch der LittleFingerView wird ein UIPanGestureRecognizer zugewiesen. Der GestureRecognizer kann Events nach bestimmten Gesichtspunkten unterscheiden, ein PanGestureRecognizer reagiert auf Ziehbewegungen und unterteilt sie je nach Anzahl der verwendeten Finger. Die Auswertung der Events findet während der Ausführung der Gesten statt, damit keine ”Überflutung” der TCP-Verbindung stattfindet, sendet die FingerView nur dann neue Winkel an die ShadowHandModel-Klasse wenn eine Ziehbewegung 10 Pixel lang in eine beliebige Richtung stattgefunden hat. Dieser Wert ist in der Methode panned: zu finden und in der Abbildung dargestellt.

```
/*
 checks if the touch-event has changed its position for a given value
 */
int change = 10;

if (  (abs(lastpointx - point1.x) > change) ) {
    check2 = YES;
    lastpointx = point1.x;
    NSLog(@"YES");

}
if (  (abs(lastpointy - point1.y) > change) ) {
    check3 = YES;
    lastpointy = point1.y;
    NSLog(@"YES");

}

if ((check2 || check3)) {
    NSString* text = [[NSString alloc] initWithFormat:@"%@ %f %@ %f %@ %f", joint2, self.
        angle2, joint3, self.angle3, joint4, self.angle4];
    [self.hand setJointsWithStringHand:text];
    lastpointx = point1.x;
    lastpointy = point1.y;
}
[self.labeljoint1 setText: [[NSString alloc] initWithFormat:@"J1/J2=%.1f", self.angle2]];
[self.labeljoint2 setText: [[NSString alloc] initWithFormat:@"J3=%.1f", self.angle3]];
[self.labeljoint3 setText: [[NSString alloc] initWithFormat:@"J4=%.1f", self.angle4]];
check2 = NO;
check3 = NO;
```

Abbildung 5.4: Abschnitt der `panned:`-Methode der die Gesten erkennt und den Winkeln zuordnet

Wenn ein Gelenk seine Minimum- oder Maxiumwinkel erreicht hat, dann beendet die FingerView das Senden neuer set-joint-angle-Aufträge und stellt die Winkelwerte in der InfoView mit rotem Text dar. Die konkrete Übersetzung von Bildschirmpixel zu Winkeländerung befindet sich ebenfalls in der OneFingerView in der -(void)panned:(id)sender-Methode, siehe Abbildung. Die Winkeländerung wird berechnet aus dem Ergebnis der `[self.panRecognizer translationInView:self]`-Methode, sie gibt eine relativen Punkt zum Anfang der Bewegung aus. Der neue Winkel self.angle= self.angle + ((point1xy)/200) wird dann aus dem alten Winkel ebenfalls in der ”panned:”-Methode berechnet.

Aus diesem x-Koordinaten- oder y-Koordinatenwert wird dann der Wert berechnet der auf den neuen Winkel addiert werden muss. Die SubFingerView

```
if ((((UIGestureRecognizer*)sender).state == UIGestureRecognizerStateChanged))
{

    [self.labeljoint1 setTextColor:[UIColor whiteColor]];
    [self.labeljoint2 setTextColor:[UIColor whiteColor]];
    [self.labeljoint3 setTextColor:[UIColor whiteColor]];


    CGPoint point1= [self.panRecognizer translationInView:self];

    NSLog(@"x=%f y=%f", point1.x, point1.y);


    if ([self.panRecognizer numberOfTouches] == 1){

        self.angle2= self.angle2 + ((point1.y)/200);
        if (self.angle2 > [[range2 objectAtIndex:1] doubleValue]) {
            self.angle2 = [[range2 objectAtIndex:1] doubleValue];
            [self.labeljoint1 setTextColor:[UIColor redColor]];
            [self getWarning];

        }
        if (self.angle2 < [[range2 objectAtIndex:0] doubleValue]) {
            self.angle2 = [[range2 objectAtIndex:0] doubleValue];
            [self.labeljoint1 setTextColor:[UIColor redColor]];
            [self getWarning];

        }

    }

    if ([self.panRecognizer numberOfTouches] == 2){

        self.angle3= self.angle3 + ((point1.y)/200);
        if (self.angle3 > [[range3 objectAtIndex:1] doubleValue]) {
            self.angle3 = [[range3 objectAtIndex:1] doubleValue];
            [self.labeljoint2 setTextColor:[UIColor redColor]];
            [self getWarning];

        }
        if (self.angle3 < [[range3 objectAtIndex:0] doubleValue]) {
            self.angle3 = [[range3 objectAtIndex:0] doubleValue];
            [self.labeljoint2 setTextColor:[UIColor redColor]];
            [self getWarning];
        }
```

Abbildung 5.5: ”int change” = Pixel-Intervall zum Aussenden der neuen Winkel an die Hand

hat am unteren Ende eine View die das fünfte Gelenk steuert, da für eine Geste mit drei Fingern in der FingerView kein Platz ist, ist eine Zwei-finger Lösung gewählt.

5.3.2 ThumbView

Die ThumbView erstellt drei verschiedene Views davon ist eine die eigene View die `ThumbView` die den Daumen steuert und zwei davon sind `OneFingerViews` die auch in der FingerView verwendet sind. Die ThumbView ist mit 450 Pixeln bedeutend größer als die OneFingerView mit 220 Pixeln, denn die drei-fingrigen Gesten der DaumenView-Steuerung verlangen mehr Platz um ausgeführt zu werden ohne das die Geste die designierte View verlässt. Auch die ThumbView sendet erst neue TCP-Anfragen wenn eine Geste mindestens 10 Pixel fortbewegt wurde.

5.3.3 SliderView

Die SliderView wird ebenfalls vom SliderViewController geladen und enthält soviele Slider wie es bewegbare Gelenke gibt. Jeder Freiheitsgrad hat einen vertikalen Slider der immer den aktuellen Winkel aus dem ShadowHandModel widergibt. Die Slider sind einfache UISlider-Elemente die gedreht wurden um in eine Reihe auf den Bildschirm zu passen.

5.4 Implementierung der Posenauswahl

Die Posenauswahl ist in der Klasse `RPosesViewController` implementiert, sie ist eine Matrix aus Buttons die auf dem Hauptfenster dargestellt werden und enthält keine Unterklassen. Die Anwendung erhält im AppDelegate, die Daten für die einzelnen Posen aus einer Datei der `predefinedPoses.plist`, die plist ist eine Xml-Datei.

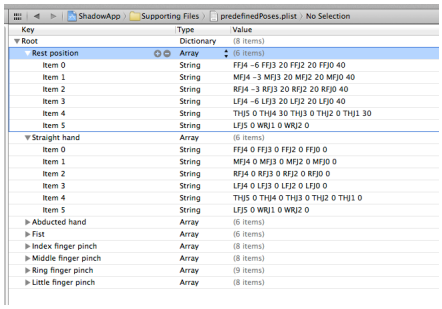

Abbildung 5.6: Ansicht der ”predefinedPoses.plist”

5.5 Implementierung der Griffauswahl

Die Griffauswahl ist als `RGraspViewController` implementiert, dieser enthält zwei UIImageViews zur Darstellung des aktuellen Griffs und des aktuellen Objekts. In der Mitte der GriffauswahlView ist eine TableView die durch den RGraspDatabaseListViewController gesteuert wird. Der RGraspDatabaseListViewController stellt passende Griffe aus der ”graspLibrary.txt” in der Tabelle dar, und zeigt einem die Anzahl der passenden Ergebnisse an. Wenn man ein Ergebnis auswählt wird der Griff als Anfrage an das ShadowHandModel übergeben. Die Auswahlmöglichkeit für die Griffe ist durch zwei Buttons gegeben, welche beim betätigen ein Popover einblenden.

5.6 Implementierung der SliderView

Die SliderView biete das präzise Steuern per Regler von Daumen, Zeigefinger, Mittelfinger, Ringfinger und kleinem Finger an. Die SliderView wird auch vom RSliderViewController in der SliderView dargestellt. Es handelt sind wieder um eine einfache View ohne Unterklassen. Die SliderView hat verfügt immer über aktuelle Gelenkwinkel aus dem ShadowHandModel.

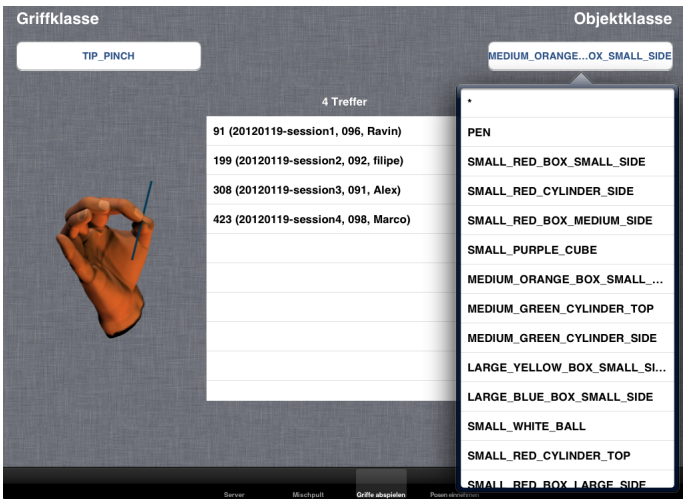

Abbildung 5.7: Griffauswahl-View

5.7 Implementierung der Kommunikation - TCP-Connect

Das Verbindungsmanagement findet in der `TCPConnect`-Klasse statt. Dort wird die Verbindung `+(TCPConnect)connection` als ein Singleton gehalten [Vli94b], sodass zu jeder Zeit nur eine TCP-Verbindung bestehen kann. Der socket der Verbindung ist vom Typ GCDAsyncSocket, aus dem verwendeten Framework "CocoaAsyncSocket" [Han12]. Diese asynchrone Kommunikation läuft auch bei WLAN-Verbindungsproblemen ohne das die Anwendung vergeblich in einen Deadlock läuft, denn die Verbindung ist in einen Hintergrundthread ausgelagert und kann somit auch den Mainthread nicht behindern, der die Oberflächen darstellt. Erst wenn ein Ergebnis aus dem Thread der TCP-Klasse vorliegt wird dieses Ergebnis an den Mainthread zurückgegeben. Die Klasse TCPConnect implementiert einen GCDAsynSocketDelegate und die connection hat einen GCDAsyncSocket als socket. Das Framework bietet die Möglichkeit TCP-Anfragen mit Nummern zu versehen und benachrichtigt einen über den erfolgreichen Erhalt einer Antwort, sodass die TCPConnect-Klasse in der Lage ist eine von der `ShadowHandModel` gekommene get- oder set-Anfrage zuverlässig mit der richtigen Antwort zu bestätigen. Die TCPConnect-Klasse definiert zu diesem Zweck ein `TCPRequestDelegate`-Protokoll, dieser benachrichtigt dann alle Klassen die sich als TCPRequestDelegate angemeldet haben und die Methoden des Protokolls implementieren müssen, hier in diesem Fall ist das nur die ShadowHandModel-Klasse. Die TCPConnect-Klasse implementiert weiterhin eine Delegate-Protokoll TCPConnectionInfoDelegate, welches von der RServerViewController umgesetzt wird und den aktuellen Verbindungsstatus der +(TCPConnect*)connection überwacht, sodass wenn sich die TCP-Verbindung disconnected, der RServerViewController eine Änderung der ServerView veranlasst. Hier sieht man gut den MVC-Ansatz, das Model ändert sich, die TCPConnect-Klasse und die ShadowHandModel-Klasse sind Teil des

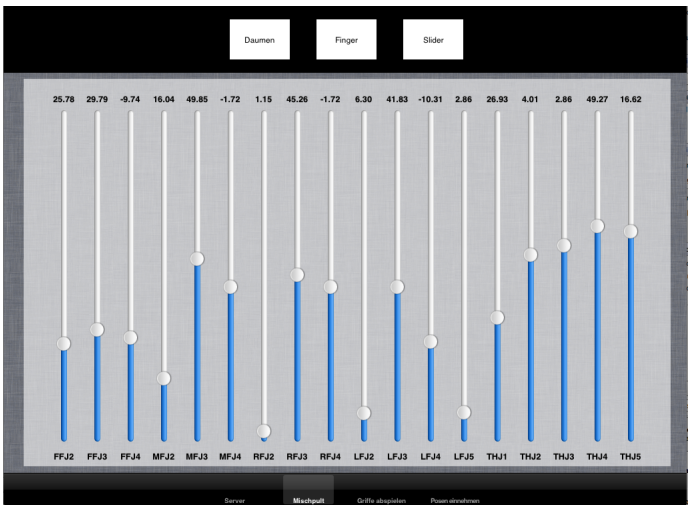

Abbildung 5.8: SliderView

Models, sie reichen die Daten weiter und der ViewController als Controller wird benachrichtigt und veranlasst eine Änderung der ServerView.

```objc
#import <Foundation/Foundation.h>
#import "GCDAsyncSocket.h"

@class RServerViewController;
@protocol TCPConnectionInfoDelegate <NSObject>
- (void)hasConnected;
- (void)hasDisconnected;
@end

@class ShadowHandModel;
@protocol TCPRequestDelegate <NSObject>
- (void)succeeded:(long)tag message:(NSData*)message;
@end


@interface TCPConnect : NSObject <GCDAsyncSocketDelegate> {
}

@property (nonatomic, retain) GCDAsyncSocket *socket;
@property (nonatomic, weak) id<TCPConnectionInfoDelegate> infoDelegate;
@property (nonatomic, weak) id<TCPRequestDelegate> reqDelegate;

-(void) write:(NSString*)string withTag:(long)tag;
-(void) connect:(NSString *)host withPort:(short)port;
-(void) disconnect;


+(TCPConnect*)connection;

@end
```

Abbildung 5.9: Protkolldefinition der Delegates

Kapitel 6

Zusammenfassung und Ausblick

Das implementierte Werkzeug ist in der Lage die Aufgaben als Remote-User-Interface zur Robotersteuerung wahrzunehmen. Der Benutzer kann die Hand direkt mit dem mobilen Endgerät steuern und auch die anderen Aufgaben, wie Griffe abspielen und Posen einnehmen, ausführen. Die Anwendung hat ein alternatives Benutzerinterface das sich von dem eines Desktop-PCs absetzt, und ist so entworfen, dass sich die Bedienung zukünftig noch ändern lässt. Die Steuerung der Hand ist mit der designten Bedienung einfacher als mit der Bedienung durch traditionelle Slider im Desktop-PC-Programm. Das iPad ist für diesen Anwendungszweck geeignet, würde aber von Vibrationsmöglichkeiten profitieren, die 9.7 inch Bildschirmdiagonale bieten ausreichend Platz zur Steuerung von 20 Freiheitsgraden. Das Werkzeug auf dem Tablet-PC ermöglicht eine schnellere Steuerung der Shadow-Hand als es andere Laptops oder Desktop-PCs bieten. Da der Entwicklungsprozess und die Erprobung der Steuerung mit dem GraspIt!-Simulator stattgefunden hat ist ein ausgiebiger Test und der zukünftige Einsatz der Anwendung an der Shadow Dextrous Hand geplant.

Literaturverzeichnis

[Bre] Stephen A Brewster. Multimodal interaction and proactive computing.

[Bre02] Stephen Brewster. Overcoming the lack of screen space on mobile computers. *Personal and Ubiquitous Computing*, 6:188–205, 2002.

[Com08] Shadow Robot Company. Shadow dexterous hand c5 technical specification. Technical report, 2008.

[CSD12] University Columbia Computer Science Department. *GraspIt v2.2*, 2012. Available online at `http://www.cs.columbia.edu/\~cmatei/graspit/html-manual/graspit-manual.html`; Stand vom 7. November 2012.

[EKM] Florian Echtler, Gudrun Klinker, and Technische Universität München. A multitouch software architecture.

[FSF12] Inc. 2007 Free Software Foundation. Gnu general public license. Website, 2012. Available online at `http://www.gnu.org/licenses/gpl-3.0-standalone.html`; Stand vom 23 October 2012.

[Han12] Robbie Hanson. Cocoaasyncsocket. git-repository, 2012. Available online at `https://github.com/robbiehanson/CocoaAsyncSocket` Stand vom 26 October 2012.

[Inc12a] Apple Inc. ios human interface guidelines. Website, 2012. Available online at `http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf` Stand vom 23 October 2012.

[Inc12b] Apple Inc. ios reference. Website, 2012. Available online at `http://developer.apple.com/library/ios/documentation/uikit/reference/uiview_class/UIView_Class.pdf` Stand vom 23 October 2012.

[iSLA12a] iOSX Software License Agreement. Xcode. Website, 2012. Available online at `http://www.apple.com/legal/sla/` Stand vom 23 October 2012.

[iSLA12b] iOSX Software License Agreement. Xcode. Website, 2012. Available online at `http://www.apple.com/legal/sla/` Stand vom 23 October 2012.

[LMBS07] Rock Leung, Karon Maclean, Martin Bue Bertelsen, and Mayukh Saubhasik. Evaluation of haptically augmented touchscreen gui elements under cognitive load. In *In Proceedings of ICMI 2007*, pages 12–15. ACM Press, 2007.

[Lum03] Joanna Lumsden. A paradigm shift: Alternative interaction techniques for use with mobile & wearable devices. In *Proc. of the 13th Annual IBM Centers for Advanced Studies Conference CASCON'2003*, pages 197–210. IBM Press, 2003.

[Nar88] Sundar Narasimhan. Dexterous robotic hands: Kinematics and control. *MASTER'S THESIS, MIT ARTIFICIAL INTELLIGENCE LABORATORY*, 1988.

[PP93] In The Presence and Nancy S. Pollard. Planning grasps for a robot hand. In *In Proc. IEEE International Conf. on Robotics and Automation*, pages 723–728, 1993.

[VH03] Matthias Veit and Stephan Herrmann. Model-view-controller and object teams: A perfect match of paradigms. In *In AOSD'03 [2*, pages 140–149. ACM Press, 2003.

[Vli94a] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[Vli94b] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, October 31, 1994.

[VSMS] Rama Vennelakanti, Anbumani Subramanian, Sriganesh Madhvanath, and Sriram Subramanian. Counting on your fingertips – an exploration and analysis of actions in the rich touch space.

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen - benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit der Einstellung der Bachelor-Arbeit in den Bestand der Bibliothek des Departments Informatik einverstanden.

(Ulrike Schäfer)