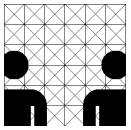
Diplomarbeit
im Studiengang Informatik

Multimodale Sensorfusion zur Bestimmung der Pose einer Wiimote zur Telemanipulation von Service Robotern

am Arbeitsbereich für
Technische Aspekte Multimodaler Systeme
Fachbereich Informatik
Fakultät für Mathematik, Informatik, Naturwissenschaften
Universität Hamburg

vorgelegt von
Martin A. Noeske
Februar 2014



betreut von
Dr. Norman Hendrich
Prof. Dr. Jianwei Zhang

Abstract

The acceptance of service robots increases with the intuitiveness and simplicity of the underlying Human-Robot-Interaction. At the moment we are far from a user being able to point at something or even show how to do something, in a way that would make a robot understand and achieve goals by executing actions autonomously or maybe even learn how to do this.

The Wiimote-Controller is a widely spread, inexpensive and, given its price, relatively accurate input device that has been used in the area of game consoles for years. This circumstance gives rise to the thought that extending the Wiimotes application to robotics might be a good idea.

In this diploma thesis a system was created, that uses the features of the Wiimote to give the user literally something in his hand, so he can interact better with a service robot. For this purpose, a software was developed which can read the sensor data of the Wiimotes accelerometers and gyroscopes and fuse them together to a combined impression of better quality. Based on this, different control strategies for different robotic platforms have been developed and evaluated experimentally, qualitatively. The Wiimote has proved itself as a device being able to improve Human-Robot-Interaction.

Kurzfassung

Serviceroboter erfahren eine höhere Benutzerakzeptanz, wenn die Mensch-Roboter-Interaktion sich möglichst intuitiv und einfach gestaltet. Derzeit sind wir noch weit davon entfernt, dass einem Roboter durch den Benutzer etwas gezeigt oder vorgeführt werden kann und dieser autonom eine Aktion ausführt oder gar erlernt.

Da der Wiimote-Controller ein weitverbreitetes, günstiges und zum Preis verhältnismäßig genaues Eingabegerät ist, welches für Spielekonsolen schon lange erfolgreich genutzt wird, liegt der Gedanke nahe es in der Robotik einzusetzen.

In dieser Diplomarbeit wurde ein System geschaffen, das die Fähigkeiten der Wiimote nutzt, um einem Benutzer buchstäblich etwas in die Hand zu geben, mit dem er besser mit einem Serviceroboter interagieren kann. Dazu wurde eine Software geschrieben, die die Sensordaten der Gyroskope und Accelerometer der Wiimote auslesen und diese zu Daten von besserer Qualität fusionieren kann. Es wurden darauf aufbauend verschiedene Steuerungskonzepte auf verschiedenen Roboterplattformen experimentell qualitativ evaluiert und es hat sich herausgestellt, dass die Wiimote geeignet ist, um die Interaktion zu verbessern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Ich versichere hiermit, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den 28. Februar 2014 Unterschrift: .....
(Martin A. Noeske)

Einleitung

1

Die Mensch-Computer-Interaktion und spezieller die Mensch-Roboter-Interaktion ist ein zentrales Forschungsgebiet im Bereich des Human-Centered Computing. Bei diesem Ansatz steht der Mensch im Mittelpunkt und Serviceroboter werden als Systeme betrachtet, welche bestimmte Dienste für den Menschen erbringen können. Für eine hohe Akzeptanz ist es unerlässlich, dass die Menschen möglichst einfach und intuitiv mit den Robotern interagieren können.

Kaum ein Laienbenutzer ist gewillt sich mit Eigenheiten von Steuerungssoftware zu beschäftigen. Ein Benutzer möchte dem Serviceroboter Dinge zeigen können, wie "Hier ist der Knopf für den Fahrstuhl", "Da ist ein Tisch", "Hier schaltet man das Licht ein" und der Serviceroboter soll die Orte schnell verinnerlicht haben. Falls ein Roboter einen Bewegungsablauf nicht beherrscht, so möchte man es ihm vormachen und er soll es danach ausführen können. Auch eine natürliche Kommunikation durch z.B. Gesten ist wünschenswert.

Der Roboter soll in die Lage versetzt werden, die Bewegungen des Menschen "zu verstehen". Da dies mit aktuellen Methoden der Bildverarbeitung nur schwer und daher sehr begrenzt möglich ist, sind alternative Ansätze von übergeordnetem Interesse. Eine Möglichkeit hierfür besteht darin, dass der Mensch Sensoren an die Hand bekommt, welche seine Bewegungen messen und die Messdaten vom Roboter interpretiert werden. Solche Sensoren sind mittlerweile weit verbreitet und haben schon längst den Massenmarkt erobert. Moderne Spielekonsolen machen von ihnen Gebrauch.

Ein solches dediziertes Eingabegerät, welches ausgestattet mit einer Reihe von Sensoren geliefert wird, ist die Wiimote von Nintendo. Im Kontext von Videospielen wird sie bereits benutzt, um die Eingabe der Kommandos in Form von Bewegungen zu erfassen und ein Feedback zum Benutzer, mittels LEDs und Vibration zu kommunizieren.

1.1 Motivation

Der Gedanke liegt nahe, dass dieses Gerät sich eignen könnte, um die Interaktion mit den Roboterplattformen des Arbeitsbereiches TAMS intuitiver zu gestalten.

Ein Benutzer könnte z.B. dem TAMS Serviceroboter (TASER) im wahrsten Sinne des Wortes zeigen, wo sich Gegenstände von Interesse, wie Tassen, Türklinken oder Fahrstuhlknöpfe befinden, anstatt es über Koordinaten im Computer ausdrücken zu müssen. Ein *learning by demonstration* wird so überhaupt erst ermöglicht.

Eine bequeme Ansteuerung des Mitsubishi PA10 Armes, an dem die Shadow-Hand montiert ist, würde es ermöglichen den Arm in eine Position zu bringen, von der aus Greifexperimente besser durchgeführt werden können. Aktuell besteht die Alternative darin, dass Koordinatenpaare und Winkel per Tastatur eingegeben werden.

Bei der Durchführung von Handmanipulationen könnten die Messdaten mit den Sensoren der Wiimote überprüft werden. Sie könnte als "intelligentes" Objekt, bzw. *"sensing object"* genutzt werden und damit Daten anderer Sensoren (etwa taktiler) ergänzen.

Wenn die Pioneer Plattformen mittels Wiimote an eine gewünschte Position gefahren werden können, damit sie dort (teil-)autonom studentische Experimente durchführen, wäre es für die experimentierenden Studenten nicht mehr nötig einen Laptop in der Hand zu tragen, um Fahrtkommandos an den Pioneer abzusetzen. Im Außeneinsatz (beim sog. "Out-Door-Pioneer") kann dieser Vorteil noch deutlicher zu Tage treten, wenn Laserscans vom Gelände gemacht werden sollen, um sie später weiterzuverarbeiten.

Ein weiteres mögliches Anwendungsgebiet ist die Modulation von Bewegungsparametern modularer Roboter ([NKH+12]). Durch Änderung der Parameterwerte wird die Charakteristik von Bewegungswellen maßgeblich verändert. So ist direkt ersichtlich, welche Auswirkung die jeweilige Änderung – etwa der Amplitude – hat.

In allen diesen möglichen Anwendungsgebieten ist aktuell die Alternative eine Eingabe von Werten mittels Tastatur.

1.2 Ziel der Arbeit

In der vorliegenden Diplomarbeit wird untersucht, wie sich durch den Einsatz der Wiimote die Mensch-Roboter-Interaktion zwischen Benutzern und den Robotern am Arbeitsbereich TAMS verbessern lässt.

Hierzu müssen zunächst die Daten von der Wiimote auf einen Computer/Roboter transferiert und dort verarbeitet werden. Aufbauend auf die native CWiid Bibliothek zur Kommunikation mit der Wiimote, wird eine Software geschrieben, die es ermöglicht Daten von der Wiimote zu empfangen, anzuzeigen, zu verarbeiten und die Wiimote als 3D-Modell zu visualisieren. Verschiedene Steuerungskonzepte werden auf verschiedenen Plattformen implementiert und deren Vor- und Nachteile gegeneinander abgewogen. Die Daten der Gyroskope und Accelerometer werden fusioniert und für Experimente zur neigungsbasierten Steuerung genutzt, sowie zur

Verbesserung der Bestimmung der Orientierung der Wiimote. Es werden auf dem Kalman-Filter basierende Ansätze gewählt, bei denen die Daten der Accelerometer gegenüber den Gyrokopdaten lageabhängig unterschiedlich gewichtet werden. Damit ist eine gute Abschätzung möglich, wie weit sich die Wiimote eignet, um in der Robotik eingesetzt zu werden.

1.3 Arbeiten in einem ähnlichen Umfeld (Related Work)

Da die Wiimote schon seit 2006 für jedermann erhältlich ist, sind eine Reihe wissenschaftlicher, sowie spielerischer Arbeiten und Projekte rund um das Gerät entstanden. Zu den populärsten Projekten und meist gesehenen YouTube Videos, die mit der Wiimote zu tun haben, zählen mit Sicherheit die von Johnny Chung Lee[1]. Seine Arbeiten zu Head-Tracking oder "Wiimote als Whiteboard" waren zweifelsfrei für Hardwarehersteller inspirierend. Eine wissenschaftliche Arbeit, die sich offenkundig von Johnny Lee hat inspirieren lassen, ist [BPRR11], wo Whiteboards mittels Wiimote implementiert werden.

In [PGUv10] haben Petric und Gams die IR-Kameras mehrerer Wiimotes benutzt, um Positionen von Markern im 3D-Raum zu rekonstruieren. Etwas später wird in [DZ12] ein ähnlicher Ansatz gewählt. Eine direktere Interaktion ist in [OV09] vorgestellt, bei der ein Serviceroboter "an der Leine" geführt werden kann. Weitere wissenschaftliche Arbeiten mit der Wiimote sind zum Thema Gestenerkennung verfasst worden, wie [BSRI10] und [DHS08].

Das erfassen von Bewegungen (*Motion Capturing*) unter Zuhilfenahme der Wiimote und MotionPlus spielt in der medizinischen Rehabilitation eine Rolle, wie in [HZJH12] erörtert.

Projekte und Communities, welche die Protokolle der Wiimote reverse-engineert haben sind: (1) die open source community Wiili, (2) das Projekt GlovePie, (3) WiimoteProject.com und (4) wiire.org.

Durch deren Erfahrungen und Dokumentationen sind die o.g. wissenschaftlichen Arbeiten ermöglicht worden, da dort oft Bibliotheken, wie die CWiid, genutzt werden.

1.4 Marktübersicht über alternative Hardware

In diesem Abschnitt wird ein Überblick über alternative Hardware gegeben, welche prinzipiell für diese Arbeit anstelle der Wiimote hätte gewählt werden können.

Die Problemstellung der Positionsbestimmung ist für alle innovativen 3-D Eingabegeräte zu lösen. Verschiedene Hersteller haben dazu verschiedene Lösungsansätze

[1] www.johnnylee.net

gewählt. Im Grunde wird immer versucht Rückschlüsse auf die aktuelle Position und Orientierung der Hände oder des gesamten Körpers der Benutzer zu ziehen. Prinzipiell kann dies geschehen, indem man die Benutzer mit Sensoren versieht oder sie beobachtet und aus den Bildern die gesuchten Daten errechnet.

Die Wiimote von Nintendo, sowie der SixAxis (oder auch DualShock 2 bzw. 3) Controller von Sony, als auch der Nachfolger namens PlayStation Move gehen den ersteren Weg. Microsoft beschreitet mit der Kinect den letzteren Weg.

Eine kurze Übersicht über Vor- und Nachteile der anderen Controller

DualShock hat je 3 Achsen, Gyroskop, Accelerometer, 2 Joysticks, Vibrationseinheit, LEDs; ist somit der Wiimote sensorisch sehr ähnlich. Quelloffene Bibliotheken zum Auslesen der Sensordaten waren aber (für Linux) nicht auffindbar.

Move-Controller ist die Antwort von Sony auf die Wiimote; hat 9 DOF durch je 3 Achsen Accelerometer, Gyroskop, Magnetometer. Die leuchtende Kugel erlaubt eine Detektion im Raum. Ein quelloffenes SDK wurde von Sony angekündigt, wurde aber für diese Diplomarbeit nicht rechtzeitig veröffentlicht.

Kinect Kamera erzeugt zwei Bilder: ein Infrarotbild und ein "normales" Farbbild; erlaubt somit das Errechnen einer 3D-Position von Menschen (Infrarotkörpern); die Problemstellung liegt bei der Kinect im Bereich der Bildverarbeitung und nicht Sensorik. Das macht sie für die Problemstellung dieser Diplomarbeit nur wenig geeignet.

1.4.1 Entscheidung für die Wiimote

Die Wiimote wird in Abschnitt (3.1) detailliert vorgestellt. Hier sei erwähnt, dass sie schon seit 2006 auf dem Markt ist. Sie hatte also im Vergleich zu den anderen Controllern vier Jahre mehr Zeit von Nutzern evaluiert zu werden. Mögliche anfängliche "Kinderkrankheiten" konnten beseitigt werden und das Wichtigste: Eine große Entwicklergemeinde hatte Zeit (Wiibrew-Projekt [Gem14b]) das Gerät und seine technischen Fähigkeiten sehr genau und detailliert zu untersuchen und zu beschreiben. Ferner konnte freie Software für die Kommunikation mit diesem Gerät auf den Markt gebracht werden, welche ebenfalls über Jahre reifen konnte (siehe Abschnitt 4.1). Ein weiteres Argument für einen robusten Sensor, welchen man im Bedarf nachkaufen kann ist der günstige Preis der Wiimote im Vergleich zu den anderen Controllern.

Diese Argumente machen die Wiimote als Eingabegerät für diese Arbeit geeigneter als die zuvor genannten, wenngleich sich dies in einigen Monaten, oder Jahren, anders präsentieren kann.

1.4.2 Sensoren als Bauteile

Gegen Ende der Bearbeitungszeit dieser Diplomarbeit sind Sensoren als Bauteile auf dem Markt gekommen, die in Leistungsfähigkeit mit denen der Wiimote vergleichbar, oder ihnen sogar überlegen sind. Das vorgesehene Anwendungsgebiet ist Flugzeug/Hubschraubermodellbau. Diese Mikrochips besitzen 9-10 DOF (je 3 Achsen Magnetometer, Gyroskop, Accelerometer und ggf. ein Barometer). Bei einem Preis von 10 Euro sind sie aktuell günstiger als eine Wiimote, erfordern allerdings weitere Hardware, wie Arduinos oder Raspberry Pis, damit sie über I^2C ausgelesen und deren Daten verarbeitet werden können. Für die Sensoren werden die Nachfolgechips der Wiimote MPU6050, HMC5883L und als Barometer BMP085 benutzt. Alternativ, falls das 3 Achsen Accelerometer durch 2 einfachere ersetzt wird, folgende Chipsätze: L3G4200D, ADXL345, HMC5883L, BMP085. Diese Hardware wird der Vollständigkeit halber erwähnt, war aber bei Anmeldung dieser Diplomarbeit nicht zu einem angemessenen Preis erhältlich.

1.5 Bibliotheken für die Verwendung der Wiimote an einem Computer

Es wurden seit dem Erscheinen der Wii und damit auch der Wiimote eine ganze Reihe Bibliotheken geschrieben, mit denen man die Wiimote an andere Geräte als die Wii-Spielekonsole anschließen und benutzen kann. In diesem Abschnitt wird ein kleiner Überblick über einige Bibliotheken zur Verwendung der Wiimote an einem Computer gegeben. Die Liste ist nicht als vollständig zu verstehen, enthält aber die bekanntesten Bibliotheken. Warum letztlich die Wahl auf die CWiid fiel wird in Abschnitt 4.1 genauer erläutert. Sie hatte alle nötigen Eigenschaften, war gut ausgereift, lief verhältnismäßig stabil und die Installation war einfach. Das lässt sich leider nicht von allen Bibliotheken sagen. Manche ließen sich trotz angestrengter Versuche gar nicht installieren.[2] Da die in dieser Arbeit produzierte Software (Abschnitt 5) auf der TASER-Plattform (Abschnitt 3.2.2) lauffähig sein soll, kommen nur Bibliotheken in Frage, die auch unter Linux lauffähig sind.

GlovePIE[3] ist eine Bibliothek für Windows, welche eine hohe Verbreitung und Akzeptanz besitzt. Sie wurde auch an unserem Fachbereich in Projekten zum Interaktionsdesign eingesetzt. **GlovePIE** bietet ein eigenes Interface und kann dem Betriebssystem gegenüber als Joystick auftreten oder aber Tastatureingaben emulieren. Mit dieser Bibliothek lassen sich auf sehr einfache Weise Tasten der Wiimote auf Tasten der Tastatur abbilden. Es ist auch möglich die Software Abläufe lernen zu lassen und sie damit zur Erkennung von einfachen Gesten zu nutzen. So kann

[2] Ubuntu 10.04, 10.10 bzw. 11.04 und openSUSE 11.3 sowie 11.4

[3] `www.glovepie.org`, besucht Juli 2011

mit **GlovePIE** in wenigen Zeilen Code eine Anweisung, wie *Drehung im Uhrzeigersinn entspricht dem Drücken der ENTER-Taste*, ausgedrückt werden. **GlovePIE** ist eine mächtige und ausgereifte Bibliothek, welche mit sehr geringen Programmierkenntnissen, die Möglichkeit bietet, mit Hilfe von Skripten interessante Dinge zu vollbringen. Leider ist sie nicht gut geeignet, um Daten von der Wiimote auszulesen und in einer anderen Programmiersprache weiterzuverarbeiten. Das war aber auch nicht das Entwicklungsziel dieser Bibliothek. Derzeit ist sie ausschließlich unter Windows-Systemen lauffähig.

Für das Betriebssystem Linux wurden die **CWiid** (siehe Abschnitt 4.1), die **lg3d-wii**, die **libwiimote**, **PerlWiimote** und die **WMD**-Bibliothek veröffentlicht. Die **lg3d-wii**-Bibliothek[4] wurde als Eingabegerättreiber entwickelt, damit die Wiimote wie ein Joystick oder anderes Eingabegerät an einem Linuxsystem benutzt werden kann. Diese Bibliothek macht von ihrer Intention her den Zugriff auf die Rohdaten der Wiimote unwahrscheinlich. Da initiale Installationsversuche fehlschlugen, wurden weitere Tests in dieser Arbeit vermieden.

Die Bibliothek **WDM**, welche zum Generieren von Maus- und Tastatur-Events mit einer Wiimote gedacht war, ist aus denselben Gründen, wie die letztgenannte nicht gut geeignet, weil der Zugriff auf die Rohdaten erschwert ist. **WDM** und **PerlWiimote** haben beide noch den Nachteil, dass sie in Programmiersprachen geschrieben sind (Python bzw. Perl), welche für diese Diplomarbeit mangels Sprachmächtigkeit ungeeignet sind.

Die Angaben zur **libwiimote** sind zum Teil etwas widersprüchlich. Es lassen sich Belege finden, dass sie nicht weiterentwickelt wurde und in den meisten Linux-Distributionen ihre Unterstützung eingestellt und sie durch die **CWiid** ersetzt wurde. Andererseits soll sie vollständig in die **CWiid** eingeflossen sein, bzw. sie wurde lediglich in die **CWiid** umbenannt. Es was hier nicht möglich abschließend zu klären, ob es zwei Bibliotheken waren oder ob es eine ist, welche umbenannt wurde, möglicherweise auf Grund von Lizenzrechten oder Urheberrechten. Beispielprogramme deuteten auf Grund ziemlich verschiedener Syntax auf unterschiedliche Bibliotheken hin. Andererseits ist ein Überarbeitung und Weiterentwicklung einer Bibliothek mit frischer Namensgebung denkbar. Da der angegebene Funktionsumfang beider Bibliotheken im Wesentlichen gleich ist[5], ist es für diese Arbeit von niedriger Relevanz, ob es sich vielleicht um die gleiche Bibliothek handelt. Lauffähige Quellen für die **libwiimote** wurden nicht gefunden.

Für das Mac OS X wurden die **DarwiinRemote**, **Remote Buddy** und die **Wiinstrument** entwickelt. In dieser Arbeit wurden sie aber nicht getestet, weil die Software quelloffen und linuxbasiert sein soll.

[4] `https://www.ohloh.net/p/lg3d-wii`

[5] angeblich soll die **libwiimote** die Lautsprecherausgabe gut unterstützt haben, was sie in diesem Punkt der **CWiid** überlegen gemacht hätte

1.6 Anwendungsmöglichkeiten für die Wiimote

Die Anwendungsmöglichkeiten der Wiimote werden in der vorliegenden Arbeit in zwei unterschiedliche Anwendungsklassen unterteilt. Die eine Klasse bilden Anwendungsgebiete, welche die Sensoren der Wiimote in einfacher Weise nutzen und durch Multisensordatenfusion (siehe Abschnitt 2.1.2) unter Umständen wesentlich verbessert werden können, aber im Grunde auch ohne eine Sensorfusion implementierbar sind. Sie werden hier Anwendungen der *simplen Klasse* genannt. Die andere Klasse bilden die Anwendungen, welche durch Multisensorfusion überhaupt erst möglich werden. Diese Klasse wird hier *komplexe Klasse* genannt.

Zu den simplen Anwendungen zählen hauptsächlich alle Arten von Steuerungsaufgaben, wie die Steuerung von:

- Pioneer-Roboterplattform: Hierbei wird die Plattform in zwei Dimensionen gesteuert und kann ggf. noch auf einen Knopfdruck reagieren, indem z.B. ein Greifer schließt.
- TASER-Roboter-Plattform: Die Steuerung der mobilen Plattform ist prinzipiell die gleiche, wie die des Pioneers. Durch den montierten PA10-Arm kommen aber sechs weitere Freiheitsgrade hinzu.
- Shadowhand: Diese fünffingerige Hand kann ebenfalls mit einer Wiimote gesteuert werden, wobei die Abbildung zwischen den 27 Freiheitsgraden der Hand und denjenigen der Wiimote eine Hauptaufgabe bei der Implementierung darstellt.
- Im Rahmen der von Dennis Krupke programmierten Simulationsumgebung für modulare Roboter ([KLZ+12]) kann die Wiimote für die Evaluation geeigneter Parameter, wie Phase und Amplitude, dazu benutzt werden, um Eigenschaften eines entwickelten Fortbewegungsmodells während der Laufzeit zu modifizieren und zu evaluieren ([NKH+12]).

Diesen simplen Anwendungen ist gemein, dass sie im Prinzip eine Messgröße auf eine Stellgröße abbilden und beide linear verlaufen. So kann z.B. die Neigung einer Wiimote bestimmt und zur Einstellung einer Geschwindigkeit benutzt werden. Dies kann unter Benutzung der Accelerometer alleine geschehen, wobei die Ergebnisse unter Zuhilfenahme der Gyroskope teilweise wesentlich verbessert werden können.

Komplexere Anwendungsmöglichkeiten für die Wiimote ergeben sich im Rahmen des HANDLE Projektes oder als intelligentes Eingabegerät zur Steuerung von Robotern.

- Sensing Object: Für Manipulationsaufgaben mit der Shadowhand kann die Wiimote als ein *sensing object* genutzt werden, welches mit seinen fusionierten Sensordaten, die Daten der Roboterhand ergänzen kann und damit eine bessere Auswertung und Interpretation der Szene ermöglicht. Auf diese Weise kann die Datenpluralität, im Vergleich zum Einsatz des derzeitigen sensing object (eine "Cola"-Dose) alleine, erhöht werden.

- Gestenerkennung: Durch das Erkennen von Gesten kann die Wiimote für eine intuitivere Interaktion mit einem Roboter genutzt werden.
- Lernen von Bewegungsabläufen: Wenn die Pose einer Wiimote hinreichend gut bestimmt werden kann, dann kann die mit ihr durchgeführte Bewegung direkt von einem Manipulator nachgefahren und so von einem Roboter gelernt werden. So kann einem Serviceroboter z.B. ein Weg beigebracht werden, der durch bestimmte Punkte zu verlaufen hat.

Die Implementation der Lernverfahren oder Gestenerkennungsverfahren, für welche Hidden-Markov-Modelle, Fuzzy-Logik oder andere Lerntechniken, wie neuronale Netze nötig wären, gehören nicht zum Fokus dieser Arbeit.

1.7 Aufbau der Arbeit

Im ausführlichen Grundlagenteil (Kapitel 2) werden alle für diese Diplomarbeit nötigen Konzepte und Begriffe erläutert. Insbesondere die Konzepte *Eulerwinkel*, *Rotationsmatrizen*, sowie *Multi-Sensordatenfusion* werden im ersten Teil beschrieben, während der zweite Teil recht ausführlich auf verschiedene Inertialsensoren, insbesondere Beschleunigungssensoren (Accelerometer) und Drehratensensoren (Gyroskope) auf Basis der MEMS-Technologie eingeht.

Es folgt die Beschreibung der verwendeten Hardware (Kapitel 3), wobei als Erstes die Wiimote als zentrales Element vorgestellt wird. Dabei wird auf die Kamera, die Inertialsensoren (3-Achsen Accelerometer) und die Erweiterungscontroller MotionPlus (3-Achsen Gyroskop), sowie Nunchuk und deren Leistungsfähigkeit eingegangen. Im zweiten Abschnitt des Kapitels 3 werden die Experimentalplattformen vorgestellt. Das sind der Pioneer als mobile Plattform, der TASER (TAMS Service Roboter), sowie der PA-10 Arm von Mitsubishi mit angebauter Hand von Shadow als ein Vertreter der Manipulatoren.

Kapitel 4 beschreibt die in dieser Arbeit verwendeten Programmbibliotheken und geht auf programmiertechnische Grundlagen ein. Zu diesen Grundlagen zählt das Java Native Interface, welches es ermöglicht Bibliotheken, welche in nativen Sprachen geschrieben wurden, von JAVA aus zu nutzen. Die wichtigste Bibliothek, die zu nennen ist, ist die CWiid, eine C-Bibliothek, die den Zugriff auf die meisten Wiimote Funktionen ermöglicht. Player/Stage, ROS und RCCL sind hier direkt, bzw. indirekt genutzt worden, um Plattformen (Player/Stage) bzw. Manipulatoren (ROS/RCCL) zu steuern und werden daher ebenfalls kurz vorgestellt.

Es folgt dann Kapitel 5, in dem die im Rahmen dieser Arbeit produzierte Software präsentiert wird. Angefangen wird mit einem Überblick über die Komponenten, sowie Programmierdetails, wie der Zugriff über JNI auf die CWiid funktioniert und wo die Grenzen dieses Konzeptes liegen. Dann folgt ein Abschnitt über die graphische Bedienoberfläche und die Visualisierung mittels JAVA-3D. Im Abschnitt zu

den Implementationsdetails wird beschrieben, wie die gesammelten Daten als XML repräsentiert und gespeichert werden, wie die Daten gefiltert werden können, sowie der Aufbau der selbsterstellten Klasse, bzw. nachladbaren JAVA-Bibliothek (JNI-Wrapper), die mittels JNI auf die CWiid zugreift. Im letzten Teil dieses Kapitels befindet sich die Beschreibung der Kalibrierungsdaten, sowie des zugehörigen selbstentwickelten Konzeptes, welches es ermöglicht diese Daten, aber auch zukünftige Daten, ggf. von anderer Länge, in der Wiimote zu speichern und wieder auszulesen.

Kapitel 6 bildet den experimentellen Teil dieser Arbeit, in welchem mobile Plattformen, sowie Manipulatoren, zunächst mit den Knöpfen der Wiimote digital gesteuert wurden. Im nächsten Schritt wurden die Joysticks kalibriert und eine analoge Steuerung mittels Joysticks vorgestellt. Es folgen Experimente zur Steuerung mittels Inertialsensoren. Dazu war eine Kalibrierung dieser Sensoren nötig. Es wurde auch die IR-Kamera kalibriert, damit die gemessenen Drehraten der Gyroskope, welche mit Hilfe der gleichmäßigen Drehraten von Plattenspielern ermittelt wurden, auch validiert werden konnten. Aus Sicherheitsgründen wurde bestimmt, ob die Wiimote sich in der Hand befindet, oder unbenutzt ist, z.B. im freien Fall oder auf einem Tisch liegend. So kann eine Notabschaltung realisiert werden. Darauf aufbauend werden in Abschnitt 6.3.6 weitere neigungsbasierte Steuerungskonzepte vorgestellt, welche mittels Sensordatenfusion zu besseren Ergebnissen gebracht werden. Das Kapitel schließt ab mit einem Experiment bei dem zwei Wiimotes an dem Arm einer Testperson befestigt werden und dadurch ein simuliertes, schematisches Gebilde aus Schulter, Oberarm, Ellenbogen und Unterarm den Bewegungen des Benutzers folgt und somit gesteuert wird.

Den Abschluss der Arbeit bildet Kapitel 7, in dem die Ergebnisse zusammengefasst werden, auf weitere Anwendungsgebiete und zugehörige Veröffentlichungen verwiesen wird, sowie ein Ausblick gegeben wird, wie weitere Arbeiten in dieselbe Richtung aussehen könnten.

Im Anhang zu dieser Arbeit folgen die vollständigen Tabellen zu den Experimenten, Bilder der Haltevorrichtungen, die zur Kalibrierung benutzt wurden, sowie die Beschreibung der Verzeichnis bzw. Paketstruktur der erstellten Software. Der beiliegende Datenträger enthält die Datenblätter der Sensoren, verwendete Quellen, sofern sie digital vorlagen und die erstellte Software selbst.

Grundlagen

2

In diesem Kapitel werden die Grundlagen für das Verständnis der vorliegenden Arbeit erörtert. Zusammen mit Kapitel 3 (Hardware) und Kapitel 4 (Verwendete Software) bildet es das Fundament auf dem diese Arbeit fußt. Zunächst werden die hier benutzten Begriffe und Konzepte aus Mathematik, Physik und Robotik kurz geklärt. Dazu zählen die unterschiedlichen Arten zur Darstellung von Rotation. Als nächstes werden Grundlagen der Sensordatenfusion vorgestellt und der Abschnitt schließt mit kurzen Bemerkungen zur Gestenerkennung mittels Inertialsensoren ab, gefolgt von einer ausführlicheren Betrachtung verschiedener Arten von Inertialsensoren. Welche Steuerungskonzepte mit diesen Grundlagen realisiert werden können und wie, ist Gegenstand von Kapitel 6 (Experimentelle Evaluation von Steuerungskonzepten für Roboter mittels Wiimote).

2.1 Grundbegriffe und Grundkonzepte

Roboter, Manipulator, Service Roboter

Der Begriff *Roboter* geht auf ein Science-Fiction Theaterstück von Josef und Karel Čapek zurück, welches im Jahre 1920 unter dem Titel "Rossumovi Univerzální Roboti" bekannt wurde [Ča04]. Seitdem unterlag der Begriff dem Wandel der Zeit und heute wird, wie in der vorliegenden Diplomarbeit unter *Roboter* ein *Computersystem* verstanden, welches mit seiner Umgebung interagieren und diese verändern kann. Nach VDI Richtline 2860 ist ein Roboter ein universell einsetzbarer Bewegungsautomat mit mehreren Achsen, dessen Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei programmierbar sind.

Als *Manipulator* wird hier ein Roboter verstanden, welcher über mehrere Freiheitsgrade verfügt und Objekte manipulieren, beispielsweise greifen, bewegen und hinlegen, kann.

Ein *Serviceroboter* erbringt Dienstleistungen für den Menschen. In dieser Arbeit wird stets davon ausgegangen, dass ein Serviceroboter *autonom* und *intelligent* agieren kann. Dabei wird eine Lernfähigkeit vorausgesetzt. Ferner wird hier davon ausgegangen, dass der Serviceroboter auf einer mobilen Plattform montiert ist und über einen oder mehrere Manipulatoren verfügt.

Eine **Pose** oder äquivalent **Lage** bezeichnet (im technischen Sinne) die Position *und* Orientierung eines Roboters, Manipulators oder anderen Objektes.

Teleoperation und Telemanipulation

Unter **Teleoperation** wird die Steuerung eines Roboters oder Manipulators aus einer Distanz verstanden. Es ist notwendig, dass ein einseitiger Kanal vom Teleoperateur zum Roboter besteht. Wünschenswert ist aber, dass es einen bidirektionalen Kanal gibt und auch etwaige Sensordaten zum Benutzer übertragen werden können. So könnte ein Operateur Daten von Sensoren, beispielsweise Kamera- und Laserdaten, auf einem Monitor verfolgen, während er die Teleoperation durchführt.

Anwendungsgebiete sind allgemein Gebiete, welche für Menschen zu gefährlich, oder lebensfeindlich sind. Dazu zählen große Höhen, Tiefen (im Wasser), hohe bzw. niedrige Temperaturen oder andere Einflüsse, wie Strahlungen, welche für den Menschen zulässige Grenzwerte überschreiten. Ein aktuelles Beispiel für Telemanipulation stellen die in den havarierten Reaktoren von Fukushima eingesetzten Roboter, welche mittels Telemanipulation aus selbsterklärenden Gründen, eingesetzt werden. Ein direkter Einsatz von Menschen wäre derzeit unverantwortlich.

Obwohl die Begriffe der **Teleoperation** und **Telemanipulation** oft synonym benutzt werden, ist für eine Telemanipulation von der Wortherkunft ein Manipulator von Nöten. Eine fahrbare Plattform kann zwar ferngesteuert, damit also teleoperiert werden, sie kann aber nichts im eigentlichen Sinne manipulieren. Nach diesem Hinweis wird auch in dieser Arbeit nicht scharf zwischen den beiden Begriffen unterschieden.

Physikalische Grundlagen

Die **Gravitation** ist eine der 4 physikalischen Grundkräfte[1] (auch *fundamentale Wechselwirkungen* genannt) und bildet die physikalische Grundlage für viele Berechnungen und Störungen, welche in dieser Arbeit behandelt werden. Aus theoretischer Sicht ist die Reichweite der Gravitation unbegrenzt und sie selbst lässt sich nicht abschirmen. Die Gravitationskonstante $\boldsymbol{g}$ ist eine Fundamentalkonstante der Physik und unter ihnen am wenigsten genau bestimmt. Der Wert für $\boldsymbol{g}$ wird in Mitteleuropa mit

$$g = 9,81 m/s^2 \tag{2.1}$$

angenommen. Die Gravitation ist eine Kraft und wirkt auf Körper als Beschleunigung. Im Umkehrschluss gilt also: Für Beschleunigung ist also immer eine Kraft notwendig. Beschleunigung wird definiert als das Verhältnis von Kraft zu Masse:

[1] neben den drei anderen: Starke Kernkraft, Elektromagnetische Kraft, Schwache Kernkraft

$$\vec{a} = \frac{\vec{F}}{m} \tag{2.2}$$

Da auf einen sich frei bewegenden Körper eine Kraft nicht wirken kann, ohne seine Geschwindigkeit in Betrag und/oder Richtung zu ändern, wird in der Physik jede beliebige Änderung des Geschwindigkeitsvektors (ein Bremsvorgang wäre dann eine negative Beschleunigung) als Beschleunigung betrachtet. Sie kann somit auch als zeitliche Ableitung des Geschwindigkeitsvektors $\dot{\vec{v}}$ und 2. Ableitung des Ortsvektors $\ddot{\vec{s}}$ nach der Zeit dargestellt werden. Es gilt somit:

$$\vec{a}(t) = \dot{\vec{v}}(t) = \ddot{\vec{s}}(t) \tag{2.3}$$

Von dieser wichtigen Tatsache wird an mehreren Stellen dieser Arbeit Gebrauch gemacht.

In dieser Arbeit geht es hauptsächlich darum, aus gemessenen Sensordaten auf Position und Lage eines Objektes zurückzuschließen. Das Bestimmen der Lage ist verhältnismäßig einfach (siehe 6.3). Für die Berechnung der Position muss zunächst die Beschleunigung $\vec{a}$ über die Zeit t gemessen werden. Auf diesen Prozess haben Messfehler, Rauschen und andere Störgrößen besonders starken Einfluss.

Im nächsten Schritt kann die Geschwindigkeit v errechnet werden:

$$\vec{v} = \vec{a} \cdot t \tag{2.4}$$

Um die Position eines Körpers schätzen zu können, muss man seine zurückgelegte Strecke ausrechnen. Man nimmt hierbei an, dass die Messung sich in hinreichend kleine Zeitabschnitte Δ_t zerlegen lässt und innerhalb dieser die Geschwindigkeit konstant bleibt. Messwerte der Accelerometer der Wiimote werden grundsätzlich mit 100Hz geliefert, also kann in dieser Arbeit $\Delta_t = 10ms$ und die Geschwindigkeit innerhalb der Δ_t als konstant angenommen werden. Es gilt für die Änderung der Wegstrecke Δ_l und damit indirekt die Position dann:

$$\Delta_l = \vec{a} \cdot t \cdot \Delta_t \tag{2.5}$$

Da die Messwerte der Wiimote hier zwar in leicht schwankenden, aber immer bekannten und als fest angenommenen Δ_t geliefert werden, ist für eine Positionsbestimmung noch die Beschleunigung zu messen. Diese wird hier im Messverhältnis zu $\boldsymbol{g}$ angegeben. Es werden also Messwerte m_i für die Beschleunigung a als Vorfaktoren mit der Gravitation multipliziert. Aufgrund der Messbereiche der Accelerometer der Wiimote (Siehe Datenblatt A.3, Seite XII) kann man für die einzelnen m_i Werte zwischen -3.0 und +3.0 annehmen. Es wird also die Beschleunigung $\vec{a}$ hier mit Hilfe von $\boldsymbol{g}$ ausgedrückt:

$$\vec{a} = m_i \cdot g \tag{2.6}$$

Damit ändert sich die Formel 2.5 zu:

$$\Delta_l = m_i \cdot g \cdot t \cdot \Delta_t \tag{2.7}$$

Für die gesamte Wegstrecke l gilt dann:

$$l = \sum (m_i \cdot g \cdot t \cdot \Delta_t) \tag{2.8}$$

Falls man die Δ_t als beliebig klein annehmen kann, gilt:

$$l = \lim_{\Delta_t \to 0} \left(\sum (m_i \cdot g \cdot t \cdot \Delta_t) \right) = \int_0^T (m_i \cdot g \cdot t) \, \mathrm{d}t \tag{2.9}$$

Für die Genauigkeit der Sensoren der Wiimote kann man 10ms als hinreichend klein betrachten, sodass die Gleichung 2.9 als theoretisch hinreichend fundiert betrachtet werden kann.

Inertialsystem Als Inertialsystem[2] bezeichnet man ein physikalisches System in dem das newtonsche Trägheitsgesetz gilt. Im Allgemeinen kann man sich die alltägliche Umgebung darunter vorstellen. Körper in Bewegung versuchen auf Grund der Massenträgheit ihre Bewegung in Betrag und Richtung beizubehalten. Sie bewegen sich gleichmäßig und gleichförmig, falls keine Kraft auf sie einwirkt. Wenn nicht explizit anders gesagt, tritt bei unseren Systemen grundsätzlich immer Reibung auf und keine Bewegung kann ohne Energieaufwand dauerhaft fortbestehen. Dies ist eine Konsequenz aus den Eigenschaften unserer umgebenden Welt und nicht aus denen eines Inertialsystems, gilt allerdings dennoch stets bei hier betrachteten Systemen.

Corioliskraft Die Corioliskraft ist eine der Trägheits- bzw. Scheinkräfte. Diese können nur in beschleunigten Bezugssystemen auftreten. In Inertialsystemen kann es solche Kräfte nicht geben. Für deren Auftreten sind zwei unterschiedlich wirkende Kräfte und die Massenträgheit nötig. Sie werden Scheinkräfte genannt, weil es vom Bezugssystem abhängig ist, ob man die Existenz dieser Kräfte zur Erklärung einer Bewegung heranziehen muss oder nicht.

Wenn in einem rotierenden Bezugssystem zwei Personen sich einen Ball zuwerfen würden, so käme ihnen die Flugbahn des Balles, in ihrem (rotierenden) Bezugssystem geradlinig vor. Für einen Betrachter aus einem Inertialsystem heraus dagegen würde die Bewegung des Balles nicht etwa einen Kreis, wie es im ruhenden Fall wäre, oder gar eine geradlinige Bewegung, sondern eine gekrümmte Bahn ergeben. Die Corioliskraft würde als resultierende der Zentrifugalkraft und der Massenträgheit

[2] lat.: *iners* träge, untätig

senkrecht zur Rotationsachse und senkrecht zur Bewegungsrichtung des Balles wirken.

Sie bildet die physikalische Grundlage der meisten Gyroskope. In MEMS-Technologie wurde noch kein anderes Funktionsprinzip eingesetzt (siehe Abschnitt 2.2.2).

Statistische Grundlagen

Konzepte aus diesem Abschnitt werden in der Software und deren Beschreibung verwendet.

Der *arithmetische Mittelwert* ist die Summe aller Werte geteilt durch deren Anzahl. Der *geometrische Mittelwert* ist für n Werte die n-te Wurzel aus dem Produkt aller Werte. Nach diesem Schema kann man auch einen *quadratischen* oder *kubischen Mittelwert* bilden. Bei n sortierten Messwerten teilt der *Median* die Stichprobe genau in der Mitte. Kommen dafür zwei (unterschiedliche) Werte in Frage, so gibt jeder von beiden einen gültigen Median ab, von denen einer auszuwählen ist. Der Mittelwert kann ein Wert sein, welcher nie gemessen wurde. Als Median können nur Messwerte dienen.

Der häufigste Wert in einer Stichprobe bzw. Verteilung heißt *Modus*. Gibt es mehr als ein (lokales) Maximum in der Stichprobe bzw. Verteilung, so nennt man sie *bimodal*, wenn zwei Maxima vorliegen, sonst *multimodal*.

Die *Varianz* ist die Summe der quadrierten Abweichungen aller Messwerte vom (arithmetischen) Mittelwert geteilt durch die Anzahl der Messwerte. Als Formel für n Messwerte: $Varianz = \frac{\sum_{i=0}^{n-1}(x_i - x_{mittel})^2}{n}$

Die *Standardabweichung* ist die (positive) Quadratwurzel aus der Varianz. Es gilt: $\sigma_X = \sqrt{Var(X)}$

Messung, Messfehler und Störgrößen

Da eine Messung immer einen Vergleich darstellt, auch wenn sie mit einem Gerät durchgeführt wird, kann es keine exakte Messung einer physikalischen Größe geben [OM84]. Messungen werden nach ihrer *Genauigkeit*, also der Übereinstimmung von gemessenem und tatsächlichem Wert, bewertet. Für Genauigkeit ist die Präzision, also möglichst gute Wiederholbarkeit, Voraussetzung. Bei der Wiimote hat das Gyroskop eine deutlich höhere Präzision, als das Accelerometer.

Wenn eine Messung durchgeführt wird, können Gerätefehler, methodische Fehler, personenbezogene Fehler, systematische Messfehler und Zufallsfehler auftreten. Fehler der erstgenannten Fehlerquelle sind im Rahmen dieser Arbeit nicht festgestellt worden. Die beiden nächstgenannten Fehlerquellen stehen inhärent mit der Person des Experimentators in Zusammenhang. Es wird daher davon ausgegangen, dass das

Messinstrument (Wiimote) zuverlässig funktioniert, die Versuche sinnvoll konzipiert wurden und die Experimente das messen, was sie messen sollen.

Das Hauptaugenmerk richtet sich hier auf die letztgenannten beiden Fehlerquellen [DSN10]. Systematische Messfehler werden durch Kalibrierung der Messvorrichtung beseitigt. Wie man Zufallsfehler in den Experimenten zu dieser Diplomarbeit vermindern kann, wird mittels verschiedener Filter erprobt. Der Einfluss von Hysterese, also einem unterschiedlichen Messverhalten, bei steigender und fallender Messgröße, ist bei Inertialsensoren immer vorhanden. Hier wird er aber als minimal angenommen. Temperaturabhängiges Messverhalten wurde in den Experimenten bei der Wiimote festgestellt. Hierfür wird aber eine erneute Kalibrierung an besonders warmen bzw. kalten Tagen, vor Experimentbeginn, als eine ausreichende Lösung betrachtet. Auf das Einbeziehen der spezifischen Sensorkennlinie zur Auswertung wurde verzichtet. Der Einfluss von Störgrößen, wie elektromagnetischen Wellen (Mikrowellen oder Mobilfunk), wird hier als nicht vorhanden angenommen.

2.1.1 Repräsentation von Rotation

Die Rotation ist eine lineare Transformation. Man kann Rotation mittels des Objektkoordinatensystems des zu drehenden Objektes, mittels des Weltkoordinatensystems oder einer Beziehung zwischen den beiden ausdrücken. So können jeweils zwei der Aussagen unter gewissen Umständen die gleiche Drehung beschreiben:

- Drehe das Objekt Obj um 30° entlang seiner y-Achse (des Objektkoordinatensystems) im Uhrzeigersinn.
- Drehe das Objekt Obj um 30° entlang der z-Achse des Weltkoordinatensystems gegen den Uhrzeigersinn.
- Drehe das Objekt Obj so, dass die x-Achse des Objektkoordinatensystems und die y-Achse des Weltkoordinatensystems einen Winkel von 30° einnehmen.

Es wurden verschiedene Arten der Beschreibung von Rotation für verschiedene Anwendungsfälle entwickelt. Drei mögliche Repräsentationsformen von Rotation werden hier kurz vorgestellt:

- Eulersche Winkel: sie werden in dieser Arbeit hauptsächlich benutzt, um die Orientierung der Wiimote zu beschreiben.
- Drehmatrizen: bilden die Grundlage für die Rotation und das Integrieren der Orientierung über die Zeit und damit das Berechnen von $\vec{g}$ in der hier erstellten Software.
- Quaternionen: werden oft wegen ihrer Effizienz in Software zur 3D-Simulation und 3D-Darstellung benutzt. So geschieht es auch bei den Bibliotheken, die in dieser Arbeit benutzt werden.

Eulersche Winkel

Mittels Eulerscher Winkel (benannt nach Leonhard Euler) kann die Orientierung eines Körpers im Raum angegeben werden. Die Winkel beschreiben eine Drehung um bestimmte Achsen des Objektes und geben damit eine Transformation vom globalen Koordinatensystem ins Koordinatensystem des Körpers und umgekehrt an. Da die Drehachsen des Körpers und die Reihenfolge der Rotationen frei gewählt werden kann, ergeben sich mehrere mögliche unterschiedliche, aber gleichwertige Definitionen für Eulersche Winkel. In dieser Arbeit wird die Roll-Nick-Gier-Konvention (im Gegensatz zur DIN 9300 Yaw-Pitch-Roll, Z, Y', X") aus der Luftfahrt benutzt, welche im folgenden Abschnitt vorgestellt wird.

Roll-Nick-Gier Winkel sind drei objektbezogene Drehwinkel, welche auch auf Fahrzeuge angewendet werden. Die Wurzeln liegen in der Luftfahrt, die Begriffe wurden aber mittlerweile auf Wasser- und Landfahrzeuge übertragen. Der Schnittpunkt der drei rechtwinkligen Achsen des Objektes liegt in seiner (definierten) Mitte. Dann sind die drei Winkel Rollen (*roll*), Nicken (*pitch*) und Gieren (*yaw*) wie folgt definiert:

- Rollen (*roll*), auch gelegentlich *Querneigungswinkel* genannt, gibt die Drehung um die Längsachse des Fahrzeuges an. Die Längsachse liegt in Bewegungsrichtung des Fahrzeuges und ist meist mit x-Achse bezeichnet.
- Nicken (*pitch*) ist die Drehung um die senkrecht zur Längsachse (horizontal) verlaufende Achse (Querachse, meist y-Achse). Falls dieser Winkel 0 ist, so ist das Fahrzeug plan.
- Gieren (*yaw*) beschreibt die Drehung um die auf der x/y-Ebene senkrecht stehende z-Achse, die Vertikalachse. Aus der Begriffswelt der Aeronautik ist auch der Ausdruck *heading* entliehen.

Diese Winkel sind sehr leicht intuitiv verständlich und falls eine Reihenfolge definiert wird, in der die Rotationen auszuführen sind, wie oben geschehen, so ist damit eine eindeutige Orientierung beschreibbar. Allerdings treten mit dieser der Luftfahrt entliehenen Beschreibungsweise in ihren Grenzfällen Probleme auf. So ist es bei einem Fahrzeug oder beliebigen Körper unklar, welcher sein *roll* und welcher sein *pitch* Winkel ist, wenn das Fahrzeug genau senkrecht auf die Erde zeigt. In diesem Fall wäre die Längsachse nicht notwendigerweise in Bewegungsrichtung und sie kann sowohl als *roll*, als auch als *pitch* bezeichnet werden. Es ist verständlich, warum solche Gesichtspunkte in der anfänglichen Definition keine Rolle spielten. In Flugzeugen, U-Booten und Autos stellt in dem Moment, wenn "vorne" und "unten" die gleiche Richtung beschreiben, eine Definitionslücke der eulerschen Winkel ein untergeordnetes Problem dar. In der Robotik und Computergraphik müssen diese Grenzfälle aber besonders behandelt werden, damit merkwürdige Effekte vermieden werden. Es gibt mehrere solcher Grenzfälle in denen sich die eulersche Orientierung per definitionem

schlagartig ändern kann. Diese speziellen Orientierungen der Wiimote werden in der hier erstellten Software gesondert behandelt.

Drehmatrix

Jede beliebige Rotation im $\mathbb{R}^3$ (um einen Winkel α) lässt sich mittels maximal dreier Rotationen (mit Winkeln $[\phi, \theta, \psi]$) um die kartesischen Achsen darstellen. Werden die Drehungen als Rotationsmatrizen geschrieben, so sieht die Matrix für die Drehung um die x-Achse so aus:

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix} \tag{2.10}$$

Die Drehung um die y-Achse sieht wie folgt aus:

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{2.11}$$

Eine Drehmatrix für die Drehung um die z-Achse hat die Form:

$$R_z(\psi) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.12}$$

Will man um eine beliebige Ursprungsgerade drehen, so repräsentiert man sie günstigerweise mit einem normierten Vektor: $\vec{n} = (n_1, n_2, n_3)^T$

Die Drehmatrix $R_{\vec{n}}$ der Drehung um diese Gerade ist erheblich komplexer aufgebaut, entspricht aber nur dem Matrixprodukt $R_{\vec{n}} = R_x \cdot R_y \cdot R_z$ der drei Rotationsmatrizen.

(Zwecks besserer Lesbarkeit und aus Platzgründen wurden die Klammern in der Notation der trigonometrischen Funktionen zulässigerweise weggelassen und $\cos\alpha$ sowie $\sin\alpha$ durch C_α bzw. S_α ersetzt.)

$$R_{\vec{n}}(\alpha) = \begin{pmatrix} C_\alpha + n_1^2(1-C_\alpha) & n_1 n_2(1-C_\alpha) - n_3 S_\alpha & n_1 n_3(1-C_\alpha) + n_2 S_\alpha \\ n_2 n_1(1-C_\alpha) + n_3 S_\alpha & C_\alpha + n_2^2(1-C_\alpha) & n_2 n_3(1-C_\alpha) - n_1 S_\alpha \\ n_3 n_1(1-C_\alpha) - n_2 S_\alpha & n_3 n_2(1-C_\alpha) + n_1 S_\alpha & C_\alpha + n_3^2(1-C_\alpha) \end{pmatrix} \tag{2.13}$$

Unter bestimmten Bedingungen kann diese Matrix vereinfacht werden. Von diesen Vereinfachungen wird in der Software, die dieser Arbeit zugrunde liegt, ausgiebig Gebrauch gemacht.

Approximation von kleinen Winkeländerungen Wenn eine kleine Orientierungsänderung eines Körpers in Bezug auf ein globales Bezugssystem beschrieben werden soll, wie es oft beim Tracking von Messeinheiten (siehe Abschnitt 2.2.4) der Fall ist, so können für kleine Winkel die trigonometrischen Funktionen approximativ durch die Winkel selbst substituiert werden. Falls eine Drehmatrix benutzt wird, um Winkeländerungen innerhalb von kleinen Zeitintervallen zu beschreiben, so ist davon auszugehen, dass die Winkel, welche zu beschreiben sind, auch hinreichend klein sind. Auf jeden Fall kann ein Zeitintervall so klein gewählt werden, dass die maximale Winkeländerung die geforderte Bedingung erfüllt. Es gilt dann für kleine Winkel ϕ, θ und ψ, dass der Sinus eines Winkels näherungsweise durch den Winkel selbst ersetzt werden kann und für den Kosinus der Wert 1 angenommen werden kann. Wenn man diese Approximation benutzt und das Produkt der Winkel, auf Grund ihrer sehr kleinen Werte vernachlässigt, so vereinfacht sich die Rotationsmatrix (2.13) zu:

$$R = R_x \cdot R_y \cdot R_z \approx \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{pmatrix} \tag{2.14}$$

Quaternionen

Quaternionen, auch Hammilton Zahlen genannt, sind eine Erweiterung der reellen Zahlen und bilden ihr eigenes Zahlensystem, ähnlich wie die komplexen Zahlen. Quaternionen lassen sich aus den reellen Zahlen durch Hinzunehmen von drei neuen Zahlen i,j,k bilden. Damit erzeugt man, analog zu den komplexen Zahlen, einen Realteil und einen Vektorteil $< i, j, k >$, welcher als ein dreikomponentiger Imaginärteil betrachtet werden kann.

Ein Quaternion kann geschrieben werden als: $x_0 = x_1 \cdot i + x_2 \cdot j + x_3 \cdot k$

Ferner gelten die Eigenschaften:

(1) $i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$
(2) $i \cdot j = -i \cdot j = k$
(3) $k \cdot i = -i \cdot k = j$
(4) $k \cdot j = -j \cdot k = i$

Beim Rechnen mit Quaternionen auf $< +, \cdot >$ gilt das Assoziativgesetz und das Distributivgesetz. Für Multiplikation der Imaginärzahlen i,j,k und generell der Quaternionen gilt nicht das Kommutativgesetz. (Realzahlen können aber kommutativ an i,j,k multipliziert werden.) Quaternionen bilden also einen Schiefkörper und eignen sich damit zur Darstellung von Rotation, welche ebenfalls nicht kommutativ ist.

Vergleich der Ansätze

Die Anschaulichkeit von Rotation in ihrer Quaternionen-Schreibweise ist gegenüber dem Ansatz der Drehmatrix-Schreibweise etwas schwerer zugänglich. Da aber statt der neun Werte einer Rotationsmatrix nur noch vier Werte für eine Repräsentation benötigt werden, reduziert sich die Berechnung von 27 Multiplikationen und 18 Additionen, welche für das Berechnen des Matrixproduktes $R_a \cdot R_b$ zweier 3×3 Rotationsmatrizen benötigt werden, auf die Berechnung eines Quaternionenproduktes $q_1 \cdot q_2$, welches nur 16 Multiplikationen und 12 Additionen benötigt. Das macht Quaternionen für Berechnungen in der Computergraphik effizienter als Drehmatrizen [Koc08, Abschnitt 7.1]. Besonders hoch ist der Effizienzgewinn, wenn viele Matrizenmultiplikationen nötig sind.

Eulersche Winkel sind zweifelsfrei die anschaulichste Repräsentation, eignen sich damit gut für die Beschreibung der Lage eines Objektes, aber nicht gut für Berechnungen, da sie im strengen Sinne Definitionslücken aufweisen.

In vielen Fällen, wenn die Effizienz nicht gänzlich im Vordergrund steht, sind Drehmatrizen ein guter Kompromiss zwischen Effizient und Anschaulichkeit. Sie wurden in der in dieser Diplomarbeit erstellten Software für die interne Darstellung der Rotation gewählt.

2.1.2 Multisensor Datenfusion

Mit *Multisensor Datenfusion* bezeichnet man den Prozess, bei dem die Signale mehrerer (verschiedenartiger) Sensorquellen zusammengeführt und vorverarbeitet werden, mit der Absicht ein *besseres* Gesamtsystem zu erzeugen, als es jeder einzelne Sensor sein könnte [SK08, Kapitel 25, Seite 585ff.]. *Besser* kann hierbei einen Gewinn an Genauigkeit, Zuverlässigkeit oder nach einem anderen Kriterium bedeuten.

Daten von IMUs können zusätzlich mit Sensordaten, welche absolute Positionen oder Orientierungen in Bezug auf ein (Welt-)Koordinatensystem liefern, fusioniert werden. Ein Magnetometer kann als obere Schranke des Orientierungsmessfehlers einer IMU dienen. Um den Positionsfehler zu minimieren, können Daten von GPS-Empfängern und IMUs fusioniert werden. Im Automobilbereich ist auch die Fusion anderer Daten üblich, um "intelligente" Fahrzeuge zu konstruieren [HLGS08].

Dies sind Beispiele für *direkte* Sensorfusion. Die Experimente (auf Seite 96 in 6.3.4) zur Bestimmung, ob eine Wiimote in der Hand gehalten wird, oder auf einem Tisch liegt, benutzen auch menschliches Kontextwissen, sind also damit eine *indirekte* Multi Sensor Datenfusion.

Das Ergebnis einer Multisensor Datenfusion wird *Lagebild* genannt. Für dieses Lagebild ist es von entscheidender Bedeutung, wie die zu fusionierenden Informationen interpretiert, sowie gewichtet werden und wie deren Aktualität berücksichtigt wird.

Üblicherweise unterscheiden sich die Verfahren zur Sensorfusion in ihrem Umgang mit dieser Informationsgewichtung, Informationsinterpretation und dem *Information Aging*.

Häufig anzutreffende Multisensorfusionsmethoden sind:

- Gewichtetes Mittel: Es können *arithmetische*, *geometrische* oder *harmonische* Mittelwerte gebildet werden, in deren Bildung die Daten der unterschiedlichen Sensorquellen unterschiedlich gewichtet einfließen. Die Gewichtung kann auch noch von anderen Parametern abhängen und sich im Verlauf ändern. So kann es manchmal sinnvoller sein das Gyroskop, manchmal das Accelerometer stärker zu gewichten. In dieser Arbeit werden sogar die Gewichte 0.0 (gar nicht) und 1.0 (ausschließlich) benutzt, abhängig von der Stellung der Wiimote.
- Kalman-Filter: Wenn sowohl die mathematische Struktur des Systems, als auch die auftretenden Messfehler bekannt sind, so eignet sich das Kalman-Filter gut, um Daten aus mehreren Quellen zu fusionieren [TBF05][Seite 40ff.]. Hierbei kann für eine Wiimote ein lineares Verhalten mit bestimmten Wahrscheinlichkeiten für bestimmte Punkte und eine stochastische Normalverteilung der Messfehler (Rauschen) angenommen werden.
- Methoden der kleinsten, gewichteten Quadrate: Dieses Verfahren eignet sich nicht nur dazu, Parameter einer (linearen) Funktion zu bestimmen, sondern kann unter der Annahme einer bekannten Funktion auch für die Fusion von Daten benutzt werden. So kann man annehmen, dass die Messdaten von einer Wiimote während eines Experimentes einer linearen Funktion folgen und die letzten n-Messpunkte mittels dieses Verfahrens (linear) approximieren.
- Andere Verfahren: Neben diesen Verfahren können Sensordaten mittels der Fuzzy-Set-Theorie mit der Fuzzy-Logik, mittels bayesscher Verfahren [Pun99], Dempster-Shafer-Verfahren oder neuronalen Netzen fusioniert werden.

Sowohl Kalman-Filter, gitterbasierte Ansätze[3], sequentielle Monte Carlo Verfahren, als auch Dempster-Shafer-Verfahren lassen sich mittels der allgemeinen bayesschen Gleichungen modellieren [KC03]. In dieser Arbeit wird von Verfahren auf Basis neuronaler Netze kein Gebrauch gemacht.

Pendelbasierte Roboterplattformen ("Segway-artige Roboter") machen oft von Kalman-Filtern Gebrauch im hochfrequenten Bereich bei den Gyroskopdaten und im niederfrequenten Bereich für die Accelerometerdaten [LJ09]. Somit lässt sich eine schnell-reaktive Plattform realisieren, die die Neigung in Echtzeit ausgleicht. Manche der oben genannten Verfahren sind bereits in Mikrochips realisiert, wie im Motion Processor MPU-6000 von InvenSense.

[3] bei denen die Gitterzellen selbst Samples der zugrundeliegenden Wahrscheinlichkeitsverteilung für jeden Zustand sind

2.1.3 Gestenerkennung

Intuitiv ist *Information* nötig, damit aus einer Bewegung eine Geste werden kann. Kurtenbach und Hulteen [KH90] unterscheiden zwischen einem "Goodbye-Winken" und dem Druck auf einen Knopf. Beides sind Bewegungen, aber nur die erstere von beiden hat auch eine Semantik und kann damit überhaupt Information tragen. Harling und Edwards [HE97] fordern für eine Geste nicht einmal Bewegung. Statische Gesten werden somit auch zulässig. In der Mensch-Computer-Interaktion interessiert man sich vorrangig für Gesten der Arme, bzw. Hände und des Kopfes. Gesten können prinzipiell durch am Körper getragene Sensoren (**gerätebasiert**) oder durch externe Sensoren (meist **kamerabasiert**) erkannt werden. Bei der Gestenerkennung durch externe Sensoren werden oft Kameras eingesetzt und zwar bis zu vier Kameras für eine Szene. Mit Mitteln der Bildverarbeitung kann dann die Position und Körperhaltung des Menschen berechnet und in sinnvolle Gesten mit einer Semantik umgesetzt werden. Mit der Kinect-Kamera (Abschnitt 1.4) und passender Software kann solch ein Ansatz realisiert werden.

Zu den am Körper getragenen Sensoren zählen Datenhandschuhe, welche sehr gute Erkennung von Handgesten bzw. Fingergesten ermöglichen, wie unter anderem die Diplomarbeit von Hanno Scharfe [Sch10] gezeigt hat. Allerdings sind für das Erkennen von Armgesten Inertialsensoren (Abschnitt 2.2) erforderlich. Falls diese, wie die Wiimote mit der eingesteckten MotionPlus Erweiterung, nicht nur aus Accelerometern, sondern auch aus Gyroskopen bestehen, eignen sie sich besonders gut zur Erkennung von Armgesten. Um Gesten mittels Inertialsensoren zu erkennen, geht man prinzipiell wie folgt vor:

- Im ersten Schritt wird eine Frequenzanalyse des Zeitsignals durchgeführt. Das Erkennen von Gesten im Frequenzbereich ist nämlich deutlich einfacher, als im Zeitsignal.
- Im zweiten Schritt muss die ausgeführte Geste mit allen Gesten verglichen werden, welche im System vorhanden sind. Diejenige "Mustergeste", welche den höchsten Wert in einem definierten Ähnlichkeitsmaß besitzt, wird dann als vermutlich ausgeführte Geste angenommen. Ist die gemessene Ähnlichkeit zu klein, so kann die ausgeführte Geste nicht erkannt werden.
- Falls die Mustergesten nicht bereits mit einer Library oder dem erkennendem System geliefert werden, so müssen sie zuerst erlernt werden. Dazu werden die unterschiedlichen zu erkennenden Gesten am Besten mehrfach vorgemacht und das System muss sie lernen. Dieser Schritt ist optional, aber falls das System lernt, so muss es das natürlich zwischen den beiden vorherigen Schritten tun.

Als Lernstrategien kommen in Frage: Bayessche Netze, Hidden-Markov-Modelle (HMM) oder neuronale Netze. Wobei die drei Vorgehensweisen keineswegs disjunkt sind. So werden oft neuronale Netze als Hidden-Markov-Modelle implementiert, damit sie Gesten lernen. Andererseits eignen sich Markov-Ketten für eine approxima-

tive Inferenz von Bayesschen Netzen. Die Bibliotheken **GlobePIE** und **WiiGee** haben für ihre Gestenerkennung zunächst Bayessche Netze benutzt und stiegen zunehmend auf neuronale Netze um. GlovePIE hat von Haus aus einige Gesten, wie "Drehung im Uhrzeigersinn" und "Drehung gegen den Uhrzeigersinn", bereits eingebaut. Diese müssen natürlich nicht neu erlernt werden.

2.1.4 Algorithmen zur Fouriertransformation

Für manche Anwendungsfälle kann es nützlich oder nötig sein, dass man vorliegende Signale nicht im Zeitbereich, sondern im Frequenzbereich betrachtet und analysiert. Die **Fouriertransformation** [Fli91, GRS05] ist dabei das Mittel der Wahl. Nach Anwendung der Transformation kann man in der Fouriertransformierten die Frequenzanteile deutlich sehen.

Leider eignet sich die allgemeine Formel der (kontinuierlichen) Fouriertransformation

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{2.15}$$

nicht für zeitdiskrete Signale, wie sie von der Wiimote geliefert werden. Die Ursache liegt in der kontinuierlichen Frequenzvariable ω. Also muss diese abgetastet werden und kann dann durch eine diskrete Variable **k** ersetzt werden, welche auf einer diskreten Zeitvariablen **n** basiert. Auf diese Weise kommt man zu einer diskreten Fouriertransformation (DFT) wie in [DSN10, Seite 143ff.] beschrieben. Um mit DFTen rechnen zu können, ist es wichtig, dass die Abtastdauer des diskreten Signals nicht kürzer sein darf, als die Dauer des kontinuierlichen Signals, welches von Interesse ist, und das Abtasttheorem von Nyquist-Shannon nicht verletzt wird. Da die Abtastrate bei der Wiimote vorgegeben ist (10ms), kann man nur die Anzahl der Samples, also die Fenstergröße des diskreten Signals variieren.

Früher war die Berechnung der DFT aufwendig und langsam, nämlich mit einer Komplexität von N^2 für N Samples. Dies ergibt sich aus der Zahl der für die Berechnung nötigen komplexen Multiplikationen. Der Aufwand wuchs also quadratisch mit der Zahl der Abtastpunkte und machte damit das Verfahren für größere N nicht attraktiv. Als Cooley und Tukey im Jahr 1965 [CT65] einen Algorithmus vorgestellt haben, welcher die diskrete Fouriertransformierte verhältnismäßig schnell berechnen konnte (Komplexität $Nlog_2N$), änderte sich dies grundlegend. Das war der Grundstein für die Verfahren, welche in die Klasse der (diskreten) schnellen Fouriertransformationen (Fast Fourier Transform, abgekürzt FFT) fallen.

Der Algorithmus von Cooley und Tukey ist ein Radix-2-Algorithmus, d.h. er arbeitet auf einer Samplegröße, welche die Größe s^2 haben muss. Die Radix-2^n-Algorithmen reduzieren die Zahl der (komplexen) Multiplikationen deutlich (Radix-4 um 25%

und Radix-8 um ca. 40%), machen sie aber weniger flexibel in der Wahl der Fenstergröße. Sie muss natürlich von einer Größe s^{2^n} sein. Hierbei ist n vorgegeben durch die Radix (n=2 bei Radix-4 und n=3 bei Radix-8 Algorithmen) und s kann frei gewählt sein. Je effizienter der Algorithmus, welcher gewählt wurde, desto unflexibler die Fenstergröße, so dass man entweder geneigt ist kleinere Fenster zu wählen, oder Signale verarbeiten muss, welche keinen Beitrag leisten. Kurz zur Veranschaulichung: Zulässige Fenstergrößen sind bei Radix-8: 8,64,512,4096,32.768 oder 262.144 gesammelte Samples. Sollte man nun 267.000 Samples verarbeiten wollen, muss das Fenster entweder kleiner gewählt oder geteilt werden oder man wählt die nächstgrößere Potenz von 8. Dies ist aber ein extremes Beispiel. Im Allgemeinen arbeiten die im Mittel sehr effizient und für unsere Zwecke reicht der originale Radix-2-Algorithmus vollkommen aus.

Es gibt auch Algorithmen für beliebiges N, falls N keine Primzahl ist ([Sin69]). Die Fensterbreite wird dann in Teiler von N zerlegt und die einzelnen Teilstücke werden bearbeitet. Trivialerweise gilt:

$$N = N_1 N_2 N_3 ... N_l \tag{2.16}$$

Algorithmen für beliebiges N sind in vielen praktischen Fällen effizient, programmiertechnisch jedoch z.T. sehr viel aufwendiger zu implementieren [Rab79].

In der vorliegenden Diplomarbeit werden Radix-2-Algorithmen benutzt, welche auf bereits gesammelten Daten (offline) arbeiten. Damit tritt die Frage nach Effizienz in den Hintergrund. Es ist lediglich auf die Wahl der Fenstergröße zu achten. Ob die günstig war, kann man daran erkennen, ob das Signal, welches sich aus der invertierten DFT (IDFT) ergibt, mit dem Originalsignal identisch ist. Die diskrete Fouriertransformation wird in der vorliegenden Arbeit benutzt, um Experimente auszuwerten und auf Anwendbarkeit im Bereich der Gestenerkennung zu überprüfen. Es ist die Erwartung da, dass sich mittels der DFT die Daten der Wiimote noch besser auswerten und damit in weiteren Bereichen anwenden lassen. Für eine echtzeitfähige Verarbeitung der Gesten müssten weitergehende Experimente mit verschiedenen der hier vorgestellten Ansätze durchgeführt werden.

2.2 Inertialsensoren

Der Begriff *Inertialsensoren* bildet einen Oberbegriff für Sensoren, welche auf Basis der Massenträgheit Kräfte messen können. Meist sind das translatorische und rotatorische Beschleunigungskräfte. Für translatorische Kräfte sind das Beschleunigungssensoren (Accelerometer) und für rotatorische Kräfte sind es Drehratensensoren, welche manchmal auch Kreiselinstrumente[4] bzw. Gyroskope genannt werden.

[4] streng genommen nicht ganz korrekt

Beide Sensortypen haben eine lange Entwicklung hinter sich und es gibt sie in verschiedenen Bauarten und unterschiedlichen Empfindlichkeiten mit unterschiedlicher Messgenauigkeit.

Neben den größeren mechanischen Sensoren, welche Eigenschaften aus der Mechanik oder aber auch Optik für ihre Funktion nutzen, gibt es auch mikrosystembasierte Sensoren, welche heutzutage die meiste Verbreitung besitzen. Zu den Inertialsensoren werden oft auch das Inklinometer und das Magnetometer gezählt, welches im eigentlichen Sinne mit Massenträgheit nichts zu tun hat. Es folgt zunächst die Beschreibung der Beschleunigungssensoren und der Drehratensensoren.

2.2.1 Beschleunigungssensoren (Accelerometer)

Den einfachsten, und damit grundlegenden Fall eines Accelerometers bildet ein **Federpendel**[5], wie es in Abb.2.1 zu sehen ist. Es besteht aus einer Masse, welche an einer Feder befestigt ist und schwingen kann und einer Dämpfungsvorrichtung[6]. Bei einer Krafteinwirkung folgt das Verhalten der elastischen Feder dem Hookeschen Gesetz[7] und verformt sich damit proportional zur einwirkenden Belastung. Da ein Federpendel nach Krafteinwirkung einen harmonischen Oszillator bildet, welcher nach einer gewissen Zeit durch Gravitation und Reibung in eine Ruhelage einpendeln würde, ist es nicht gerade optimal zum Bestimmen von Beschleunigungskräften. Es ist aber geeignet, um das Funktionsprinzip von Beschleunigungssensoren zu verdeutlichen. Falls die Auslenkung eines Federpendels gemessen wird, so ist die anfängliche Amplitude proportional zur beschleunigenden Kraft. Es gilt:

$$F = m \cdot a \tag{2.17}$$

Wobei **F** die beschleunigende Kraft, welche gemessen wird, repräsentiert, und **m** der beschleunigten Masse und **a** der Beschleunigung entsprechen.

Dieses mit einer Dämpfungsvorrichtung versehene Federpendel bildet zusammen mit der einwirkenden Kraft ein klassisches mechanisches Schwingungssystem 2. Ordnung, für das sich nach der Allgemeinen mechanischen Schwingungsgleichung die Gleichung 2.17 zu

$$F_{Wirkend} - F_{Dämpfungsvorrichtung} - F_{Feder} = m \cdot \ddot{x} \tag{2.18}$$

$$F_{Wirkend} = m\ddot{x} + c\dot{x} + kx \overset{2.17}{=} m \cdot a \tag{2.19}$$

[5] Gelegentlich auch als *Feder-Masse-Schwinger* bezeichnet

[6] z.B. einem Kolben, welcher sich durch eine zähe Flüssigkeit bewegen kann (Mechanische Schwingung)

[7] Ein Sonderfall des **Elastizitätsgesetzes**, welches ein Spezialfall des **Stoffgesetzes** der Physik ist

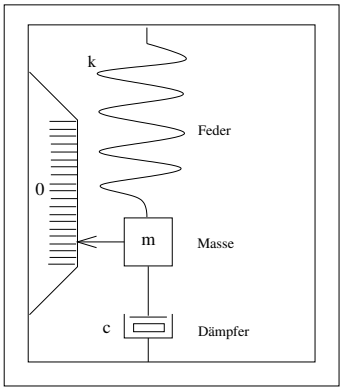


Abbildung 2.1: Ein einfaches Federpendel

umformen lässt. Hierbei ist die Beschleunigung **a** aus Gleichung 2.17 als 2.Ableitung der Längenänderung des Pendels, abgeleitet nach der Zeit, in Gleichung 2.18 und 2.19 mit $\ddot{x}$ dargestellt. In Gleichung 2.19 ist **c** der Dämpfungskoeffizient der Dämpfungsvorrichtung, **k** die Federkonstante und **x** die gemessene Längenänderung der Feder. $\dot{x}$ und $\ddot{x}$ sind dann wie schon erwähnt die 1. und 2. Ableitung von x nach der Zeit. Natürlich ist die Umstellung der Gleichung 2.19 einfach und ergibt:

$$a = \ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x \tag{2.20}$$

Da das Federpendel ein lineares, zeitinvariantes System[8] (LZI) bildet, kann mittels Laplacetransformation [GRS05, Fli91] ihre Übertragungsfunktion berechnet werden zu:

$$G_m(s) = \frac{x(s)}{\ddot{x}(s)} = S_m \frac{\omega_n^2}{s^2 + 2D\omega_n s + \omega_n^2} \tag{2.21}$$

wobei $\omega_n = \sqrt{k/m}$ die Resonanzfrequenz des Systems ist, $D = \frac{1}{2}c\sqrt{k/m}$ die Dämpfungsgrad und $S_m = m/k$ die mechanische Sensitivität sind [AL02]. Wie bei allen LTI-Systemen solcher Art, hängt das Einschwingverhalten des Systems vom Dämpfungsgrad ab. Je höher er gewählt ist, desto schneller pendelt sich das System nach einer Krafteinwirkung wieder ein.

MEMS Accelerometer

Mikrosystembeschleunigungssensoren haben sich in den letzten Jahren weit auf dem Markt verbreitet. Eine Ursache ist sicherlich in der steigenden Nachfrage der Automobilbranche zu sehen, welche unter anderem ihre Airbags mit Sensoren dieser

[8]Englisch: LTI - linear time-invariant system

Bauart ausgestattet hat. Das hat einen bedeutenden Beitrag zur günstigen Massenfertigung dieser Sensoren geleistet. Diese wiederum machte sie für die Spiele(konsolen)industrie und die Mobilfunkbranche verfügbar. Die letzten beiden haben Inertialsensoren in großem Maßstab produziert und fördern damit neben der Verbreitung auch deren Präzision stark.

Alle Mikrosystembeschleunigungssensoren bilden das Prinzip des Feder-Masse-Pendels (Abb. 2.1) auf die eine oder Art ab. Es sind grundsätzlich Strukturen aus Silizium, welche oft auch in Silizium eingekleidet sind, von denen eine die Feder abbildet und die andere die Masse. Üblicherweise werden MEMS-Accelerometer auf Basis einer von dreien physikalischen Eigenschaften produziert. Es gibt **kapazitive Accelerometer**, **piezoelektrische Accelerometer** und **piezoresistive Accelerometer**. Die piezoelektrischen Accelerometer brauchen leider für ihre Funktion Wechselstrom. Diese Inkompatibilität mit Gleichstrom macht sie für die Anwendung im Bereich kleiner inertialer Messeinheiten für Robotik, Automobil, Spiele und Mobilgeräte nahezu unbedeutend.

Piezoresistive Accelerometer

Diese Beschleunigungssensoren basieren auf dem piezoresistiven Effekt, welcher die Veränderung des elektrischen Widerstandes eines Material beschreibt, wenn Druck oder Zug drauf ausgeübt wird. Dieser Effekt wurde Anfang des 20. Jahrhunderts entdeckt und Accelerometer, welche diesen Effekt ausnutzten wurden Ende der 1970 Jahre hergestellt und zeigten damit etwas, was bis dahin als unwahrscheinlich galt, nämlich, dass siliziumbasierte MEMS-Beschleunigungssensoren mit einer angemessenen Genauigkeit herstellbar sind.

Kapazitive Accelerometer

Unter den Beschleunigungssensoren aus Basis der Mikrosystemtechnologie sind diese am meisten verbreitet. Der LIS3L02AL Beschleunigungssensor der Nunchuk (3.1.3) ist ein kapazitives Accelerometer, wie man in seinem Datenblatt (Anhang A.3) bereits auf der ersten Seite sehen kann. Obwohl das Datenblatt dazu nichts sagt, ist der Beschleunigungssensor der Wiimote (ADXL330) mit Sicherheit auf der selben Technologie gefertigt, wie der ADXL05[9] und der ADXL50[10] und damit also kapazitiv.

Bei kapazitiven Accelerometern, ändert ein Feder-Masse System die kapazitiven Eigenschaften von Kapazitätsplatten. Die Änderungen in den Kapazitäten werden

[9] Messbereich: ±5g
[10] Messbereich: ±50g

gemessen und aus ihnen kann ein Ausgabesignal abgeleitet werden, welches die einwirkende Kraft und damit die Beschleunigung repräsentiert. Ein eine Art der Anordnung besteht darin, dass in einer kleinen Siliziumstruktur ein Trägerbalken an seinen Enden an zwei Federn befestigt wird, so dass er frei schwingen kann. An diesem Balken werden metallische Platten befestigt, welche kapazitive Eigenschaften besitzen. Diese Struktur sieht aus wie ein Kamm und wird oft auch in der Fertigung so genannt. Die Zinken des Kammes bestehen aus den eben genannten Platten, welche Kapazitäten darstellen. Zwischen den Zinken dieses beweglichen Kammes werden liegen an der Wand festmontierte Platten, so dass auf eine bewegliche Platte immer eine feste folgt und umgekehrt. Diese Struktur sieht aus, wie zwei ineinander gesteckte Kämme, von denen einer fest ist und der andere sich, an zwei Federn befestigt, entlang einer Bewegungsachse frei bewegen kann.

Durch eine Beschleunigungskraft verschiebt sich der Abstand zwischen den beweglichen und festen Platten (Zinken der Kämme) und eine bestimmte Anzahl von ihnen, abhängig von der einwirkenden Kraft, kann sich berühren. Diese Berührung ändert die kapazitiven Eigenschaften des Systems und daraus kann die wirkende Kraft abgeleitet werden.

2.2.2 Drehratensensoren (Gyroskope)

Gyroskope werden benutzt, um Änderungen der Orientierung eines Körpers zu messen. Das geschieht, indem physikalische Gesetze benutzt werden, welche zu vorhersagbaren und berechenbaren Effekten bei einer Rotation führen. Grundsätzlich wirkt eine Kraft auf einen Körper ein, welche ihn entlang einer Kreisbahn zu bewegen versucht. Die Wirkung dieser Kraft kann gemessen werden. Man kann dazu mechanische, optische oder Mikrosystem-Gyroskope (MEMS - englisch: Micro-Electro-Mechanical System(s)) benutzen. Sie alle unterscheiden sich in der Art, welche physikalischen Gesetzmäßigkeiten sie benutzen, und welche Genauigkeiten sie erreichen können. Die MEMS-Technologie ist hierbei von besonderem Interesse, weil solche Sensoren in der Wiimote verbaut sind.

Mechanisches Gyroskop

Grundsätzlich basieren mechanische Gyroskope (und Gyrokompasse) auf dem Prinzip der Erhaltung des Drehimpulses [SK08, Kapitel 20]. Ein Drehimpuls ist hierbei die Neigung eines sich drehenden Objektes sich mit der selben Winkelgeschwindigkeit und an der gleichen Raumachse zu drehen, bei gleichzeitigem Fehlen von externen Drehmomenten. Der Drehimpuls I eines Objektes wird bei gegebenen Trägheitsmoment T und Winkelgeschwindigkeit ω berechnet wie folgt:

$$I = T \cdot \omega \tag{2.22}$$

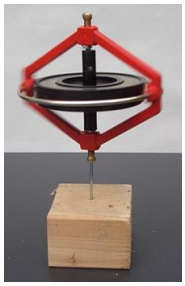


Abbildung 2.2: Das einfachste mechanische Gyroskop: ein Kreisel

Man kann ein mechanisches Gyroskop als ein geschlossenes System ansehen in dem ein Kreisel versucht seinen Drehimpuls zu erhalten. Greift an diesem Objekt eine Kraft an, welche nicht rechtwinklig oder parallel zur Drehachse ist[11], so versucht der Kreisel seinen Gesamtimpuls zu bewahren und kippt die Drehachse senkrecht zur angreifenden Kraft. Diese Änderung der Position wird in mechanischen Gyroskopen gemessen und meist in eine Winkelgeschwindigkeit umgerechnet.

Kreiselkompass Der Kreiselkompass, auch Gyrokompass genannt ist ein sich schnell drehender Kreisel, welcher in einer kardanischen Aufhängung geeignet gelagert ist. Er wird so angebracht, dass auf ihn ein äußeres Drehmoment einwirkt, solange er seine Rotationsachse nicht parallel zur Rotationsachse der Erde ausgerichtet hat und damit die Nord-Süd Richtung anzeigt. Leider konvergiert ein Kreiselkompass nicht sofort, sondern pendelt sich nach Überschwingungen mit kleiner werdender Amplitude im stabilen Zustand ein [SK08, Kapitel 20]. Für langsame Fahrzeuge, wie langsam fahrende Schiffe, ist ein solcher Kompass, welcher vollkommen unabhängig von magnetischen Einflüssen ist, sondern lediglich auf der Rotationsbewegung der Erde basiert, eine gute Ergänzung der anderen Instrumente oder kann sogar als einziger Kompass benutzt werden.

Optisches Gyroskop

Optische Gyroskope basieren auf der Ausnutzung des Sagnac-Effektes [Ste97]. Georges Sagnac hat ihn beschrieben und wollte den Nachweis der Existenz des Licht-Äthers erbringen [Sag13]. Ableitend aus seiner Forschung ist es nun möglich eine Rotation absolut, also ohne Bezugssystem, zu messen. Die Grundidee basiert auf der Tatsache, dass die Ausbreitungsgeschwindigkeit von Licht in einem Medium unabhängig davon ist, ob das Medium selbst sich bewegt. Für den Versuchsaufbau wird ein Lichtstrahl in zwei Strahlen geteilt, welche in entgegengesetzten Richtungen entlang eines Kreises geschickt werden und später wieder am Anfangspunkt gemessen

[11] beides hätte überhaupt keine Auswirkungen auf die Drehachse

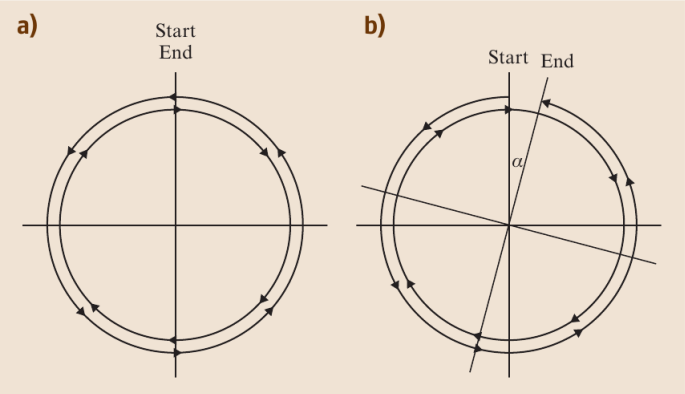


Abbildung 2.3: Optisches Gyroskop

werden.

Wenn die Strahlen eine stationäre Bahn mit Umfang $U = 2\pi \cdot r$ umkreisen, so werden sie die selbe Entfernung mit gleicher Geschwindigkeit (Lichtgeschwindigkeit **c** im entsprechendem Medium) zurücklegen und am Start/Zielpunkt nach einer Zeit $t = U/c$ ankommen. Sollte während die Lichtstrahlen unterwegs sind, sich das System um den Mittelpunkt des runden Lichtpfades (Kreises) mit einer Umdrehungsfrequenz (Drehzahl) ω drehen, so wird der Weg für den Lichtstrahl, dessen Umlaufsinn dem Drehsinn entspricht, länger. Wohingegen der Weg für den Strahl, welcher entgegen dem Drehsinn unterwegs ist, kürzer wird. In Abb. 2.3 wird der Aufbau verdeutlicht.

Es ergibt sich für den längeren Weg ("Umfang" des Kreises) $U_l = 2\pi \cdot r + \omega \cdot t_l$, wobei t_l die Zeit ist, die für die längere Wegstrecke vom Licht gebraucht wird. Für die kürzere Strecke ergibt sich dann: $U_k = 2\pi \cdot r - \omega \cdot t_k$, wobei hier t_k die Zeit ist, die der andere Lichtstrahl für die kürzere Strecke braucht. Da gilt $U_l = c \cdot t_l$ und $U_k = c \cdot t_k$ und $t_l = 2 \cdot \pi \cdot r(c + \omega \cdot r)$ und $t_k = 2 \cdot \pi \cdot r(c - \omega \cdot r)$ folgt für die Zeitdifferenz $\Delta t = t_l - t_k$ die Formel:

$$\Delta t = 2\pi \cdot r\left(\frac{1}{c - \omega \cdot r} - \frac{1}{c + \omega \cdot r}\right) \tag{2.23}$$

Diese Zeitdifferenz kann im Zielpunkt gemessen werden und aus ihr die Umdrehungsfrequenz berechnet werden.

Da Δt sehr klein ist und damit eine Quelle von Messungenauigkeiten bildet, werden oft kilometerlange Glasfaserkabel in optischen Gyroskopen benutzt, welche sie auf ein Gewicht von 2-3 Pfund und eine mittlere Größe bringen. Die Genauigkeit von optischen Gyroskopen ist aber den anderen beiden Techniken überlegen.

Eine andere Möglichkeit der Messung besteht darin, nicht die Umlaufzeit, sondern den Phasenversatz zu messen. Dabei wird mit einem Laser eine stehende Welle

erzeugt und die Phaseninterferenz im Messpunkt bestimmt und aus ihr kann die Drehung des Systems errechnet werden..

Während der Aufbau des Originalexperimente in welchem weißes Licht und Spiegel benutzt wurden noch ziemlich groß war, schrumpfen heutzutage die optischen Gyroskope, in denen meist polarisiertes Laserlicht in verschiedenen Wellenlängen benutzt wird auf zunehmend kleine Größen bei verbesserter Genauigkeit. Die Gruppe um Prof. Jacob Scheuer [Sch09] hat offenbar das Ziel ein laserbasiertes, optisches Gyroskop mit der Größe eines Stecknadelkopfes für den Massenmarkt, speziell Smartphones, zu entwickeln.

Mikrosystemgyroskope - MEMS

Mikrosystemgyroskope (englisch: micro-elektromechanical system(s) - MEMS) nutzen entweder den piezoelektrischen Effekt aus oder bedienen sich vibrierender Strukturen. Gyroskope der letztgenannten Gruppe sind unter den Mikrosystemgyroskopen die weitaus häufigsten. Ihr Aufbau basiert auf Energietransfer, welcher begründet durch die Coriolisbeschleunigung, zwischen den Vibrationsmodi auftritt.

Auf einen Körper, welcher sich in einem mit einer Drehrate ω in Bezug auf ein Inertialsystem drehenden Bezugssystem mit lokaler Geschwindigkeit $\vec{v}$ geradlinig bewegt, wirkt eine Corioliskraft $\vec{F_c}$ mit:

$$\vec{F_c} = 2m \cdot (\vec{v} \times \vec{\omega}) \tag{2.24}$$

Da die Masse m der Messeinheit auf die die Kraft $\vec{F_c}$ wirkt, als bekannt vorausgesetzt werden darf, kann aus der Gleichung 2.24 für die Corioliskraft, durch Anwendung von Gleichung 2.3 eine massenunabhängige Gleichung für die Coriolisbeschleunigung wie folgt formuliert werden:

$$\vec{a_c} = 2 \cdot (\vec{v} \times \vec{\omega}) \tag{2.25}$$

Das Messprinzip in MEMS-Gyroskopen basiert darauf, dass eine lokale, lineare Geschwindigkeit erzeugt wird und die durch die Rotation der Messeinheit, und damit des Bezugssystems, entstehende Corioliskraft gemessen wird. Frühere MEMS-Gyroskope haben vibrierende Quarzkristalle benutzt, um die nötige lineare Bewegung zu erzeugen. Heutzutage werden die vibrierenden Strukturen in Silizium gegossen [SK08, 20.2.3]. Es folgt ein Auszug aus den vielen unterschiedlichen vibrierenden Strukturen für MEMS-Gyroskope.

Stimmgabelgyroskop Diese Gyroskope[12] nutzen eine vibrierende Struktur, welche aussieht wie eine Stimmgabel. Durch eine Rotation des Bezugsrahmens werden

[12] Auch als *tuning fork gyroscopes* bekannt

die Zacken der Gabel aus der Ebene gedrückt. Diese Veränderung wird gemessen und findet unter Anderem bei [FHA98] in Form des *InertiaCube* im Bereich des Kopfbewegungstrackings Anwendung.

Gyroskop auf Basis eines vibrierenden Rades Diese Gyroskope[13] basieren darauf, dass eine radförmige Struktur um ihre Rotationsachse oszilliert. Eine Rotation des externen Bezugsrahmens führt dazu, dass die radähnliche Struktur kippt bzw. sich neigt. Diese Neigung kann gemessen und aus ihr die Drehrate berechnet werden.

Weinglasresonatorgyroskop Die Teile dieser Gyroskope (Englisch: *Wine-Glass Resonator Gyroscopes*) schwingen streng genommen nicht selbst, sondern werden zur Schwingung angeregt. Durch die Einwirkung der Corioliskraft entsteht in dieser Struktur eine Resonanz, welche am Knotenpunkt der einzelnen Bögen gemessen werden kann. Aus dieser Resonanz kann auch hier die Rotation errechnet werden.

Zusammenfassung Die Vorteile der in MEMS-Technologie gefertigter Gyroskope sind ihre geringe Größe, das geringe Gewicht sowie die zunehmend geringer werdenden Kosten. Mit steigender Genauigkeit und einem geringen Stromverbrauch haben sie mechanische und optische Gyroskope aus dem Gebiet der Robotik weitgehend verdrängt. Sie entwickeln sich auch in anderen Bereichen zu den dominierenden Sensortypen, da sie keine rotierenden Teile besitzen, und damit ein Verschleiß, wie es bei Mikrochips üblich ist, faktisch nicht vorhanden ist.

Bei MEMS-Gyroskopen liegt ein sehr gutes Preis-/Leistungsverhältnis vor, im Vergleich zu den anderen Herstellungsarten. Ob eine Technologie prinzipiell Besseres leisten kann, als eine andere, kann hier nicht gesagt werden. Am Beispiel der mechanischen Gyroskope kann man zunächst im niedrig-preisigem Sektor sagen, dass mechanische Gyroskope nicht so genau sind, wie die anderen beiden Techniken. Andererseits kann man die Genauigkeit nahezu beliebig erhöhen, wenn man genug Geld in die Konstruktion investiert. Wenn man z.B. Quarzkugeln in der Größe eines Tischtennisballes, welche mit Niob beschichtet sind, bei 1,8 Grad K mit 10.000 U/min rotiert, so können Interferenzen des supraleitenden Quarzes gemessen werden und damit Messgenauigkeiten von 1/40.000.000 Grad erreichen. Mechanische Gyroskope dieser Qualität sind aber Institutionen wie der NASA und der Stanford University für Weltraummissionen, wie **Gravitiy Probe B** vorbehalten [FE08].

Bei optischen Gyroskopen erhöht sich die Präzision mit der Größe, dem Gewicht und dem Preis. So sind die MEMS-Sensoren ein sehr guter Kompromiss aus Preis-/Leistung und dem Verhältnis von Größe zu Genauigkeit. Vermutlich wird das der dominierende Sensortyp der Zukunft sein. Alle Gyroskoptypen haben gemeinsam,

[13] Englisch: *vibrating wheel gyroscopes*

dass sie nicht perfekt messen können. Eine, auch noch so kleine Messungenauigkeit, wird sich also über die Zeit kumulieren und zu einer Sensordrift führen. Diese Drift lässt sich mit den bisher genannten Sensoren, welche immer nur eine relative Positions- bzw. Bewegungsänderung messen können, nicht beseitigen. Dazu braucht man Sensoren, welche absolute Werte, wie die derzeitige Orientierung liefern können. Inertiale Messeinheiten, in denen absolute Positionsgeber mit den bisherigen Inertialsensoren mittels Sensorfusion gekoppelt wurden, können diese Drift eingrenzen.

2.2.3 Inertiale Messeinheit (IMU) und Trägheitsnavigationssystem (INS)

Das Kernstück eines Trägheitsnavigationssystems (*inertial navigation system* - INS) ist die inertiale Messeinheit (*inertial messurement unit* - IMU). Sie ist ein System, welches zur Bestimmung der Position und Orientierung von Objekten dient. Grundsätzlich hat eine IMU mindestens sechs Sensoren. Das sind drei Accelerometer, und drei Gyroskope welche orthogonal zueinander entlang der gedachten kartesischen Achsen der IMU angeordnet sind. Diese Daten können mit weiteren Sensordaten fusioniert und damit ergänzt werden (siehe dazu 2.1.2). Das können absolute Positionsgeber, beispielsweise das GPS[14] sein, oder Sensoren, welche einen bestimmten Fehler eingrenzen, wie das bei Magnetometern in Form von Kompassen der Fall ist. Diese können ihre Lage in Bezug auf das Magnetfeld der Erde mit einer Maximalabweichung bestimmen.

Am häufigsten werden inertiale Messeinheiten heutzutage auf Basis der MEMS-Technologie gefertigt und auf einem, oft sehr kleinen, Chip angebracht. In den folgenden Abschnitten wird davon ausgegangen, dass das betrachtete System nur Sensoren besitzt, die auch in einer Wiimote zu finden sind und keine weiteren.

Kardanisch aufgehängte Messeinheit

Eine Messeinheit in einer kardanischen Aufhängung[15] ist im optimalen Fall an den Achsen des Weltkoordinatensystems orientiert und liefert auch Werte, welche bereits in dieses Weltkoordinatensystem umgerechnet sind[16]. Für große Systeme, wie Wasser-, Boden-, oder Luftfahrzeuge, welche im Großteil der Betriebszeit ihre Orientierung nur wenig ändern, eignen sich solche IMUs besser, als in kleinen, wendigen Objekten, wie die Wiimote eines ist. In der Seefahrt werden solche IMUs bei langsamen Schiffen auch heute noch oft eingesetzt. Messeinheiten, die kardanisch aufgehängt sind, haben zwei Nachteile. Sie haben auf Grund der Aufhängung eine gewisse Größe und die Qualität der Aufhängung hat direkten Einfluss auf die Qualität der gemessenen Werte. Wie schon erwähnt, besteht der Vorteil in der fehlenden

[14] *global positioning system* - Ein System, welches mittels (fast) geostationärer Satelliten eine genaue Positionsbestimmung auf der Erde ermöglicht.

[15] englisch *gimbal IMU*

[16] Im theoretischen Optimalfall ist die Aufhängung reibungsfrei und reagiert instantan.

Umrechnung der Werte in ein Weltkoordinatensystem, bzw. in einem Orientierungsmessfehler, welcher sich nicht kumuliert, sondern als obere Schranke die Genauigkeit der kardanischen Aufhängung besitzt. Das heißt, sollte die IMU buchstäblich mal "aus dem Lot geraten", so ist die Maximalablenkung von vornherein klar und es ist zu erwarten, dass die Vorrichtung bei einer der nächsten Bewegungen des Systems, oder von ganz alleine, sich in ihre gewünschte, stabile Lage zurückbewegt.

Für die Lagebestimmung kann entweder die Auslenkung der kardanischen Aufhängung benutzt werden, oder aber Gyroskope werden am System selbst und nicht an der kardanischen Aufhängung der IMU befestigt. Größere und teurere kardanisch aufgehängte Messeinheiten können auch in drei Achsen kreiselstabilisiert gelagert werden. Solche Geräte sind in der Raumfahrt verbreitet und können eine Steuerungsgrundlage für Bewegungen von Körpern im All bilden.

Die Bestimmung der Bewegung kann so genau sein, dass teure inertiale Navigationssysteme nur auf ihrer inertialen Messeinheit basierend sowohl die Eigendrehung der Erde (15°/h) als auch die Umdrehung um die Sonne (0,041°/h) herausrechnen können. Dass ihre Genauigkeit um ein Vielfaches wächst, bei Zuhilfenahme von absoluten Positionsgebern, wie weiter oben erwähnt, ist leicht einsichtig.

Strapdown IMU

Bei einer *strapdown*[17] Messeinheit sind die Sensoren körperfest am System angebracht, dessen Beschleunigung oder Drehrate gemessen werden. Deshalb liefern sie Werte im Koordinatensystem des Messkörpers[18] und nicht im Weltkoordinatensystem, wie im kardanischen Fall. Um diese Messwerte in das Weltkoordinatensystem umzurechnen, muss ein deutlich höherer Aufwand betrieben werden. Abbildung 2.4 zeigt schematisch, wie das im Einzelnen aussieht.

Man sieht, dass die gelieferten Gyroskopwerte über die Laufzeit integriert werden müssen, um zu jedem Zeitpunkt eine möglichst präzise Näherung für die Orientierung der IMU zu haben. Nur bei bekannter Orientierung ist es möglich die Messwerte der Beschleunigungssensoren in das Weltkoordinatensystem zu projizieren. Je genauer die gemessene[19] Orientierung der tatsächlichen Orientierung entspricht, desto genauer kann der Einfluss der Gravitation gefiltert werden. Die Gravitation kann anhand der Messdaten der Beschleunigungssensoren alleine nicht erkannt werden. Nachdem der Einfluss der Gravitation optimalerweise beseitigt, aber zumindest minimiert wurde, kann eine globale Beschleunigung berechnet werden. Ab diesem Zeitpunkt gleicht das Vorgehen bei der strapdown IMU dem der kardanisch aufgehängten. Die Beschleunigung ins Weltkoordinatensystem umgerechnet muss integriert werden und mit der anfänglichen Initialgeschwindigkeit verrechnet

[17] englisch *strapdown* - flugkörperfest, fahrzeugfest
[18] Im sogenannten *body frame*
[19] man kann hier auch von einer Schätzung sprechen

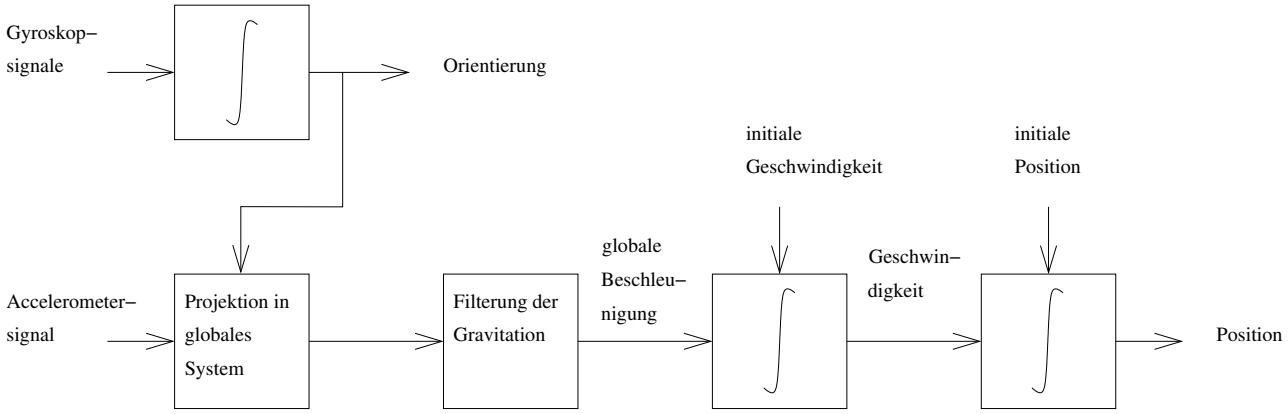


Abbildung 2.4: Schematisches Funktionsprinzip einer Strapdown-IMU

werden, damit man die aktuelle Geschwindigkeit bekommt. Die aktuelle Geschwindigkeit ihrerseits muss dann noch einmal integriert werden und die Initialposition berücksichtigt werden, damit man die aktuelle Position berechnen kann.

Es ist leicht ersichtlich, dass bei diesem Vorgehen zwei Fehlerquellen vorhanden sind:

- Die fehlerhaft bestimmte Orientierung fließt direkt als Fehler ein. In Abb 2.4 entspricht dieser Fehler dem oberen Quadrat. Hinzu kommt, dass je ungenauer die Orientierung bestimmt wird, desto größer der Einfluss der fehlerhaft korrigierten Gravitation wird. Das entspricht der ganzen unteren Reihe aus der genannten Abbildung. Ein Fehler in der Orientierung kann sich innerhalb kürzester Zeit stark kumulieren und hat als Fehlerquelle auf Dauer den größten Einfluss [Kon00, Abschnitt 2] [Woo07, Abschnitt 6.2.3].
- Die zweite Fehlerquelle ist die doppelte Integration. Jede Messungenauigkeit leistet einen quadratischen Beitrag zum Messfehler. Die doppelte Integration sorgt dafür, dass kleine Messabweichungen in sehr kurzer Zeit, also im Sekundenbereich, bzw. bei günstigen Sensoren sogar im Subsekundenbereich, zu einem stark anwachsenden Fehler führen.

Falls man nicht in der Lage ist den Orientierungsfehler zu begrenzen, oder in gewissen Abständen mittels absoluter Orientierungsgeber zu minimieren, führt die Positionsbestimmung recht schnell zu einem *random walk* [Woo07, Abschnitt 6.1.3].

Die zweite Fehlerquelle ist natürlich nicht in der Messung, sondern in der Mathematik begründet. Hätte man eine optimale Messung[20], so würde die doppelte Integration keinen Beitrag zum Fehler leisten. Natürlich gibt es keine optimale Messung, also muss auch dieser Fehlerquelle Aufmerksamkeit geschenkt werden.

Da in den meisten strapdown Messeinheiten des low-cost Bereiches, wie sie sich

[20] die es per definitionem von *Messung* gar nicht geben kann, dass *Messen* immer Vergleichen bedeutet und die Messwerte bei genügend hoher Wiederholungszahl letztlich auf eine Normalverteilung hinauslaufen

auch in der Wiimote befinden, die Sensoren mittels MEMS-Technologie hergestellt werden, müssen diese beiden Fehlerquellen, insbesondere die Berechnung der Orientierung, bei der Herleitung der Bewegungsgleichungen, genauer betrachtet werden. Dies geschieht im folgenden Abschnitt.

2.2.4 Bewegungsgleichungen zur Bestimmung der Position und Orientierung einer inertialen Messeinheit

Im trivialen Fall sind die Bewegungsgleichungen für Beschleunigungssensoren leicht nachvollziehbar. Bei Vernachlässigung von Störeinflüssen und optimalen, reibungsfreien, kardanisch aufgehängten Beschleunigungssensoren ist der Fall einfach zu lösen. Es muss noch vorausgesetzt werden, dass man eine reibungsfreie und optimal arbeitende kardanische Aufhängung besitzt, welche ohne Zeitverzug sofort auf Änderungen der Orientierung ohne Überschwingen und Reibung reagiert.

In der Realität wird man so ein Gerät natürlich nicht zur Verfügung haben, aber für den trivialen Fall der Bewegungsgleichungen nehmen wir solch ein optimales Accelerometer an. Die Darstellung hier orientiert sich an [AL02, Abschnitt 2.4] für diesen trivialen Fall und nutzt [Woo07, Abschnitt 6] um den komplexeren Fall der strapdown Messeinheit zu entwickeln.

Da im einfachen Fall unser Accelerometer kardanisch aufgehängt ist, liefert es uns die Beschleunigungsdaten unseres Systems bereits in Weltkoordinaten. Diese Beschleunigungsdaten ($\vec{a}$) müssen nur noch über die Zeit (t) integriert werden, damit man daraus die Geschwindigkeit ($\vec{v}$) berechnen kann und nach einer Integration der Geschwindigkeit ($\vec{v}$) nach der Zeit (t) erhält man die Position des Systems. Damit ergibt sich für die Positionsbestimmung in erster Näherung die folgende Formel [AL02, 2.4.1.]:

$$Position\ p = \int_0^t \dot{p}(t)\ dt = \iint_0^t \ddot{p}(t)\ dt \tag{2.26}$$

Wobei p den Positionsvektor, $\dot{p}$ den Geschwindigkeitsvektor bezeichnen. Die Beschleunigung, mit $\vec{a}$ vektoriell dargestellt, welche von einem INS gemessen werden kann, ist hier mit $\ddot{p}$ als (doppelte) Ableitung des Positionsvektors dargestellt. Man beachte, dass hier von einer wirklichen Beschleunigung ausgegangen wird. Meist werden von Sensoren aber die Kräfte gemessen, welche am System wirken. In diesem Fall muss man die Gravitationskraft ($\vec{g}$) ebenfalls in der Gleichung berücksichtigen und diese aus den gemessenen Kräften heraus rechnen.

Bei einer strapdown-Messeinheit (siehe 2.2.3), welche lediglich körperfeste Kräfte misst, genau so, wie das die Wiimote macht[21], muss die Orientierung der Messeinheit bestimmt werden, damit der Einfluss des Vektors ($\vec{g}$) der Gravitationskraft

[21] Im Gegensatz zu bereits gefilterten und umgerechneten Beschleunigungen, wie im trivialen Fall

durch Addition eines entgegengesetzten Vektors ($-\vec{g}$), beseitigt werden kann. Dazu muss die Orientierung der Messeinheit und Bezug auf das Weltkoordinatensystem möglichst genau bestimmt werden, damit $-\vec{g}$ möglichst exakt errechnet werden kann. Auf den sehr ungünstigen Einfluss von falsch berechnetem $\vec{g}$ wurde bereits vorher eingegangen. Es wird im folgenden Abschnitt zunächst beschrieben, wie die Orientierung bestimmt werden kann. Danach folgen die auf ihrer Grundlage gebildeten Bewegungsgleichungen für die Geschwindigkeit und aus ihnen hergeleitete Positionsgleichungen.

Tracking der Orientierung einer strapdown IMU

Für die Darstellung der Orientierung eines Körpers sind zwei Koordinatensysteme notwendig. Das eine ist das körpereigene Koordinatensystem, welches in den Gleichungen mit dem Index k[22] bezeichnet wird. Das andere ist das Weltkoordinatensystem, welches in den Gleichungen mit dem Index g[23] versehen wurde. Die Beschreibung einer Orientierung ist nichts anders, als eine Umrechnungsvorschrift zwischen diesen beiden Koordinatensystemen, welche hier als Drehmatrix gegeben ist. (Es sind andere Repräsentationsformen möglich siehe dazu Abschnitt2.1.1. Für die Darstellungen im folgenden Abschnitt wurde [Woo07, Abschnitt 6] genutzt.)

Um die Orientierung einer IMU oder eines INS in Bezug auf ein globales Referenzsystem (Weltkoordinatensystem) zu tracken, wird die Winkelgeschwindigkeit $\omega_k(t) = (\omega_{kx}(t), \omega_{ky}(t), \omega_{kz})^T$, gegeben als zusammengesetztes Signal der inkrementellen Drehratensensoren, "auf-integriert"[24]. Nachfolgend wird die Orientierung als eine 3x3 Drehmatrix C dargestellt[25], bei der jede Spalte einen Einheitsvektor im Körperbezugssystem darstellt, welcher in den Koordinaten des globalen Bezugssystems ausgedrückt wird. Wenn Messfehler und Rundungsfehler vernachlässigt werden und die Drehmatrix somit eine perfekte Darstellung der Orientierung ist, dann kann die Vektorgröße $\vec{v}_k$ aus dem Bezugssystem des Körpers im globalen Bezugssystem ausgedrückt werden als:

$$\vec{v}_g = C \cdot \vec{v}_k \tag{2.27}$$

Da das Inverse einer Drehmatrix ihre Transponierte ist (siehe Abschnitt 2.1.1), kann die Inverstransformation zu Gleichung 2.27 wie folgt formuliert werden:

$$\vec{v}_k = C^T \cdot \vec{v}_g \tag{2.28}$$

[22] Englisch *body frame*

[23] Der Buchstabe g, anstatt von w wurde gewählt, um eine Verwechslung mit dem griechischen ω zu vermeiden. G steht hier für *globales Koordinatensystem* bzw. englisch: *global frame*

[24] Im vereinfachten Fall ist es die Bildung einer Summe

[25] In der Implementation in Abschnitt 5 wird aus Effizienzgründen eine orthogonale (4x4) Matrix benutzt. Das Mathematische Prinzip bleibt aber natürlich das selbe.

Es muss für das Tracken der Orientierung die Drehmatrix C über die Zeit getrackt werden. Falls die Orientierung zu einem Zeitpunkt t durch C(t) gegeben ist, so erhält man die Drehratenänderung[26] von C zum Zeitpunkt t durch:

$$\dot{C}(t) = \lim_{\delta t \to 0} \frac{C(t+\delta t) - C(t)}{\delta t} \tag{2.29}$$

Wobei in Gleichung 2.29 der Ausdruck $C(t+\delta t)$ als Produkt zweier Matrizen

$$C(t+\delta t) = C(t) \cdot A(t) \tag{2.30}$$

geschrieben werden kann. Wobei $A(t)$ die Drehmatrix ist, welche die Änderungen im Körperbezugssystem beschreibt, welche sich zwischen den Zeitpunkten t und $t+\delta t$ ereigneten. Sie repräsentiert also die Drehbewegung, welche sich im Intervall δt ereignete. Falls das Intervall δt hinreichend klein ist, so kann diese Änderung als sehr klein angenommen werden und die Winkel $\delta\phi$, $\delta\theta$ und $\delta\psi$ können nach Gleichung 2.14 (aus Abschnitt 2.14 auf Seite 19) approximiert werden und A(t) lässt sich schreiben als:

$$A(t) \overset{2.14}{=} I + \delta \cdot \Psi \tag{2.31}$$

Mit:

$$\delta \cdot \Psi = \begin{pmatrix} 0 & -\delta\psi & \delta\theta \\ \delta\psi & 0 & -\delta\phi \\ -\delta\theta & \delta\phi & 0 \end{pmatrix} \tag{2.32}$$

Durch Substitution erhält man:

$$\begin{aligned} \dot{C}(t) = & \lim_{\delta t \to 0} \frac{C(t+\delta t) - C(t)}{\delta t} \\ \overset{2.30}{=} & \lim_{\delta t \to 0} \frac{C(t) \cdot A(t) - C(t)}{\delta t} \\ \overset{2.31}{=} & \lim_{\delta t \to 0} \frac{C(t) \cdot (I + \delta\Psi) - C(t)}{\delta t} \\ = & C(t) \cdot \lim_{\delta t \to 0} \frac{\delta\Psi}{\delta t} \end{aligned} \tag{2.33}$$

Im Grenzfall $\delta t \to 0$ gilt die Approximationsannahme für kleine Winkel aus Abschnitt 2.14 und nach Gleichung 2.14 gilt:

$$\lim_{\delta t \to 0} \frac{\delta\Psi}{\delta t} \overset{2.14}{=} \Omega(t) \tag{2.34}$$

[26] an dieser ist man hauptsächlich für das Aufsummieren interessiert.

Mit:

$$\Omega(t) = \begin{pmatrix} 0 & -\omega_{kz}(t) & \omega_{ky}(t) \\ \omega_{kz}(t) & 0 & -\omega_{kx}(t) \\ -\omega_{ky}(t) & \omega_{kx}(t) & 0 \end{pmatrix} \tag{2.35}$$

was die antisymmetrische[27] Form des Winkelgeschwindigkeitsvektors[28] $\omega_k(t)$ darstellt. Damit die Orientierung getrackt werden kann, muss die Differentialgleichung

$$\dot{C} = C(t) \cdot \Omega(t) \tag{2.36}$$

Welche die Lösung

$$C(t) = C(0) \cdot exp\left(\int_0^t \Omega(t)dt\right) \tag{2.37}$$

besitzt. Wobei C(0) die initiale Orientierung des zu messenden Körpers darstellt, an welchem die inertiale Messeinheit befestigt ist.

Für eine reale Implementierung der Theorie muss man Integrationsverfahren benutzen, welche mit den diskreten Signalsamples, welche üblicherweise von einer IMU geliefert werden, arbeiten können. Ein kontinuierliches Signal $\omega_b(t)$ wird von realen Geräten normalerweise nicht geliefert, sondern ist abgetastet und quantisiert. Digitale Signale lassen sich vereinfacht mit der Mittelpunktsregel[29] zeitlich integrieren. Man kann auch andere Verfahren 2. oder 3. Ordnung benutzen, um für die jeweilige Anwendung angemessenere Ergebnisse zu erzielen. Wie man mit diesen Signalen bei der Implementation umgeht, wird im entsprechendem Abschnitt dieser Arbeit behandelt. Für eine Darstellung der Theorie sollen die Ausführungen hier genügen.

Tracking der Position einer strapdown IMU

Wie schon im unteren Teil der schematischen Abbildung 2.4 von Seite 35 ersichtlich kann bei bekannter Orientierung C auch die Position p mittels doppelter Integration über die Zeit und Filterung des Gravitationsvektors $\vec{g}$ bestimmt werden [Sav98].

Damit die Position einer strapdown inertialen Messeinheit getrackt werden kann, muss das Signal der Accelerometer, also die Beschleunigung $\vec{a}_k(t) = (\vec{a}_{kx}(t), \vec{a}_{ky}(t), \vec{a}_{kz}(t))^T$, in das Weltkoordinatensystem und damit das globale Bezugssystem projiziert werden. Das geschieht mittels der Gleichung:

[27] Sie wird auch *schiefsymmetrisch* genannt. Eine Eigenschaft von ihr ist, dass die ihre negative Matrix gleich der Transponierten ist

[28] auch gelegentlich mit *Drehrate* bezeichnet

[29] oft auch als *Rechteckregel* referenziert

$$\vec{a}_g(t) = C(t) \cdot \vec{a}_k(t) \tag{2.38}$$

Im nächsten Schritt wird für die Berechnung der Geschwindigkeit der IMU der Einfluss der Gravitation gefiltert:

$$\vec{v}_g(t) = \vec{v}_g(0) + \int_0^t \vec{a}_g(t) - \vec{g}_g dt \tag{2.39}$$

$$p_g(t) = p_g(0) + \int_0^t \vec{v}_g(t) dt \tag{2.40}$$

Wobei, wie auch schon vorher üblich, $\vec{v}_g(0)$ die initiale Geschwindigkeit, $p_g(0)$ die Initialposition der Messeinheit angeben und $\vec{g}_g$ den Gravitationsvektor bezeichnet. Alle Größen werden im Weltkoordinatensystem des globalen Bezugssystems angegeben.

In der Theorie lassen sich die Formeln gut berechnen. Für eine Implementation ist es allerdings nötig die digitalen Messwerte tatsächlich zu integrieren. Durch die Mittelpunktsregel vereinfachen sich die Gleichungen 2.39 und 2.40 zu:

$$\vec{v}_g(t + \delta t) = \vec{v}_g(t) + \delta t \cdot (\vec{a}_g(t + \delta t) - \vec{g}_g) \tag{2.41}$$

$$p_g(t + \delta t) = p_g(t) + \delta t \cdot \vec{v}_g(t + \delta t) \tag{2.42}$$

Offenkundig hängt die Genauigkeit der Rechnung maßgeblich von a_g, v_g und g_g ab. Und alle diese drei Größen werden komplett durch die akkurate Bestimmung der Orientierung C dominiert. Es ist leicht ersichtlich, dass bei jeder Implementation für eine möglichst präzise Bestimmung der Orientierung C ein beträchtlicher Aufwand getrieben wird, damit die Drift sich in einem gewissen Rahmen hält.

Da alle Sensoren auf Basis der MEMS-Technologie und alle Sensoren in einem vertretbaren Preisniveau, auch eine gewissen Messungenauigkeit bei der Bestimmung von C liefern, ist eine dauerhaft driftfreie Implementation, ohne zusätzliche Sensoren, welche (zumindest intervallsweite) absolute Positionen bzw. Orientierungen liefern, gar nicht möglich. Die Frage besteht vielmehr darin, wie man die Drift eindämmen kann und möglichst eine obere Schranke für sie findet. Genaueres dazu steht in den entsprechenden Abschnitten.

2.2.5 Weitere Inertialsensoren - Absolute Positionsgeber

Bisher vorgestellte Sensoren liefern immer relative Werte, also Differenzen zwischen dem vorherigem Messwert und dem aktuellen, wie z.B. eine Beschleunigungsänderung $\Delta\vec{a}$ oder eine aktuelle Winkelgeschwindigkeit (Drehrate) von $x°/Sekunde$ (bzw. $\frac{rad}{s}$). In diesem Abschnitt wird kurz auf weitere Sensoren eingegangen, welche im Zusammenhang mit Inertialsensoren und inertialen Messeinheiten stehen, nämlich Sensoren, welche absolute Werte liefern können und damit auf die Fragen:

- Wo befindet sich das System gerade?
- Welchen Winkel/Welche Orientierung hat das System gerade in Bezug auf das Magnetfeld/die Gravitation der Erde?

Solche Sensoren sind **Inklinometer** (Neigungsmesser), **Magnetometer** ("Kompasse") und **GPS**. Im strengen Sinne sind diese Sensoren keine Inertialsensoren, aber die Grenzen sind teils unscharf, so kann man Kompasse auf Basis von Gyroskopen, als Gyrokompasse[30] bauen und Inklinometer lassen sich unter Zuhilfenahme von Accelerometern bauen, wie in Formel 6.10 auf Seite 103 und dem zugehörigen Abschnitt gezeigt. Da sie inertiale Messeinheiten und inertiale Navigationssysteme oft ergänzen, werden sie hier kurz vorgestellt.

- Einfachste Inklinometer können mit einem an einem Faden befestigten Gewicht und einer Skala realisiert werden. Eine höhere Genauigkeit erreichen Accelerometer, welche über drei Achsen die Summe der Kräfte bilden, deren Verhältnis berechnen, im Bereich von $\pm 1\vec{g}$ aktiv sind und so nach Formel 6.10 den Winkel α zur Gravitation berechnen können. Noch bessere Ergebnisse kann man durch ein anderes Prinzip erreichen: Eine elektrisch leitende Flüssigkeit (z.B. das flüssige Metall Quecksilber) wird entlang einer Widerstandsbahn in einem Gefäß geführt. Bei Neigung kann diese Bahn verschieden stark in die Flüssigkeit eintauchen und liefert andere Messwerte.
- Magnetometer sind Geräte, welche Magnetfelder messen können. Das einfachste Funktionsprinzip hat der bekannte magnetische Kompass. Zur Lagebestimmung sind 3D-Magnetometer geeignet, welche in allen Orientierungen das Magnetfeld der Erde messen können und damit der Messeinheit das Berechnen der Orientierung in Bezug auf das Magnetfeld der Erde ermöglichen. Leider messen solche Geräte alle magnetischen Signale und werden leicht gestört durch fremde Magnetfelder. Ihr Einsatz in manchen Gebieten wird erschwert und man muss unter Umständen Gebrauch von Filterungstechniken machen, damit man magnetisches Rauschen oder andere Störsignale filtern kann.

[30] Diese können Änderungen in der Nord/Süd-Ausrichtung anzeigen, ohne das Magnetfeld der Erde dafür nutzen zu müssen

- Das Global Positioning System (GPS) ist ein satellitengestütztes System, welches theoretisch in der Lage ist die Position eines Objektes im Zentimeterbereich anzugeben. In Outdoor-Navigation hat es sich längst durchgesetzt. Im Indoorbereich, insbesondere bei Roboternavigation oder Positionsbestimmung von kleinen Objekten eignet es sich nicht, auch wenn mit Differentiellen GPS die Messgenauigkeit signifikant verbessert werden kann. Für den Outdoor-Einsatz ist es aber auch in der Robotik als die Inertialnavigation unterstützendes System, welches den Messfehler beschränkt, sehr nützlich.

Alle diese Sensorarten eignen sich gut, um den maximalen Messfehler inertialer Messeinheiten einzudämmen. Leider hat die Wiimote solche Sensoren nicht. Ein Gerät, welches der Wiimote am nächsten käme und z.B. ein dreiachsiges Magnetometer besitzt, ist die Move von Sony. Warum für in der vorliegenden Arbeit die Entscheidung gegen die Move und für die Wiimote fiel, ist im entsprechendem Abschnitt (1.4) erklärt.

2.3 Zusammenfassung

In diesem Kapitel wurden zunächst für das Verständnis dieser Arbeit relevante Grundbegriffe und Grundkonzepte aus Physik, Mathematik und Robotik vorgestellt. Danach wurde das Grundprinzip der Funktionsweise von Inertialsensoren vorgestellt und Möglichkeiten, wie sie hergestellt werden können. Danach folge eine kurze Einführung in Multisensordatenfusion und in Gestenerkennung. Dort wurde auch kurz auf Algorithmen eingegangen, welche sich für Gestenerkennung eignen und deren Komplexität betrachtet.

Hardware

3

Der folgende Teil beschreibt die in der vorliegenden Diplomarbeit benutzte Hardware näher. Abschnitt 3.1 behandelt das Eingabegerät mit dem hier gearbeitet wird: Die Wii Remote (kurz Wiimote) ist das Eingabegerät, welches genutzt wird, um mit der Spielekonsole *Wii* von Nintendo zu spielen. Die Wiimote ist also ein Joystick der neueren Generation. In Abschnitt 3.2 folgen die Plattformen mit denen experimentiert wurde. Die ersten Tests und Versuche wurden mit der Pioneer Plattform gemacht (Abschnitt 3.2.1), um dann mit dem TAMS Serviceroboter (TASER) weitergehende Fähigkeiten zu implementieren. Der TASER ist in Abschnitt 3.2.2 genauer beschrieben.[1] Die Ansteuerung des PA10 Armes von Mitsubishi mittels CWiid, Java und RCCL ist auch Teil dieser Arbeit gewesen und wird in Abschnitt 3.2.3 behandelt. Im Abschnitt 3.2.4 wird die fünffingerige Hand von Shadow vorgestellt, mit der auch experimentiert wurde.

3.1 Wiimote von Nintendo

Nintendos Spielekonsole Wii ist mit über 82 Mio. verkauften Exemplaren[2] eine der meistverkauften Spielekonsolen weltweit. Im Jahre 2007 lag sie laut Hersteller sogar unter den Neuerscheinungen vor der Xbox 360 und PlayStation 3. Die Tatsache dass mit jeder verkauften Wii auch mindestens eine Wiimote gekauft wurde, macht sie zu einem der meistverkauften und praxiserprobten innovativen Eingabegeräte. Das besondere und innovative an der Wiimote ist ihre höhere Zahl an Freiheitsgraden und die damit verbundene neu geschaffene Möglichkeit einer noch intuitiveren Steuerung, als das bei herkömmlichen Joysticks möglich ist. Diese Fähigkeiten des Gerätes werden von Spielern sehr geschätzt und sind wohl ursächlich für die hohen Verkaufszahlen. Die Umsetzung der natürlichen und leicht zu erlernenden, abstraktionsfreien Bewegungssteuerung wird oft gelobt.

In Abbildung 3.1 sind zwei verschiedene Wiimotes abgebildet. Jede Wiimote besitzt 12 Knöpfe, von denen vier ein Steuerkreuz bilden. Oben links in der Ecke ist

[1] `http://tams-www.informatik.uni-hamburg.de/research/robotics/service_robot`

[2] `http://www.nintendo.co.jp/ir/pdf/2013/annual1303e.pdf#page=22`Nintendos Finanzbericht, 2013, Seite 22

der Powerknopf, welcher softwaretechnisch ein anderes Signal liefert, als die übrigen Knöpfe und ist damit gesondert zu behandeln (siehe 4.1). Zusätzlich zu einem gelieferten Signal schaltet er die Wiimote ein bzw. aus. Es gibt einen “Feuerknopf“ auf der Rückseite (Knopf ”B”) und einen großen, runden Knopf (Knopf ”A”) vorne.

Ferner hat die Wiimote die Knöpfe “Plus“, “Minus“, “Home“, “1“ und “2“. Die Belegung der Knöpfe innerhalb von Spielen und Software ist frei. Auch sind die Bezeichnungen der Knöpfe bei manchen Modellen anders. Die Knöpfe “1“ und “2“ sind gleichzeitig zu drücken, damit sich die Wiimote in den Bluetooth Modus *sichtbar* begibt. Das kann auch durch Drücken des speziellen ”Sync”-Knopfes im Batteriefach auf der Rückseite erreicht werden.

Abbildung 3.1: Vorder- und Rückseite von zwei Wiimotes

Die Maße der Wiimote sind 148 mm an Länge, 36,2 mm Breite und 30,8 mm Dicke laut Herstellerangaben. Innerhalb dieses Gehäuses bildet der Bluetooth Broadcom BCM2042 SoC (System On Chip) Controller das Herzstück. Er unterstützt den I^2C-Bus (siehe Abschnitt 3.1.3) und verarbeitet die Accelerometerdaten, welche durch das ADXL330 Accelerometer von AnalogDevices® (siehe Datenblatt im Anhang A.3) bereitgestellt werden. Ferner werden dort die Daten der Infrarotkamera vorverarbeitet. Auf der Oberseite der Wiimote befinden sich noch vier lichtemittierende Dioden (LEDs) und ein Lautsprecher. Eine Vibrationseinheit ist im Gehäuse ebenfalls verbaut. Diese drei Komponenten werden ebenfalls durch den BCM2042 SoC gesteuert.

3.1.1 Technische Details der Wiimote

Die im folgenden Abschnitt gemachten Angaben haben als Quelle das WiiBrew-Projekt[3] ([Gem14b], welches ein Unterprojekt vom Wii-Homebrew-Projekt[4] ([Gem14a]) ist. Dieses wiederum betreibt den Wii-Homebrew-Channel für die Wii, mit dem es möglich sein soll eine alternative Firmware auf der Wii zu betreiben. Nicht alle Teile der Projekte sind von gleicher Relevanz für diese Arbeit. Die Mitglieder dieser Projekte haben viel Zeit darauf verwendet um die Spielekonsole Wii und

[3] `www.wiibrew.org`
[4] `http://www.wii-homebrew.com/`

die dazugehörigen Steuer-/Eingabegeräte, wie **Wiimote**, **MotionPlus** und **Nunchuk**, aber auch viele andere auseinander zu bauen und deren Funktionsweise sehr detailliert zu verstehen und zu veröffentlichen.[5] Man kann folgende Erkenntnisse für diese Arbeit nutzbar machen:

Die Wiimote benutzt das Standard HID (Human Interface Device) Bluetooth Protokoll zur Kommunikation mit der Wii Spielekonsole. Das erleichtert die Kommunikation mit der Wiimote für einen externen Kommunikationspartner, welcher nicht die Spielekonsole, sondern z.B. die hier vorliegende Software, ist. Leider werden nicht die Standarddatentypen und der standardisierte HID Descriptor benutzt. Warum das so gemacht ist, kann abschließend nur Nintendo® beantworten, es wird aber vermutet, dass es nicht geschah, um Geheimnisse zu hüten, sondern weil das HID Bluetooth Protokoll in seiner Standardversion gar kein Eingabegerät mit 6 DOF und Knöpfen vorgesehen hat. Letztlich ist für diese Arbeit ausreichend, dass es Software gibt, die mit der Wiimote über Bluetooth kommunizieren kann. Mit Hilfe von solcher Software (CWiid 4.1) ist es möglich die Daten der Wiimote mit einer Frequenz von 100Hz abzufragen und Antworten zu verschicken. So kann man alle 10ms Daten von den Accelerometern, Gyroskopen und der IR-Kamera bekommen und verarbeiten.

Das ADXL330 Accelerometer (MEMS Technologie, siehe Seite 31, Abschnitt 2.2.2) von AnalogDevices® (siehe Datenblatt im Anhang A.3) misst die Beschleunigung über alle drei Achsen in einem Bereich von -3g bis +3g. Der Chip selbst kann die Daten mit 0,5Hz bis 1600Hz in x/y Richtung und 0,5Hz bis 550Hz in Richtung der orthogonalen z-Achse liefern. Um Bandbreite und Energie zu sparen werden diese theoretischen Daten leider stark gedrosselt. Der Chip kann die Gravitation ("passive Beschleunigung"), als auch Bewegung, Vibration und Aufprall bzw. Erschütterung messen. Seine Maße betragen 4mm × 4mm × 1,45mm:

Die eingebaute monochrome Infrarotkamera der Wiimote hat einen Fotochip mit der Auflösung von 128x96 Pixel. Durch eine achtfache Subpixelverarbeitung ergibt sich ein Bild von 1024x768 Pixel. Der Öffnungswinkel der Kamera beträgt 33 Grad horizontal und 23 Grad vertikal. Lichtquellen im Bereich von 940nm werden doppelt so gut wahrgenommen, wie diejenigen im Bereich von 850nm. Theoretisch ist die Kamera in der Lage jedes helle Objekt zu erkennen, aber durch das Infrarotpassfilter am Eingang wird nur Licht im infrarotem Bereich durchgelassen. Ferner unterscheidet der Fotochip die Stärke mit der ein Punkt erkannt wird. Jeder Punkt kann in eine von drei Intensitätsstufen eingeteilt werden. Um die zu übertragende Datenmenge zu reduzieren, werden die Daten vorverarbeitet und auf vier sichtbare Infrarotpunkte reduziert. Von einem externen Computer aus kann man also nur vier Koordinatenpaare ablesen und eine Intensitätsstufe von 1 bis 3 pro Paar.

Von den 16KB EEPROM können 6KB vom Benutzer frei beschrieben werden. Das wird spielerisch genutzt, um Avatare bzw. Spielerprofile zu speichern. In der vorliegenden Arbeit werden in dem Bereich die ermittelten Kalibrierungsdaten gespeichert

[5] ein solcher Prozess wird allgemein *reverse engineering* genannt

(beginnend an der Adresse 0x0030). Ferner lassen sich einige der internen Register der Wiimote lesen und beschreiben.

Die Vibrationseinheit ("Rumble") kann mit verschiedenen Frequenzen angesprochen werden, um unterschiedliches Vibrationsfeedback zu geben. Erfolgreich ausprobiert wurden hier 100ms und 250ms Intervalle. Der Lautsprecher lässt sich ansteuern, indem man ihn zunächst initialisiert (0x01 in 0xA20009 schreiben), dann die Lautstärke einschaltet (0x08 auf 0xA20001 schreiben). Im Anschluss lässt sich eine Frequenz mittels einer 7 Byte langen Sequenz erzeugen (7 Byte auf 0xA20001-0xA20008). Die Ansteuerung des Lautsprechers wurde nicht weiter verfolgt, aber es soll Entwickler geben, die mit hinreichendem Erfolg Musik mit der Wiimote machen. Mit Energie wird die Wiimote über zwei AA Batterien versorgt. Induktionsakkus machen das Öffnen des Gehäuses zum Aufladen überflüssig.

3.1.2 Orientierung der Wiimote

Da in der Literatur der Gebrauch von Orientierungen und Achsen von Objekten uneinheitlich ist, werden diese im folgenden Abschnitt für die Wiimote definiert. Die Vorderseite der Wiimote wird in dieser Arbeit stets als die Seite referenziert, an der sich die Infrarotkamera befindet. Dies ist als Analogie zum menschlichen Auge, welches auch unser "vorne" ist, leicht eingängig. Beim Auslesen der Werte der Wiimote werden, falls eine Beschleunigung (Gravitation) in dieser Richtung anliegt, größer werdende y-Werte ausgelesen. Das ist die positive y-Achse. Die entgegengesetzte Richtung ist dann die negative y-Achse und "hinten" bei der Wiimote. "Oben" sind die elf Knöpfe und die negative z-Achse (den gelieferten Werten folgend), während die Batteriefächer sich dann "unten" in Richtung positiver z-Achse befinden. "Rechts" und "links" ergeben sich dann, wenn der Betrachter hinter der Wiimote steht und sie mit der Rückseite auf dem Boden liegt. So ergibt sich die Richtung der positiven x-Achse, wenn man hinter der Wiimote stehend sie im Uhrzeigersinn dreht, sodass die Knöpfe nach rechts zeigen. Die entgegengesetzte Lage (Knöpfe nach links weisend) zeigt die negative x-Achse an.

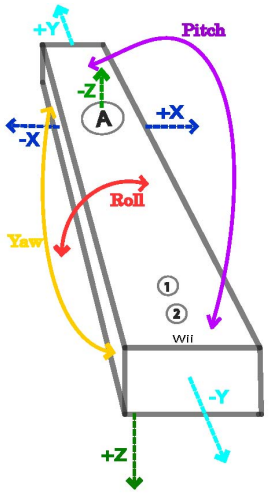


Abbildung 3.2: Definition der Orientierung der Wiimote

Roll ist somit, angelehnt an die Luftfahrt, eine Drehung um die y-Achse, **Pitch** um die x-Achse und eine Drehung um die z-Achse entspricht **Yaw**. Es wird angenommen, dass die Wiimote von hinten betrachtet wird und der Uhrzeigersinn sich wie in einem linkshändigen Koordinatensystem ergibt.

3.1.3 Erweiterungscontroller für die Wiimote

Erweiterungscontroller für die Wiimote (auch genannt "Wiimote Erweiterungen" oder kurz "Erweiterungen") findet man sehr viele auf dem Markt. In dieser Arbeit wurden die MotionPlus (siehe 3.1.3) mit ihren dreiachsigen Gyroskopen und die Nunchuk, welche einen Joystick, 2 Knöpfe und ein 3 achsiges Accelerometer besitzt, verwendet.

Auf diese Erweiterungen wird in ihren Abschnitten detaillierter eingegangen. Der Überblick hier ist keineswegs vollständig und könnte es auch nicht lange bleiben, da laufend neue Erweiterungen entwickelt werden. Prinzipiell gibt es zwei Arten von Erweiterungen: die einen, die rein mechanisch die Wiimote erweitern, meist sind es Kunststoffgehäuse, die eine andere Art der Bewegung oder des Haltens bzw. Greifen der Wiimote ermöglichen. Dazu gehören Lenkräder für Autorennen oder Schläger für Sportarten wie Golf und Tennis. Die Wiimote wird in diese Gehäuse hineingesteckt. Sie erweitern die sensorischen Möglichkeiten der Wiimote nicht. Ferner gibt es Erweiterungen, welche über den I²C Bus angeschlossen werden. Sie können die Wiimote mit mächtigen Sensoren erweitern und beinhalten grundsätzlich elektronische Bauteile, welche über den I²C-Bus mit der Wiimote kommunizieren. Zu ihnen zählen die MotionPlus und Nunchuk.

I²C Erweiterungen
MotionPlus
Nunchuk
Balance Board
Classic Controller
Vitality Sensor
Guitar Hero Guitar
Guitar Hero Drums
Gehäuseerweiterungen
Wheel
Sport
Gun

Tabelle 3.1: Kleiner Ausschnitt der Erweiterungen für die Wiimote

Wii MotionPlus Erweiterung

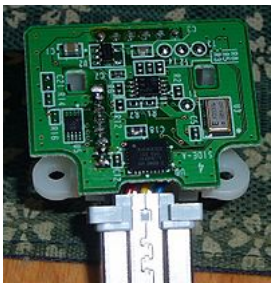


Abbildung 3.3: Der Gyroskop Chip der MotionPlus von oben

Die MotionPlus ist quaderförmig und misst 3cm in der Länge und 3,5cm in der Breite. Mit eingesteckter MotionPlus ergab eine Messung für die Wiimote inklusive der Erweiterung 172mm in der Länge. Die MotionPlus wird in eine Wiimote über den I²C-Bus (siehe 3.1.3) eingesteckt. Sobald sie eingesteckt wird, meldet sie sich bei der Wiimote an und schreibt in ein bestimmtes Register den Wert, welcher spezifisch für eine MotionPlus Erweiterung ist. Damit weiß das SoC der Wiimote, dass eine MotionPlus eingesteckt und betriebsbereit ist. Sobald die Software die MotionPlus einschaltet, wird sie aktiviert und das Gyroskop liefert Werte. Die MotionPlus ist in der Lage eine andere Erweiterung durchzuleiten. Dazu werden die Daten der zweiten Erweiterung über den I²C-Bus grundsätzlich in veränderter Form weitergegeben.

Abhängig von der Erweiterung werden die Bits unterschiedlich geshiftet. Es ist also anzunehmen, dass ursprünglich nur die Möglichkeit für eine Erweiterung zur Zeit von Nintendo vorgesehen wurde und die MotionPlus nachträglich als ein Gerät zum "Dazwischenschalten" entwickelt wurde.

Im Inneren der MotionPlus befindet sich ein IDG-600 Gyroskopchip der Firma InvenSense[6] (siehe Abbildung 3.3 und 3.4 für eine Sicht von oben und unten auf den Chip). Berichte von Bastlern, welche Reverse-Engineering betrieben haben gehen auseinander, ob der ursprünglich für zwei Achsen ausgelegte Chip für Nintendo umgebaut wurde, oder ein EPSON TOYOCOM XV3500CB für die Yaw Richtung benutzt wird. Das MEMS Gyroskop der MotionPlus kann Messungen über alle drei Achsen durchführen. Es ist hierbei nicht wichtig, ob die den Winkelbeschleunigungen oder Radialbeschleunigungen zugrunde liegenden Kräfte gemessen werden. Ein Rückschluss auf die gesuchten Größen ist immer möglich. Die Messgenauigkeit der drei Gyroskope ist deutlich höher als die Genauigkeit der Accelerometer, welche der Wiimote zur Verfügung stehen.

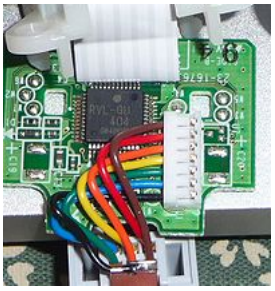


Abbildung 3.4: Der Gyroskop Chip der MotionPlus von unten

WiimotePlus

Die WiimotePlus Fernbedienung ist ein Controller, welcher auf den Markt kam, als diese Diplomarbeit im Entstehen war. Die WiimotePlus ist seit dem 14.November 2010 im Handel erhältlich.[7] Im Wesentlichem vereint sie eine Wiimote Fernbedienung mit einer MotionPlus Erweiterung in einem Gehäuse. (Zu MotionPlus siehe 3.1.3) In der vorliegenden Arbeit wurde noch mit der klassischen Wiimote und der Erweiterung MotionPlus gearbeitet. Die WiimotePlus ist zu dieser Kombination technisch gleichwertig. Das geht aus Testberichten und Erfahrungsberichten von Benutzern hervor.

Der Erweiterungscontroller Nunchuk

Der **Nunchuk** Erweiterungscontroller[8] ist ein Gerät, welches die Wiimote um weitere sensorische Möglichkeiten erweitert. Der Controller ist 113mm hoch, 38mm breit und 37mm dick, hat zwei Knöpfe, einen analogen Joystick und ein drei Achsen Accelerometer von STMicroelectronics®. Der Chip mit der Bezeichnung LIS3L02AL ist in MEMS-Technologie als kapazitives Accelerometer gefertigt. Das Accelerometer der Nunchuk hat einen Arbeitsbereich von -2g bis +2g, welcher etwas kleiner

[6] `invensense.com`
[7] Quelle:www.nintendo.de, Stand Dezember 2010
[8] oder auch einfach kurz "Nunchuk" genannt, mit weiblichem Genus

ist, als der des Accelerometer der Wiimote, dafür ist die angegebene Auflösung mit 0,5mg bei 100Hz vom Chip her deutlich höher, als die des Chips der Wiimote (siehe dazu auch die Datenblätter im Anhang A.3) Eine Nunchuk, welche über den I²C Bus direkt, oder aber auch durch eine MotionPlus durchgeschleift, an eine Wiimote angeschlossen werden kann, erhöht deutlich die Menge an Sensordaten, welche gesammelt und ausgewertet werden können.

I²C Bus

I²C ist ein serieller Datenbus, welcher von Phillips Semiconductors entwickelt wurde. Seit der Spezifikation 1.0 im Jahre 1992 hat sich die maximale Datenrate vom 100KB/s über 400KB/s zu maximal 3,4MB/s (Version 2.0 im Jahre 1998) erhöht. Der Adressraum umfasste zunächst 112 Knoten, wurde jedoch auf 10 Bit erweitert und kann jetzt 1136 Knoten adressieren. Der als Master-Slave konzipierte Bus erlaubt generell mehrere Master. Die Kommunikation läuft über eine Datenleitung (SDA serial data line) und eine Taktleitung (SCL serial clock line). Dies ist mit ursächlich für den Alternativnamen *TWI* (Two-Wire Interface). Andererseits mag eine gewisse Unsicherheit über Schutzrechte des Namens I²C dazu ebenfalls beigetragen haben. Tatsächlich ist das Logo, nicht jedoch der Name I²C von Phillips geschützt worden.

Wenn das I²C Protokoll auf elektrischen Leitern implementiert wird, dann meist so, dass ein ständiger High-Pegel (VDD) zur Datenübertragung beginnend auf Low (Ground) gesetzt wird. Zur Erkennung von Kollisionen hören alle Busteilnehmer auf der Leitung und lesen auch ihr eigenes Signal. Liegt auf dem Bus ein Low-Signal an, obwohl ein Teilnehmer ein High-Signal senden möchte, dann ist der Bus belegt und es wird nach einer zufälligen Zeit noch mal gesendet.

I²C wird für Peripheriegeräte benutzt, welche nicht auf Geschwindigkeit angewiesen sind. Es hatte in Deutschland durch Chipkarten (Krankenkassenkarten) an Bedeutung erlangt. Für größere Entfernungen und hohe, fehlerfrei zu sendende Datenmengen eignet sich I²C nur wenig bis gar nicht. Als Bus für Erweiterungscontroller der Wiimote und in bestimmten Bereichen von eingebetteten Systemen hingegen, ist der Bus mit seinem Protokoll sehr gut geeignet.

3.1.4 Fazit und Begründung der Entscheidung für die Wiimote

Nachdem in Abschnitt 1.4 dargestellt wurde, welche alternative sensorische Hardware hätte verwendet werden können und auf Grund welcher Nachteile sie nicht genutzt worden ist, wird in diesem Abschnitt dargelegt, warum die Wiimote als Eingabegerät gewählt worden ist. Gleichzeitig soll er als eine Zusammenfassung dienen.

Die Wiimote besitzt folgende Eigenschaften bzw. Vorteile:

- Durch die hohe Stückzahl ist sie als Eingabegerät und Sensor günstig in der Anschaffung und leicht wiederzubeschaffen im Falle eines Defektes/Verlustes.
- Die Genauigkeit der Sensoren ist gemessen am Preis sehr gut. Sie ist auf jeden Fall ausreichend, dass Millionen von Spielern damit zufrieden sind. Das alleine ist kein Kriterium, aber ein Hinweis auf die Güte des Produktes.
- Durch die lange Zeit (seit 2006), die das Produkt am Markt ist, ist es sehr ausgereift.
- Sie ermöglicht eine natürliche und intuitive Steuerung.
- Neben 3-Achsen Accelerometern lassen sich 3-Achsen Gyroskope (MotionPlus) anschließen und die Daten der Infrarotkamera können ebenfalls für eine Positionsbestimmung herangezogen werden.
- Als *System on Chip* (SoC) ist sie ein eingebettetes System und hat damit Vorteile durch vorhandene Rechenleistung. Eine Vorverarbeitung der Kameradaten ist möglich.
- Sie lässt sich ggf. durch den I^2C Bus erweitern oder an andere Systeme anschließen.
- Mittels HID Profil kann sie über eine Bluetooth-Schnittstelle angesprochen und an Computer angeschlossen werden.
- Der innere Aufbau des Gerätes ist weitgehend bekannt, da es Projekte gibt, die sich genau damit beschäftigen (Wiibrew).
- Es gibt bereits freie Software auf dem Markt, die die Kommunikation mit dem Gerät sehr vereinfachen (CWiid in Abschnitt 4.1).

Wie bereits in Abschnitt 1.4 erwähnt, wurde die Wiimote gegenüber den anderen möglichen Controllern bevorzugt, wegen ihrer längeren Marktpräsenz, der besseren Verfügbarkeit von (freier) Software für das Gerät. Abschließend kann man abschätzen, dass die Experimente (siehe Abschnitt 6) zu zeigen haben, wie gut die Wiimote für diese Diplomarbeit geeignet ist. Die Erwartung ist natürlich, dass die Wiimote gut geeignet ist auch wenn die Move von Sony einen Sensor (Magnetometer) mehr hat und für die visuelle Datenanalyse mächtiger ist. Es ist aber sehr anzunehmen, dass die Vorteile der Wiimote im Vergleich zu den in Abschnitt 1.4 aufgeführten Controllern überwiegen.

3.2 Evaluationsplattformen

Im Arbeitsbereich TAMS gibt es einige Roboterplattformen. Hier werden ein paar davon vorgestellt, mit denen im Rahmen dieser Diplomarbeit gearbeitet wurde. Da die initialen Tests zur Fernsteuerung einer mobilen Plattform mit der Pioneer Plattform gemacht wurden, wird mit dieser angefangen. Die neue Outdoor-Variante des Pioneers hat sogar einen eigenen Computer on-board, sodass ein komfortablerer Umgang möglich wurde, indem nicht jedes Mal ein externes Notebook auf der Plattform montiert werden musste. Das reduzierte nicht nur die Gefahr, dass das System Schaden nimmt, wie bei einer Kollision, bei der das Notebook herunterfällt, sondern auch die Zuverlässigkeit des Systems wurde durch eine konstante Hardware mit festen Treibern erhöht.

Die Steuerung des TASERs ist zwar von der konzeptionellen Seite gleich der des Pioneers, aber aus Gründen der unterschiedlichen Implementierungen der Steuerungssoftware, wurden auch Fahrtests mit dem TASER gemacht und er wird auch hier beschrieben.[9]

Ein kurzer Überblick über den Mitsubishi PA10-6C und die daran angebaute fünffingerige Hand von der Shadow-Company, runden die Vorstellung der Roboter und Manipulatoren ab, mit denen experimentiert wurde.

3.2.1 Pioneer

Die Pioneer Roboterplattformen bei TAMS wurden hergestellt und vertrieben durch die *ActivMedia Robotics*. Heute hat das Unternehmen sein Tätigkeitsfeld erweitert und firmiert seit Juni 2010 unter dem Namen *Adept MobileRobots Inc*[10]. Den Namen verdankt die mobile Roboterplattform, welche für Forschungszwecke produziert wurde und immer noch weiterentwickelt und verkauft wird, der Tatsache, dass es das erste erfolgreiche Projekt der damals noch universitätsnahen Gruppe war.

Abbildung 3.5: Mobile Pioneer Roboterplattformen (Quelle: TAMS Website)

Der erste Pioneer wurde 1995 fertig gestellt. Insgesamt gibt es im Arbeitsbereich TAMS einen Pioneer der ersten und zwei der 2. Generation. Ausgestattet sind die mobilen Plattformen mit Sonarsensoren, Motoren und Akkus. Es können dann noch weitere Komponenten hinzubestellt werden. So hat ein Modell einen Greifer, mit dem es Objekte festhalten, anheben und transportieren kann. Auf einem anderen ist ein Laserscanner von SICK montiert.

[9] zur Steuerung: während die Pioneers sich mit der Player/Stage Umgebung leicht steuern lassen, wird der TASER sehr viel hardwarenäher angesteuert

[10] http://www.mobilerobots.com/

Zum Steuern werden externe Computer in Form von Laptops benutzt, welche alle nötigen Berechnungen machen und über serielle Schnittstellen, USB und ggf. FireWire mit den mobilen Plattformen oder anderen angeschlossenen Geräten kommunizieren. Die Pioneers der zweiten Generation haben leistungsfähigere Motoren und Akkus, als die der ersten Generation. Aber beide Modelle können laut Herstellerangaben Geschwindigkeiten von über $1\frac{m}{s}$ erreichen.

In der vorliegenden Arbeit wurden sie benutzt, um die initialen Steuerungsversuche mit der Wiimote durchzuführen. Beide Modelle ließen sich gut fernsteuern. Abbildung 3.5 zeigt die Pioneers von der Website des Arbeitsbereiches TAMS.

Pioneer 3 AT ("Outdoor Pioneer")

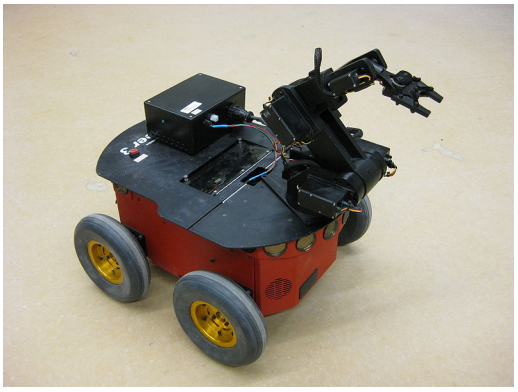


Abbildung 3.6: Pioneer 3 AT der "Outdoor Pioneer"

Die mobile Roboterplattform der dritten Generation "Pioneer 3"liegt in der Outdoor-Variante vor. Seit dem Pioneer 2 werden Modelle für den Indoorbereich (Pioneer DX) und für den Outdoorbereich (Pioneer AT) von Adept MobileRobots bereitgestellt.

Dem AB TAMS steht ein Pioneer 3 AT zur Verfügung. Dieser Roboter ist bedeutend größer und robuster gebaut, als die Vorgängermodelle. Was ihn besonders nützlich macht, ist der bereits fest eingebaute Computer, auf welchem ein modernes Linux-Betriebssystem installiert ist, mit dem sich die Bluetooth-Schnittstelle und die WLAN Karte problemlos ansprechen lassen. Durch die optionalen 8 Sonarsensoren vorne und hinten, sowie dem optional benutzbaren Laserscanner auf dem Gerät, eignet sich dieses Modell besonders gut für den Outdooreinsatz, aber auch für Indoor Anwendungen.

Mit der Player/Stage und CWiid Software wurde diese Plattform als weitere Testplattform für Telemanipulation benutzt. Die Abbildung 3.6 zeigt einen Pioneer 3.[11]

[11] ActivMedia Pioneer 3-AT Roboter am Georgia Institute of Technology, Bild von: Jiuguang Wang

3.2.2 Der TAMS Serviceroboter TASER

Der TASER ist die Service Roboter Plattform des Arbeitsbereiches TAMS.[12] Erworben und zusammengebaut wurde diese Plattform im Dezember 2003.

Seitdem wird die Software stetig weiterentwickelt. Während der Entstehung dieser Diplomarbeit fand auch ein Upgrade der Hardware und des gesamten Betriebssystems des TASERs statt. Die ursprüngliche Hardware des TASERs besteht aus einer veränderten Version der MP-L655[13] mobilen (Roboter) Plattform von NEOBOTIX[14]. Die Plattform hat von Haus aus Differentialencoder zur Steuerung, zwei Laserscanner, jeweils einen vorne und einen hinten, sowie eine Traglast von 150 Kg. Auf dieser Plattform sind acht Bleiakkus, wie sie in PKW benutzt werden, eingebaut, sowie ein Gehäuse ("Oberkörper"), welches die beiden PA10-6C Industriearme von Mitsubishi halten kann (vgl. dazu den Abschnitt 3.2.3). Die Akkulaufzeit beträgt ca. 6 Std. Die acht Akkus sind gegliedert in zwei parallel geschaltete Gruppen, in denen jeweils vier 12V Akkus in Reihe geschaltet wurden. Somit ergibt sich eine Betriebsspannung von 48V für den TASER und die möglichen PA10-6C Arme. Als Endeffektoren sind dreifingerige Greifer ("Hände") von Barrett Technologies Inc.[15] montiert. Auf einer Neige-Schwenk-Vorrichtung ("Pan-Tilt-Unit") sind 2 IEEE1394-Kameras befestigt, welche es ermöglichen Stereobilder aufzunehmen. Eine omnidirektionale Kamera war bei der Aufnahme des Bildes (Abb.3.7) ebenfalls noch befestigt, ist aber z.Z. nicht im Einsatz. Ein Industrie PC mit einem Pentium IV 2.4 GHz Prozessor verarbeitet die Signale und steuert den Roboter. Das Betriebssystem ist ein OpenSUSE Linux. Das frühere Betriebssystem mit einem 2.4 Kernel wird auf eine aktuelle Version von OpenSUSE aktualisiert, sodass der aktuelle 2.6 Kernel eine einfache Anbindung von Bluetooth ermöglicht. Es lässt sich auch ein Windowssystem booten.

Abbildung 3.7: Der TASER mit 2 montierten PA10 und einer Omnidirektionalen Kamera

[12] TAms SErvice Roboter
[13] http://www.neobotix.de/de/products/MP-L655.html
[14] http://www.neobotix.de/
[15] http://www.barrett.com/robot/index.htm

3.2.3 Der PA10-6C Manipulator von Mitsubishi Heavy Industries Ltd.

Der PA10-6C ist ein industrieller Roboterarm (Manipulator), mit sechs Gelenken und sechs Freiheitsgraden, welcher von Mitsubishi Heavy Industries Ltd. (MHI) hergestellt und vertrieben wurde. Seine maximal zulässige Traglast beträgt 10 Kg. Mit 1317mm Länge ist der Arm mit einem menschlichen Arm vergleichbar.

Das Gewicht von 38 Kg ist für Industriearme nicht schwer (Der MHI-PA10-7C wiegt 40 Kg). Der Name setzt sich aus *Portable (general-purpose intelligent) Arm* zusammen, mit der Traglast und der Angabe der Gelenkzahl. Das Nachfolgemodell PA10-7C hat 7 Gelenke und ist von Hause aus mit Drehmomentsensoren ausgestattet, welche mehr Möglichkeiten eröffnen (wie z.B. eine "zero gravity" Anwendung). Der PA10-6C wird bei TAMS aus Sicherheitsgründen nicht mit den möglichen 100V, sondern 48V betrieben [BL08, Abschnitt 2.2, Seite 15ff.]. Somit ergibt sich eine tatsächliche Traglast, von weniger als 10 Kg. Im praktischen Betrieb sind es ungefähr 6 Kg.

Abbildung 3.8: Portable Arm 10-6C vom Mitsubishi Heavy Industries Ltd. mit Barret-Hand

Die Positionsgenauigkeit[16] und Wiederholgenauigkeit[17] wird herstellerseitig im Millimeterbereich angegeben(±0,1mm in [HKK+03, Tabelle 2]). Der Arm kann von einem Standard-(Industrie-)PC mit einer ARCNet-Karte[18] gesteuert werden. So geschieht es auch auf der TASER (siehe 3.2.2) Plattform. Beide Arme des AB TAMS werden mittels RCCL (Abschnitt 4.3) gesteuert.

Durch die Denavit-Hartenberg-Parameter und Grenzwerte der Gelenke wird der Arm mathematisch beschrieben. Auf eine so umfangreiche Darstellung wird hier allerdings verzichtet. Das wäre für Anwendungen, welche sich mit inverser Kinematik beschäftigen, oder eine eigene Steuerung entwickeln, unumgänglich. In [Bru09, Kapitel 4.1.1, Seite 36ff.] findet sich eine ausführliche Darstellung der Werte. Da die vorliegende Arbeit RCCL zur Steuerung im kartesischen bzw. Gelenkwinkelraum nutzt, kann davon ausgegangen werden, dass die Software für die Einhaltung der

[16] Wie genau eine Position angefahren werden kann
[17] Ob beim nächsten Mal auch wirklich die gleiche Position "getroffen" wird
[18] Attached Resources Computer Network

zulässigen Werte sorgt (oder aber in einer singulären Stellung blockiert, worauf man dann aber keinen direkten Einfluss[19] hat (siehe 4.3).

Abschließend kann bemerkt werden, dass der Arm für eine zitter- und ruckelfreie Bewegung alle 10ms einen Befehl erhalten muss. Dieses Echtzeitverhalten (*hard realtime*) kann durch die Token-Ring Eigenschaften von ARCNet eingehalten werden. Durch das umhergereichte Token ist gesichert, dass die Gelenke, welche im Geschwindigkeits- oder Drehmomentmodus betrieben werden können, rechtzeitig mit neuen Anweisungen versorgt werden und flüssig arbeiten können (vgl. [Lab09, Abschnitt 2.4.1] und ausführlicher in [Sch04, Anhang B]).

3.2.4 Die fünffingerige Hand von Shadow

Die rechte Hand[20] von der *Shadow Robot Company Ltd.*[21] in der Version C5, welche sich im Arbeitsbereich befindet, ist eine pneumatische fünffingerige Hand, welche versucht so nah wie möglich die 24 Freiheitsgrade der menschlichen Hand abzubilden [Ltd08, Kapitel 1].

Als neuartiger Manipulator ist die Hand von Shadow einer durchschnittlichen männlichen Hand nachempfunden. Sie hat zusammen mit den pneumatischen Muskeln, den Sensoren und den Ventilen ein Gewicht von 3,9 Kg. Die pneumatischen Muskeln fassen ein Luftvolumen von 0,015 Litern. Von der Materialbeschaffenheit ist die Hand, welche im eigentlichen Sinne ja ein Unterarm ist, gefertigt aus einer Kombination aus Metall und Kunststoff, wobei der Unterarm aus Stahl ist und Acetyle, Polycarbonate ("Fingernägel") und Polyurethane ("Fleisch") die Hauptbestandteile der Hand bilden.

Abbildung 3.9: Dexterous Hand von Shadow (©Shadow Robot Company 2008)

Als Betriebsmittel werden elektrischer Strom (Elektronik: 0,7A bei 8V, Ventile: 1A max bei 28V) und Druckluft (worst case: 24 Liter/Min) verwendet. Durch die 40 pneumatischen Muskeln ist es möglich jeden Finger, mittels der zugehörigen Sehne, ähnlich einer menschlichen Hand zu steuern. Genau genommen kann die Shadow C5 Hand manche Konfigurationen leichter erreichen und halten, als ein ungeübter Mensch. Bei der Roboterhand lassen sich alle Finger unabhängig von einander steuern und Griffsynergien vermeiden.

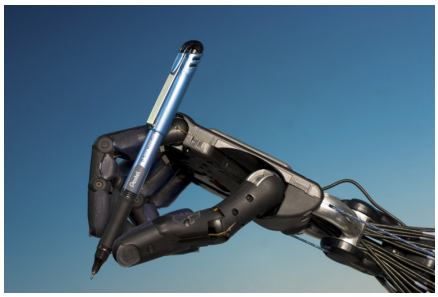
Über Hall-Effekt-Sensoren kann die Rotation eines jeden Gelenkes mit einer 0,2 Grad Genauigkeit, gemessen werden. Diese Zahl kann mittels eines 12-Bit Analog/Digital-

[19] Man könnte allerdings in dem Trajektoriengenerator des quelloffenen RCCL Änderungen am Code vornehmen

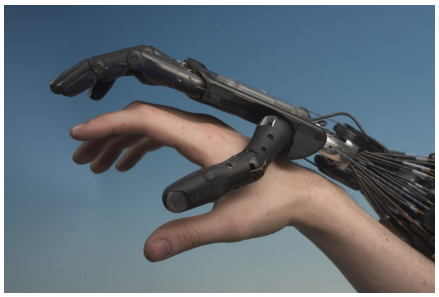
[20] Englischer Originalname *Shadow dextreous (pneumatic) Hand C5*

[21] http://www.shadowrobot.com

wandlers (ADC) auf den CAN-Bus gegeben und mit einer maximalen Frequenz von 180Hz von außen abgefragt werden. Durch Drucksensoren lässt sich der Druck in jedem Muskel messen und mit einer 12-Bit Auflösung im Bereich 0-4 Bar ausgeben. Aus kinematischer Sicht hat der Daumen 5 Gelenke und 5 Freiheitsgrade (DOF). Jeder Finger besitzt 3 Gelenke, aber nur 2 Freiheitsgrade. Das ist genau, wie bei einer menschlichen Hand, bei der auch die oberen beiden Gelenke eines jeden Fingers, nämlich das distale und das proximale Gelenk, über die gleiche Sehne kontrolliert werden und das Fingergrundgelenk den zweiten Freiheitsgrad darstellt. Die Genauigkeit der Steuerung wird mit ±1 Grad über die gesamte Bewegungsspanne vom Hersteller angegeben. Abschließend sei erwähnt, dass die rechte Hand von Shadow auch zu einer linken Hand umgebaut werden kann.



(a) mit einem Stift in der Hand



(b) im Vergleich mit einer menschlichen Hand

Abbildung 3.10: Eine C5 Shadow Hand im Vergleich zu einer menschlichen Hand

3.3 Zusammenfassung

In diesem Abschnitt wurde die Hardware näher beschrieben, welche in dieser Arbeit benutzt wurde. Als Eingabegerät und dasjenige, mit dem am meisten gearbeitet wurde, ist die Wiimote (Abschnitt 3.1) ganz zu Beginn beschrieben. Die Pioneer-Roboter (Abschnitt 3.2.1), auf denen die anfänglichen Fahrexperimente und erste Implementierungen realisiert wurden, folgen darauf. Der TASER-Roboter (Abschnitt 3.2.2) wurde für Fahrexperimente genutzt. Da er auf ein neues Betriebssystem umgestellt wurde, war für diese Diplomarbeit eine Steuerung seines PA10-Armes nicht möglich. Für Experimente mit einem Manipulator wurde ein separater, wandmontierter PA10 (beschrieben in Abschnitt 3.2.3) genutzt. Als weitere genutzte Hardware folgt die Hand von Shadow (Abschnitt 3.2.4).

Verwendete Software

4

Das vorliegende Kapitel behandelt die in dieser Arbeit verwendete Software. Zunächst ist das die CWiid-Bibliothek, welche es ermöglicht die Daten einer Wiimote über Bluetooth an einem PC zu verarbeiten. Dann folgt ein Abschnitt über die in dieser Diplomarbeit hauptsächlich verwendete Programmiersprache JAVA und *Java Native Interface*, welches hier genutzt wurde, um die in C geschriebene CWiid-Bibliothek nutzen zu können. Wobei auf das während dieser Arbeit erstellte C Programm, welches mittels JNI auf die CWiid zugreift, in Abschnitt 5.2.3 des nächsten Kapitels eingegangen wird. Die Abfolge in dem vorliegenden Kapitel geht weiter mit der Robot Control C Language (RCCL), welche zur Steuerung des Mitsubishi PA10 Armes verwendet wird. Es folgt eine Beschreibung der Player-Stage-Gazebo-Umgebung, welche benutzt wurde, um einen Pioneer zu steuern. Das Kapitel schließt ab mit ROS, welches ein Werkzeug in Form einer Programmsammlung ist, mit dem man verschiedene Roboterplattformen ansteuern kann. Es wird hier für einen PR2 Roboter und einen KuKa-7-Gelenk Manipulator benutzt.

4.1 Die CWiid Bibliothek

Programme und Bibliotheken, die sich mit dem Thema *Wiimote* im Allgemeinen und *Wiimote am PC* im Speziellen auseinandersetzen, gibt es viele. Allerdings sind sie von unterschiedlicher Güte. Eine Suchanfrage bei *www.sourceforge.net*[1] ergibt derzeit mehr als 70 Treffer. Leider sind nicht alle Projekte, die dort gehostet werden, aktiv oder gar fertig gestellt worden.

Für diese Arbeit wurden nicht alle diese Treffer durchprobiert. Vielmehr wurde nach einem Programm oder einer Bibliothek gesucht, welche bereits ausgereift, getestet und möglichst schon vielfach im Einsatz ist. Es stellte sich heraus, dass die von L. Donnie Smith entwickelte CWiid-Bibliothek[2] diese Eigenschaften weitgehend besitzt. Sie ist in den Internetrepositories der gängigen Linux-Distributionen (z.B. bei Ubuntu, OpenSUSE oder Debian) bereits vorhanden oder lässt sich mühelos herunterladen und nachinstallieren.

[1] Dies ist zur Zeit der "Marktführer"für das kostenlose Hosting von kooperativen Projekten zur Herstellung freier Software im Amateur- bzw. Hobbybereich

[2] http://abstrakraft.org/cwiid/

Die von L. Donnie Smith geschriebene CWiid-Bibliothek ermöglicht es, sich mit der Wiimote-Fernbedienung über Bluetooth zu verbinden und Daten von ihr auszulesen. Man kann hierbei folgende Daten auslesen:

- Daten von den Beschleunigungssensoren über alle drei Achsen
- Daten aller Knöpfe (insbesondere des Steuerungskreuzes)
- Infrarotdaten der IR-Sensoren
- interner Speicher (sowohl Register, als auch EEPROM)

Sofern mindestens eine Erweiterung, wie Nunchuk (siehe 3.1.3) oder MotionPlus (siehe 3.1.3) eingesteckt ist, lassen sich zusätzlich

- die Daten der Gyroskope über alle drei Achsen (MotionPlus)
- die Beschleunigungssensoren der Nunchuk
- der analoge Joystick der Nunchuk
- sowie die beiden Knöpfe der Nunchuk

auslesen. Die CWiid-Bibliothek erlaubt schreibend den Zugriff auf den Speicher der Wiimote. Bedingt lässt sich auch der Lautsprecher der Wiimote ansteuern. Dies ist natürlich überhaupt nicht im Fokus der vorliegenden Arbeit, aber es gibt Bibliotheken, die angeblich ganze mp3-Dateien über diesen Lautsprecher abspielen können.[3]

Aufbau der CWiid Bibliothek

Die Quellen der CWiid Bibliothek bestehen aus insgesamt fünf Kompiliereinheiten, von denen vier selbstständige Programme darstellen:

common Das ist der einzige Teil, welcher kein selbstständiges Programm darstellt. Er stellt gemeinsame Funktionalitäten zur Verfügung. Dazu gehören die (Bluetooth) Verbindung mit der Wiimote, das Auswerten der Datenpakete und verschiedene andere Funktionen.

lswm Dieses Programm ist im Sinne des unter Linux bekannten *ls* Befehls und seiner Erweiterungen. Es listet die derzeit sichtbaren und/oder verbundenen Wiimotes auf. Es ist ein kleines, kurzes, oft aber sehr nützliches Programm.

wmdemo Mit diesem Programm kann man sich in einer Textkonsole fast alle Stati der Wiimote anzeigen lassen und die meisten auch setzen. Man kann z.B. die Daten der Accelerometer einmalig abfragen oder sich ständig über Änderungen, der Knöpfe, der IR-Daten, sowie der angeschlossenen Erweiterung(en) informieren lassen. In der Rückkommunikation kann man die LEDs oder das Vibrationsfeedback (*Rumble Feature*) ein- und ausschalten. Ein Zugriff auf die EEPROMs oder Lautsprecher der Wiimote ist nicht möglich.

[3] www.sourceforge.net

wmgui Als leichtgewichtiges Programm kann diese grafische Bedienoberfläche alles, was *wmdemo* kann. Der Vorteil liegt darin, dass sich alle Informationen "auf einen Blick" präsentieren. Das ist nicht nur beim Verstehen der IR-Daten nützlich. Auch die Accelerometer und ggf. Gyroskopdaten (bei angeschlossener MotionPlus) werden mit Balken signalisiert. Mit dieser Komponente kann man auch die internen Speicher der Wiimote lesen und beschreiben. Beim Schreibvorgang verhält sich die hier benutzte Version etwas instabil. Eine Ansicht der GUI ist in Abb. 4.1 zu sehen.

wminput Dieses Programm lässt die Wiimote laut Website und Dokumentation zu einem vollwertigen Eingabegerät werden, welches als Maus(ersatz), Joystick oder für die Unterstützung von Präsentationen genutzt werden kann. Leider war ein Test mit diesem Treiber nicht erfolgreich.[4]

Nach erfolgreicher Übersetzung und Installation stehen dem Benutzer dann die vier Programme (*lswm wmdemo wmgui wminput*) zusätzlich zu den bisherigen zur Verfügung.

Für die vorliegende Arbeit wurde das Programm *wmdemo* als Grundlage genommen und zu einer *shared library* verändert. Diese wird dann mittels JNI (Siehe Abschnitt 4.2.1) in der hier entwickelten Software (Abschnitt 5) benutzt.

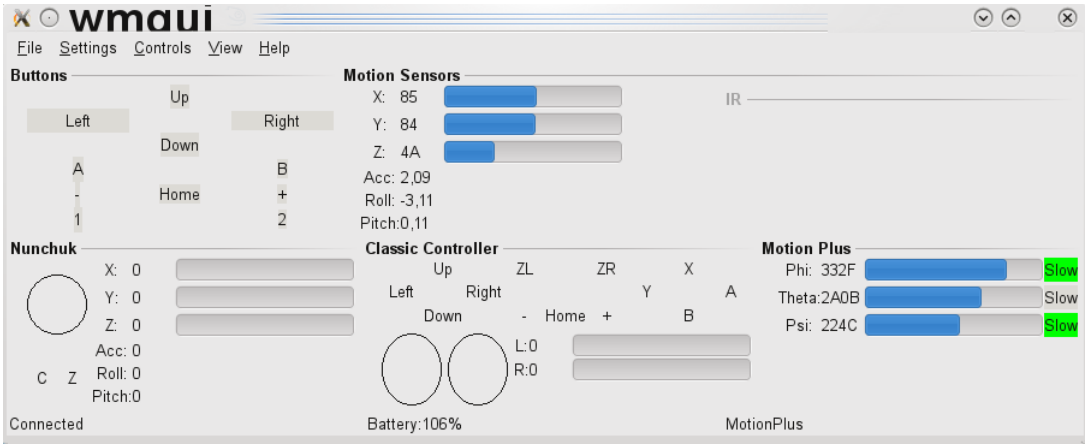


Abbildung 4.1: Die grafische Bedienoberfläche der CWiid (wmgui)

Details und Abschließende Bemerkungen zur CWiid

Die CWiid ist ein hilfreiches Tool, hat aber auch ein paar problematische Eigenschaften. Sie gibt bei Fehlern direkt Informationen auf die Fehlerkonsole aus und die Standardbehandlung von Fehlern besteht in einem Aufruf der Methode *exit(1)*.

[4]Der erste Versuch brachte das System (OpenSUSE) komplett zum Erliegen. Tastatur, Maus und X-Server konnten erst nach 2 intensiven Arbeitstagen wieder in Betrieb genommen werden

Für die Nutzung als shared library ist das ziemlich ungünstig. Bei einem Fehler wird nämlich das ganze (übergeordnete) Programm (auch mit)beendet.

Die while(true)-Endlosschleife, in welcher Daten von der Wiimote über Bluetooth geholt werden, stellt die CWiid in den Mittelpunkt der Verarbeitung. Für eine Bibliothek, welche ressourcenschonend sein und sich starten, pausieren, beenden und wieder starten lassen soll, ist dieses Vorgehen ungünstig. Beim Pausieren werden auf JAVA-Seite dann die Daten zwar empfangen, aber dort schlichtweg verworfen. Das ist auch nötig, da nicht angenommene Daten zu einem Message-Queue Überlauf in der CWiid führen.

Die CWiid bietet zwei Betriebsmodi an. Das sind der *push*- und *pull*-Modus. Beim push-Modus muss man die CWiid nach einem aktuellen Datum fragen, im pull-Modus wird eine callback-Funktion aufgerufen. Der letztere Modus ist allerdings leider so realisiert, dass man bei Wahl von zu großen Intervallen mit Datenverlust rechnen muss und bei zu klein gewählten Intervallen das gleiche Datum mehrfach abgefragt wird und man nicht entscheiden kann, ob es sich dabei um eine Kopie oder identische, aufeinanderfolgende Daten handelt. Für die Realisierung der shared library wurde daher der push-Modus gewählt, in dem die CWiid selbst über eine callback-Funktion auf JAVA Listener zugreift (siehe dazu Abschnitt 5.2.3). Zeitstempel, welche direkt in der Endlosschleife noch in C gesetzt werden, machen jedes Datum eindeutig.

Falls die Batterien der Wiimote schwach sind, bzw. ihr Zustand sich im Betrieb ändert (aus drei anzeigenden LEDs werden zwei), stürzt das Programm im push-Modus undeterministisch ab. Bei Problemen mit dem Bluetooth-Interface beendet sich das Programm ebenfalls, als shared library stürzt es sogar direkt ab. Obwohl die CWiid in manchen Fällen in ihrer Fehlerbehandlung unausgereift scheint, ist sie im Normalfall eine sehr nützliche Hilfe in der eine Menge Implementationsarbeit und Reverse-Engeneering steckt. Das macht sie nicht nur für diese Diplomarbeit sehr wertvoll, sondern auch für das ROS-Projekt, wo die Wiimote-Kommunikation von **ROS-Knoten** mittels CWiid realisiert wurde.

4.2 Programmiertechnische Grundlagen

In der vorliegenden Arbeit wurde die Programmiersprache JAVA [Jav99a] genutzt. Wenn es nötig war, auf externe Bibliotheken zuzugreifen, wie die in C geschriebene CWiid, geschah dies durch das *Java Native Interface* (JNI).

Der Name JAVA hat sich ausgehend von einer indonesischen Insel [Ull07, Kapitel 1] über die Lieblingskaffeesorte der Entwickler zu einer Programmiersprache und später zu einem maschinenunabhängigen Gesamtkonzept entwickelt [Fla05]. So entstanden die Namen für die Entwicklungs-Roadmap *Jakarta* und die Technologie *JAVA-Beans*.

Eine Systemunabhängigkeit mit Programmen, die in einer virtuellen Umgebung ausgeführt werden, ist zwar seit UCSD-Pascal und dessen p-Code ("Pseudocode") nicht neu, die große Stärke von JAVA besteht jedoch in der einheitlichen API [Jav98, Jav99c], welche zentral verwaltet wird. Dafür ist der kleine Nachteil in Kauf zu nehmen, dass Funktionalität, welche nativ auf einem System zur Verfügung gestellt wird (z.B. die CWiid), auf einem umständlich wirkendem Weg für JAVA verfügbar gemacht werden muss. Dazu kommt prinzipiell Kommunikation über Sockets in Frage, oder wie im Falle dieser Arbeit über das *Java Native Interface*.

4.2.1 Das Java-Native-Interface (JNI)

Das *Java Native Interface* (kurz JNI [Gor98]) ist eine Programmierschnittstelle (API), die es ermöglicht von JAVA aus native Programme, Programmteile oder Bibliotheken zu nutzen. Ebenso ist es möglich, mittels JNI von nativer Software aus in Java geschriebene Software zu nutzen. Da JNI ein sehr mächtiges Konzept von entsprechender Komplexität ist, wird in dieser Diplomarbeit nur ein Ausschnitt benutzt. So wurde auf das native Exception-Handling und das Erzeugen einer eigenen JVM und deren Aufruf aus C heraus verzichtet. Es kann jedoch bei Bedarf schnell nachimplementiert werden [Lia99, Seite 73ff, Seite 174]. Hier wurde der Ansatz gewählt, dass jede Logik und Aktion von Java ausgeht, somit eine JVM aus C heraus nicht aufgerufen werden soll. Behandeln von Ausnahmen im nativen Code ist nicht sinnvoll möglich, weil fast jeder Fehler, der in der CWiid auftritt auch unmittelbar zu einem Programmende führt.[5]

Das Java Native Interface wird in der Spezifikation von Sun Microsystems beschrieben [Jav97, Lia99]. Schon im JDK 1.0 wurde ein Interface für nativen Zugriff eingeführt. Das Netscape Java Runtime Interface (JRI) und Microsoft Raw Native Interface (RNI) [Mic99] erlauben ebenfalls nativen Zugriff. Vorteil von JNI gegenüber RNI und JRI ist die Binärkompatibilität zwischen den einzelnen JVM Versionen. Die JVM ist ein Vermittler zwischen Host-Hardware und dem JAVA Programm. Die Bibliothek, welche über JNI aufgerufen wird kann aber als außerhalb dieser JVM und dieser Vermittler-Struktur aufgefasst werden.

Java ruft nativen Code auf

Es wird hier als native Sprache C benutzt. Codebeispiele sind direkt der erstellten Software entnommen. Die Methodensignatur einer aufzurufenden nativen Methode ist systematisch aufgebaut: Sie beginnt mit dem Wort *Java*, gefolgt vom optionalen Packetnamen in vollständiger Tiefe und dann vom Klassennamen. Der letzte Teil

[5] So z.B. ein Fehler bei der Bluetoothverbindung, weil kein Bluetooth-USB-Stick eingesteckt ist; oder eine leere Batterie bei der Wiimote

des Namens ist der Methodenname. Der Methodenname, so wie er in JAVA lautet, muss oft etwas geändert werden, weil manche Zeichen in nativen Methoden mit anderen zu maskieren sind. Als Trennzeichen zwischen den einzelnen Teilen wird ein Unterstrich benutzt, in JAVA ist es ein Punkt. Die Übergabeparameter der JNI-Funktionen sehen immer so aus: `(JNIEnv *, jobject,...)` gefolgt von weiteren optionalen Parametern.

Durch den Pointer auf die `JNIEnv` wird eine Kommunikation mit dem aufrufenden Thread und über ihn mit der JAVA-Laufzeitumgebung überhaupt erst ermöglicht. Der Pointer darf nicht im nativen Code gespeichert werden, weil er sich zwischen einzelnen Aufrufen ändern kann und es mit Sicherheit tut, falls die native JNI-Methode von verschiedenen Threads aufgerufen wird.

Das JAVA-Objekt, an welchem die native Methode gerade aufgerufen wird, ist repräsentiert durch das Schlüsselwort `jobject`. Diese Referenz bleibt stabil, solange die native Methode arbeitet. Sobald sie zurückgekehrt ist, obliegt es der JVM und damit dem Garbage-Collector, wann die Referenz ungültig und das Objekt gelöscht wird. Soll das Objekt über die native Methode hinaus benutzt werden, so ist eine globale Referenz zu erzeugen (`(*env)->NewGlobalRef(env, jobject)`), welche auch vom Programmierer selbst wieder freigegeben werden muss (`DeleteGlobalRef`). Eine Globale Referenz ist die einzige Möglichkeit, wie eine Beständigkeit zwischen Methodenaufrufen erzeugt werden kann und damit eine Möglichkeit, der sich ein callback-Mechanismus von der nativen Sprache aus, bedienen kann.

Auf diese beiden Parameter, welche jede native JNI-Funktion haben muss, können weitere vom Programmierer festgelegte Parameter folgen. Ein Beispiel für eine native JNI-Funktion, wie sie in der libcwiidJNIWrapper.so zu finden ist:

```
JNIEXPORT void JNICALL Java_wiijavalibrary_comm_WiimoteProxy_toggle_1Messages(JNIEnv *env,
jobject jni_input_object, jboolean jni_state)
```

Java wird von nativem Code aufgerufen

Hier sind zwei Fälle zu unterscheiden. In einem wird von einem nativen Programm (z.B. in C oder C++) aus der Zugriff auf JAVA gewünscht und das Programm erzeugt sich eine Virtual Machine (JVM). Konzeptionell erzeugt sich das Programm eine JVM, holt sich Zeiger auf die Laufzeitumgebung (`JNIEnv*`) und kann dann Objekte erzeugen und (ggf. an den Objekten) Methoden erzeugen. Nach erledigter Arbeit wird die JVM beendet.

Deutlich schwieriger ist der Fall zu behandeln, wenn man vom nativen Code eine oder sogar mehrere callback-Funktionen implementieren möchte. Um ein Observer-Modell zu implementieren und die Daten der Wiimote möglichst in Echtzeit zu bekommen, wird dieser Weg hier beschritten. An dieser Stelle wird erläutert, wie eine callback-Funktion mit einem einzelnen C-Thread realisiert werden kann.

Nachdem die geschriebene und fertig kompilierte *shared library* in JAVA geladen wurde, können native Methoden mit ihrer Hilfe benutzt werden. Bevor ein nativer Thread einen *callback* ausführen kann, muss ihm seine erzeugende Umgebung bekannt gemacht werden. Da die Gültigkeit des `JNIEnv*` Pointers auf den Methodenaufruf selbst beschränkt ist, muss ein callback-Thread die Pointer auf die JVM und `jobject` mittels globaler Referenz behalten. Dies war bei der Erstellung des Programmcodes für diese Diplomarbeit nicht ganz einfach.

4.2.2 Java 3D

Die Java 3D Klassenbibliothek ist eine Sammlung von Javaklassen, welche eine Abstraktion von der konkreten Hardware erlauben. Ganz im Sinne der betriebssystemunabhängigen Programmiersprache Java ist auch die Java 3D API auf verschiedener Hardware und verschiedenen Betriebssystemen lauf- und einsatzfähig. So ist es mit Java 3D unter Windows mittels Direct 3D oder Unixderivaten (Linux) auf OpenGL[6] Basis möglich, sehr effizient auf die (Grafik)hardware des Computersystems zuzugreifen, um 3D Grafiken zu erstellen, zu manipulieren und schließlich als sich bewegende Teile auch darzustellen.

Seit dem Jahre 2004 ist Java 3D ein Open Source Projekt, welches von einer Gemeinde weiterentwickelt wird. Die Version 1.0 wurde von Sun Microsystems im Jahre 1998 freigegeben und ist aktuell als Java 3D 1.5 erhältlich.

In Java 3D wird ein rechtshändiges Koordinatensystem verwendet, in welchem sich ein Szenengraph befindet. Innerhalb des Szenengraphen können Objekte hierarchisch angeordnet werden. Hauptsächlich sind es Elemente des Typs *Branchgroup*, die ihrerseits noch Unterelemente vom Typ *Transformgroup* haben können. Eine *Branchgroup* wird benutzt, um eine zusammenhängende Einheit aus mehreren einfacheren Teilkomponenten zu definieren. Durch eine *Transformgroup* ist es dem Ganzen (der *Branchgroup*) oder Teilen möglich, sich zu bewegen. Als Bewegungsmatrizen werden hauptsächlich die Translation und Rotation benutzt, aber auch Scherungen und Skalierungen sind natürlich möglich. Damit Objekte sich bewegen oder auf Umwelt bzw. Zeit reagieren können, muss ein *Behavior* implementiert werden. Das kann beispielsweise nach einer festgelegten Zeit eine Rotation oder eine Aktion, wie eine Änderung der Textur, auslösen, falls andere Objekte oder die Maus in die Nähe kommen.

Ferner können mit Java 3D unter anderem eigene Texturen entworfen, geladen und auf die Objekte gelegt, die Eigenschaften von Licht festgelegt und sogar mehrere Kamerastandorte definiert werden. Diese können sich zur Laufzeit flexibel bewegen.

In der hier entwickelten Software werden Open-source Texturen der Wiimote geladen und zur Simulation der Bewegung der Wiimote(s) verwendet. Für die animierte

[6] OpenGL läuft auch unter Windows

Darstellung wurden eigene *Behavior*, sowie *Branchgroup* und *Transformgroup* implementiert.

4.3 Die Robotersteuerungsbibliothek RCCL

Die Robotersteuerungsbibliothek RCCL (englischer Name: Robot Control C Library) ist eine Zusammenstellung von Funktionen, welche in der Programmiersprache C geschrieben wurde, mit deren Hilfe man leichter und intuitiver Roboterarme recht hardwarenahe steuern kann. Es wurde früher hauptsächlich für die Steuerung vom PUMA-Manipulatoren benutzt, da diese an wissenschaftlichen Einrichtungen und z.T. in der Industrie weit verbreitet haben.
Dadurch, dass RCCL freie Software ist, welche für Lehrzwecke ohne weiteres benutzt, verbreitet und verändert werden darf, kann ihr Bekanntheitsgrad, vor allem an Universitäten, erklärt werden. In RCCL gibt es einzelne Modelle für Roboter(manipulatoren). So kann das benötigte Modul für einen Manipulator geladen werden und RCCL kann der Benutzerin eine Menge[7] Berechnungen der kinematischen Ketten abnehmen. Hier im Arbeitsbereich TAMS wird RCCL benutzt, um die PA10-6C (siehe 3.2.3, Seite 55) Arme von TAMS anzusteuern. Das ist möglich, weil die Bibliothek um Modelle zur Unterstützung der Arme von Thorsten Scherer im Rahmen seiner Dissertation [Sch04] erweitert wurde. Ein weiteres Anwendungsgebiet von RCCL ergibt sich aus der Diplomarbeit [Lab09] von Gunter Labes für die Lehre.
RCCL kann die Kinematik sowohl im kartesischen, als auch im Gelenkwinkelraum berechnen. Es genügt also meistens die kartesischen Koordinaten des TCP[8] oder die Gelenkwinkel des zu steuernden Manipulators anzugeben. Die Berechnungen sind bis auf singuläre Stellungen[9] grundsätzlich erfolgreich. Mittels RCCL können keine Dynamik-Gleichungen gelöst werden.
RCCL wurde 1983 an der Purdue Universität von Vincent Hayward geschrieben [HR86] (Version 1.0). 1985 wurde ein verbessertes Release an der McGill Universität von John Lloyd [Llo85] im Rahmen seiner Master Thesis produziert. In den Jahren 1987-1988 wurden größere Verbesserungen vorgenommen, als die Multi-Robot und Multi-CPU Unterstützung eingebaut wurden (durch Lloyd, Parker, Mcclain [LPM88]). Diese im Herbst 1988 veröffentlichte Version wurde *Version 3.0* genannt. Die Version 4.0 besteht nach eigenen Angaben (Abschnitt 1.1 in [LH92]) aus Bugfixes, Verbesserungen der Dokumentation und einem allgemeinen Aufräumen. 1996 wurde RCCL echtzeitfähig durch Portierung auf geeignete Betriebssysteme. Das war die letzte offizielle Version (5.0). Es folgte noch die Ergänzung 1999 (Version 5.0) an der Universität Bielefeld zur Unterstützung von PA10 Armen (siehe Abschnitt 3.2.3)

[7] im Optimalfall sogar alle
[8] tool center point, die "Mitte der Hand" eines Manipulators
[9] Das sind Stellungen in denen die Determinante der Jacobimatix 0 ist, damit die Matrix nicht (eindeutig) lösbar und nicht invertierbar ist

und 2003 (Version 5.1.4) an der Universität Hamburg die Unterstützung für PA10-6C (siehe 3.2.3). Die letzten beiden Versionen sind beide durch Thorsten Scherer entwickelt [Sch04].

4.4 Die Player-Stage-Gazebo Umgebung

Durch das Player-Projekt ist eine freie, quelloffene Software entstanden, welche für Forschungszwecke nutzbar ist und auch dafür entwickelt wurde [GVH14]. Seitdem es im Jahre 2000 von Brian Gerkey, Richard Vaughan und Andrew Howard in der "University of Southern California" in Los Angeles begonnen wurde, hat sich der Name über "Player/Stage" und "Player/Stage/Gezebo" auf den schlichten Namen "Player-Project" wiederbesonnen. Die in diesem Projekt entwickelte Software kann man als Schnittstelle zwischen Roboterplattform und höheren Funktionen bezeichnen. Es ist einerseits eine Roboterbetriebssystem-Middleware, andererseits eignet es sich zur Visualisierung und Simulation. Die Software ist lauffähig auf allen POSIX-fähigen Betriebssystemen (Linux, Unix, BSD, Mac OS X). Das Projekt selbst besteht aus drei Komponenten:

- Player Netzwerk Server ermöglicht die Abstraktion von konkreter Hardware. Wenn die Treiber für die Hardware geschrieben und geladen sind, dann spielt es nach oben keine Rolle, welcher Roboter unten die Kommandos ausführt. Ein Programm kann, sofern es abstrakt genug formuliert war, auf einer anderen Plattform mühelos weiterbenutzt werden. Als Programmiersprache werden C, C++, Ruby und Python direkt unterstützt, andere (Java) können aber als Bibliotheken nachgerüstet werden. Im Player-Projekt werden Treiber für die Roboter der Pioneer-Reihe (Abschnitt 3.2.1), die mobilen Segway-Plattformen und viele andere Treiber direkt mitgeliefert.
- Stage ist die 2D Simulationsumgebung, welche auf Grund ihrer Skalierbarkeit auch geeignet ist, um mehrere hundert Roboter zu simulieren. Man kann damit nicht nur eigene Programme ausprobieren, sondern auch mittels Player auf simulierte Sensoren und Geräte zugreifen. Es ist auch eine Mischung aus realen und simulierten Robotern in einer Umgebung möglich.
- Gazebo ist eine 3D Simulationsumgebung, die im eigentlichen Sinne 2,5D ist, weil die Höhe nicht variabel ist, sondern alle Objekte die gleiche Höhe besitzen. Wie zuvor Stage, kann auch Gazebo alleine, oder mit dem Player-Server zusammen betrieben werden.

Diese Software in der Gesamtheit ist eine gute Lehr- und Lernsoftware, welche oft in Roboter-Kursen an Hochschulen, aber zunehmend auch in Schüler-Projekten Anwendung findet. Nachdem die Hardware-Treiber einmalig konfiguriert wurden, lassen sich Programme mit wenig Aufwand schreiben. In der Player Umgebung wurden die ersten Experimente dieser Arbeit auf der Pioneer Mobilplattform durchgeführt.

4.5 ROS - Robot Operating System

Das *Robot Operating System* [Gar14], welches kein Betriebssystem im eigentlichen Sinne, sondern eher — wie schon Player/Stage — eine Robotik-Middleware ist, wurde 2007 am Stanford Artificial Intelligence Laboratory im Rahmen des Projektes "Stanford AI Robot (STAIR)" entwickelt und es wird heute vom Robotikinstitut Willow Garage maßgeblich betreut und weiterentwickelt. Es setzt auf ein Betriebssystem, wie Linux, Mac OS X oder Windows auf und unterstützt viele verschiedene Programmiersprachen, mit denen Pakete oder ROS-Knoten geschrieben werden können. Es werden C, C++, C#, Ruby, Python heute schon unterstützt und für eine Java Anbindung gibt es derzeit Bibliotheken, welche über JNI auf C++ zugreifen. Eine vollständige Realisierung der Java-Bibliotheken in Java (ohne JNI) wird derzeit angestrebt. An dieser Stelle kann nur ein grober Überblick über ROS gegeben werden und einige ausgewählte Teile werden näher beleuchtet. ROS setzt sich aus drei konzeptuellen Ebenen zusammen[10]:

- *ROS Filesystem Level* ist die Sammlung der Programme und Tools, welche mit ROS zusammen geliefert werden. So ist es mit diesen Tools, welche als Ergänzung des darunterliegenden Betriebssystems zu verstehen sind, möglich direkt in bestimmte ROS-Verzeichnisse zu springen und dort einen Kompilierprozess anzustoßen, oder zu erfragen, welche Knoten derzeit aktiv sind und welche Dienste sie anbieten. Das geht alles ohne, dass man Kenntnis vom ROS-Installationsort oder den aktiven Diensten haben muss. Bestandteile des Filesystem Levels sind:

  Packages bilden die kleinste Einheit, in der sich ROS-Software zusammenfassen lässt. Ein Package kann Laufzeitprozesse (Knoten), Daten, Konfigurationsdateien oder andere Dateien, die sich sinnvoll zusammenfassen lassen, beinhalten.

  Manifestdateien (manifest.xml) beinhalten u.a. Paketmetadaten, Lizenzinformationen und paketspezifische Abhängigkeiten.

  Stacks sind Sammlungen von Packages, welche eine bestimmte Funktion anbieten. So gibt es einen "navigation stack" in ROS, bei dessen Installation auch die zugehörigen spezifischen Abhängigkeiten bedient werden.

  Ferner gibt es *stack manifests*, die Manifestdateien für Stacks, *message types* und *service types*, welche die Beschreibungen für Nachrichten, bzw. Dienste beinhalten.

- *ROS Computation Graph Level* bildet die Ebene in der die eigentliche Kommunikation stattfindet und algorithmische Logik angesiedelt ist. Es ist ein

[10] Sofern eine übliche deutsche Übersetzung vorhanden war, wurde diese gewählt. Bei ROS-spezifischen Eigennamen wurden die englischen Bezeichnungen entweder im Original belassen oder zumindest in Klammern angegeben.

Peer-to-Peer-Netzwerk der ROS-Prozesse, welche miteinander kommunizieren und auf gemeinsamen Daten arbeiten können. Die Grundlage dieser Ebene bilden die Knoten (*nodes*), der *Master*, der *Parameter Server*, die Nachrichten (*messages*), die Dienste (*services*, die *topics* und die *bags*. Dies sind nötige Komponenten, um eine Punkt-zu-Punkt (1:1) oder eine broadcast (m:n) Kommunikation zu realisieren, welche synchron oder asynchron stattfinden kann. Der *Computation Graph Level* setzt sich zusammen aus:

Knoten (nodes) stellen die logische Beschreibung der kommunizierenden Prozesse dar. In ihnen laufen die Algorithmen und dort werden die Informationen verarbeitet. Knoten können einzelne Teilaufgaben erfüllen, wie die Motoren steuern, Daten der Wiimote auslesen oder die Wegplanung übernehmen.

Master so heißt die Komponente, welche für den Namensdienst und die Registrierung aller anderen Komponenten verantwortlich ist. Ohne den Master könnten Knoten sich nicht sehen, miteinander kommunizieren oder Dienste aufrufen.

Nachrichten (messages) über sie kommunizieren Knoten miteinander. Sie können aus primitiven Datentypen bestehen, oder sich aus primitiven Datentypen zusammensetzen.

Topics sind im weiteren Sinne Beschreibungen von Nachrichten. Knoten können Topics abonnieren, abbestellen und Nachrichten mit einem bestimmten Topic senden. Es können beliebig viele Sender und Empfänger auf einem Topic asynchron arbeiten. Derzeit ist in ROS bereits ein *Wiimote Topic* implementiert.

Dienste (services) werden benötigt, wenn eine synchrone Punkt-zu-Punkt Kommunikation gewünscht ist. So kann ein Knoten einen Dienst anbieten und ein anderer kann diesen aufrufen. Es ist die für viele Fälle nützliche Alternative zur anonymen m:n-Kommunikation mittels Topics.

Bags und Parameter Server sind zum Aufbewahren von Daten da.

- *ROS Community Level* bezeichnet die Konzepte, welche es Communities ermöglichen ROS-Software und -Wissen auszutauschen.

  ROS Distributionen sind Sammlungen von in ihren Versionen zueinander passenden Stacks. Das Konzept ist den Linux-Distributionen entlehnt. Man kann so eine ganze Fülle konsistenter Software installieren.

  Repositories Das Konzept von ROS basiert auf einem Netzwerk von Repositories, die von Menschen oder Institutionen angeboten werden können. Dort können sie dann ihre eigene Robotik-Software entwickeln, ablegen und anbieten. Auch dies ist aus dem Linux-Umfeld bekannt.

Es gibt ferner eine ROS-Mailingliste, ein Wiki und eine Website, welche für Fragen offen steht.

Durch die Tatsache, dass ROS einige Projekte (wie auch Player) in sich aufgenommen hat, aber auch nicht zuletzt durch die beträchtlichen finanziellen Mittel, die in das ROS-Projekt geflossen sind, ist davon auszugehen, dass es sich zu einer bedeutenden Robotersoftware entwickeln wird. In dieser Arbeit wurde ein Knoten geschrieben, welcher auf Wiimotedaten abonniert und diese verarbeitet. Eine neigungsbasierte Steuerung wie in Abschnitt 6.3.6 beschrieben, wurde damit auf dem TASER (Abschnitt 3.2.2) implementiert.

4.6 Zusammenfassung

In diesem Abschnitt wurde die für diese Diplomarbeit verwendete Software vorgestellt. Zuerst wurde die Bibliothek CWiid vorgestellt, die in C geschrieben ist und den Zugriff auf die Wiimote ermöglicht. Nach einer kurzen Einleitung zur Programmiersprache JAVA wurde im nächsten Teil auf das Java Native Interface eingegangen, welches den Zugriff auf native Bibliotheken, wie die CWiid ermöglicht. Eine kurze Einführung in die JAVA-3D-Bibliothek folgte darauf, weil sie für die Visualisierung der erstellten Software genutzt wird. Es folgten Einführungen in die Robot Control C Library (RCCL), die Player-Stage-Gazebo Umgebung und das Robot Operating System (ROS). Diese Bibliotheken bzw. Umgebungen wurden benutzt, um die verwendete Hardware anzusprechen. Dabei ist RCCL für die Steuerung des PA10 Armes geeignet. Mit Player-Stage-Gazebo können die Pioneer-Plattformen gesteuert werden. ROS eignet sich theoretisch nicht nur für den TASER und den PR2[11], sondern könnte auch auf die Pioneers und den PA10 portiert werden, da es die mächtigste Bibliothek von den hier genannten ist. Es kann vermutet werden, dass auf Dauer ROS die anderen Bibliotheken im Bereich der Robotik verdrängen wird.

[11] `http://www.willowgarage.com/pages/pr2/overview`

Im Rahmen dieser Diplomarbeit erstellte Software

5

Die Software, welche im Rahmen dieser Diplomarbeit erstellt wurde, ist in drei Komponenten gegliedert. Es können Daten von der Wiimote geholt, verarbeitet und angezeigt werden. Diese Aufteilung folgt dem Model-View-Controller (MVC) Architektur- bzw. Interaktionsmuster ([LR09, Kapitel 8.2]).

Diese drei Komponenten sind modular gestaltet. Um das zu erreichen wurde verstärkt mit *Interfaces* und *abstrakten Klassen* gearbeitet. So können Teile problemlos ausgetauscht werden.

Mit dem Programm kann man:

- Messdaten als Java-Objekte persistent speichern und lesen
- Messdaten in XML speichern und wieder lesen
- alle Funktionen der CWiid zugreifen (siehe 4.1 auf Seite 58)
- Accelerometer-/Gyroskop- und Nunchukdaten visualisieren
- Wiimote(s) mittels Java-3D visualisieren
- graphische Darstellung der Daten (Plots) aufrufen
- Kalibrierungsexperimente für Gyroskop und Accelerometer starten

5.1 Aufbau der Software - Überblick

In Abbildung 5.1 sieht man die Kohärenz der logischen Softwarekomponenten und wie sie miteinander interagieren können. Softwaretechnisch kann eine Komponente von mehreren JAVA-Klassen kooperativ realisiert werden, entspricht aber meistens genau einer Klasse. Da die Software dem *Singleton* Entwurfsmuster folgt, ist der Controller mittig angeordnet und kann nur einmalig instanziiert werden. Mit dieser Komponente können die anderen Komponenten uni- bzw. bidirektional kommunizieren.

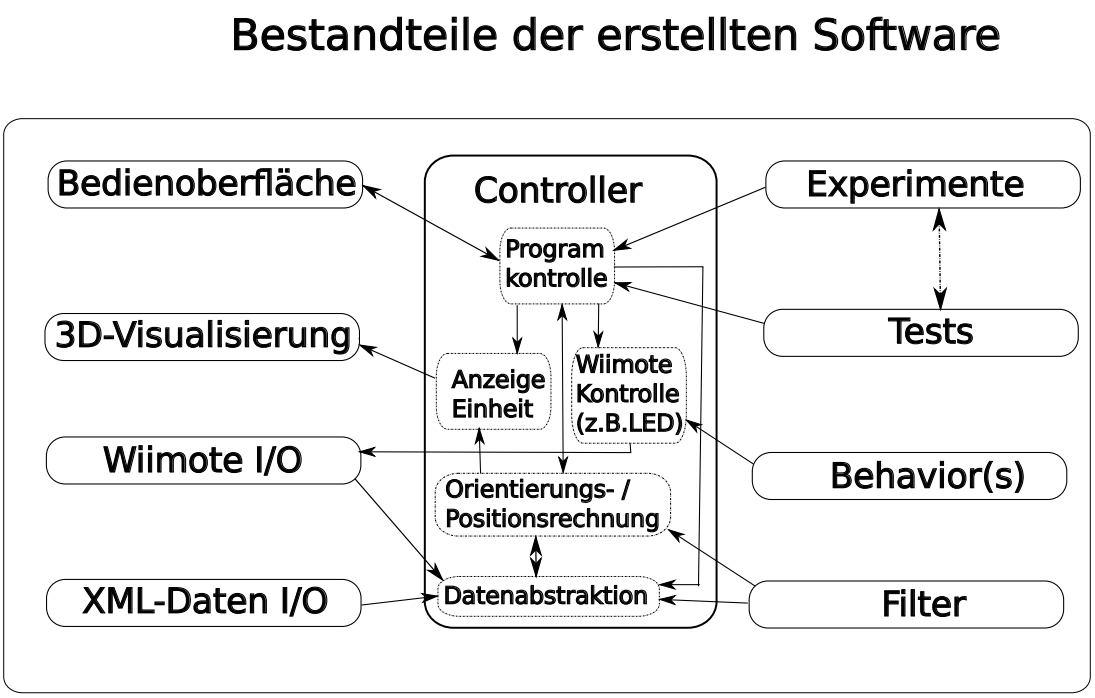


Abbildung 5.1: Aufbau der erstellten Software

So kann die graphische Bedienoberfläche (Abb.5.4, Abschnitt 5.1.2) Kommandos vom Benutzer an den Controller weitergeben und Daten anzeigen. Die 3D-Visualisierungskomponente (Abb.5.5(a), Abschnitt 5.1.2) ist unidirektional an den Controller angeschlossen und zeigt die Orientierung einer oder zwei Wiimotes an. Die Ein-/Ausgabekomponenten kommunizieren in beide Richtungen, wobei nur das Lesen der Daten von der Wiimote in regelmäßigen Intervallen (10ms) geschieht, während die Übrigen Ein-/Ausgaben einmalige Ereignisse beim Programmstart (Lesen der Konfigurationsdaten) bzw. am Ende eines Experimentes (Konfigurationsdaten in Wiimote schreiben) oder einer Experimentreihe (Daten in XML sichern) sind.

Die Komponenten *Experimente* und *Tests* repräsentieren die hier durchgeführten Experimente und ggf. dazugehörige evaluative Tests. Tests können auch kurze Programme sein, welche eine Bestimmte (Teil-)Aufgabe erledigen, bzw. einen "Proof of Concept" realisieren. Beide (Experimente und Tests) nehmen Einfluss auf die Programmkontrolle und damit das Verhalten des Controllers. Sie bestimmen, was aktuell getan wird bzw. werden kann.

Die *Behaviors*-Komponente, ist dazu entwickelt worden, dem Benutzer zu ermöglichen ein Feedback zu erhalten. Es können visuelle oder haptische Reize eingesetzt werden, die sich in ihrer Frequenz und Intensität unterscheiden können. In dieser Arbeit wurde u.A. ein *Behavior* implementiert, welches standardmäßig im Controller gesetzt wird und nach erfolgreichem Verbinden mit der Wiimote alle LEDs nacheinander aufleuchten lässt und die Vibration als taktiles Feedback ein-

setzt.

In der *Filter*-Komponente (Abschnitt 5.2.2) sind die Softwareteile zusammengefasst, welche Einfluss auf die Messdaten und Positions- bzw. Orientierungsberechnung nehmen. Sie dienen der Rauschfilterung, Datenfusion oder Vorverarbeitung der Daten und können grundsätzlich ineinander verschachtelt werden.

Die Hauptaufgabe der *Controller*-Komponente(n) liegt in der Programmkontrolle und Interaktion mit dem Benutzer. Ein Teil verarbeitet die Eingaben vom Benutzer, während ein anderer für das Feedback an die Bedienoberfläche, oder Wiimote zuständig ist. Die Art des Feedbacks kann über Behavior modifiziert werden. Ein dediziertes Modul ist für die Berechnung der Orientierung, bzw. der Position der Wiimote verantwortlich. Hier kann mit Filtern Einfluss auf die Berechnung genommen werden und es können verschiedene Algorithmen evaluiert werden.

5.1.1 Programmiertechnische Details

Die Software ist strukturiert in sieben *packages*[1]:

comm In diesem package sind die Klassen, die für die Kommunikation von und zur Programmhauptkomponente verantwortlich sind.

- **FileHandler** Ist für die XML-Ein/Ausgabe der Daten in Dateien zuständig.
- **WiimoteData** Repräsentiert einen kompletten Datensatz zu einem Zeitpunkt.
- **WiimoteDevice1, WiimoteDevice2, WiimoteProxy** Klassen zur Abstraktion von konkreten Geräten, die die Unterscheidung zwischen zwei Wiimotes erst möglich machen (siehe dazu 5.2).

Ferner sind in diesem package Interfaces und Listener definiert.

control In diesem package ist der *Controller* selbst, der zugehörige *PoseComputer* zur Berechnung der Pose der Wiimote, die verschiedenen *Behavior*, sowie die *Collector*-Klasse zum Sammeln der Wiimotedaten.

experiments Dieses package enthält die Experimente, sofern sie nicht aus zusammengesetzten Klassen, wie Tests oder parametrisierten Programmaufrufen bestehen. Experimente, welche mittels C, C++ oder ROS durchgeführt wurden, sind hier nicht vorhanden.

filter Hier sind alle Filter drin. Diese werden in Abschnitt 5.2.2 detaillierter beschrieben.

gui Die Klassen für die Bedienoberfläche, das Plotten der Graphen und die 3D-Visualisierung liegen hier.

[1] die vollständige Auflistung ist in A.2 auf Seite VIII zu sehen

tests Verschiedene Klassen zum Testen kurzer Sachverhalte, die nicht in die anderen packages einsortiert wurden. Das sind Klassen zum Lesen/Schreiben der Kalibrierungsdaten von/zur Wiimote, Berechnung der Regressionsgeraden, Armsteuerung des PA10, Plotten, sowie weitere Test-Klassen.

util Hier sind Klassen, die als Werkzeuge dienen. Die Kalibrierungsexperimente der Accelerometer und Nullwerte der Gyroskope, sowie Klassen für das Logging, als auch die Repräsentation einer Pose sind in diesem package.

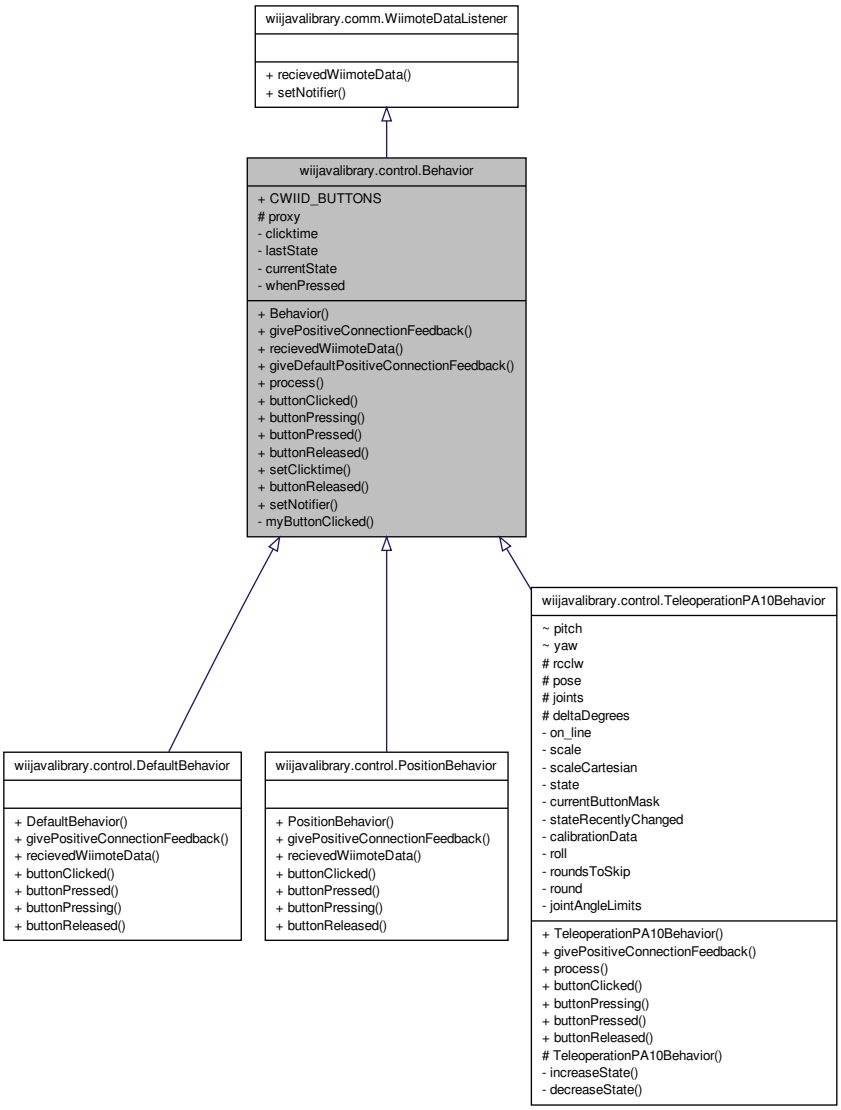
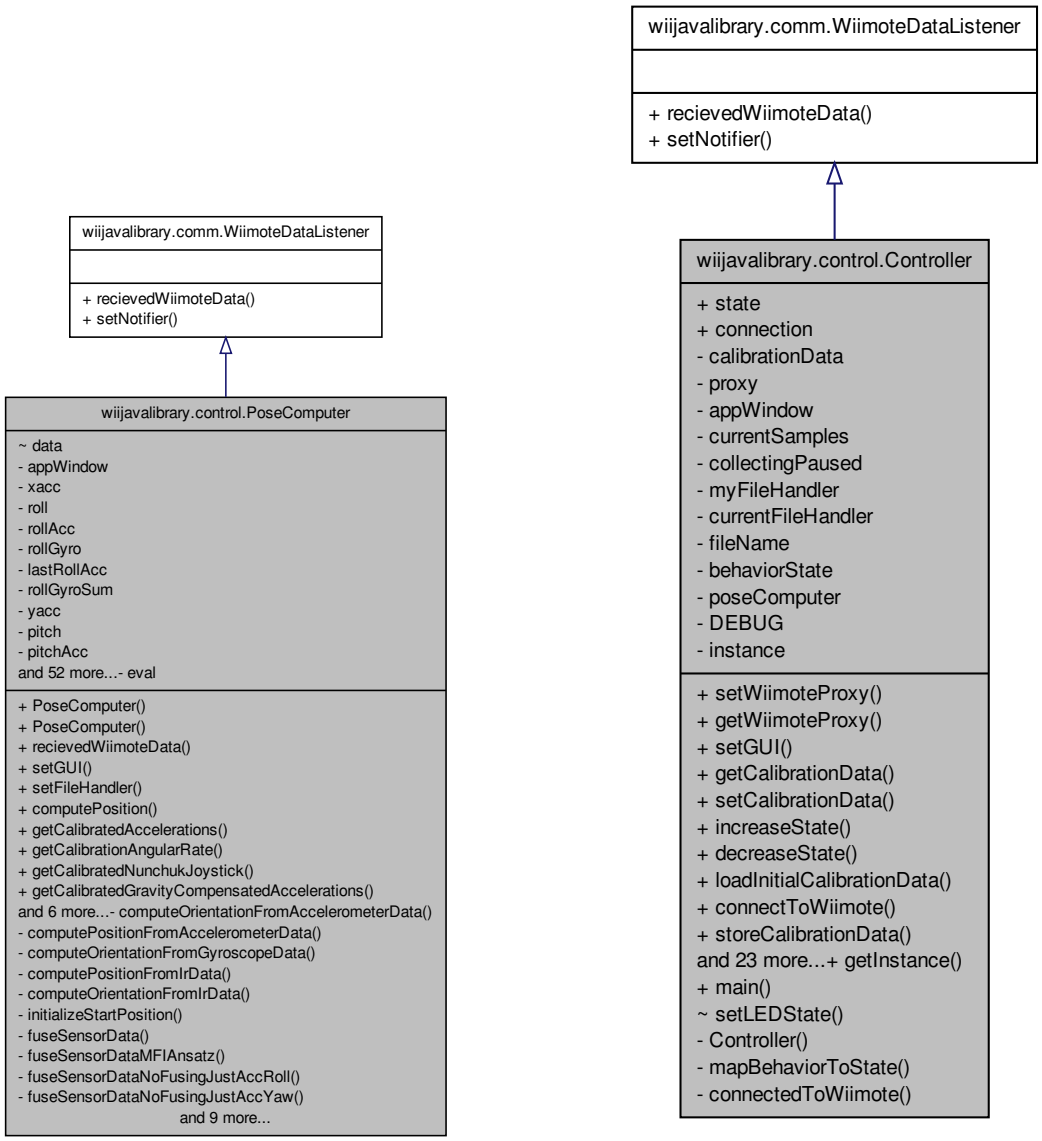


Abbildung 5.2: Klassendiagramm der Behavior-Klassen



(a) Klasse, welche die Pose einer Wiimote berechnet

(b) Zentrale Steuerungsklasse, erbt von WiimoteDataListener

Abbildung 5.3: Klassendiagramme der Klassen Controller und PoseComputer

5.1.2 Graphische Bedienoberfläche und 3D-Visualisierung

Die graphische Bedienoberfläche (GUI) der hier vorgelegten Software, erlaubt dem Nutzer mit dem Controller interaktiv zu kommunizieren. Wie in Abb. 5.4 sichtbar, sind oben in der GUI vier Textfelder angebracht, welche aktuelle Daten, wie z.B. Status der Verbindung (connected/disconnected), Entfernung zur IR-Kamera, Position der IR-Quellen und andere, bereits gefilterte bzw. fusionierte Daten, angeben. Diese Felder können vom Programmierer mit aktuell gewünschten Daten beschrieben werden. Eine Zeile tiefer sind acht Knöpfe zum Bedienen des Programms, welche am Ende dieses Abschnittes beschrieben werden.

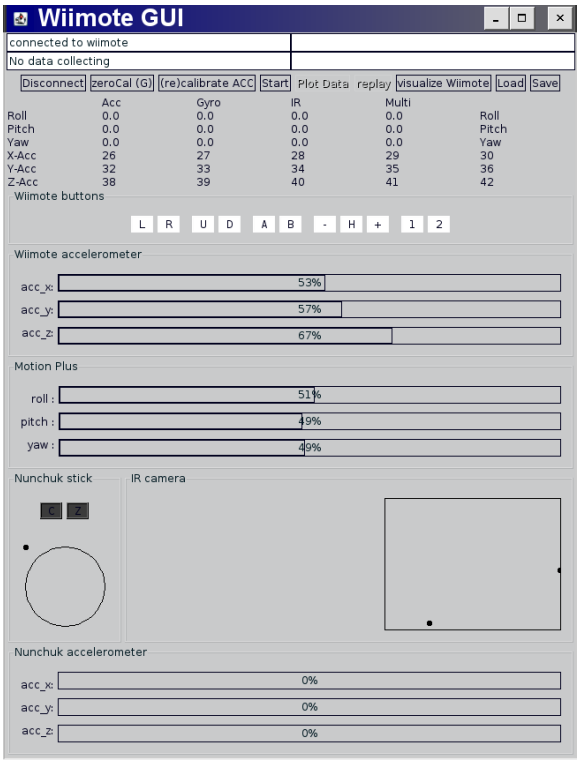


Abbildung 5.4: Die GUI der hier erstellten WiiJavaLibrary

In dem folgenden Array von Feldern können verschiedene Daten zur Laufzeit angezeigt werden. Aktuell sind das die berechneten roll, pitch, yaw Winkel der Wiimote, welche mittels Accelerometer, Gyroskop, Infrarotkamera alleine und fusioniert angezeigt werden, sowie verschiedene andere Werte nach der Anwendung verschiedener Filter.

Im mittleren Drittel der GUI folgt die Anzeige, welcher Knopf der Wiimote gerade gedrückt ist, sowie die genormten Werte (Bereich 0 bis 100) für Accelerometer, Gyroskop, während sich die Anzeige der Accelerometer der Nunchuk-Erweiterung am unteren Bildrand befindet.

Im unteren Drittel ist die Visualisierung für die Position des Nunchuk-Joysticks, sofern angeschlossen und die Darstellung der maximal vier Infrarotpunkte der Kamera (von denen in Abb.5.4 zwei sichtbar sind).

Die acht Knöpfe aus Zeile zwei, die der Bedienung des Programms dienen, sind:

- *Connect/Disconnect* Knopf zum Verbinden mit der Wiimote via Bluetooth.
- zwei Knöpfe zum Re-/Kalibrieren der Gyroskope und der Accelerometer; mit ihnen wird entweder die Nullposition der Gyroskope neu bestimmt, oder das Experiment zur Accelerometerkalibrierung gestartet.
- Knopf zum Starten/Pausieren/Beenden der Aufzeichnung der Messdaten.

- Knopf für die Anzeige von Graphen der Messdaten. Diese Anzeige ist vielfältig konfigurierbar.
- *replay* Knopf zum Wiederabspielen eines Experimentes. Dies ist wichtig, falls man verschiedene Experimente mit den selben Messdaten machen möchte. Dies ist nur möglich, indem man offline Daten einliest.
- Der *visualize Wiimote* Knopf startet die 3D-Visualisierung, beschrieben im folgenden Abschnitt (5.1.2).
- Die beiden Knöpfe auf der rechten Seite dienen dazu die XML-Daten zu laden bzw. zu speichern.

3D-Visualisierung

(a) Repräsentation der Orientierung einer Wiimote in Java 3D

(b) Darstellung eines angewinkelten Arms erstellt aus Daten von zwei Wiimotes

Abbildung 5.5: 3D-Visualisierung von einer und zwei Wiimotes, hier schematisch als Gelenk dargestellt

Die 3D-Visualisierung für eine Wiimote ist in Abb.5.5(a) zu sehen. Dabei wurde eine quelloffene Textur und Objektbeschreibung für ein JAVA3d-Wiimote-Objekt verwendet. Es werden die roll, pitch und yaw Winkel der Wiimote gesetzt, sodass man die Auswirkungen der verschiedenen Berechnungs- und Fusionsverfahren unmittelbar auf dem Bildschirm sehen kann.

Zwei Wiimotes werden in dieser Arbeit für die Steuerung eines Armes benutzt, sodass eine schematische Darstellung, wie in Abb.5.5(b) sinnvoller ist. Es werden

hierbei einfache 3D-Objekte, wie Zylinder und Kugeln benutzt, um eine Schulter mit einem Oberarm und einem Unterarm zu repräsentieren.

Obwohl sie als Plugins für die Bedienoberfläche gedacht sind, lassen sich beide 3D-Visualisierungen auch im stand-alone Modus starten. Die Berechnung der Orientierung, bzw. der Pose der Wiimote(s) muss dann allerdings in einem separaten Behavior realisiert werden, welches der Visualisierungskomponente bekannt gemacht werden muss.

5.2 Implementationsdetails

Die Software für diese Diplomarbeit wurde in JAVA geschrieben und nutzt eine in C verfasste Bibliothek, die CWiid. Für den mittels Java Native Interface realisierten Zugriff sind in der Programmiersprache C Methoden[2] geschrieben worden, die zur Laufzeit in zugehörige JAVA-Klassen geladen werden müssen. Da die CWiid nicht für den simultanen Einsatz mehrerer Wiimotes entwickelt wurde, besteht der Workaround in dieser Diplomarbeit darin, dass für jede Wiimote eine eigene Kompiliereinheit in C mit individuellem Namen erzeugt und von JAVA aus als eigene Bibliothek (shared library) geladen wird.

Aus diesem Grund, gibt es für jede zu nutzende Wiimote eine eigene *WiimoteDevice* Klasse. Dieses Vorgehen ist hier pragmatisch gewählt, weil es sehr unwahrscheinlich ist, dass mehr als zwei Geräte simultan benutzt werden und der Aufwand für eine Namensänderung und Neukompilierung im Falle weiterer Wiimote-Geräte deutlich kleiner ist, als das Neuschreiben der quelloffenen CWiid-Bibliothek, was die Alternative zu dem Vorgehen hier darstellt.

Zwar kann die CWiid, wie in Abschnitt 4.1 beschrieben, in zwei Betriebsmodi, im *push*- oder *pull*-Modus, genutzt werden, jedoch hatte der *pull*-Modus, bei dem man die Daten aktiv von der CWiid holt, im praktischen Einsatz erheblich größere Latenzen, als der andere Modus und führte letztlich sogar zu Datenverlust, in Form nicht abgefragter Messdaten.

Daher wird im JAVA-Teil der Software der *push*-Modus der CWiid genutzt, der alle 10ms ein neues Datum liefert. Hierzu wird der native Thread der CWiid benutzt, um eine Liste von Listenern auf der JAVA Seite zu informieren. Dies hat zur Folge, dass die Methode *recievedWiimoteData* der Klasse WiimoteDataProxy vom nativen CWiid-Thread aufgerufen wird, und von dort aus die Abarbeitung der WiimoteDataListener-Liste des WiimoteProxy angestoßen wird. Also sollten WiimoteDataListener in ihrer *recievedWiimoteData* Methode nicht zu lange verweilen. Es bringt die Einschränkung mit sich, dass alle Berechnungen innerhalb dieser Listener Liste bis zum Auftreten des nächsten Datums abgearbeitet sein müssen. Diese Zeitspanne entspricht in der Regel 10 Millisekunden. Während der Verarbeitung sind

[2] siehe dazu Abschnitt 5.2.3

sleep(), oder Warten (*wait()*) nicht zulässige Operationen. Falls die Zeitbedingungen von 10ms verletzt werden, so findet die Datenverarbeitung der Wiimotedaten nicht mehr in Echtzeit statt. In weniger günstigem Falle läuft die Messagequeue der CWiid voll und das Programm stürzt ab. Da aktuell die Verarbeitung der Daten weit unter dieser Grenze liegt, kann das Problem nicht auftreten, sollten aber komplexere Verarbeitungen der Daten gewünscht werden, so müsste ein *Producer-Consumer* Modell im WiimoteProxy implementiert werden, bei dem der native Thread mit *recievedWiimoteData* die Daten in eine Datenqueue packt (*produce*) und ein neu zu erstellender Thread sie sich bei Bedarf der Reihe nach holt (*consume*).

5.2.1 Aufbau und Speicherung der Daten

Während der Ausführung werden zwei Arten von Daten generiert und gespeichert. Die einen sind die laufend gesammelten Daten vom Typ *WiimoteData*. Die anderen sind die Kalibrierungsdaten vom Typ *CalibrationData*. Die Kalibrierungsdaten werden sowohl auf Festplatte, als auch im EEPROM Speicher der Wiimote vorgehalten. So können bei einem Arbeitsplatzwechsel die zugehörigen Kalibrierungsdaten aus jeder kalibrierten Wiimote beim Programmstart ausgelesen werden.

Zum Sichern auf Festplatte werden beide Datenarten in ein XML-Format umgewandelt und als ASCII-Text gespeichert oder über ein Netzwerk transportiert. Für das Umwandeln in XML wird die Bibliothek **XStream**[3] benutzt. XStream ist quelloffen und steht unter der BSD-Lizenz[4] zur Verfügung. Die XStream Bibliothek wurde gewählt, weil sie folgende Vorteile hat:

- Sie ist sehr einfach zu nutzen.
- Die zu serialisierenden Objekte müssen nicht angepasst oder anderweitig geändert werden.
- Anders, als bei anderen XML-Serialisierungsverfahren, ist es unnötig Abbildungsvorschriften zwischen XML und JAVA-Objekten zu definieren.
- Ganze Objektgraphen können abgelegt werden. Objekte als Attribute von Objekten werden also auch serialisiert. Dabei werden auch zyklische und mehrfache Referenzen automatisch erkannt und aufgelöst.

Ausschlaggebend für die Benutzung dieser Bibliothek war, dass sie ohne weiteren Aufwand benutzbar ist. Man muss nur angeben, welcher Klasse das Objekt angehört (*WiimoteData.java*), welches in XML gewandelt werden soll und wie das oberste XML-Tag heißen soll. In dieser Arbeit heißt es grundsätzlich *wiimotedata*. Objekte werden mit einem *ObjectOutputStream* geschrieben und durch den passenden *ObjectInputStream* gelesen. Als Benutzer kann man alle Arbeit der Bibliothek überlassen.

[3] `http://xstream.codehaus.org/`
[4] `http://www.opensource.org/licenses/bsd-license.php`

Aufbau der Daten

Die Objekte *CalibrationData* und *WiimoteData* dienen dazu die entsprechenden Daten zu repräsentieren. Die *WiimoteData* Klasse enthält alle nötigen Attribute, um die gemessenen Daten der Wiimote mit angeschlossener Nunchuk und der MotionPlus zu repräsentieren. In Listing 5.1 sind die oberen 37 Zeilen der Klasse *WiimoteData* zu sehen. Dort befinden sich die Attribute der Klasse. Die Kalibrierungsdaten sind etwas anders aufgebaut, wie man in Listing 5.3 sehen kann. Dort sind die bereits ausgerechneten Regressionsgeraden und wegen ihrer besonderen Bedeutung die Null-Werte, sowie die Werte für ±1g gespeichert. Dazu folgt im entsprechendem Abschnitt mehr.

Aufbau von WiimoteData Für den Zeitstempel wird der Datentyp *long* benutzt, welcher zum Erzeugungszeitpunkt des Objektes noch im C-Programm gesetzt wird. Die Attribute *type*, *btID* und *notes* sind vom Typ *String*. Im Feld *type* wird gespeichert, ob die gesammelten Daten vom Gyroskop, Accelerometer, der IR-Kamera oder der Nunchuk sind. So kann in einem Datensatz nach Daten von einem Sensor gefiltert werden. Jedes Datum hat die Bluetooth-Adresse der Wiimote gespeichert, welche es generiert hat. Das geschieht im Attribut *btID*. Im Notizfeld können weitere Einträge eingetragen werden. Es ist hauptsächlich angelegt worden, damit es mit sehr wenig Aufwand später auch vom C-Programm mit Daten gefüllt werden kann. Dort ist Platz für Annotationen, welche das Auswerten der Daten, wenn sie in XML vorliegen einfacher machen können.

Die restlichen Attribute sind primitive Datentypen, wie *boolean*, *short*, *int* bzw. ein *int[]* Array. Sie sind für die jeweilige Information möglichst kompakt und kompatibel zu den C-Pendants gewählt. In den booleschen Variablen werden die Zustände der LEDs und der Vibrationseinheit kodiert, sowie die Information, ob das Gyroskop gerade im **schnellen** oder **langsamen** Modus Daten liefert. Als *short* werden die Werte der Accelerometer der Wiimote und der Nunchuk, sowie die Position des Joysticks und die Zustände der Knöpfe von Nunchuk, Wiimote und der Batterieladezustand kodiert. Die Daten der Gyroskope mussten zwecks Kompatibilität zum Programm in C als *int* und die der IR-Kamera als *int[]* Array implementiert werden. Von der IR-Kamera werden maximal 4 Punkte und 3 Intensitätsstufen der Signale geliefert, sodass dieses Array von der Größe 12 sein kann. Es werden für jeden Punkt Abszisse, Ordinate und Intensität kodiert und dann folgt der nächste Punkt mit einem Tripel. Das Array kann auch leer sein, falls keine IR-Quellen entdeckt werden. Für alle übrigen Werte gilt: Sofern der zugrundeliegende Sensor keine Werte liefert, werden sie auf 0 gesetzt. Stellt 0 einen gültigen Wert dar, so wird -1 genommen. So kann es passieren, dass für den Wert der Beschleunigung der Nunchuk der Wert -1 gelesen wird, falls der Erweiterungscontroller nicht angeschlossen ist.

Listing 5.1: Attribute der *WiimoteData* Klasse

```
public class WiimoteData implements Serializable,Cloneable{
        private static final long serialVersionUID = 8469041770597246340L;

        private long timeStamp;
        private long nanoStamp;
        private short x_acc;
        private short y_acc;
        private short z_acc;
        private short buttons;
        private short battery;
        private boolean rumble;
        private boolean led1;
        private boolean led2;
        private boolean led3;
        private boolean led4;
        private int[] ir_Data;

        private int x_rot;
        private int y_rot;
        private int z_rot;
        private boolean slow_speed_x;
        private boolean slow_speed_y;
        private boolean slow_speed_z;


        private short nunchuk_stick_x;
        private short nunchuk_stick_y;
        private short nunchuk_buttons;
        private short nunchuk_acc_x;
        private short nunchuk_acc_y;
        private short nunchuk_acc_z;

        private String type;
        private String btID;
        private String notes;
//...
}
```

Listing 5.2: Ein Beispieldatensatz der Wiimote in XML

```
<object-stream>
  <wiimotedata>
    <timeStamp>1297275656600</timeStamp>
    <nanoStamp>32347638127878</nanoStamp>
    <x__acc>129</x__acc>
    <y__acc>143</y__acc>
    <z__acc>108</z__acc>
    <buttons>0</buttons>
    <battery>0</battery>
    <rumble>false</rumble>
    <led1>false</led1>
    <led2>false</led2>
    <led3>false</led3>
    <led4>false</led4>
    <x__rot>0</x__rot>
    <y__rot>0</y__rot>
    <z__rot>0</z__rot>
    <nunchuk__stick__x>-1</nunchuk__stick__x>
    <nunchuk__stick__y>-1</nunchuk__stick__y>
    <nunchuk__buttons>-1</nunchuk__buttons>
    <nunchuk__acc__x>0</nunchuk__acc__x>
    <nunchuk__acc__y>0</nunchuk__acc__y>
    <nunchuk__acc__z>0</nunchuk__acc__z>
    <type>acc</type>
    <btID>00:19:1D:61:4C:D6</btID>
  </wiimotedata>
</object-stream>
```

Aufbau von CalibrationData Die Kalibrierungsdaten unterscheiden sich von den gesammelten Daten, dahingehend, dass in den *CalibrationData* bereits die Ausgleichsgeraden nach einer Messreihe gespeichert werden. Zum Aufbau des Experimentes und der Messreihe siehe Abschnitt 6.3.5. Die Idee ist, dass man in den Kalibrierungsdaten eine Kennlinie für die Beschleunigungs- und Gyroskopsensoren

abspeichert. So ist es leichter möglich aus den Messdaten direkt in die Erdgravitation umzurechnen. Nach der allgemeinen Geradengleichung

$$g = a \cdot x + b \tag{5.1}$$

ergibt sich die Erdgravitation (g) sofort, nach dem man einen Messwert für die Variable x einsetzt.

Die dazugehörigen Parameter der Geraden wurden nach der Methode der kleinsten Fehlerquadrate bzw. dem Algorithmus von Marquardt-Levenberg ermittelt [GMW81, Mor78, Shu10]. Damit ergibt sich, dass in den Kalibrierungsdaten, wie in Listing 5.3 ersichtlich, die sechs Steigungen der jeweiligen Geraden (in Gleichung 5.1 als a und in Listing 5.3 als *_gradient* notiert) und die y-Abschnitte der selbigen (in Gleichung 5.1 mit b und in Listing 5.3 mit *_offset* bezeichnet) für die drei Accelerometer und Gyroskope der Wiimote festgehalten werden. Der Datentyp ist jeweils eine Gleitkommazahl mit doppelter Präzision (ein JAVA *double*). Ferner speichern die Kalibrierungsdaten einen Zeitstempel (*long*) und die Bluetooth-Adresse der Wiimote, von der sie stammen.

Zusätzlich zu diesen Daten werden die Ruhewerte der Accelerometer und Gyroskope gespeichert, sowie die Werte für die negative und positive Gravitation. Zwar könnten diese aus der gespeicherten Geradengleichung berechnet werden, jedoch sind sie für die Minimierung von Messfehlern in der Ruheposition vorteilhaft. Um den Einfluss der Erdbeschleunigung auf die Bewegungs- und Positionsberechnung klein zu halten, werden diese Werte ebenfalls direkt gespeichert und mit den Messwerten in geeigneter Weise verrechnet. Die Klasse, welche die Kalibrierungsdaten repräsentiert, hat zusätzlich noch die Information über die Länge der Kalibrierungsdaten in Bytes und die Version der Speichervorschrift.

Eine Methode zur Erkennung und damit zur Unterscheidung der angeschlossenen Nunchuk ist nicht bekannt. Somit werden keine Kalibrierungsdaten für sie gespeichert, weil sie der angeschlossenen Nunchuk nicht zuzuordnen wären.

Listing 5.3: Attribute der Kalibrierungsdaten

```
public class CalibrationData {

        public static final int lenghtInBytes = 98;

    private double x_acc_gradient;
    private double x_acc_offset;
    private double x_acc_zero;
    private double x_acc_plusOneG;
    private double x_acc_minusOneG;

    private double y_acc_gradient;
    private double y_acc_offset;
    private double y_acc_zero;
    private double y_acc_plusOneG;
    private double y_acc_minusOneG;

    private double z_acc_gradient;
    private double z_acc_offset;
    private double z_acc_zero;
    private double z_acc_plusOneG;
    private double z_acc_minusOneG;
```

```
23    private double x_gyro_gradient;
      private double x_gyro_offset;
      private double x_gyro_zero;

      private double y_gyro_gradient;
28    private double y_gyro_offset;
      private double y_gyro_zero;

      private double z_gyro_gradient;
      private double z_gyro_offset;
33    private double z_gyro_zero;

      private long timeStamp;

      private String btId;
38  //...
    }
```

Listing 5.4: Kalibrierungsdaten in XML

```
 1 <object-stream>
     <calibrationdata>
       <x__acc__gradient>27.13704506931142</x__acc__gradient>
       <x__acc__offset>133.0711111111111</x__acc__offset>
       <y__acc__gradient>26.645002627050633</y__acc__gradient>
 6     <y__acc__offset>127.50999999999999</y__acc__offset>
       <z__acc__gradient>27.616898360965745</z__acc__gradient>
       <z__acc__offset>122.81444444444443</z__acc__offset>
       <x__gyro__gradient>0.0079214195183776 94</x__gyro__gradient>
       <x__gyro__offset>-64.2031051964512</x__gyro__offset>
11     <y__gyro__gradient>0.00808871704858892 3</y__gyro__gradient>
       <y__gyro__offset>-62.24754435994931</y__gyro__offset>
       <z__gyro__gradient>0.008392852646686082</z__gyro__gradient>
       <z__gyro__offset>-62.567332065906214</z__gyro__offset>
       <timeStamp>1299947668327</timeStamp>
16     <btId>00:17:AB:39:FE:75</btId>
     </calibrationdata>
   </object-stream>
```

Aufbau von den XML Dateien Listing 5.4 zeigt die Kalibrierungsdaten als XML Datei und in Listing 5.2 sind exemplarische WiimoteDaten zu sehen. Man kann erkennen, dass beide Dateien mit dem Tag *<object-stream>* beginnen und dem passenden Tag *</object-stream>* enden. Dies ist bereits in der XStream Bibliothek so implementiert. Es ist ersichtlich, dass überall dort, wo in den Variablennamen vorher ein Unterstrich war, jetzt ein doppelter Unterstrich vorhanden ist. Sonst sind die Tagnamen den Variablennamen sehr ähnlich und für Menschen leicht lesbar. Dies ist eine für diese Arbeit sehr nützliche Eigenschaft der XStream Bibliothek.

5.2.2 Filter

In der vorliegenden Software sind Filter realisiert durch Klassen, welche empfangene Objekte vom Typ *WiimoteData* bekommen, diese verarbeiten und Objekte vom selben Typ zurückgeben. So können mehrere Filter benutzt werden, welche hintereinander geschachtelt werden und damit mächtigere Eigenschaften besitzen können, als die primitiven Filter. Da die Software (im *Controller*, bzw. *WiimoteProxy*) erwartet, dass ein Filter gesetzt wird, ist beim Initialisieren das *PassThrough*-Filter per default gesetzt, welches die empfangenen Daten ohne Änderung oder Verzögerung durchreicht.

Als weitere Filter stehen u.a. zur Verfügung: *AverageFilter*, *DelayFilter*, *LowPassFilter*, *MedianFilter*, *MultiModalFilter*, *OscilationFilter*, *StateFilter*, *CompositeFilter*, *DistributionFilter*, *KalmanFilter*, *PartikelFilter*, *CalibrationFilter*

Das sind die Eigenschaften der Filter:

AverageFilter Das AverageFilter oder Mittelwert Filter bildet den arithmetischen Mittelwert über die letzten n Datensätze. Das n wird bei der Erzeugung des Objektes angegeben. Wird kein n angegeben, so geht das Filter von n=*3* aus. Vorteilhaft an diesem Filter ist, dass es (weißes) Rauschen glättet. Allerdings verzögern sich die Daten linear mit wachsendem n um den Faktor n*10ms bzw. das Intervall in dem die Daten im Schnitt ankommen.

CompositeFilter Mit diesem Filter können mehrere Filter hintereinander geschaltet, oder anders kombiniert werden.

DistributionFilter Ist eine Distribution der Werte bekannt, so kann dieses Filter eingesetzt werden. Unter der Annahme eine Normalverteilung lässt sich damit Signalrauschen mildern.

DelayFilter Dieses Filter ist dazu da, um die Messdaten verzögert auszuwerten. Dies ist im praktischen Einsatz gelegentlich nützlich.

LowPassFilter Ist ein Tiefpass Filter, welches hochfrequentes Rauschen entfernt bzw. mildert.

MedianFilter Das MedianFilter gibt den Median einer Messreihe von n Werten zurück. Initial ist n=3. Ausreißer werden damit komplett bereinigt.

MultiModalFilter In diesem Filter wurden mehrere z.T. eigene Filteransätze ausprobiert, welche sowohl Gyroskop-, als auch Accelerometerdaten berücksichtigen. Eine Berücksichtigung der IR-Daten ist in Zukunft möglich.

5.2.3 JNI Wrapper

Die Datei *cwiidJNIWrapper.c* ist auf dem Beispielprogramm *wmdemo.c* aus der CWiid aufgesetzt. Sie wird zur **libcwiid.so** kompiliert und kann in JAVA-Klassen als shared library geladen werden. Da jede shared library in JAVA nur ein Mal geladen werden kann, ist es nötig für mehrere zu nutzende Wiimotes ebenso viele shared libraries zu erzeugen.

Die Hauptaufgabe der shared library besteht darin, dass sie mittels eines Programmthreads der CWiid regelmäßig die Daten der Wiimote über Bluetooth abfragt und die folgende Methode:

```
void cwiid_callback(cwiid_wiimote_t *wiimote, int mesg_count,union cwiid_mesg mesg[], struct timespec *timestamp)
```

In dieser Methode werden von der JVM die gewünschten Klassen und Datentypen geholt, Objekte von ihnen erzeugt, diese mit den empfangenen Werten befüllt und die Methode von der JAVA Listener-Klasse aufgerufen.

```
result = (*env)->NewObject(env, wiiDataClass, cid,acc_x,acc_y,acc_z,buttons,rot_x,rot_y,rot_z,low_speed_x,
      low_speed_y,low_speed_z,jrumble,jled1,jled2,jled3,jled4,jni_ir_data,jtype,jbtID,nun_acc_x,nun_acc_y,
      nun_acc_z,nun_but,nun_stick_x,nun_stick_y );
if (result == NULL){printf("error creating WiimoteData...");}
jclass cls = (*env)->GetObjectClass(env, jni_callback_obj);
jmethodID mid = (*env)->GetMethodID(env, cls, "recievedWiimoteData", "(Lwiijavalibrary/comm/WiimoteData;)V");
if (mid == NULL) {printf("cwiid_callback: mid is null\n");}
else {(*env)->CallVoidMethod(env, jni_callback_obj, mid, result);}
```

Das Listing zeigt das Erzeugen der *WiimoteData* und den Aufruf der Methode *recievedWiimoteData(WiimoteData data)* aus dem nativen Thread. Falls etwas schief geht, so hat die JVM entsprechende Objekte oder Methoden nicht gefunden und es wird eine Fehlermeldung ausgegeben. Eine Behandlung von Fehlern auf dieser Ebene wäre sehr aufwendig und findet nicht statt. Während des praktischen Einsatzes wurden keine Fehler solcher Art beobachtet.

5.2.4 Wiimote Eingabe/Ausgabe der Kalibrierungsdaten

Da es mittels der shared library möglich ist, die CWiid Methoden zum Zugriff auf das eingebaute EEPROM der Wiimote zu benutzen, werden die Kalibrierungsdaten in der Wiimote gespeichert und beim Programmstart von ihr gelesen.

Ein Zugriff auf das EEPROM über die CWiid ist nur Zeilenweise, also in Blöcken von 16 Byte möglich. Damit beliebige Daten geschrieben werden können, wird das byte Array in Blöcke der Größe 16 zerlegt und in solchen in das EEPROM beginnend mit der Adresse 0x0030 geschrieben. Da die Initialdaten im Speicher zufällige Werte haben und prinzipiell jeder in das EEPROM schreiben kann, müssen korrupte Daten erkannt werden können. In Version 1 wurden hierzu 98 Bytes in die Wiimote geschrieben, wobei das vorletzte Byte eine 1 als Versionsnummer und das letzte Byte die Parität (XOR) darstellten. Dies bietet guten Schutz gegen korrumpierte Daten. Zufällige Daten würden nämlich mit einer Wahrscheinlichkeit von $\frac{1}{256}$ als korrekte Daten gelesen und vermutlich auf semantischer Ebene als bedeutungslos erkannt werden. Als die Kalibrierungsdaten im Zuge der Programmentwicklung komplexer wurden, ist das Verfahren noch etwas verbessert worden. In der 2. Version ist das Array 186 Byte lang, wobei an erster Stelle die Versionsnummer steht und im letzten Byte die Parität notiert ist. Wie die Datentypen in Byte gewandelt werden sieht man in Listing 5.5 und die Zerlegung der longs in ein Bytearray in Listing 5.6

Diese nun aktuell verwendete Art der Speicherung wurde hier entwickelt, damit ab Version 3 eine allgemeine, generelle Darstellung genutzt werden kann, falls zukünftige Entwicklungen dies erfordern sollten. Dabei könnte im ersten Byte die Version stehen, gefolgt von der Arraylänge und im letzten Byte die Parität. Diese Idee ist für den Fall entstanden, dass die Software weitergenutzt und weiterentwickelt wird.

Listing 5.5: Aufbau der *toByteArray* Methode der *CalibrationData*

```
public static byte[] toByteArray(CalibrationData data) {
    byte[] result = new byte[lenghtInBytes];
    byte[] array = null;
    for (int i = 1; i <= 23; i++) {
      if (i == 1) array = longToByteArray(Double.doubleToRawLongBits(data.x_acc_gradient));
      if (i == 2) array = longToByteArray(Double.doubleToRawLongBits(data.x_acc_offset));
      if (i == 3) array = longToByteArray(Double.doubleToRawLongBits(data.y_acc_gradient));
      if (i == 4) array = longToByteArray(Double.doubleToRawLongBits(data.y_acc_offset));
      if (i == 5) array = longToByteArray(Double.doubleToRawLongBits(data.z_acc_gradient));
      if (i == 6) array = longToByteArray(Double.doubleToRawLongBits(data.z_acc_offset));

      if (i == 7) array = longToByteArray(Double.doubleToRawLongBits(data.x_gyro_gradient));
      if (i == 8) array = longToByteArray(Double.doubleToRawLongBits(data.x_gyro_offset));
      if (i == 9) array = longToByteArray(Double.doubleToRawLongBits(data.y_gyro_gradient));
      if (i == 10) array = longToByteArray(Double.doubleToRawLongBits(data.y_gyro_offset));
      if (i == 11) array = longToByteArray(Double.doubleToRawLongBits(data.z_gyro_gradient));
      if (i == 12) array = longToByteArray(Double.doubleToRawLongBits(data.z_gyro_offset));

      if (i == 13) array = longToByteArray(Double.doubleToRawLongBits(data.x_nunchuk_acc_gradient));
      if (i == 14) array = longToByteArray(Double.doubleToRawLongBits(data.x_nunchuk_acc_offset));
      if (i == 15) array = longToByteArray(Double.doubleToRawLongBits(data.y_nunchuk_acc_gradient));
      if (i == 16) array = longToByteArray(Double.doubleToRawLongBits(data.y_nunchuk_acc_offset));
      if (i == 17) array = longToByteArray(Double.doubleToRawLongBits(data.z_nunchuk_acc_gradient));
      if (i == 18) array = longToByteArray(Double.doubleToRawLongBits(data.z_nunchuk_acc_offset));

      if (i == 19) array = longToByteArray(Double.doubleToRawLongBits(data.x_nunchuk_joystick_gradient));
      if (i == 20) array = longToByteArray(Double.doubleToRawLongBits(data.x_nunchuk_joystick_offset));
      if (i == 21) array = longToByteArray(Double.doubleToRawLongBits(data.y_nunchuk_joystick_gradient));
      if (i == 22) array = longToByteArray(Double.doubleToRawLongBits(data.y_nunchuk_joystick_offset));
      if (i == 23) array = longToByteArray(data.timeStamp);

      for (int j = 0; j < 8; j++) {       result[((i-1)*8)+j+1] = array[j];     } //start with the 2. element
    }
    result[0] = version;
    result[lenghtInBytes-1] = 0;
    for (int i = 0; i < result.length-1; i++) {
      result[lenghtInBytes-1]=((byte) (result[lenghtInBytes-1]^result[i]));}
    return result;
}
```

Listing 5.6: Aufbau der *longToByteArray* Methode

```
private static byte[] longToByteArray(long data) {
  byte[] result = new byte[] {
    (byte)((data >> 56) & 0xff),
    (byte)((data >> 48) & 0xff),
    (byte)((data >> 40) & 0xff),
    (byte)((data >> 32) & 0xff),
    (byte)((data >> 24) & 0xff),
    (byte)((data >> 16) & 0xff),
    (byte)((data >> 8 ) & 0xff),
    (byte)((data >> 0) & 0xff),
  };
  return result;
}
```

Experimentelle Evaluation von Steuerungskonzepten für Roboter mittels Wiimote

6

Im experimentellem Teil dieser Arbeit wurde evaluiert welche Steuerungskonzepte sich mit einer Wiimote und ihren Erweiterungen realisieren lassen und wie sie realisiert werden können. Für eine eindeutige Steuerung einer mobilen Plattform im kartesischen (2D-)Raum reicht es bereits, wenn man das Triple (x,y,θ) manipulieren kann, wohingegen man für eine eindeutige Pose eines Manipulators das Sextupel (x,y,z,θ, ϕ, ψ) bearbeiten muss. Hierbei sind (x,y,z) Koordinaten im (kartesischen) Raum und (θ, ϕ, ψ) Winkel des Manipulators bzw. der Plattform in Bezug auf die Koordinatenachsen. Im 3D-Raum ist die Zahl der Freiheitsgrade zwar deutlich größer, einfache Steuerungskonzepte können jedoch bereits mit den Knöpfen der Wiimote und deren Steuerkreuz realisiert werden. Die so erhaltenen Digitalen Signale können auf einfache Weise zur Steuerung genutzt werden (6.1).

Nach erfolgter Kalibrierung der Sensoren (6.3.1), bei der Kennlinien von Wiimote, MotionPlus und Nunchuk-Joystick bestimmt werden, können anspruchsvollere Steuerungskonzepte mittels der analogen Signale[1] realisiert werden, welche man vom Joystick der Nunchuk (6.2), sowie den Inertialsensoren der Wiimote, der Nunchuk und der MotionPlus erhält (6.3).

Um die Sicherheit der Roboter bei der praktischen Evaluation der Steuerungskonzepte zu gewährleisten, wurde zunächst ein Verfahren entwickelt, mit dem es möglich ist, zu bestimmen, ob die Wiimote sich in der Hand eines Benutzers oder in Ruhe -ggf. im freien Fall- befindet. Dies bildet die Experimentreihe im Abschnitt 6.3.4.

Inwiefern man die Pose einer Wiimote mittels doppelter Integration der Accelerometerdaten bestimmt werden und durch Sensorfusion mit Gyroskopdaten verbessert werden kann, wird in der Experimentreihe in Abschnitt 6.3.9 betrachtet.

In der in Abschnitt 6.4 vorgestellten experimentellen Versuchsreihe werden Steuerungsversuche mit zwei Wiimotes durchgeführt. Diese werden an einem menschlichen

[1] Strenggenommen sind auch diese Signale digital, da sie werte- und zeitdiskret sind

Arm montiert und steuern einen Manipulator mit sieben Gelenken durch Bestimmung der Vorwärtskinematik und einer Abbildung in den kartesischen Koordinatenraum des Kuka Manipulators und der dazu nötigen inversen Kinematik (Abschnitt 6.4).

6.1 Teleoperation mit den Knöpfen der Wiimote

Die Steuerung mittels der Knöpfe oder des Steuerkreuzes der Wiimote stellt das verarbeitungstechnisch am wenigsten aufwendige Konzept dar. Es ist nicht nötig aus den Signalen etwas zu berechnen. Vielmehr kann auf bestimmte Ereignisse reagiert werden oder Knopfdrücke können bestehende Aktionen modifizieren.

6.1.1 Steuerung der mobilen Plattformen

Für die Steuerung der mobilen Plattformen mittels Knöpfen und Steuerkreuz wurden drei Varianten von Steuerungskonzepten in Betracht gezogen:

Variante 1 - feste Geschwindigkeit

Diese Variante besteht darin, dass in der mobilen Plattform eine bestimmte Bewegungsgeschwindigkeit fest eingetragen wird und die Plattform sich in die Richtung bewegt, welche am Steuerkreuz gedrückt wird, solange die entsprechende Taste gehalten wird. Wird die Taste losgelassen oder fällt die Wiimote aus, so bleibt die Plattform stehen. Diese Art der Steuerung ist einfach in der Implementierung und für Benutzer leicht zu verstehen. Es ist nicht möglich auf diese Art Kurven zu fahren. Die Steuerung wirkt etwas “abgehackt“, weil die Plattform vorwärts oder rückwärts fahren kann und dann auf der Stelle wendet, um weiter zu fahren.

Variante 2 - inkrementelle/dekrementelle Geschwindigkeit

Bei diesem Steuerungskonzept kann die Geschwindigkeit der Plattform mittels wiederholtem Drücken eines Knopfes erhöht bzw. verringert werden. Wenn ein Benutzer den Knopf des Steuerkreuzes in eine Richtung, wie zum Beispiel ”vorne” ein Mal drückt, fährt die Plattform los. Ein wiederholtes Drücken des selben Knopfes erhöht die Geschwindigkeit, wohingegen das Drücken des Knopfes für die entgegengesetzte Richtung die Geschwindigkeit erniedrigt. Mehrmaliges Drücken bringt die Plattform zum Stillstand, bevor sie sich in die entgegengesetzte Richtung bewegt. Bei dieser Variante ist es möglich auch Kurven zu fahren. Allerdings würde bei einer naiven Implementierung im Falle des Ausfalles der Steuerung (Wiimote), die Plattform fahren, ohne abzubremsen, was nicht akzeptabel ist. Daher ist es unumgänglich eine

decreasing-Funktion einzubauen, welche die Geschwindigkeit der Plattform gegen Null konvergieren lässt, falls keine Knöpfe gedrückt werden. Eine Lineare *decreasing*-Funktion mit einem decrease von 20% der Geschwindigkeit pro 10s und einem Stillstand nach 5 Sekunden ohne jeden Knopfdruck, ist für Tests ausreichend. Alternativ (oder zusätzlich) kann die In-Hand-Erkennung aus Abschnitt 6.3.4 (Seite 96) verwendet werden, um die Plattform zum Stillstand zu bringen. Dies wäre natürlich ein Vorgriff auf ein komplexeres Steuerungskonzept.

Variante 3 - Integration der vorherigen Ansätze

Es liegt nahe das Steuerkreuz für die Richtungsangaben zu nehmen und zwei andere Tasten, wie z.B. die "+" und "-" Tasten für das Erhöhen bzw. Erniedrigen der Geschwindigkeit. Wie schon vorher muss im Falle des Loslassens der Tasten oder Ausfalles der Steuerung die Plattform entweder sofort stehen bleiben, oder die Geschwindigkeit zügig gegen Null konvergieren lassen.

6.1.2 Steuerung des Manipulators

Kartesischer Raum

Im kartesischen Raum lässt sich ein Manipulator steuern, indem man das Steuerkreuz der Wiimote auf zwei Achsen des globalen Koordinatensystems legt und zwei weitere Knöpfe, wie z.B. "+" und "-" auf die dritte Achse. So kann das Steuerkreuz für Steuerung in der Ebene (x,y-Ebene) und die beiden anderen Knöpfe für die Höhe (z-Achse) genutzt werden. Auf diese Art lassen sich Bewegungen in allen drei Achsen des (globalen) Koordinatensystems beschreiben und man kann den Manipulator in die positive oder negative Richtung bewegen. Mit zwei weiteren Knöpfen ("1" und "2") kann man darüberhinaus noch die Schrittweite variabel machen. So kann die Position des Endeffektors angegeben werden.

Die Orientierung des Endeffektors hat drei weitere Freiheitsgrade[2]. Für eine sinnvolle Steuerung des Manipulators müssen diese auch veränderbar sein. Man kann es auch erreichen, in dem man die 12 Knöpfe der Wiimote auf die 6 Freiheitsgrade legt und jeweils einen zum Inkrementieren und einen zum Dekrementieren benutzt. Auf diese Art ist die Geschwindigkeit, mit der sich der Manipulator bewegt, nicht mehr steuerbar, weil schlichtweg keine Knöpfe mehr da wären. Es ist daher sinnvoller einen anderen Weg zu gehen:

Es werden hier zwei Steuerungszustände eingeführt. Mittels einer weiteren Taste ("Home") kann zwischen den beiden Zuständen alterniert werden. In dem einen werden die Position, wie oben beschrieben, kartesisch gesteuert, im anderen die Orientierung mittels der drei Winkel ϕ, ψ und θ, welche sich entweder fest am globalen

[2]Englisch:Degree Of Freedom (DOF)

Koordinatensystem orientieren, oder an das Koordinatensystem des Manipulators gebunden werden müssen.

Für eine intuitive Steuerung bietet es sich an, zuerst die Orientierung des Manipulators festzulegen und fest zu belassen und dann die Steuerung umzuschalten, um die Position zu verändern. Dieses Vorgehen ist deshalb sinnvoll, weil die Änderung der Orientierung, auch wenn sie noch so klein ist, teils große Änderungen an den Gelenkstellungen des Manipulators, insbesondere in der Nähe von singulären Stellungen, erfordert. Es kann passieren, dass die Zwischengelenke, also diejenigen, die zwischen dem ersten und dem letzten Gelenk sind (wobei das erste an der Basis befestigt ist und das letzte den Endeffektor trägt) sehr große Änderungen ausführen müssen, damit die geforderten Orientierungen erreicht werden können. Ein solches Verhalten ist aber gerade in der Nähe des Zieles oft unerwünscht.

Es ist in den meisten Fällen also sinnvoller zunächst die Orientierung, dann die Position zu steuern.

Gelenkwinkelraum

Im Gelenkwinkelraum kann ein Manipulator gesteuert werden, indem man jedem Gelenk zwei Knöpfe zuordnet, welche dieses steuern können. Hierbei bedeutet ein Knopf Drehung in die eine Richtung (etwa CW[3]). Sein Antagonist dreht dann das Gelenk in die andere Richtung (z.B. CCW[3]). Für den Manipulator PA-10-6C von Mitsubishi (Abschnitt 3.2.3), welcher sechs Gelenke hat, ergeben sich dann zwölf Knöpfe, die nötig sind, um ihn im Gelenkwinkelraum zu steuern. Zwei weitere Knöpfe sind erforderlich, falls man die Geschwindigkeit ebenfalls veränderbar machen möchte. Die Wiimote bietet nicht genügend Knöpfe, um das direkt zu erreichen. Es ist also hier zwingend nötig manche Knöpfe mehrfach zu belegen und entweder mittels umschaltbarer Zustände die Knöpfe mehrfach zu belegen, oder man definiert bestimmte Knöpfe als "Sondertasten", welche die Knopfeigenschaften so ändern, wie die *Shift* oder *Strg* Tasten es bei herkömmlichen Tastaturen auch tun. In dieser Arbeit wird der Ansatz von Zuständen, in denen die Wiimote sich befindet, bevorzugt. Diese können auch über die LEDs angezeigt werden. So können die Gelenke eins bis drei, genau, wie oben bei den kartesischen Achsen beschrieben, angesteuert werden. Mittels der Zustandsumschaltung, werden die Knöpfe auf die Gelenke vier bis sechs gelegt, was, wie oben beschrieben, der Steuerung der Orientierung entsprach. Dadurch kann auf einfache Weise eine Steuerung der Gelenkwinkel realisiert werden.

[3]CW: clock-wise Uhrzeigersinn (UZS) CCW: counter clock-wise Gegen den Uhrzeigersinn (GZS)

6.2 Kalibrierung und Teleoperation mittels Joystick der Nunchuk

Da der Analogjoystick der Nunchuk viele Werte zwischen Ruheposition und Maximum liefern kann, wird er hier als *analog* bezeichnet[4]. Mit diesen analogen Signalen ist eine Steuerung von Geschwindigkeiten und Intensitäten gut möglich. Diese Art der Steuerung von Robotern (bzw. Maschinen im allgemeinen), ist schon seit langer Zeit bekannt und daher für Benutzer intuitiv und leicht zu erlernen.

6.2.1 Kalibrierung des Wertebereiches des Nunchuk-Joysticks

Da die Pose von mobilen Plattformen im 2D-Raum mit zwei Freiheitsgraden (Translation und Rotation) beschrieben werden kann, eignet sich ein Joystick für die Steuerung sehr gut. Man kann die Intensität mit der der Steuerknüppel bewegt wird, nach einer Normierung, direkt auf die Geschwindigkeit der Plattform abbilden und die Vor-/Zurück-Richtung für das Fahren vorwärts/rückwärts benutzen, sowie die Rechts/Links Bewegung in Drehungen umsetzen. Es ist nützlich, wenn die Werte, welche vom Joystick geliefert werden und von der Plattform akzeptiert werden können, auf die selben Bereiche, hier von -1.0 bis +1.0, normiert werden. Das kann für die Plattform und den Joystick mittels der Formel:

$$Wert_{Aktuell} = \frac{Messwert}{|Maximalwert_{Richtung}|} \tag{6.1}$$

errechnet werden.[5] Hierbei ist $Wert_{Aktuell}$ der normierte Messwert, welcher nach Anwendung der Normierung hier zwischen -1.0 und +1.0 liegt oder falls keine Gleitkommazahlen gewünscht sind, auf den Bereich von (-100:100) normiert wird, was an die Prozentangabe angelehnt ist. Der *Messwert* ist derjenige Wert, welcher vom Joystick geliefert wird. $Maximalwert_{Richtung}$ gibt den maximalen Wert des Joysticks an, welcher während der Kalibrierungsphase für diese Richtung ermittelt wurde. Bei den Joysticks der Nunchuk liegen die Wertebereiche zwischen -127 und 128, bzw. 0 und 255 falls sie vorzeichenlos interpretiert werden. Leider kann nicht jeder Joystick, produktionsbedingt, die maximal möglichen Wertebereiche voll ausschöpfen. Daher erfolgt die Normierung.

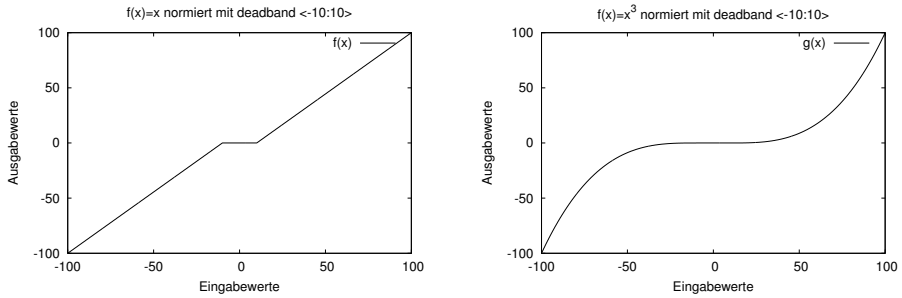
6.2.2 Steuerung der mobilen Plattformen

Nach einer Kalibrierung, bei der der Joystick in Nullposition, sowie in die vier Maximalauslenkungen zu bringen ist, bleiben noch einige wenige Parameter, die variabel sind, mit denen die Steuerung verbessert werden kann:

[4] Die Werte werden zu diskreten Zeitpunkten (10ms) abgetastet und sind quantisiert. Durch den diskreten Definitions- und Wertebereich sind die Signale im formalen Sinne *digital*.

[5] Die Maximalwerte des Joysticks können abhängig von der Richtung sein, in die er bewegt wird und damit kann es bis zu vier Maximalwerte geben, aber das Prinzip bleibt gleich.

- Man kann ein *Totband* einbauen und seine Größe variieren. So fährt die Plattform nicht sofort "unkontrolliert" los, sobald mal der Joystick nicht in der Nullposition ist. Bei genormten Werten hat sich ein Totband von 0.01-0.2 bewährt.
- Die Abbildung der Joystickwerte auf die Geschwindigkeit kann nicht nur linear, sondern z.B. kubisch geschehen. Diese Idee, welche als Vorarbeit für die Diplomarbeit von Benjamin Adler [Adl08] entstand, wird hier aufgegriffen und weiter fortgesetzt. Damit ist eine feine Steuerung gerade bei langsamen Geschwindigkeiten möglich. Bei höheren Geschwindigkeiten, wo es nicht so sehr auf die exakte Geschwindigkeit ankommt, reagiert die Plattform dann intensiver auf kleine Änderungen. Das ist auch intuitiv gut nachvollziehbar, weil es dem Fahrzeugverhalten beim Autofahren ähnlich ist, wo bei höheren Geschwindigkeiten, etwa auf der Autobahn, kleine Änderungen in der Steuerung entsprechende (subjektiv) höhere Wirkung zeigen. Der Bereich in dem eine sehr feine Steuerung möglich ist, ist besonders für genaue Positionierung von Manipulatoren wichtig.



(a) linear (b) kubisch

Abbildung 6.1: Mögliche Steuerfunktionen und deren Verlauf

Abbildung 6.1 zeigt zwei mögliche Steuerfunktionen für die Umsetzung der Eingabewerte des Joysticks, hier normiert im Bereich (-100:100), in den Steuerbereich des Roboters, welcher hier auch mit (-100:100) normiert ist. Beide Funktionen besitzen ein Totband (-10:10), damit kein Steuersignal erzeugt wird, falls der Joystick nicht in einer optimalen Nullposition ist. Die Funktion von Abb.6.1(a) ist eine lineare Funktion, welche beschrieben wird durch:

$$f(x) = \begin{cases} \frac{x+10}{0.9}, & \text{wenn } x < -10 \\ \frac{x-10}{0.9}, & \text{wenn } x > 10 \\ 0, & \text{sonst} \end{cases}$$

Die Funktion von Abb.6.1(b) ist kubisch mit der Gleichung:

$$g(x) = \begin{cases} \frac{(x+10)^3}{(100-10)^3} \cdot 100, & \text{wenn } x < -10 \\ \frac{(x-10)^3}{(100-10)^3} \cdot 100, & \text{wenn } x > 10 \\ 0, & \text{sonst} \end{cases}$$

Beide Funktionen brauchen eine Fallunterscheidung für das Totband und wären ohne die Normierung des Wertebereiches etwas einfacher. So haben sie aber den Vorteil, dass die Eingabewerte direkt in Geschwindigkeitswerte umgesetzt werden können.

6.2.3 Steuerung eines Manipulators

Auch bei der Steuerung eines Manipulators bietet es sich an, die Steuerwerte nach Formel 6.1 zu normieren. Innerhalb der hardwareseitig vorgegebenen Maximalgeschwindigkeiten kann dann der Manipulator bewegt werden. Genau, wie bei der Steuerung einer mobilen Plattform, kann es von Vorteil sein, wenn man die Werte kubisch approximiert und ein Totband einbaut. Da der Joystick zwei DOF, der Manipulator im kartesischem Raum aber sechs DOF besitzt, ist es nötig die Joystick-Richtungen mehrfach zu belegen und mittels Knöpfen umschaltbar zu machen.

Ein mögliches Steuerkonzept ist das Folgende: Die Richtungen des Joysticks entsprechen jeweils einer Translation entlang einer Achse (in Weltkoordinaten), bzw. einer Drehung um diese Achse. Bewegungen mit höherer Intensität entsprechen höheren Geschwindigkeiten und für sanftere Bewegungen entsprechend analog. Falls eine bestimmte Taste der beiden Nunchuk-Tasten (z.B. die obere) beim operieren mit dem Joystick gedrückt und gehalten wird, so wird der Manipulator an der X-Achse gesteuert. Durch den Druck der anderen Taste wird die Y-Achse ausgewählt und gesteuert. Beim Drücken der beiden Tasten kann optional auf die Z-Achse gewechselt werden. Und die Nutzung des Joysticks ohne Knopf kann für eine Simulation benutzt werden, oder aber ohne Funktion sein. Dies ist als eine Sicherheitsmaßnahme zu werten, falls die Nunchuk mal herunterfällt und in einer aktiven[6] Position verbleibt. Eine Steuerung mittels beider Knöpfe in der dritten Ebene ist redundant, erleichtert aber die Operation des Manipulators in Situationen, wo in genau dieser Ebene gearbeitet wird. Typischerweise finden Teleoperationen von Manipulatoren nicht in allen drei Raumebenen gleichmäßig statt, sondern es wird während einer Operation oft eine als weitgehend fest angenommen und die anderen beiden werden gesteuert.[7]

Für die Steuerung eines PA10 Armes im **Gelenkwinkelraum** ist es unerlässlich, dass die beiden Freiheitsgrade des Joysticks mehrfach belegt werden. Hier kann mittels der Knöpfe zwischen den Gelenken umgeschaltet werden. Im ersten Zustand (Knopf 1 gedrückt) steuert der Joystick die Gelenke 1 und 2. Zustand zwei (Knopf 2 gedrückt) ist für die Steuerung der Gelenke 3 und 4 zuständig, während der dritte

[6] also wertegebenden Position mit Werten ungleich 0

[7] Beispiele: Industrieller Roboterarm, Gabelstapler, Schachmanipulator

und letzte Zustand (beide Knöpfe gedrückt) die Gelenke 5 und 6 steuert. Die Intensität kann hier, wie schon vorher, auf eine (nach Formel 6.1) normierte Geschwindigkeit abgebildet werden. Dabei würde sich das aktive Gelenk bei Nullstellung des Joysticks nicht bewegen und bei Auslenkung des selbigen in einem entsprechenden Verhältnis zur Auslenkung, z.B. linear oder kubisch mit einem Totband in die vorgesehene Richtung bewegen.

Diese Art der Steuerung ist gut geeignet, um ein Ziel inkrementell zu erreichen. Sie eignet sich auch zum Erlernen von bestimmten Bewegungsabläufen, wenn diese auf eine bestimmte Art durchzuführen sind. Ein Serviceroboter kann so lernen, wie er nach einer bestimmten Tasse im Schrank greifen kann, ohne die anderen Tassen umzustoßen.

Ein Steuerungskonzept hat sich experimentell als schwer praktikabel herausgestellt:

Eine direkte Abbildung des normierten Wertebereiches des Joysticks in einen normierten kartesischen Arbeitsraum bzw. Gelenkwinkelarbeitsraum des Manipulators. Die Implementation ist recht einfach, aber die Steuerung nur komplex zu handhaben. Da besonders in den Extrempositionen des Joysticks die Reaktionen des Manipulators sehr intensiv sein können, wurden Versuche in diese Richtung nach wenigen Experimenten aus Gründen der Vorsicht eingestellt.

6.3 Experimente mit den Inertialsensoren der Wiimote

Während die bisher vorgestellten Steuerungskonzepte im Wesentlichen Joystickeingaben und Knopfdrücke abbilden, ermöglichen die Inertialsensoren der Wiimote deutlich anspruchsvollere Steuerungskonzepte. Aus den Daten der Inertialsensoren kann der Winkel errechnet werden, welchen die Wiimote im Verhältnis zur Gravitation einnimmt. Es kann dabei statisch das aktuelle Verhältnis der beiden Vektoren betrachtet werden, in die sich der Gravitationsvektor aufteilt, wenn seine Kraft an der Wiimote angreift. Zusätzlich kann die Information mit den aus dem Gyroskop gewonnenen Daten fusioniert werden. Dieses Verfahren wird hier als **neigungsbasiert** bezeichnet und nachfolgend beschrieben. Mit diesem Verfahren ist eine Bestimmung der Pose der Wiimote nicht möglich, dafür aber die Bestimmung der Orientierung. Diese Information ist allerdings für viele Zwecke vollkommen ausreichend. Bevor auf das neigungsbasierte Verfahren eingegangen werden kann, muss zunächst die Kalibrierung der Inertialsensoren erfolgen. Wobei die Gyroskope zuerst betrachtet werden, weil sie es für spätere Steuerungen ermöglichen zu erkennen, ob die Wiimote in der Hand eines Benutzers ist oder gerade nicht benutzt wird. Für die erfolgreiche Kalibrierung der Gyroskope ist eine Kalibrierung der Kameradaten sinnvoll. Dies bildet den Anfang der Kalibrierungsexperimente.

Eine andere Vorgehensweise bildet die tatsächliche direkte Positions- und Orientierungsbestimmung der Wiimote. Dieses Verfahren ist in einem eigenen Abschnitt

(6.3.9 auf Seite 109) beschrieben, weil sein Implementationsversuch sich als sehr schwer durchführbar herausgestellt hat. Hierbei müssen die Daten der Accelerometer und Gyroskope ständig, möglichst genau und lückenlos, aufsummiert und die Accelerometerdaten doppelt integriert werden. Ferner muss man Sensordrift und andere Fehlerarten berücksichtigen. Mittels Multisensorfusion können die Fehler verringert werden, jedoch muss man sich vor Augen halten, dass auch kleine Fehler durch zweifache Integration ein starkes Gewicht bekommen. Im letzten Abschnitt wird ein Manipulator gesteuert, für dessen Steuerung Daten von zwei Wiimotes benutzt werden, die an einem menschlichen Arm befestigt sind.

6.3.1 Vorbemerkungen zu den Kalibrierungsexperimenten der Inertialsensoren

Damit man gemessene Sensordaten korrekt interpretieren kann, ist es unerlässlich die Eigenschaften der vorliegenden Sensoren zu kennen bzw. zu bestimmen (siehe Abschnitt 2.2 - Inertialsensoren). Das Verhältnis gemessener Sensorwerte zu den tatsächlich vorliegenden physikalischen Größen wird hier in *Sensorkennlinien* dargestellt. Dieser Abschnitt beschäftigt sich mit der Bestimmung der Kennlinien der Sensoren der Wiimote.

Obwohl temperaturabhängige Messfehler sowohl im Bereich der Accelerometer, als auch Gyroskope einen spürbaren Beitrag leisten können, wie man Ergebnissen von Forschungsprojekten [CF08, Kapitel 15] entnehmen kann, werden solche Messabweichungen in dieser Arbeit vollkommen außer Acht gelassen. Dies hat 2 Gründe

1. Für eine Bestimmung solcher Fehler müssen die Messungen bei unterschiedlicher Umgebungstemperatur wiederholt werden. Der Zeitaufwand steht zur Nützlichkeit dadurch gewonnener Korrekturdaten in keinem angemessenen Verhältnis.
2. Damit eine solche Korrektur erfolgreich angewendet werden kann, müssten Benutzer vor der Benutzung der Sensoren die Raumtemperatur der Software bekannt machen.

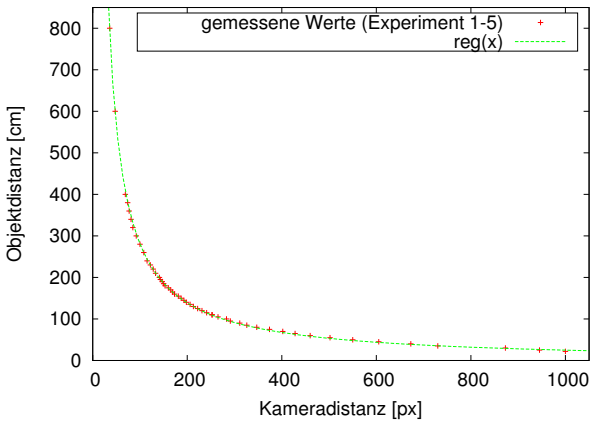
Es ist schlichtweg unrealistisch zu erwarten, dass Nutzer der Software zunächst ein Thermometer bemühen und danach die Software benutzen. Stattdessen wird hier der Fokus mehr auf die anderen Fehlerquellen gelegt, welche größere (systematische, wie stochastische) Beiträge zum Gesamtfehler haben.

Dazu wurden Vergleichsexperimente mit 3 Geräten gemacht. Es wurde jeweils eine Wiimote mit einer eingesteckten MotionPlus benutzt. Erwartungsgemäß bleiben die Daten der MotionPlus unverändert. Sie sind unabhängig von der durchleitenden Wiimote. Falls die Messergebnisse der drei Geräte sich grundlegend unterscheiden, so wird in dem jeweiligen Abschnitt gesondert drauf hingewiesen. Anderenfalls darf man davon ausgehen, dass die Geräte sich bis auf individuelle Kennlinien gleichartig verhalten.

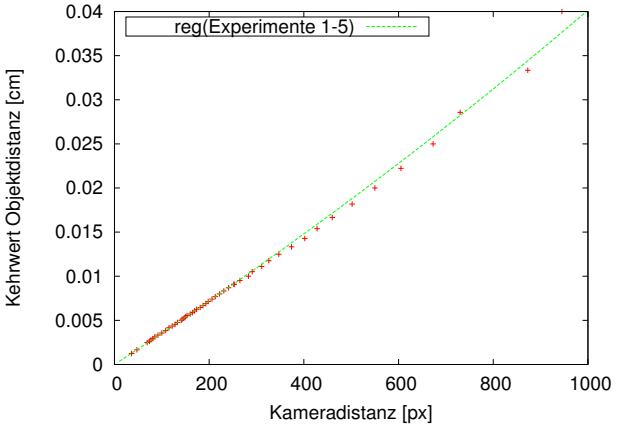
6.3.2 Kalibrierungsexperimente mit der Infrarotkamera

Damit die Infrarotkamera zum Bestimmen von Entfernungen eingesetzt werden kann, ist es nötig das Verhältnis zwischen Objektdistanz d_o und Pixelgröße des Objektbildes der Kamera d_c zu bestimmen. Dazu wurde eine Sensorbar von der Wiimotekamera beobachtet und sowohl die Entfernung zwischen Sensorbar und Wiimote d_o (in cm) als auch die Distanz der Pixel, welche den Bildern der Sensorbar LEDs entsprechen d_c (in Pixeln), gemessen. In Anhang A.1.2 sind die Messwerte in Tabellen aufgeführt. Abbildung 6.2 zeigt das Verhältnis als Graph geplottet. Damit der lineare Verlauf sichtbar wird, sind als Ordinate die Kehrwerte der Messwerte der Pixeldistanz ($\frac{1}{d_c}$) eingetragen. Für die Experimente wurden d_o zwischen 22cm und 9m vermessen. Die anfänglichen Messpunkte sind 5cm von einander entfernt.

Ab einer Entfernung von 2m wurden alle 10cm und später alle 20cm Messpunkte bestimmt. Über 4m hinaus wurden nur noch 2 Messpunkte bei 6m und 8m eingetragen. Die maximal sichtbare Entfernung für die Sensorbar betrug 9 Meter und 40 Zentimeter. Auf diese Entfernung entsprachen die LEDs der Sensorbar bei guten Bedingungen einem Pixel im Kamerabild. Eine Regressionsgerade durch die Messpunkte zeigt, dass die Kamera, wie erwartet, die Entfernungen sehr linear abbildet. Damit eignet sie sich gut, um im Rahmen ihrer Genauigkeit Entfernungen zur Sensorbar als Infrarotquelle zu bestimmen.



(a) Tatsächlich gemessene Werte



(b) Linearität bei reziproker Darstellung der Ordinate

Abbildung 6.2: Verhältnis der Messungen von Objektdistanz (Sensorbar-Kamera) zu angezeigter Pixeldistanz im Kamerabild

Da die Objektdistanz (d_o) wächst, während die Kameradistanz (d_c) kleiner wird und umgekehrt, kann ein reziproker linearer Zusammenhang angenommen werden, welcher sich mit einer Hyperbelfunktion gut beschreiben lässt. Die allgemeine Geradengleichung $g = a \cdot x + b$ (5.1) kann umgewandelt werden zur Hyperbelgleichung:

$hyp(x) = \frac{a}{x} + b$

Mittels der **Methode der kleinsten Quadrate** können die Parameter **a** und **b** zu a=28455.9 mit ±98.12 (≈0.3448%) und b=-3.54514 mit ±0.7887(≈22.25%) bestimmt werden. Die Werte in Abbildung 6.2(a) können approximiert werden mit der Hyperbelgleichung:

$$d_o[cm] = \frac{28455.9}{d_c[px]} - 3.54514 \tag{6.2}$$

Hierbei ist d_c in Pixeln und d_o in Zentimetern angegeben.

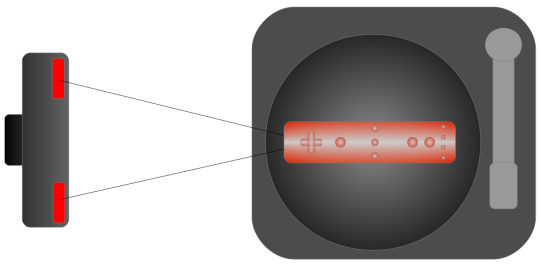
Da Abb. 6.2(b) lediglich den linearen Verlauf der reziproken Funktion zeigen soll, ist sie $\frac{1}{f(x)}$ und somit parametrisiert: $h(x) = \frac{1}{f(x)} = \frac{1}{d_o[cm]} = \frac{1}{\frac{28455.9}{d_c[px]} - 3.54514}$

6.3.3 Kalibrierung der Gyroskope

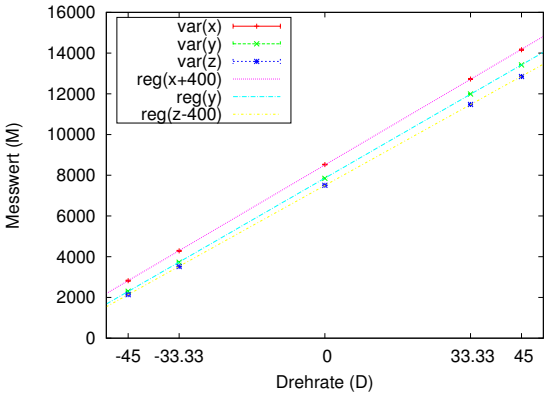
Für eine erfolgreiche Kalibrierung ist es notwendig, Geräte zu haben, welche feste Drehraten liefern, die sich gut von den Gyroskopen der Wiimote messen lassen.

Die Gyroskope der Wiimote wurden mittels Plattenspieler kalibriert. Diese drehen üblicherweise mit einer Frequenz von $33\frac{1}{3}$ bzw. 45 Umdrehungen pro Minute (UpM bzw. RPM). Für eine möglichst genaue Kalibrierung der empfindlichen Gyroskope ist es sinnvoll, die Drehgeschwindigkeit der Plattenspieler während der Kalibrierung mit Hilfe einer Lichtschranke zu validieren.

Hierzu wurden die Daten der Kamera ausgewertet, wobei eine Sensorbar als Lichtschranke diente. Der Aufbau ist in Abb.6.3(a) skizziert.



(a) schematischer Versuchsaufbau



(b) gemessene Umdrehungsraten

Abbildung 6.3: Kalibrierung der Gyroskope

Der für die Kalibrierung der Gyroskope verwendete Plattenspieler drehte recht präzise mit den erwarteten Drehraten von $33\frac{1}{3}$ im langsamen Modus, sowie 45 Umdrehungen pro Minute im schnellen Modus.

Die Abbildung 6.3(b) zeigt die mittels allgemeiner Geradengleichung (5.1) und der **Methode der kleinsten Quadrate** errechneten Regressionsgeraden für die Drehratensensoren. Der Einfachheit wegen ist die Abbildung der Drehrate (D) $\rightarrow$ Messwert (M) zu sehen und es wurde zu den Messwerten M_x der Wert 400 dazuaddiert, während bei M_z der Wert 400 abgezogen wurde, was die Lesbarkeit des Graphen erhöhen soll.

Für eine Berechnung der Drehraten aus den Messwerten müssen die Umkehrfunktionen ($f(x)^{-1}$) berechnet werden, welche sich bei Geraden in einem Schritt leicht bilden lassen zu:

$$Mg_x = 126.24 \cdot D_x + 8105 \overset{f(x)^{-1}}{\Rightarrow} D_x = \frac{Mg_x - 8105}{126.24} \tag{6.3}$$

$$Mg_y = 123.629 \cdot D_y + 7858.13 \overset{f(x)^{-1}}{\Rightarrow} D_y = \frac{Mg_y - 7858.13}{123.629} \tag{6.4}$$

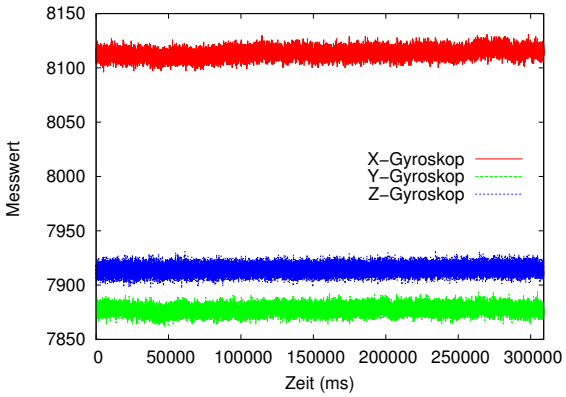
$$Mg_z = 119.149 \cdot D_z + 7898.3 \overset{f(x)^{-1}}{\Rightarrow} D_z = \frac{Mg_z - 7989.3}{119.149} \tag{6.5}$$

Wobei $Mg_{[x|y|z]}$ die Messwerte (dimensionslos) der Gyroskope darstellen und $D_{[x|y|z]}$ die Drehraten in Umdrehungen pro Sekunde ($U \cdot s^{-1}$).

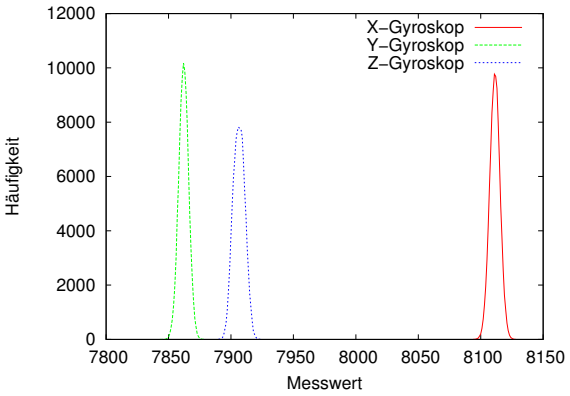
6.3.4 Erkennung von Ruhelage vs. Bewegung der Wiimote

Bei den Experimenten in diesem Abschnitt ging es darum zu erkennen, ob eine Wiimote sich in der Hand eines Benutzers befindet oder in Ruhe ist. Die empfindlichen Gyroskope der Wiimote können den natürlichen Tremor der menschlichen Hand messen. Somit wird es möglich eine Steuerung sofort abzuschalten, sobald die Wiimote die Hand des Benutzers verlässt und sich im freien Fall befindet oder auf dem Tisch bzw. Boden liegt. In Abb.6.4 sieht man den deutlichen Unterschied zwischen den Messwerten einer ruhig liegenden Wiimote und einer, die in Benutzerhand gehalten wird mit der Aufforderung sie möglichst ruhig zu halten. Die Streuung ist deutlich breiter bei den Daten, die von der Wiimote in der Hand gesammelt wurden.

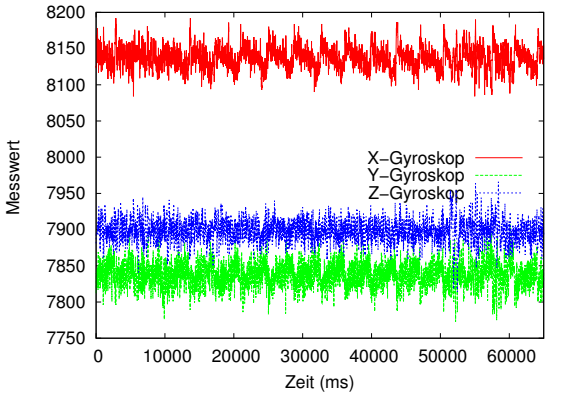
Für die Bestimmung der Streuung in Ruhe wird die Wiimote für mehrere Sekunden liegen gelassen und die Differenz des größten und des kleinsten Messwertes bestimmt. Es ergeben sich dabei für die Schwelle (**threshold**) oft Werte um 20 herum. Im



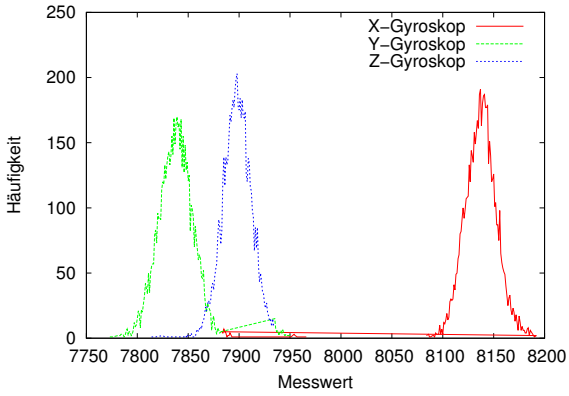
(a) Gyroskopwerte (auf Tisch liegend)



(b) Streuung (auf Tisch liegend)



(c) Gyroskopwerte (in der Hand)



(d) Streuung (in der Hand)

Abbildung 6.4: Messwerte der Gyroskope einer Wiimote in Ruhe (34859 Messwerte über 5 Min. - Abszisse: Zeit [ms] Ordinate: Gemessene Werte) und deren Häufigkeit/Streuung im Vergleich zur Messung in der Hand (Dauer 1 min)

Großteil der Experimente wurden Werte von 19.0 für die x-Achse, sowie 23.0 für die y- und z-Achsen bestimmt.

Data : Gyroskop-Messdaten mx, my, mz
Result : Eine Maßzahl ***out*** im Bereich [0..12], die angibt wie sehr sich die Wiimote in Ruhe befindet;
for $i \leftarrow 0, \leq 3$ **do**
  $factor \leftarrow 1.0$;
  switch i **do**
    case *2*
      $factor \leftarrow 1.01$
    case *3*
      $factor \leftarrow 1.16$
    end
  endsw
  if $mx \leq threshold_x * factor$ **then** out $\leftarrow out + 1$;
  if $my \leq threshold_y * factor$ **then** out $\leftarrow out + 1$;
  if $mz \leq threshold_z * factor$ **then** out $\leftarrow out + 1$;
  if $(mx + my + mz) \leq threshold * factor$ **then** out $\leftarrow out + 1$;
end
return out;

Algorithmus 1 : Entscheidung, ob die Wiimote in Ruhe ist (auf dem Tisch liegt) oder in der Hand von Benutzern

Durch Algorithmus 1 wird eine Maßzahl bestimmt, die gebildet wird aus den Gyroskopwerten für X,Y,Z und der Summe aus diesen, und angibt, wie viele davon innerhalb einer Schranke liegen. Diese Schranke wird in drei Schritten multiplikativ vergrößert, beginnend mit 1.0 über 1.01 zu 1.16. Geeignete Werte für den Faktor der Schranke wurden mittels Statistik suggeriert und experimentell bestimmt. Die Maßzahl aus Algorithmus 1, liegt zwischen 0 und 12 und gibt an, wie unbewegt die Wiimote auf einem Tisch liegt. Dabei bedeutet die 12, dass die Wiimote ganz sicher auf einem Tisch in Ruhe liegt und eine 0 notiert das Gegenteil.

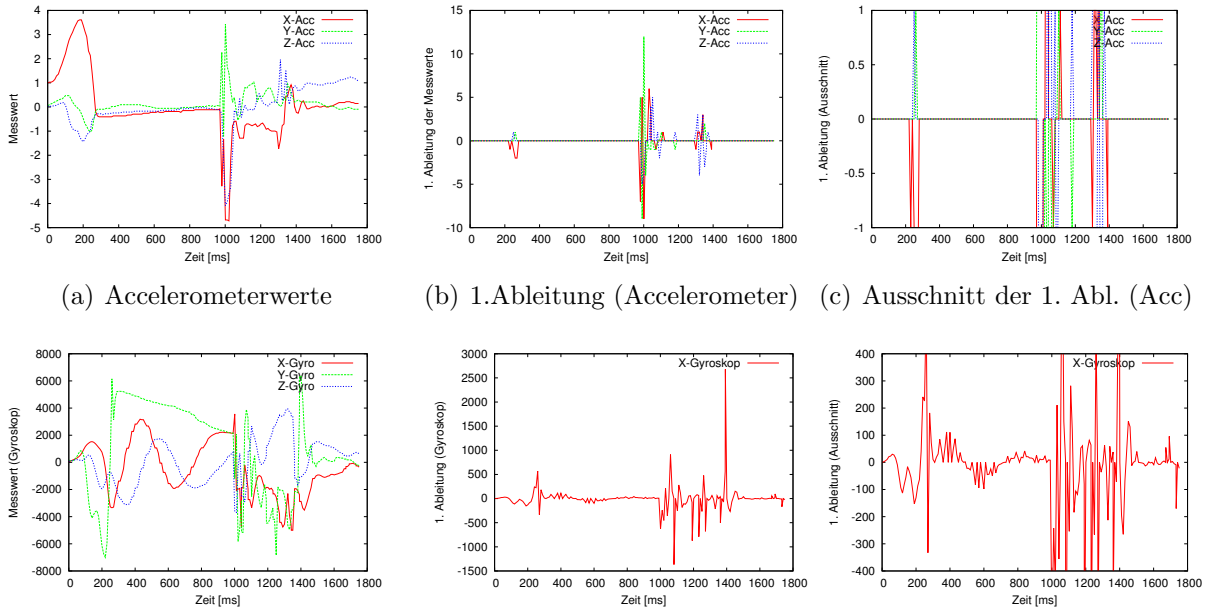
In Abbildung 6.5 sieht man die Messdaten von Accelerometern und Gyroskopen und deren Ableitungen von einer Wiimote, welche in die Luft geworfen wird. Bei den auf den Bereich von ±1g normierten Accelerometerdaten (6.5) sieht man im Bereich von ca.300ms bis ca.900ms den freien Fall mit dem bloßen Auge. Für eine Algorithmische Bestimmung wird hier ein Schwellwert von 0.1 für die Summe angenommen, somit ergibt sich für eine Wiimote im Freien Fall:

$$FreierFall = (acc_x + acc_y + acc_z) \leq 0.1 \tag{6.6}$$

Mittels der Ableitung über Differentialquotienten kann das Ergebnis, wie in Abb. 6.5(b) und 6.5(c) sichtbar, sogar deutlich verbessert werden.

Die Gyroskopwerte können hingegen nicht direkt genutzt werden, sondern müssen abgeleitet werden, wie die Abbildungen 6.5(e) und 6.5(f) zeigen. Bei einer Ableitung der Gyroskopdaten bringt allerdings ein Schwellwert von 50 sehr gute Ergebnisse und bei einem Schwellwert von 100 wird der freie Fall noch präziser erkannt. Aus Zeitgründen wurde in dieser Arbeit auf eine genaue Bestimmung der optimalen Werte verzichtet. Für die nachfolgenden Experimente wird mit einem Schwellwert von 0.1 für die Accelerometerdaten (genormt) und mit einem Schwellwert von 50 für die Gyroskopdaten gearbeitet.

Falls die Summe der Accelerometermessdaten kleiner 0.1 ist oder die Summe der Ableitungen der Gyroskopdaten um 50 oder weniger von der Null abweichen, gilt die Wiimote als sich im freien Fall befindend und die laufende Teleoperation, das Programm bzw. Experiment ist abzubrechen.



(a) Accelerometerwerte (b) 1.Ableitung (Accelerometer) (c) Ausschnitt der 1. Abl. (Acc)

(d) Gyroskopwerte (e) 1. Ableitung (Gyroskopwerte) (f) Ausschnitt 1.Abl. Gyro

Abbildung 6.5: Messwerte von einer Wiimote im freien Fall, wobei die Accelerometer in $\vec{g}$ umgerechnet sind

6.3.5 Kalibrierung der Accelerometer

Da die Wiimote (siehe 3.1) und die Nunchuk-Erweiterung (siehe 3.1.3) mit ihren jeweiligen Beschleunigungssensoren inertiale Messeinheiten mit körperfesten Sensoren (*strabdown imu* siehe Abschnitt 2.2.3) bilden, ist für eine Bestimmung der Geschwindigkeit (nach Gleichung 2.41) bzw. der Position (nach Gleichung 2.42), wie sie in

Abschnitt 2.2.4 beschrieben werden, neben der Kalibrierung der Gyroskope auch eine Kalibrierung der Accelerometer notwendig.

Auch für die Kalibrierung der Accelerometer wird eine bekannte und feste Größe benötigt, für die Messwerte bestimmt werden können, damit eine Extrapolation der Kennlinie möglich wird. Da die Accelerometer neben Kräften, welche zu einer aktiven Beschleunigung führen, auch die passive Beschleunigung in Form der Erdanziehungskraft messen können, wird diese für die Kalibrierung benutzt.

Dazu wird die zu kalibrierende Wiimote in eigens dafür angefertigte Haltevorrichtungen eingelegt, wie in Abbildung 6.6 (und A.4 auf Seite XII) sichtbar. Die Daten der Wiimote werden über einen Zeitraum von mehreren Minuten gesammelt, während die Wiimote zunächst rechtwinklig, dann parallel zur Gravitation, sowie in den Winkeln von 30°, 45° und 60° eingelegt ist. Auf diese Weise wird die konstante Gravitationskraft (wie in Abbildung 6.8 gezeigt) vektoriell in Anteile zerlegt, welche sich auf jeweils zwei Achsen der Wiimote aufspalten. So können in einem Experiment die Messdaten von jeweils zwei Sensoren für eine bestimmte, bekannte Kraft ermittelt werden. Abbildung 6.7 zeigt die Regressionsgeraden durch die 9 Messpunkte für jeden der Beschleunigungssensoren. Mit den zugehörigen Geradengleichungen (Gleichungen 6.7, 6.8, 6.9) kann man die anliegende Beschleunigungskraft aus den Messwerten berechnen.

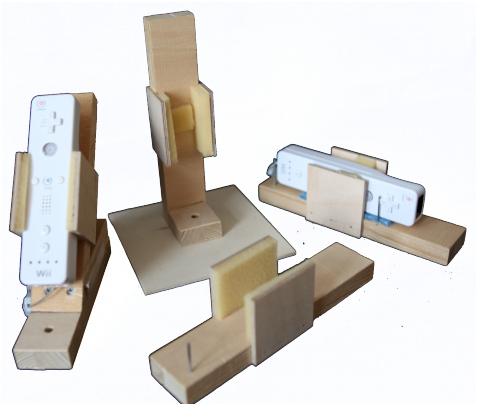


Abbildung 6.6: Bild einer Wiimote in einer Haltevorrichtung

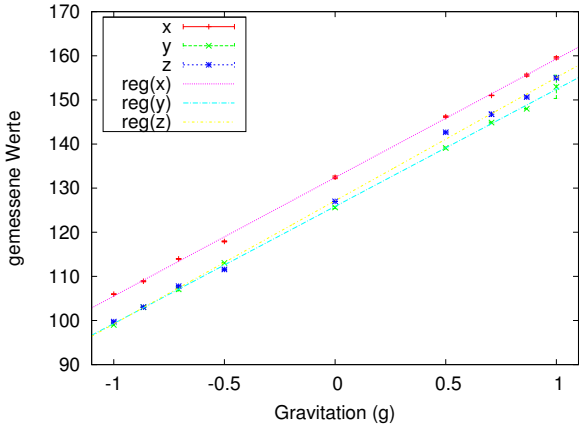


Abbildung 6.7: Regressionsgeraden durch die Kalibrierungsmesspunkte

$$Ma_x = 26.870 \cdot F_x + 132.4 \overset{f(x)^{-1}}{\Rightarrow} F_x = \frac{Ma_x - 132.4}{26.870} \tag{6.7}$$

$$Ma_y = 26.535 \cdot F_y + 125.854 \overset{f(x)^{-1}}{\Rightarrow} F_y = \frac{Ma_y - 125.854}{26.535} \tag{6.8}$$

$$Ma_z = 27.923 \cdot F_z + 127.136 \overset{f(x)^{-1}}{\Rightarrow} F_z = \frac{Ma_z - 127.136}{27.923} \tag{6.9}$$

Hierbei sind $Ma_{[x|y|z]}$ die Messwerte der Accelerometer und $F_{[x|y|z]}$ Skalare, welche die Kraft angeben, die an den Messachsen anliegt. Es ist aber zu beachten, dass die tatsächlich anliegende Kraft dem Produkt aus dem Skalar und der Gravitationskraft ($F_{[x|y|z]} \cdot \vec{g}$) entspricht. Denn die Formeln stellen einen Bezug zwischen den Messwerten der Accelerometer und der Gravitationskraft her. Bei Sensoren mit einer höheren Präzision ist ein Vorgehen, wie in [Ped13a] beschrieben, sinnvoll. Die Präzision der Wiimote ist allerdings niedrig genug, sodass für die Anwendungszwecke dieser Diplomarbeit vereinfachte lineare Modelle, wie oben vorgestellt, benutzt werden können.

Bestimmung der Nullwerte und der Messwerte für die positive und negative Gravitationskraft

Die Werte für Null und für ±1g können aus den Regressionsgeraden der Kalibrierungsexperimente berechnet werden, das bedeutet aber einen Verlust von Genauigkeit, da die Gerade selbst durch eine Näherung entsteht. In diesem Abschnitt sind die Ergebnisse der Experimente dargestellt, welche separat die Werte für -1g, 0g und +1g für jeden Sensor einzeln bestimmen. Diese Werte zusätzlich zu den Regressionsgeraden zu speichern, verbessert die Bestimmung des freien Falls, sowie einiger Orientierungen der Wiimote.

Nullwerte	x-Achse			y-Achse			z-Achse		
Messwerte	132	133	134	125	126	127	125	126	127
Häufigkeit	1293	25482	116	1054	30615	7327	280	34890	1634

Tabelle 6.1: Nullwerte der Accelerometer und zugehörige Häufigkeit

In Tabelle 6.1 sieht man diejenigen Messwerte, welche die Wiimote überwiegend anzeigt, wenn keine Gravitationskraft auf die einzelnen Sensoren wirkt.

+1g	x-Achse			y-Achse			z-Achse		
Messwerte	159	160	161	152	153	154	152	153	154
Häufigkeit	7948	29739	1240	32	17697	9162	637	34729	1272

Tabelle 6.2: Messwerte der positiven Gravitationskraft von den Accelerometern und zugehörige Häufigkeit

Versuchsaufbau Für die Kalibrierung der Nullwerte und der ±1g Werte wurde die Wiimote in sechs verschiedenen Orientierungen in Ruhe auf einen Tisch gelegt bzw. gestellt und es wurden die Messwerte für ungefähr eine Minute aufgezeichnet. Die

-1g	x-Achse			y-Achse		z-Achse		
Messwerte	105	106	107	99	100	97	98	99
Häufigkeit	358	15181	355	26735	10095	1212	22753	621

Tabelle 6.3: Messwerte der negativen Gravitationskraft von den Accelerometern und zugehörige Häufigkeit

Orientierungen der Wiimote waren (in Klammern was auf welcher Achse gemessen wurde):

- flach liegend, Knöpfe nach oben (+1g: z-Achse, Nullwert: x,y-Achsen)
- flach liegend, Knöpfe nach unten (-1g: z-Achse, Nullwert:x,y-Achsen)
- seitlich liegend, Knöpfe nach links (-1g: x-Achse, Nullwert:y,z-Achsen)
- seitlich liegend, Knöpfe nach rechts (+1g: x-Achse, Nullwert:y,z-Achsen)
- aufrecht stehend, Kamera nach oben (+1g: y-Achse, Nullwert:x,z-Achsen)
- aufrecht stehend, Kamera nach unten (-1g: y-Achse, Nullwert:x,z-Achsen)

Damit die Wiimote stabil stand, wurden eigens dafür gebaute Haltevorrichtungen benutzt. (Bilder davon sind im Anhang A.4 auf Seite XII Abb.A.4 zu sehen.)

6.3.6 Neigungsbasierte Steuerung mittels Wiimote

Für die Berechnung der absoluten Neigung der Wiimote wird lediglich die Gravitationskraft der Erde benutzt. So lassen sich die *roll* und *pitch* Winkel der Wiimote in Bezug auf ein Weltkoordinatensystem bestimmen, nicht jedoch der *yaw* Winkel, da dieser in der Ebene liegt, welche orthogonal zur Gravitation ist.

Da die Sensoren der Wiimote so angeordnet sind, dass der Gravitationsvektor $\overrightarrow{g}$, welcher stets auf die Wiimote wirkt[8], in zwei orthogonale Wirkkomponenten zerlegt wird, ist es nicht möglich Drehungen um die Gravitationsachse (*yaw*) zu messen.

Im trivialen Fall, wenn eine der Achsen der Wiimote parallel zu $\overrightarrow{g}$ ist, kann natürlich nur eine Komponente gemessen werden. Ihr Betrag entspricht der Länge von $\overrightarrow{g}$, die anderen Sensoren zeigen 0 an. Im allgemeinen Fall jedoch zerteilt sich $\overrightarrow{g}$ wie in Abbildung 6.8 sichtbar auf zwei Vektoren, welche auf den Sensorachsen der Wiimote liegen. Dort ist zu sehen, wie die beiden Vektoren $\overrightarrow{z}$ und $\overrightarrow{y}$ sich zum Gravitationsvektor $\overrightarrow{g}$ (vektoriell) summieren und mit ihm ein rechtwinkliges Dreieck bilden. In diesem Dreieck stellt der Gravitationsvektor die Hypotenuse dar. Der Winkel zwischen Erde und Wiimote wird mit Hilfe von $\overrightarrow{z}$ und $\overrightarrow{g}$ beschrieben.

[8] Es sei denn, sie befindet sich im freien Fall. In diesem Falle wären alle gemessenen Kräfte gleich 0. Dieser Fall wird an dieser Stelle nicht berücksichtigt.

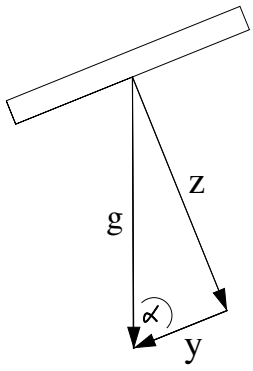


Abbildung 6.8: Der Winkel α gibt die Neigung der Wiimote zur Gravitation an. Der Winkel zur Erde ist β, hier nicht eingezeichnet. $\beta = 90° - \alpha$

Die Neigung der Wiimote kann aus $\vec{g}$ in den beiden X/Z oder Y/Z Ebenen der Wiimote nach Formel 6.10 errechnet werden.

$$\alpha_{aktuell} = arctan(\frac{\vec{y}}{\vec{g}}) \tag{6.10}$$

Sobald der Winkel α zwischen Wiimote und Gravitation, bzw. der Winkel β zwischen Wiimote und Erde, bekannt ist, lassen sich die Orientierungen *roll* und *pitch* der Wiimote bestimmen. Diese können ihrerseits für die Steuerung einer mobilen Plattform oder eines Manipulators eingesetzt werden. Letzterer kann kartesisch oder im Gelenkwinkelraum angesteuert werden.

Steuerung einer mobilen Plattform

In den hier gemachten Experimenten wurde die Pioneer Plattform wie folgt gesteuert:

Die Wiimote wird liegend in der Hand gehalten, mit den Knöpfen nach oben und der IR-Kamera nach vorne (vom Benutzer weg zeigend). Es werden die Werte für *roll* und *pitch*, sowie die Geschwindigkeiten und Drehungen der Plattform auf den Bereich von ± 1.0 normiert. Im nächsten Schritt wird *roll* auf die Drehung und *pitch* auf die Geschwindigkeit abgebildet. Wobei nicht normierte Werte für Drehungen sich im Bereich von $\pm 90°$ und die Geschwindigkeiten zwischen $+v_max$ (vorwärts) und $-v_max$ (rückwärts) befinden. Unter Beachtung der Vorzeichen können die Messwerte der Wiimote auf die Steuerungswerte der Plattform linear, kubisch oder linear/kubisch mit einem Totband abgebildet werden, wie in Abb.6.1 sichtbar. Das Prinzip ist das selbe, wie in Abschnitt 6.2.2 beschrieben. Die Steuerung mit einem Totband wurde sowohl kubisch als auch linear realisiert und ist je nach Anwendungsfall unterschiedlich gut geeignet.

Steuerung eines Manipulators

Die kartesische Steuerung eines Manipulators kann prinzipiell genau so realisiert werden, wie im Falle der Joysticksteuerung aus Abschnitt 6.2.3 . Nach einer Normierung der Werte in den Bereich von -1.0 bis 1.0, werden dieses Mal *roll* und *pitch* auf die sechs DOF abgebildet und eine Umschaltung durch Knöpfe ist nötig. Auch die Steuerung im Gelenkwinkelraum unterscheidet sich nicht prinzipiell von der Joysticksteuerung. Wichtig ist zu beachten, dass bei beiden Steuerungsarten die Veränderungen inkrementell sind. Eine direkte Abbildung des gesamten Messbereiches der Wiimote in den Zielwertebereich bietet sich nicht an. Bei dieser Art der Abbildung würde eine Maximalauslenkung der Wiimote einem Maximalstand eines Gelenkes, bzw. einem Randpunkt im Arbeitsbereich des Manipulators entsprechen. Messwerte im Messbereich der Wiimote würden direkt auf Werte im Arbeitsraum bzw. Gelenkwinkelraum abgebildet werden. Ein solches Steuerungskonzept ist besonders im Falle der neigungsbasierten Steuerung überhaupt nicht praktikabel.

6.3.7 Modulation von Bewegungsparametern modularer Roboter

Innerhalb der Diplomarbeit von Dennis Krupke entstand eine Simulationssoftware für modulare Roboter. Ihr Aufbau und ihre Funktionsweise sind in [KLZ+12] und der Diplomarbeit selbst detaillierter beschrieben.

In Zusammenhang mit der vorliegenden Diplomarbeit wurde die Wiimote benutzt, um Fortbewegungsparameter von modularen Robotern, wie in [NKH+12] beschrieben, zu modellieren. Dabei wurden Experimente gemacht, bei denen durch Neigung der Wiimote die Amplitude der Fortbewegung und der Winkel *yaw* zwischen den Modulen im Simulator variiert wurde. Die Knöpfe können eingesetzt werden, um z.B. noch die Phase zu verändern. Damit können Auswirkungen der veränderten Parameter direkt im Simulator beobachtet werden. Auf dem beiliegenden Datenträger befindet sich ein Video, auf dem man die Ergebnisse gut erkennen kann. Die Software aus [KLZ+12] ist geeignet, um neben simulierten auch reale, modulare Roboter zu steuern.

Dies ist nicht Gegenstand der vorliegenden Diplomarbeit. Während der Simulationsexperimente hat sich gezeigt, dass mittels Wiimote eine wesentlich intuitivere Modulation der Parameter möglich ist, als durch die Eingabe von Zahlenwerten alleine und die Auswirkungen nahezu instantan beobachtet und gegebenenfalls verbessert werden können.

6.3.8 Berechnung der Orientierung der Wiimote mittels Kalman-Filter basierter Sensorfusion von Accelerometer- und Gyroskopdaten

Sensordatenfusion der Messdaten von Accelerometern und Gyroskopen kann, wie in [Ped13b] beschrieben, mittels vereinfachter Kalman-Filter realisiert werden. Verein-

facht ist das Verfahren deshalb, weil kein vollständiges Modell aller Orientierungen der Wiimote, mit zugehörigen Wahrscheinlichkeiten für die Korrektheit der Accelerometerdaten erstellt wird. Stattdessen werden bei der Sensordatenfusion hier drei Fälle mit unterschiedlichen Wahrscheinlichkeiten für die Validität der Accelerometerdaten unterschieden und diese, abhängig von ihrer Wahrscheinlichkeit, mit den Gyroskopdaten fusioniert. Grundsätzlich werden die Gyroskopdaten direkt als Eulerwinkel interpretiert und auf die Transformationsmatrix der darzustellenden Wiimote draufmultipliziert. Falls zu erwarten ist, dass durch Fusion mit den Accelerometerdaten Verbesserungen zu erwarten sind, werden die Gyroskopdaten modifiziert und mit den Accelerometerdaten verrechnet. Bei der Fusion sind folgende Fälle unterschieden worden:

- Eine der Achsen der Wiimote verläuft parallel zur Erdgravitationskraft.
- Genau eine der Achsen der Wiimote verläuft orthogonal zur Erdgravitationskraft.
- Alle übrigen Orientierungen der Wiimote

Für eine Unterscheidung, welche Orientierung und damit welcher der o.g. Fälle vorliegt, werden die Messdaten aus Abschnitt 6.3.5 für ±1g und Null als Schwellwerte benutzt (Tabellen 6.1, 6.2 und 6.3). Wenn ein Accelerometerwert im Bereich von 0.9 bis 1.1 eines der Schwellwerte liegt, wird angenommen, dass die entsprechende Orientierung vorliegt. Ein kleineres Intervall als dieses hat sich in Versuchen nicht bewährt. Das liegt an der gegebenen Genauigkeit der Accelerometer.

Der eindeutigste Fall liegt vor, falls eine der Achsen der Wiimote parallel zur Gravitationskraft verläuft. Die anderen beiden Achsen sind dann orthogonal zu ihr und ein Achsenwinkel beträgt 0° während die anderen beiden jeweils 90° im Verhältnis zu $\vec{g}$ messen. Insgesamt gibt es sechs solcher Orientierungen. Bei jeder von ihnen kann mittels der Accelerometer ein Eulerwinkel nicht bestimmt werden, nämlich derjenige, der um die Gravitationsachse dreht. Die anderen beiden haben den Wert Null. Bei diesen Orientierungen wird die Wahrscheinlichkeit für die Korrektheit der Accelerometerdaten mit 1.0 angenommen und die gemessene Gravitation wird idealisiert zu einem Einheitsvektor, bei dem zwei Komponenten Null sind. Darauf aufbauend kann die Orientierung der repräsentierten Wiimote korrigiert werden. Der Winkel δ zwischen dem idealisiertem Gravitationsvektor und dem Vektor der parallelen Achse sollte 0° betragen, wird aber meist echt größer sein und bildet damit den zu korrigierenden Fehler. Er wird durch Bildung von gewichteten Mittelwerten bei einem Gewichtungsverhältnis von 100:1 (Gyroskop:Accelerometer) iterativ verringert.

$$\delta_{fused} = \frac{Eulerwinkel_{gemessen} \times Gewichtsfaktor + \delta}{Gewichtsfaktor + 1} \tag{6.11}$$

Da der Eulerwinkel, welcher um die Gravitationsachse dreht, aus den Accelerometerwerten nicht berechenbar ist, darf dieser auch nicht durch sie verändert werden.

Die anderen beiden Eulerwinkel der repräsentierten Wiimote müssen geeignet dahingehend geändert werden, sodass der Gesamtfehler gegen Null läuft. Dies geschieht nacheinander. Dazu wird zunächst um die Rotationsachse des ersten Eulerwinkels um den Winkel δ_{fused} in eine Drehrichtung ($+\delta_{fused}$) gedreht, und mit der Drehung um die entgegengesetzte Drehrichtung ($-\delta_{fused}$). Diejenige Drehrichtung, welche den Gesamtfehler verringert, wird beibehalten, die andere verworfen. Anschließend wird der Prozess, beginnend mit der Neuberechnung von Gleichung 6.11 für den zweiten (variablen) Eulerwinkel, wiederholt. Auf diese Weise konvergiert mit der Zeit die Orientierung der repräsentierten Wiimote gegen die Orientierung der realen Wiimote, bis auf eine Drehung um die Gravitationsachse.

Das sind die sechs Orientierungen der Wiimote, mit zugehörigem idealisierten Gravitationsvektor und ihrem nicht erkennbaren Eulerwinkel: (Die anderen beiden Eulerwinkel werden zur Fehlerminimierung bei der Fusion verändert)

- flach liegend, Knöpfe nach oben ($\vec{g} =< 0, 0, 1 >$ nicht messbar: yaw)
- flach liegend, Knöpfe nach unten ($\vec{g} =< 0, 0, -1 >$ nicht messbar: yaw)
- seitlich liegend, Knöpfe links ($\vec{g} =< -1, 0, 0 >$ nicht messbar: pitch)
- seitlich liegend, Knöpfe rechts ($\vec{g} =< 1, 0, 0 >$ nicht messbar: pitch)
- aufrecht stehend, Kamera oben ($\vec{g} =< 0, 1, 0 >$ nicht messbar: roll)
- aufrecht stehend, Kamera unten ($\vec{g} =< 0, -1, 0 >$ nicht messbar: roll)

Diese sechs Orientierungen der ersten Gruppe mit einer Wiimoteachse parallel zu $\vec{g}$ und den beiden anderen orthogonal zu $\vec{g}$, bilden Spezialfälle der zweiten Gruppe, wo nur noch genau eine Achse orthogonal zu $\vec{g}$ ist. Es wird eine Achse als orthogonal angesehen, sobald ihr gemessener Wert kleiner als $0.15 \times |\vec{g}|$ ist.

Eine orthogonale x-Achse entspricht einer Wiimote, welche von einer (auf dem Tisch) liegenden Position frontal hoch- bzw. runter geneigt wird. Über die Messwerte der Accelerometer kann dabei der Eulerwinkel *pitch* (Neigung) bestimmt werden.

Die y-Achse ist orthogonal zu $\vec{g}$, wenn die Wiimote liegt, oder parallel zur Erde gehalten wird. Dabei kann sie um die y-Achse gerollt werden. Aus den anderen beiden Achsen kann der Eulerwinkel *roll* ermittelt werden.

Wenn die z-Achse orthogonal zu $\vec{g}$ ist, dann steht die Wiimote aufrecht und durch seitliches Neigen wird der Eulerwinkel *yaw* (Gieren) verändert. Grundsätzlich gilt für jede der Orientierungen aus dieser Gruppe, dass nur ein Eulerwinkel bestimmt werden kann, nämlich derjenige, welcher um die zu $\vec{g}$ orthogonale Achse dreht, aber für seine Bestimmung die Messwerte der beiden anderen Achsen benötigt werden (vgl. Formel 6.10, Abschnitt 6.3.6).

Die Verknüpfung der Accelerometer- und Gyroskopdaten findet in der vorliegenden Software auf geometrische Art statt. Zunächst werden die gemessenen Accelerometerdaten zu einem Gravitationsvektor $\vec{g}_{mess}$ zusammengesetzt und dieser wird, durch geeignete Wahl eines Divisors, auf die Länge 1 normiert. Es gilt:

$$\vec{g}_{mess} = \left\langle \frac{x_{acc}}{|x_{acc}| + |y_{acc}| + |z_{acc}|}, \frac{y_{acc}}{|x_{acc}| + |y_{acc}| + |z_{acc}|}, \frac{z_{acc}}{|x_{acc}| + |y_{acc}| + |z_{acc}|} \right\rangle \quad (6.12)$$

Bei der Sensordatenfusion geht es darum, den Winkel(fehler) $\epsilon = \vec{g}_{mess} \measuredangle \vec{g}_{Welt}$ zwischen dem gemessenen Gravitationsvektor $\vec{g}_{mess}$ und dem tatsächlichen Gravitationsvektor $\vec{g}_{Welt}$ zu minimieren.

Da dies direkt nicht möglich ist, kann es nur über eine Veränderung des jeweils messbaren Eulerwinkels geschehen. Sei $\delta_{[roll|pitch|yaw]}$ der jeweilige Unterschied zwischen dem (mittels Accelerometerwerte) tatsächlich gemessenem Eulerwinkel und dem intern repräsentierten. Dann gilt, dass der Gesamtfehler ϵ maximal so groß sein kann, wie die Summe der Eulerwinkelfehler ($\delta_{\sum}$).

$$\epsilon \leq |\delta_{roll}| + |\delta_{pitch}| + |\delta_{yaw}| = \delta_{\sum} \quad (6.13)$$

Für jede Orientierung aus dieser Gruppe wird der fusionierte Eulerwinkel ϕ_{fused} wie folgt bestimmt:

$$\phi_{fused} = \frac{Euler_{Acc}}{5} + \frac{4 \times Euler_{Gyro}}{5} \quad (6.14)$$

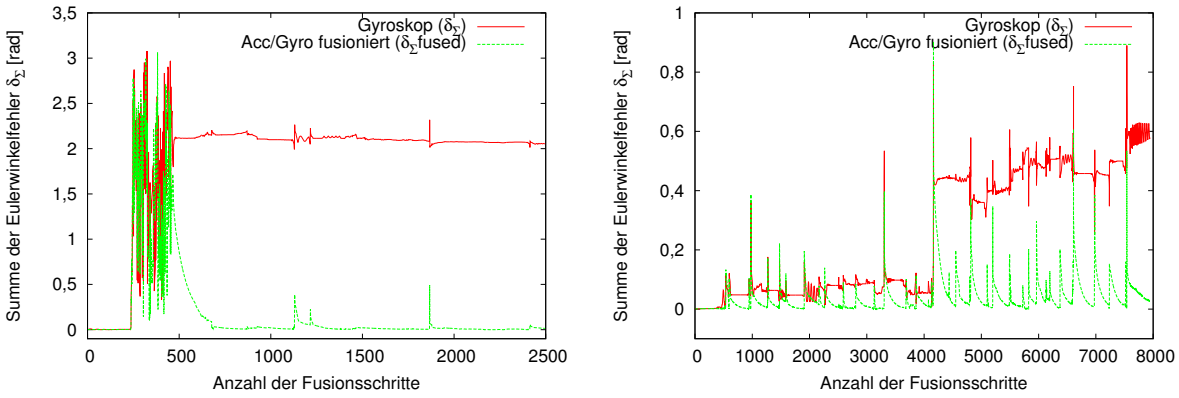
Der Eulerwinkel aus den Accelerometerdaten kann direkt errechnet werden, während der Gyroskopwinkel aus dem Mittelwert der aktuellen Gyroskopdaten und dem δ_{fused} aus dem vorherigen Zeitschritt gebildet wird.

Auf diese Weise wird der Gesamtfehler ϵ minimiert. In den Orientierungen der dritten Gruppe, werden die Gyroskopdaten unverändert an die Visualisierung und interne Positionsbestimmung weitergegeben.

In Abbildung 6.9 sind die summierten Orientierungsfehler der reinen Gyroskopdaten den Orientierungsfehlern der fusionierten Daten gegenübergestellt. Dabei wird die Orientierung, welche aus den Accelerometerdaten errechnet wird, als die Zielvorgabe angenommen. Auf der Abszisse ist die Anzahl der Fusionsschritte aufgetragen. Eine Fusion findet nur statt, wenn eine bekannte Orientierung aus den Orientierungen der Gruppe 2 vorliegt und damit aus den Accelerometerdaten berechnet werden kann. Dies geschieht mit einer Frequenz von 100 Hz. Die 8000 Fusionsschritte in Abb.6.9(b) fanden also in etwa 8 Sekunden statt.

Die Ordinate ist beschriftet mit der Fehlersumme $\delta_{\sum}$ der Eulerwinkel in Radiant. Diese kann theoretisch im Bereich von $[0..2\pi]$ liegen, wird aber praktisch grundsätzlich kleiner sein. Das liegt z.T. daran, dass sich mit dem Accelerometer nur maximal zwei Eulerwinkel zur selben Zeit bestimmen lassen.

Anhand der Abb.6.9(a) kann man erkennen, dass bei einer Auslenkung der Orientierung, welche so stark ist, dass sie einer zufälligen Orientierung gleichkommt, anhand der Gyroskopdaten alleine keine Verbesserung der Fehlersumme $\delta_{\sum}$ zu erwarten ist, während mittels Sensorfusion die Fehlersumme minimiert werden kann.



(a) Start bei zufälliger Orientierung

(b) Start bei bekannter Orientierung und Akkumulation von Winkelfehler (Gyroskop)

Abbildung 6.9: Gegenüberstellung von fusionierter Summe der Orientierungsfehler (aus Gruppe 2) $\delta_\sum fused$ und der Fehlersumme der reinen Gyroskopdaten $\delta_\sum gyro$

In Abb.6.9(b) ist dargestellt, wie sich der Winkelfehler bei reinen Gyroskopdaten über die Zeit aufsummiert, während die fusionierten Daten immer wieder auf ein Minimum korrigiert werden können. Es sei noch ein Mal darauf hingewiesen, dass die Abszisse nicht die Zeit, sondern die Fusionsschritte darstellt. Wenn keine Fusion möglich ist (etwa, weil die aktuelle Orientierung nicht aus den Accelerometern berechnet werden kann), vergeht Zeit, aber die Fusionsschritte werden nicht inkrementiert. Daraus und aus der Tatsache, dass der Gyroskopfehler nur schwach wächst, kann man erkennen, dass das Gyroskop auch ohne Sensorfusion über weite Abschnitte gute Messdaten liefert. Es ist aber auch erkennbar, dass ohne eine Sensorfusion auf Grund der Drift eine dauerhafte Benutzung des Eingabegerätes Wiimote nicht möglich ist.

Warum der Winkelfehler nicht 3π betragen kann:

Rein theoretisch könnte jede dargestellte Achse exakt in die entgegengesetzte Richtung der Messachse zeigen und damit könnte eine Summe von 3π entstehen. Dies ist aber praktisch nicht möglich und kann folgendermaßen illustriert werden: Eine Drehung um π eines Eulerwinkels, gefolgt von einer Drehung eines anderen Eulerwinkels, ebenfalls um π bilden die maximale Auslenkung. Eine Drehung um den dritten Eulerwinkel mit dem Wert π würde zur ursprünglichen Orientierung führen und damit einer Identität gleichkommen. Somit ist ein $\delta_\sum > 2\pi$ nicht möglich, da jeder Versuch den Winkelfehler eines Eulerwinkels zu vergrößern im Verringern eines anderen Winkelfehlers resultieren würde.

6.3.9 Bestimmung der Position und Orientierung der Wiimote mittels doppelter Integration der Daten der Inertialsensoren

Anfänglich war es geplant, dass die durch die kalibrierten Accelerometer gemessene Beschleunigung der Wiimote integriert werden kann, damit aus ihr eine Geschwindigkeit berechnet werden kann und dann durch wiederholte Bildung eines Integrales die Position aus den Beschleunigungsdaten errechnet wird.

Im Rahmen dieser Diplomarbeit wurden nach intensiver Literaturrecherche letztlich jedoch keine Experimente in diese Richtung durchgeführt. In [SC09] wird ein erheblich größeres, teureres und genaueres Accelerometer benutzt, als es die Wiimote besitzt und die Position wird gerade mal für 37s noch mit annehmbarer Genauigkeit bestimmt.

In [LG11] wird mit einem Accelerometer (ADXL-335) gearbeitet, welches dem der Wiimote (ADXL-330) sehr ähnlich ist, jedoch eine höhere Genauigkeit ($\pm$5g) und eine höhere Samplingrate (200Hz im Vergleich zu max. 100 Hz bei der Wiimote) besitzt. Die Autoren zeigen, dass bei einer Drehung nach 5s der Positionsfehler 62cm beträgt. Bei linearen Beschleunigungen zeigen sie, dass nach 2 Sekunden ein Positionsfehler von 6,88 Metern vorliegt und sich mittels Kalmanfiltern dieser Fehler auf 3,58 Meter reduzieren lässt. Solche Werte sind für die Steuerung eines Manipulators durch Beschleunigungsmessung der Wiimote inakzeptabel. Mittels "smoothing-techniques" wird das Verfahren dort zwar verbessert, allerdings lassen sich diese Ideen nicht auf die vorliegende Arbeit anwenden, weil hier kein Vorwissen über die (Steuerungs)-Bewegung des Benutzers vorhanden ist.

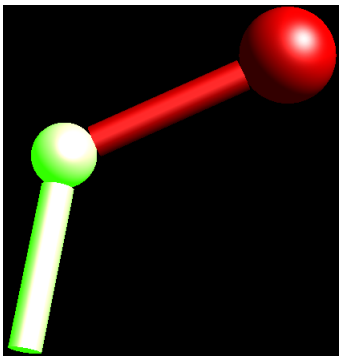
In [LP01] fassen die Autoren den Fehler mit 176cm pro Minute zusammen, während in [NMK05] (ADXL-250) nach einer Zeit von 4-8 Sekunden ein Positionsfehler entsteht, welcher für die Zwecke dieser Diplomarbeit ebenfalls zu groß ist. Die Sensoren in den zitierten Arbeiten entsprechen zwar nicht exakt denen der Wiimote, da aber die Wiimote die mit Abstand günstigsten Sensoren besitzt, ist zu erwarten, dass ihr Driftverhalten bestenfalls so gut sein wird, wie die untere Schranke der Messergebnisse in den zitierten Arbeiten. Mit diesen Ergebnissen und einer Samplingrate, welche maximal halb so groß ist, wie in den zitierten Arbeiten, wurden hier keine Experimente mit doppelter Integration der Accelerometerwerte der Wiimote gemacht, da davon auszugehen ist, dass die Positionsbestimmung viel zu ungenau sein würde für eine Teleoperation bzw. erfolgreiche Manipulatorsteuerung.

6.4 Experimente mit mehreren Wiimotes gleichzeitig

Mittels der durchgeführten Experinete sollte geklärt werden, ob sich die Wiimote dazu eignet die Orientierung eines menschlichen Armes sensorisch abzugreifen, damit diese auf einen Roboterarm übertragen werden kann. Es wurden dazu zwei Wiimotes

an einem menschlichen Arm befestigt, eine am Oberarm und eine am Unterarm, wie in Abb.6.10 gezeigt.

Als Visualisierung diente ein schematisches Modell einer Schulter mit einem Oberarm, wie in Abb.6.10(a) dargestellt. Dabei wir das Schultergelenk durch eine Kugel (Farbe rot) modelliert, der Oberarm durch einen breiteren Zylinder (ebenfalls rot), während Elenbogen und Unterarm durch entsprechend kleinere Figuren dargestellt werden. Das sind in Abb.6.10(a) ebenfalls eine Kugel und ein Zylinder, jeweils im Radius verringert und in einer anderen Farbe (blau).



(a) schematisch dargestellt in JAVA 3D

(b) menschlicher Arm mit zwei befestigten Wiimotes

Abbildung 6.10: Darstellung eines Armes mit zwei Wiimotes

Wie bereits in Abschnitt 4.1 ausgeführt, wurde die CWiid Bibliothek für den Betrieb mit einer Wiimote entwickelt. Aktuelle Versionen der JVM erlauben es nicht, dass eine shared library mehrfach geladen wird. Dies zusammengenommen führt dazu, dass für die Experimente dieses Abschnittes für jede zu nutzende Wiimote eine eigene Version der CWiid kompiliert und als shared library geladen werden musste.

Da die Bewegung des Unterarmes von der Bewegung des Oberarmes kinematisch abhängt, dies aber schwerer zu handhaben ist, wurde hier der Ansatz gewählt, dass die Bewegungen der Wiimote am Oberarm aus den Bewegungen der am Unterarm befestigten Wiimote herausgerechnet wurden. Dies geschieht, indem die Transformationsmatrix des Unterarmes mit der inversen Matrix des Oberarmes multipliziert wird. Auf diese Weise konnte realisiert werden, dass die früheren Glieder der kinematischen Kette keinen Beitrag zu den Orientierungen der späteren Glieder haben und sich die Berechnung der Orientierung des Unterarmes deutlich vereinfacht. So können die gemessenen Orientierungen getrennt betrachtet werden und genau wie in den Experimenten zuvor mittels Sensordatenfusion verbessert werden.

Fazit

7

In dieser Diplomarbeit wurde der Frage nachgegangen, ob sich durch den Einsatz alternativer Interaktionswerkzeuge, die Mensch-Roboter-Interaktion (am Arbeitsbereich TAMS) verbessern lässt. Dazu wurde die Wiimote, ein Controller aus dem Bereich der Videospiele, als Kommunikationsgerät gewählt. Um die Wiimote einsetzen zu können, wurde eine Software geschrieben, die mittels des Java Native Interfaces auf eine native C-Bibliothek zugreift und so die Daten der Wiimote erhält. Die hier erstellte Software kann in ihrem Kern für die Darstellung der Daten, das persistente Speichern und Wiederabspielen, sowie die Visualisierung der Wiimote benutzt werden. Intern werden die Multisensordaten fusioniert. Dazu wurden verschiedene Ansätze erprobt und ein Ansatz gewählt, der auf dem Kalman-Filter basiert.

7.1 Zusammenfassung der Ergebnisse

Zunächst wurde im Rahmen dieser Diplomarbeit nach einem geeigneten Eingabegerät gesucht. Nachdem sich die Wiimote als geeignet dargestellt hat, wurden Bibliotheken zur Kommunikation mit der Wiimote evaluiert. Mit der CWiid wurde eine quelloffene, unter Linux lauffähgie Bibliothek gefunden. Für die in dieser Diplomarbeit erstellte Software wurde ein Zugriff auf die CWiid mittels JNI realisiert, sodass mobile Plattformen (Pioneer), sowie der Mitsubishi PA-10 Arm gesteuert werden können.

Zu diesem Zweck werden verschiedene Steuerungskonzepte mittels Wiimote entwickelt und miteinander verglichen. Es können ausschließlich die Knöpfe der Wiimote benutzt werden, nur die Accelerometersensoren oder Gyroskope und Accelerometer fusioniert. Die Steuerung selbst kann auch auf verschiedene Arten erfolgen. Dabei hat sich herausgestellt, dass eine direkte Abbildung vom Arbeitsraum des Eingabegerätes auf den Arbeitsraum des Roboters nicht sinnvoll ist. Besser ist es, wenn z.B. abhängig von der Neigung die Richtung (und ggf. Geschwindigkeit) verändert wird.

Ferner können Daten mehrerer Wiimotes zeitgleich verarbeitet werden. Das wird benutzt, um die vereinfachte Kinematik eines menschlichen Armes zu berechnen und seine Bewegung im Computer zu simulieren.

Im Arbeitsbereich TAMS kann die Wiimote eingesetzt werden um den (PA10) Arm zu bewegen. Jetzt kann ein Experimentator bei Greifexperimenten mit der Shadow-Hand sich auf die Hand konzentrieren und muss sich nicht damit beschäftigen, welche Koordinaten einzugeben sind, damit der Arm an die gewünschte Position fährt.

7.2 Ausblick

Aufbauend auf diese Arbeit könnte eine Steuerung eines Roboterarmes implementiert werden, bei der die Position des visualisierten (menschlichen) Armes berechnet wird und der zu steuernde Roboterarm mittels inverser Kinematik an die gewünschte Position gebracht wird. Der erste Teil, bis zur Visualisierung im Computer ist erledigt. Das Aufbauende wird einer zukünftigen (Abschluss)arbeit überlassen.

Da in dieser Arbeit die Drift orthogonal zur Gravitation nicht bestimmt werden kann, weil es keine Möglichkeit gibt es sensorisch zu erfassen, kann diese Drift mit der Zeit beliebig groß werden. Um das einzuschränken könnten weitere Versuche mit zusätzlichen Sensoren, wie einem 3-Achsen-Magnetometer gemacht werden. Die Sony Move [Ped12], welche der Wiimote sensorisch sehr ähnlich ist, würde sich dazu vermutlich gut eignen. Ferner kann in die Richtung der Sensorfusion von Kinect-Kamera (bzw. Sony-Eye-Kamera [PVK13]) und solchen Eingabegeräten geforscht werden. Die leuchtende Kugel kann gut in Bildern ausfindig gemacht werden.

Was die Steuerungskonzepte in dieser Arbeit angeht, so können weitere Untersuchungen zu anderen Steuerungsparametern wie Kraft, Geschwindigkeit, Beschleunigung, Drehmomente oder Gelenkwinkel folgen. Leider konnte eine Steuerung über z.B. Beschleunigung hier nicht realisiert werden, weil es den Rahmen dieser Arbeit zu stark ausgedehnt hätte.

Prototypisch lassen sich die Steuerungskonzepte allerdings, wie man in [CNPP13] sehen kann, benutzen und ausbauen.

Mittels der in dieser Arbeit angewendeteten Prinzipien kann aus einem Arduino[1] (Nano) Board und Inertialsensoren als Bauteile ein eigenes Steuerungsgerät gebaut werden. Dabei ist zu erwarten, dass durch Verwendung modernerer Chipsätze dieses neue Gerät der Wiimote überlegen sein würde. So lassen sie die mobilen Plattformen oder Roboterarme leicht um weitere Sensoren erweitern, die nützliche Informationen liefern können.

Aufbauend auf weiterentwickelte Steuerungskonzepte können nun Lernstrategien für die Serviceroboter entwickelt werden, sodass ihre Manipulatoren z.B. vorgegebenen Bahnen folgen oder sie lernen können, wo sich Gegenstände befinden. Dies kann durch Implementation der aktuell üblichen Lernverfahren (etwa neuronale Netze) geschehen.

[1] http://www.arduino.cc/en/Main/ArduinoBoardNano

Die Einsatzmöglichkeiten reichen bis zur Entwicklung einer Segway-artigen auf Pendeln basierenden Roboterplattform mit zwei Rädern, wie in [LJ09] beschrieben.

Danksagung

Bedanken möchte ich mich bei all denen, die mich bei der Erstellung dieser Diplomarbeit unterstützt haben.

Meinem Erstbetreuer Dr. Norman Hendrich gilt hier ein besonderer Dank, der sich wirklich immer für mich Zeit genommen hat, sodass ich mit den "möglichsten und unmöglichsten" Anliegen jederzeit zu ihm kommen konnte. Im Laufe der Zeit ist dabei meinerseits ein tiefer Respekt für die breite fachliche Kompetenz dieses Mannes gewachsen.

Selbstverständlich danke ich auch meinem Zweitbetreuer Prof. Dr. Jianwei Zhang für seine konstruktiven Vorschläge. Ferner gilt mein Dank allen übrigen Mitarbeitern des Arbeitsbereiches TAMS, die mich so freundlich dort berherbergt haben. Dabei seien Dr. Denis Klimentjew, Jan Gries und Ben Adler auf Grund ihrer konstruktiven Ideen, Vorschläge und z.T. tatkräftiger Unterstützung, besonders erwähnt. Ein herzlicher Dank geht auch an Hr. Dieter Florstedt vom AB TIS, der problemlos und nahezu instantan die Halterungen für die Kalibrierungsexperimente gebaut hat, sowie Dennis Krupke, der mit mir auf dem Boden herumgekrochen ist, um die Kameradistanz zu kalibrieren. Ihm und Nils Meins, Jan Bruder sowie Gunter Labes danke ich auch für erheiternde Gespräche, wenn es mal etwas zähflüssiger lief.

Für die erste Korrekturlesung danke ich Wiebke Eggers, Ralf Engelken, Doreen Jirak und Zehra Öztürk.

Ganz besonders herzlich danke ich allen TeilnehmerInnen der Korrektur-Lese-(Geburtstags)-Party: Betty, Dennis, Elli, Lasse, Lu, Nils, Nora, Pati, Thomas, Wiebke - Ihr wart spitze. Für das gemeinsame Durchgehen (bzw. Vorlesen) der Korrekturanmerkungen danke ich nochmals Lasse und Dennis, der seinen Freitagnachmittag und Abend dafür verwendet hat. Mögen die jeweiligen Partnerinnen dafür Verständnis haben. Die Verantwortung für etwaige, verbliebene (Recht)schreibfehler möchte ich gerecht geteilt wissen. ⌣̈

Zu guter letzt danke ich meiner Familie und meinen Freunden, insbesondere meinen Eltern, meiner Partnerin, sowie all denjenigen lieben Menschen, die mich unterstützt, an mich geglaubt und z.T. mich auf diesem Weg, bei dem ich mich gelegentlich sehr schwer getan habe, getragen haben.

Ohne Euch und Eure aufbauenden Worte hätte ich dieses Werk (in dieser Form) sicherlich nicht fertiggestellt.

Vielen Dank

Literaturverzeichnis

[Adl08] Adler, Benjamin: *Dynamische Navigation und Bewegungssteuerung eines mobilen Robotersystems in Innenräumen*, Universität Hamburg, Diplomarbeit, 2008 90

[AL02] Almeida, Jorge N. ; Lobo, Sousa: *Inertial sensor data integration in computer vision systems*, University of Coimbra, Diplomarbeit, 2002 26, 36

[BL08] Baier-Löwenstein, Tim: *Lernen der Handhabung von Alltagsgegenständen im Kontext eines Service-Roboters*, Universität Hamburg, Dissertation, 2008 54

[BPRR11] Bosetti, M. ; Pilolli, P. ; Ruffoni, M. ; Ronchetti, M.: Interactive whiteboards based on the WiiMote: Validation on the field. In: *Interactive Collaborative Learning (ICL), 2011 14th International Conference on*, 2011, S. 269–273 3

[Bru09] Bruder, Jan: *Praktische Realisierung einer haptischen Telerobotik-Steuerung für eine interaktive Nutzung*, Universität Hamburg, Diplomarbeit, 2009 54

[BSRI10] Balakrishna, D. ; Sailaja, P. ; Rao, R.V.V.P. ; Indurkhya, B.: A novel human robot interaction using the Wiimote. In: *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, 2010. – ISBN 978–1–4244–9319–7, S. 645 –650 3

[CF08] Christensen, Rolf ; Fogh, Nikolaj: *Inertial Navigation System*. 2008 93

[CNPP13] Ciobanu, Vlad ; Noeske, Martin ; Popescu, Nirvana ; Petrescu, Adrian: Robot telemanipulation system. In: *17th International Conference on System Theory, Control and Computing*, 2013 112

[CT65] Cooley, James ; Tukey, John: An Algorithm for the Machine Calculation of Complex Fourier Series. In: *Mathematics of Computation* 19 (1965), Nr. 90, S. 297–301 23

[DHS08] Decker, Markus ; Hohlfeld, Christian ; Schmid, Fabian M.: *Bewegungserkennung mit dem WiiMote Controller*. Projektbericht, 2008. – abgegeben 30. Januar 2008, HTWG Konstanz 3

[DSN10] Diniz, Paulo S. ; Silva, Eduardo A. ; Netto, Sergio L.: *Digital Signal Processing: system analysis and design*. 2. Auflage. Cambridge University Press, 2010. – ISBN 978–0521–88775–5 16, 23

[DZ12] Duo, Zhen ; Zhang, Wenhui: Constructing 3D position tracking system with Twin-Wiimote Structure. In: *Information Management, Innovation Management and Industrial Engineering (ICIII), 2012 International Conference on* Bd. 2, 2012, S. 466–469 3

[FE08] Francis Everitt, Tom L. Barry Muhfelder M. Barry Muhfelder: The Gravity Probe B Experiment "Testing Einstein's Universe" Science Result - NASA Final Report / Stanford Press Department. 2008. – Forschungsbericht 32

[FHA98] Foxlin, Eric ; Harrington, Michael ; Altshuler, Yury: Miniature 6-dof inertial system for tracking HMDs. In: *In Proc. SPIE Helmet and Head-Mounted Displays III*, 1998, S. 214–228 32

[Fla05] Flanagan, David: *Java In A Nutshell, 5th Edition.* O'Reilly Media, Inc., 2005. – ISBN 0596007736 60

[Fli91] Fliege, Norbert: *Systemtheorie.* Teubner Verlag, 1991. – ISBN 978–3519061403 23, 26

[Gar14] Garage, Willow: *Website des Robot Operation Systems.* `http://www.ros.org/`. Version: Februar 2014 66

[Gem14a] Gemeinschaftsprojekt: *Wii Homebrew Projekt.* Website, 2014. – Online verfügbar auf `http://www.wii-homebrew.com/`; zuletzt besucht am 28.2.2014 44

[Gem14b] Gemeinschaftsprojekt: *Wiibrew Projekt.* Website, 2014. – Online verfügbar auf `http://wiibrew.org/wiki/Main_Page`; zuletzt besucht am 28.02.2014 4, 44

[GMW81] Gill, P. ; Murray, W. ; Wright, M.: *Practical optimization.* London : Academic Press, 1981 80

[Gor98] Gordon, Rob: *Essential JNI: Java Native Interface.* Prentice Hall PTR, 1998. – ISBN 0-13-679895-0 61

[GRS05] Girod, Bernd ; Rabenstein, Rudolf ; Stenger, Alexander: *Einführung in die Systemtheorie.* 3. Auflage. Teubner B.G. GmbH, 2005. – ISBN 978–3519261940 23, 26

[GVH14] Gerkey, Brian ; Vaughan, Richard ; Howard, Andrew: *Website des Player(-Stage-Gezebo) Projektes.* `http://playerstage.sourceforge.net/`. Version: Februar 2014 65

[HE97] Harling, Philip A. ; Edwards, Alistair D. N.: Hand Tension as a Gesture Segmentation Cue. In: *Proceedings of Gesture Workshop on Progress in Gestural Interaction.* London, UK : Springer-Verlag, 1997. – ISBN 3–540–76094–6, 75–88 22

[HKK+03] Higuchi, Masaru ; Kawamura, Takeya ; Kaikogi, Takaaki ; Murata, Tadashi ; Kawaguchi, Masataka: *Mitsubishi Clean Room Robot*. 2003 54

[HLGS08] Herpel, T. ; Lauer, C. ; German, R. ; Salzberger, J.: Multi-sensor data fusion in automotive applications. In: *Sensing Technology, 2008. ICST 2008. 3rd International Conference on*, 2008, S. 206 –211 20

[HR86] Hayward, Vincent ; Richard, Paul P.: Robot manipulator control under Unix rccl: A robot control "C"library. In: *Int. J. Rob. Res.* 5 (1986), December, 94–111. http://dx.doi.org/10.1177/027836498600500407. – DOI 10.1177/027836498600500407. – ISSN 0278–3649 64

[HZJH12] Harbert, S.D. ; Zuerndorfer, J. ; Jaiswal, T. ; Harley, L.R.: Motion capture system using Wiimote motion sensors. In: *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, 2012. – ISSN 1557–170X, S. 4493–4496 3

[Jav97] JavaSoft: *Java Native Interface Specification, Release 1.1 (Revised May, 1997)*. ftp://ftp.javasoft.com/docs/jdk1.1/jni.ps, Mai 1997. – For later modifications, see documentation of JDK1.2betaX 61

[Jav98] *Java Documentation Page*. http://java.sun.com/docs/, 1998 61

[Jav99a] *Java Technology Home Page*. http://java.sun.com/, Januar 1999 60

[Jav99b] *Microsoft Technologies for Java Home Page*. http://www.microsoft.com/java/, Januar 1999 118

[Jav99c] JavaSoft: *Java Development Kit, Version 1.2*. http://java.sun.com/products/jdk/1.2/, Januar 1999. – See the online documentation. JDK 1.2 was released in December 1998 as Java 2 61

[KC03] Koks, Don ; Challa, Subhash: An introduction to bayesian and dempster-shafer data fusion / Defence Science and Tech Org. 2003. – Forschungsbericht 21

[KH90] Kurtenbach, Gord ; Hulteen, Eric: Gestures in Human-Computer Communication. In: Laurel, Brenda (Hrsg.): *The Art and Science of Interface Design*. Addison-Wesley Publishing Co., 1990, S. 309–317 22

[KLZ+12] Krupke, Dennis ; Li, Guoyuan ; Zhang, Jianwei ; Zhang, Houxiang ; Hildre, Hans P.: Flexible Modular Robotic Environment for Research and Education. In: *26th European Conference on Modelling and Simulation. ECMS2012*, 2012 7, 104

[Koc08] Koch, Thomas: *Rotationen mit Quaternionen in der Computergrafik - Eine Demonstration am Beispiel OpenGL*, Fachhochschule Gelsenkirchen, Diplomarbeit, 2008 20

[Kon00] Kong, Xiaoying: *Inertial Navigation System Algorithms for low cost IMU*, The University of Sydney, Diss., 2000 35

[Lab09] Labes, Gunter: *Simulation des parallelen Betriebs zweier MHI PA10-6C Roboterarme in RCCL*, Universität Hamburg, Diplomarbeit, 2009 55, 64

[LG11] Lele, M.A. ; Gu, J.: Evaluation of solid state accelerometer sensor for effective position estimation. In: *Intelligent Control and Automation (WCICA), 2011 9th World Congress on*, 2011, S. 959–964 109

[LH92] Lloyd, John ; Hayward, Vincent: Multi-RCCL User's Guide / McGill University. 1992. – Forschungsbericht 64

[Lia99] Liang, Sheng: *Java Native Interface: Programmer's Guide and Reference*. 1st. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 0201325772 61

[LJ09] Lee, Hyung-Jik ; Jung, Seul: Gyro sensor drift compensation by Kalman filter to control a mobile inverted pendulum robot system. In: *Industrial Technology, 2009. ICIT 2009. IEEE International Conference on*, 2009, S. 1–6 21, 113

[Llo85] Lloyd, John: *Implementation of a Robot Control Development Environment*, McGill University, Diplomarbeit, 1985 64

[LP01] Liu, H.H.S. ; Pang, G.K.-H.: Accelerometer for mobile robot positioning. In: *Industry Applications, IEEE Transactions on* 37 (2001), Nr. 3, S. 812–819. http://dx.doi.org/10.1109/28.924763. – DOI 10.1109/28.924763. – ISSN 0093–9994 109

[LPM88] Lloyd, J. ; Parker, M. ; McClain, R.: Extending the RCCL programming environment to multiple robots and processors. In: *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, 1988, S. 465 –469 vol.1 64

[LR09] Lahres, Bernhard ; Rayman, Gregor: *Praxisbuch Objektorientierung: Das umfassende Handbuch*. 2. Bonn : Galileo Press, 2009. – ISBN 978–3–8362–1401–8 69

[Ltd08] Ltd., Shadow Robot C.: *Shadow Dexterous Hand C5 Technical Specification*, 2008. (6) . – Handbuch und Technische Spezifikationen 55

[Mic99] Microsoft: *Integrating Java and COM*. Follow links "Technical Information" and "Technical Articles" at [Jav99b], Januar 1999 61

[Mor78] Moré, J. J.: The Levenberg-Marquardt Algorithm: Implementation and Theory. In: *G.A. Watson*. Berlin : Springer Verlag, 1978 (Lecture Notes in Math. 630), S. 105–116 80

[NKH+12] Noeske, Martin ; Krupke, Dennis ; Hendrich, Norman ; Zhang, Jianwei ; Zhang, Houxiang: Interactive control parameter investigation of modular robotic simulation environment based on Wiimote-HCI's multi sensor fusion. In: *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, 2012, S. 478 –483 2, 7, 104

[NMK05] Nikbakht, S. ; Mazlom, M. ; Khayatian, A.: Evaluation of solid-state accelerometer for positioning of vehicle. In: *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*, 2005, S. 729–733 109

[OM84] Osinga, Jan ; Maaskant, Johannes W.: *Handbuch der elektronischen Messgeräte. Grundlagen, Funktionsweise u. Messmethoden*. Franzis Verlag GmbH, 1984. – ISBN 978–3772373916 15

[OV09] Olufs, S. ; Vincze, M.: A simple inexpensive interface for robots using the Nintendo Wii controller. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, S. 473–479 3

[Ped12] Pedley, Mark: Build and Calibrate a Tilt-Compensating Electronic Compass / Freescale Semiconductor, Inc. August 2012 (ISSUE 265). – Forschungsbericht. – Available at `http://cache.freescale.com/files/sensors/doc/reports_presentations/ARTICLE_REPRINT.pdf` 112

[Ped13a] Pedley, Mark: High Precision Calibration of a Three-Axis Accelerometer / Freescale Semiconductor, Inc. 2013 (Document Number: AN4399 Rev. 1, 01/2013). – Forschungsbericht. – Available at `http://cache.freescale.com/files/sensors/doc/app_note/AN4399.pdf` 101

[Ped13b] Pedley, Mark: Tilt Sensing Using a Three-Axis Accelerometer / Freescale Semiconductor, Inc. 2013 (Document Number: AN3461 Rev. 6, 03/2013). – Forschungsbericht. – Available at `http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf` 104

[PGUv10] Petrič, T. ; Gams, A. ; Ude, A. ; Žlajpah, L.: Real-time 3D marker tracking with a WIIMOTE stereo vision system: Application to robotic throwing. In: *Robotics in Alpe-Adria-Danube Region (RAAD), 2010 IEEE 19th International Workshop on*, 2010, S. 357–362 3

[Pun99] Punskaya, Elena: *Bayesian Approaches to Multi-Sensor Data Fusion*, Cambridge University Engineering Department, Diplomarbeit, 1999 21

[PVK13] Perl, Thomas ; Venditti, Benjamin ; Kaufmann, Hannes: PS Move API: A Cross-Platform 6DoF Tracking Framework. In: *Proceedings of the*

Workshop on Off-The-Shelf Virtual Reality, 2013. – Vortrag: Workshop on Off The Shelf Virtual Reality, IEEE Virtual Reality, Orlando, FL, USA; 2013-03-16 112

[Rab79] Rabiner, Lawrence R.: On the use of symmetry in FFT computation. In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 27 (1979), jun, Nr. 3, S. 233 – 239. http://dx.doi.org/10.1109/TASSP.1979.1163235. – DOI 10.1109/TASSP.1979.1163235. – ISSN 0096–3518 24

[Sag13] Sagnac, G.: L'éther lumineux démontré par l'effet du vent relatif d'éther dans un interféromètre en rotation uniforme. In: *Comptes Rendus de l'Académie des sciences* 157 (1913) 29

[Sav98] Savage, Paul G.: Strapdown inertial navigation integration algorithm design. I: Attitude algorithms. In: *Journal of Guidance, Control and Dynamics* 21 (1998), Nr. 1, S. 19–28. http://dx.doi.org/10.2514/2.4228. – DOI 10.2514/2.4228. – ISSN 0731–5090 39

[SC09] Smith, C. ; Christensen, H.I.: Wiimote robot control using human motion models. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, S. 5509 –5515 109

[Sch04] Scherer, Torsten: *A mobile service robot for automisation of sample taking and sample management in a biotechnological pilot laboratory*, Universität Bielefeld, Dissertation, 2004 55, 64, 65

[Sch09] Scheuer, Jacob: Fiber microcoil optical gyroscope. In: *Opt. Lett.* 34 (2009), Jun, Nr. 11, 1630–1632. http://dx.doi.org/10.1364/OL.34.001630. – DOI 10.1364/OL.34.001630 31

[Sch10] Scharfe, Hanno: *Physikbasierter Simulator für Greif- und Manipulationsverfahren mit Mehrfinger-Roboterhänden*, Universität Hamburg, Diplomarbeit, 2010 22

[Shu10] Shukla, Pradyumn K.: *Levenberg-Marquardt Algorithms for Nonlinear Equations, Multi-objective Optimization, and Complementarity Problems*, Technische Universität, Fakultät Mathematik und Naturwissenschaften, Dresden, Dissertation, 2010 80

[Sin69] Singleton, R.: An algorithm for computing the mixed radix fast Fourier transform. In: *Audio and Electroacoustics, IEEE Transactions on* 17 (1969), jun, Nr. 2, S. 93 – 103. http://dx.doi.org/10.1109/TAU.1969.1162042. – DOI 10.1109/TAU.1969.1162042. – ISSN 0018–9278 24

[SK08] Siciliano, Bruno (Hrsg.) ; Khatib, Oussama (Hrsg.): *Springer Handbook of Robotics*. Springer, 2008. – ISBN 978–3–540–23957–4 20, 28, 29, 31

[Ste97] Stedman, G.E.: Ring-laser tests of fundamental physics and geophysics. In: *Reports on Progress in Physics* 60 (1997), Nr. 6 29

[TBF05] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. – ISBN 0262201623 21

[Ull07] Ullenboom, Christian: *Java ist auch eine Insel*. 6., aktualisierte und erweiterte Auflage. Bonn : Galileo Computing, 2007 `http://www.galileocomputing.de/openbook/javainsel6/` 60

[Woo07] Woodman, Oliver J.: An introduction to inertial navigation / University of Cambridge Computer Laboratory. 2007. – Forschungsbericht. – ISSN 1476–2986 35, 36, 37

[Ča04] Čapek, Karel: *R.U.R. - Rossum's Universal Robots*. Penguin Classics, 2004. – ISBN 9780141182087 11

Anhang

A

Da der Anhang zu der vorliegenden Diplomarbeit recht umfangreich geworden ist, wurde der größere Teil auf den beiliegenden Datenträger ausgelagert. Das betrifft insbesondere die Datenblätter zu den verwendeten Sensoren und den Quelltext der erstellten Software.

A.1 Vollständige Tabellen der Experimentalmesswerte

A.1.1 Accelerometer Nullwerte und ±1g - Experimente 1-6

X-Achse	Messwert	129	130	131	132	133	134	135	136	137	138
	Häufigkeit	1	92	86	148	16454	19925	309	114	93	15
Y-Achse	Messwert	124	125	126	127	128	129	130	131	132	133
	Häufigkeit	3	100	104	195	26090	10332	216	102	90	5
Z-Achse	Messwert	149	150	151	152	153	154	155	156	157	158
	Häufigkeit	17	108	127	637	34729	1272	200	115	31	1

Tabelle A.1: Messdaten und deren Häufigkeit für die Nullwertekalibrierung und ±1g Experimente (Experiment 1) Anhang zu Abschnitt 6.3.5

X-Achse	Messwert	133	134			
	Häufigkeit	12818	11793			
Y-Achse	Messwert	124	125	126	127	
	Häufigkeit	18	24408	65	120	
Z-Achse	Messwert	96	97	98	99	100
	Häufigkeit	9	1212	22753	621	16

Tabelle A.2: Messdaten und deren Häufigkeit für die Nullwertekalibrierung und ±1g Experimente (Experiment 2) Anhang zu Abschnitt 6.3.5

X-Achse	Messwert	130	131	132	133	134
	Häufigkeit	8	18338	18452	5	27
Y-Achse	Messwert	99	100			
	Häufigkeit	26735	10095			
Z-Achse	Messwert	124	125	126	127	
	Häufigkeit	26	280	34890	1634	

Tabelle A.3: Messdaten und deren Häufigkeit für die Nullwertekalibrierung und ±1g Experimente (Experiment 3) Anhang zu Abschnitt 6.3.5

X-Achse	Messwert	132	133	134
	Häufigkeit	1293	25482	116
Y-Achse	Messwert	152	153	154
	Häufigkeit	32	17697	9162
Z-Achse	Messwert	126	127	
	Häufigkeit	8419	18472	

Tabelle A.4: Messdaten und deren Häufigkeit für die Nullwertekalibrierung und ±1g Experimente (Experiment 4) Anhang zu Abschnitt 6.3.5

X-Achse	Messwert	101	102	103	104	105	106	107	108	109	110	111
	Häufigkeit	11	85	144	201	358	15181	355	198	141	8	1
Y-Achse	Messwert	122	123	124	125	126	127	128	129	130		
	Häufigkeit	72	102	150	7910	7534	468	225	174	48		
Z-Achse	Messwert	122	123	124	125	126	127	128	129	130	131	
	Häufigkeit	19	91	150	268	684	14855	311	157	146	2	

Tabelle A.5: Messdaten und deren Häufigkeit für die Nullwertekalibrierung und ±1g Experimente (Experiment 5) Anhang zu Abschnitt 6.3.5

X	Messwert	154	155	156	157	158	159	160	161	162	163	164	
	Häufigkeit	66	124	158	237	582	7948	29739	1240	199	140	53	
Y	Messwert	121	122	123	124	125	126	127	128	129	130	131	132	133
	Häufigkeit	3	101	115	164	1054	30615	7327	562	207	180	140	17	1
Z	Messwert	120	121	122	123	124	125	126	127	128	129	130	131	
	Häufigkeit	39	107	147	181	645	17394	20214	1378	194	144	42	1	

Tabelle A.6: Messdaten und deren Häufigkeit für die Nullwertekalibrierung und ±1g Experimente (Experiment 6) Anhang zu Abschnitt 6.3.5

A.1.2 Messwerte der Kalibrierungsexperimente mit der Infrarotkamera

Damit die Infrarotkamera zum Bestimmen von Entfernungen eingesetzt werden konnte, wurde die Sensorbar in einer Entfernung d_o zur Kamera aufgestellt und der Abstand d_c der Bilder ihrer Infrarot-LEDs in Pixeln gemessen. In diesem Abschnitt sind die Messwerte in Tabellen aufgeführt.

Messwerte	Experiment 1												
d_o [cm]	22	25	30	35	40	45	50	55	60	65	70	75	80
d_c [px]	1000	945	873	730	673	605	550	502	460	428	402	374	347
d_o [cm]	85	90	95	100	105	110	115	120	125	130	135	140	145
d_c [px]	326	311	291	283	265	253	241	231	222	213	206	198	193
d_o [cm]	150	155	160	165	170	175	180	185	190	195	200	210	220
d_c [px]	187	181	173	169	165	160	153	150	147	143	141	133	128

Messwerte	Experiment 1								Experiment 6			
d_o [cm]	230	240	260	280	300	320	340	360	39,1	50	70	100
d_c [px]	122	115	108	100	92	85	76	77	688	505	386	263,5

Tabelle A.7: Verhältnis Entfernung Sensorbar zur Pixeldistanz im IR-Kamera Bild (Experimente 1,6)

Messwerte	Experiment 2					Exp 3		Exp 4		Experiment 5			
d_o [cm]	320	340	360	380	400	200	400	600	800	200	400	600	800
d_c [px]	86	81	77,5	74	69	139	71	47,5	36	135	40	46	36

Tabelle A.8: Verhältnis Entfernung Sensorbar zur Pixeldistanz im IR-Kamera Bild (Experimente 2,3,4,5)

Da alle Werte und Entfernungen von Hand vermessen bzw. abgelesen wurden, kann der vorletzte Wert in Experiment 1 als menschlicher Fehler betrachtet werden. Für die graphische Darstellung wurde der Messwert (340cm;76px) aus Experiment 1 als Ausreißer bzw. Messfehler behandelt und durch den Wert (340cm;81px) aus Experiment 2 ersetzt. Dieser Wert ist auch plausibler.

Zusammenfassend kann man sagen, dass 1000 Pixel in der Kameradistanz einer Objektdistanz von 21,5cm zur Sensorbar entsprechen. Die maximale Entfernung, welche noch sichtbar war, betrug 940cm. Es war möglich auf diese Distanz die Sensorbar immer noch als ein Pixel im Kamerabild zu erkennen. Allerdings war dafür eine sehr genaue, optimale Ausrichtung von Sensorbar und Wiimote nötig. Es gelang während der Experimente mit der vorliegenden Hardware nicht, größere Entfernungen als 940cm zu messen.

Hier befinden sich die zu den Experimenten gehörenden Gyroskopmesswerte

Experiment 1						Experiment 2					
X-Achse		Y-Achse		Z-Achse		X-Achse		Y-Achse		Z-Achse	
W	H	W	H	W	H	W	H	W	H	W	H
8090	2	7845	1	7892	7	8096	1	7848	13	7891	4
8092	1	7846	5	7893	18	8097	5	7849	15	7892	12
8093	2	7847	3	7894	51	8098	4	7850	58	7893	18
8094	1	7848	19	7895	94	8099	23	7851	91	7894	79
8095	15	7849	43	7896	243	8100	54	7852	216	7895	160
8096	16	7850	116	7897	424	8101	96	7853	378	7896	323
8097	29	7851	196	7898	792	8102	216	7854	656	7897	622
8098	72	7852	380	7899	1373	8103	326	7855	1079	7898	1105
8099	157	7853	668	7900	2135	8104	697	7856	1806	7899	1716
8100	290	7854	1177	7901	3069	8105	1167	7857	2535	7900	2595
8101	488	7855	1841	7902	4103	8106	1800	7858	3818	7901	3410
8102	811	7856	2855	7903	5148	8107	2584	7859	4947	7902	4456
8103	1275	7857	4151	7904	6065	8108	3706	7860	6014	7903	5396
8104	2063	7858	5473	7905	6761	8109	4964	7861	7139	7904	5958
8105	3139	7859	7000	7906	7270	8110	6226	7862	7878	7905	6329
8106	4041	7860	8214	7907	7492	8111	7006	7863	8210	7906	6482
8107	5165	7861	9148	7908	7809	8112	7724	7864	8168	7907	6535
8108	6704	7862	9503	7909	7612	8113	8163	7865	7469	7908	6616
8109	7704	7863	9319	7910	7228	8114	8130	7866	6529	7909	6230
8110	8415	7864	8795	7911	6451	8115	7380	7867	5264	7910	5923
8111	8778	7865	7596	7912	5746	8116	6620	7868	4217	7911	5284
8112	8905	7866	6422	7913	4842	8117	5444	7869	2879	7912	4381
8113	8293	7867	4898	7914	3897	8118	4151	7870	1975	7913	3515
8114	7494	7868	3517	7915	2912	8119	3004	7871	1320	7914	2670
8115	6407	7869	2421	7916	2003	8120	2033	7872	824	7915	1865
8116	5274	7870	1426	7917	1480	8121	1293	7873	461	7916	1235
8117	3819	7871	891	7918	912	8122	795	7874	289	7917	688
8118	2818	7872	497	7919	536	8123	422	7875	117	7918	434
8119	1972	7873	270	7920	301	8124	226	7876	52	7919	235
8120	1221	7874	148	7921	179	8125	118	7877	20	7920	112
8121	771	7875	75	7922	82	8126	52	7878	16	7921	44
8122	462	7876	17	7923	41	8127	20	7879	9	7922	19
8123	256	7877	6	7924	11	8128	9	7880	2	7923	11
8124	127	7878	3	7925	10	8129	5	7881	1	7925	1
8125	57	7879	3	7927	1	8130	1			7927	2
8126	37	7880	1								
8127	9										
8128	7										
8132	1										

Tabelle A.9: Messdaten und deren Häufigkeit für die Nullwertekalibrierung der Gyroskope (Experimente 1 und 2) Anhang zu Abschnitt 6.3.5

Experiment 3						Experiment 4					
X-Achse		Y-Achse		Z-Achse		X-Achse		Y-Achse		Z-Achse	
W	H	W	H	W	H	W	H	W	H	W	H
8097	4	7847	9	7888	2	8096	3	7844	2	7889	1
8098	5	7848	8	7890	5	8097	11	7845	3	7890	2
8099	4	7849	32	7891	17	8098	13	7846	8	7891	4
8100	27	7850	53	7892	22	8099	17	7847	20	7892	23
8101	61	7851	130	7893	50	8100	37	7848	40	7893	53
8102	118	7852	236	7894	117	8101	106	7849	110	7894	149
8103	212	7853	541	7895	271	8102	226	7850	217	7895	253
8104	433	7854	858	7896	506	8103	411	7851	392	7896	528
8105	836	7855	1324	7897	865	8104	782	7852	703	7897	817
8106	1450	7856	2079	7898	1534	8105	1184	7853	1133	7898	1532
8107	2159	7857	2900	7899	2404	8106	1878	7854	1807	7899	2222
8108	2981	7858	4486	7900	3369	8107	2758	7855	2803	7900	3170
8109	4148	7859	5902	7901	4493	8108	3858	7856	3884	7901	4198
8110	5458	7860	7107	7902	5438	8109	4925	7857	5242	7902	5216
8111	6898	7861	8075	7903	6271	8110	6294	7858	6437	7903	5869
8112	7950	7862	9058	7904	6998	8111	7134	7859	7512	7904	6472
8113	8823	7863	9453	7905	7193	8112	7769	7860	8039	7905	6585
8114	9402	7864	9002	7906	7646	8113	8327	7861	8628	7906	6928
8115	8989	7865	8441	7907	7356	8114	8123	7862	8286	7907	6935
8116	8420	7866	7306	7908	7440	8115	7488	7863	7697	7908	6568
8117	7529	7867	5962	7909	6721	8116	6684	7864	6574	7909	6180
8118	6264	7868	4622	7910	6382	8117	5515	7865	5354	7910	5627
8119	4821	7869	3310	7911	5593	8118	4349	7866	3989	7911	4702
8120	3474	7870	2357	7912	4694	8119	3098	7867	2848	7912	3768
8121	2380	7871	1544	7913	3728	8120	2274	7868	2066	7913	2958
8122	1593	7872	922	7914	2718	8121	1409	7869	1259	7914	2281
8123	1038	7873	497	7915	1928	8122	904	7870	799	7915	1455
8124	585	7874	256	7916	1289	8123	526	7871	445	7916	946
8125	342	7875	127	7917	778	8124	299	7872	233	7917	618
8126	161	7876	55	7918	457	8125	190	7873	132	7918	322
8127	74	7877	25	7919	222	8126	74	7874	46	7919	192
8128	34	7878	11	7920	120	8127	45	7875	13	7920	104
8129	17	7879	5	7921	38	8128	17	7876	3	7921	46
8130	5	7880	2	7922	17	8129	9	7877	3	7922	9
				7923	8	8130	1	7878	9	7923	2
				7924	2	8132	1	7879	1	7924	4
				7925	2			7881	2		
				7927	1						

Tabelle A.10: Messdaten und deren Häufigkeit für die Nullwertekalibrierung der Gyroskope (Experimente 3 und 4) Anhang zu Abschnitt 6.3.5

Experiment 5						Experiment 6					
X-Achse		Y-Achse		Z-Achse		X-Achse		Y-Achse		Z-Achse	
W	H	W	H	W	H	W	H	W	H	W	H
8099	6	7851	1	7899	3	8094	3	7843	1	7890	7
8100	11	7854	3	7900	7	8095	2	7845	1	7891	11
8101	42	7855	1	7901	32	8096	10	7847	2	7892	24
8102	68	7856	16	7902	66	8097	28	7848	14	7893	59
8103	149	7857	29	7903	177	8098	49	7849	43	7894	133
8104	303	7858	63	7904	323	8099	114	7850	88	7895	351
8105	544	7859	139	7905	591	8100	207	7851	214	7896	584
8106	920	7860	264	7906	1019	8101	405	7852	429	7897	1055
8107	1416	7861	509	7907	1545	8102	749	7853	738	7898	1813
8108	2225	7862	831	7908	2194	8103	1284	7854	1249	7899	2693
8109	3124	7863	1299	7909	3197	8104	1941	7855	1946	7900	3840
8110	4201	7864	2131	7910	3973	8105	2925	7856	2982	7901	4997
8111	5107	7865	3113	7911	4780	8106	4122	7857	4521	7902	5950
8112	6193	7866	4118	7912	5337	8107	5631	7858	5787	7903	6615
8113	6921	7867	5211	7913	5793	8108	6984	7859	7457	7904	7355
8114	7203	7868	6325	7914	6032	8109	8346	7860	8776	7905	7656
8115	7322	7869	7115	7915	6086	8110	9164	7861	9653	7906	7820
8116	6817	7870	7526	7916	5833	8111	9770	7862	10179	7907	7766
8117	6038	7871	7711	7917	5606	8112	9690	7863	9880	7908	7743
8118	5156	7872	6991	7918	5142	8113	9243	7864	9091	7909	7455
8119	4097	7873	6317	7919	4575	8114	8001	7865	7930	7910	6541
8120	3177	7874	5248	7920	4116	8115	6517	7866	6155	7911	5417
8121	2105	7875	3920	7921	3250	8116	4898	7867	4617	7912	4517
8122	1455	7876	2880	7922	2469	8117	3773	7868	3297	7913	3366
8123	907	7877	1972	7923	1860	8118	2541	7869	2242	7914	2495
8124	577	7878	1314	7924	1179	8119	1633	7870	1434	7915	1691
8125	295	7879	775	7925	699	8120	1090	7871	784	7916	1160
8126	145	7880	423	7926	393	8121	598	7872	473	7917	663
8127	69	7881	219	7927	195	8122	327	7873	195	7918	274
8128	42	7882	119	7928	116	8123	159	7874	98	7919	173
8129	13	7883	52	7929	43	8124	78	7875	38	7920	83
8130	6	7884	16	7930	14	8125	39	7876	18	7921	16
8131	2	7885	7	7931	9	8126	15	7877	7	7922	11
8132	2			7932	3	8127	4	7878	2	7923	6
				7933	1	8128	2	7879	1	7924	2

Tabelle A.11: Messdaten und deren Häufigkeit für die Nullwertekalibrierung der Gyroskope (Experimente 5 und 6) Anhang zu Abschnitt 6.3.5

A.2 In dieser Arbeit erstellte Software

Listing A.1: Verzeichnisstruktur der Software

```
./tests/SimpleBehaviorApp.java
./tests/CalibrationTest.java
./tests/CollectWiimoteDataTest.java
./tests/LoadCalibrationDataFromFileToWiimote.java
./tests/PlotValuesToTime.java
./tests/WiimoteDataToLatexTabular.java
./tests/CalculateRegressionTest.java
./tests/CalculateVarianceAndRegressionForGyroTest.java
./tests/ConfigurationFileLoaderTest.java
./tests/TwoWiimotesTest.java
./tests/StuffTest.java
./tests/WiimoteClient.java
./tests/HammingCodeExample.java
./tests/WiimoteMoveAtoBTest.java
./tests/SimpleCylinder.java
./tests/SimpleSphere.java
./tests/SimpleBehaviorApp2.java
./tests/ArmMovedByWiimotesTest.java

./util/Logger.java
./util/RawCalibrationData.java
./util/WiimoteCalibrator.java
./util/InitialCalibrationLoader.java
./util/GyroZeroCalibrator.java
./util/CalibrationData.java
./util/ZeroAndOneGValuesCalibrator.java
./util/Calibrator.java
./util/Pose.java
./util/WiiDataProcessor.java

./experiments/InHandVsOnTableExperiment.java

./filter/LowPassFilter.java
./filter/DistributionFilter.java
./filter/KalmanFilter.java
./filter/MultimodalFilter.java
./filter/PassThroughFilter.java
./filter/DelayFilter.java
./filter/StateFilter.java
./filter/WriteDataToDiskFilter.java
./filter/Filter.java
./filter/FirstFilter.java
./filter/AverageFilter.java
./filter/PartikelFilter.java
./filter/OscillationFilter.java
./filter/CalibrationFilter.java
./filter/MedianFilter.java

./gui/WiimotePanel.java
./gui/PlotManager2.java
./gui/WiimoteBehaviorApp.java
./gui/ExtensionFileFilter.java
./gui/AppWindow.java

./comm/WiimoteProxyImplOffline.java
./comm/FileHandler.java
./comm/WiimoteCalibratorFeedbackInterface.java
./comm/WiimoteDataListener.java
./comm/WiimoteData.java
./comm/WiimoteProxy.java
./comm/WiimoteDevice2.java
./comm/WiimoteProxyAbstractClass.java
./comm/WiimoteDevice1.java

./control/TeleoperationPA10Behavior.java
./control/Collector.java
./control/Controller.java
./control/PoseComputer.java
./control/PositionBehavior.java
./control/WiimoteRecordingServer.java
./control/CollectorThread.java
./control/DefaultBehavior.java
./control/Behavior.java
```

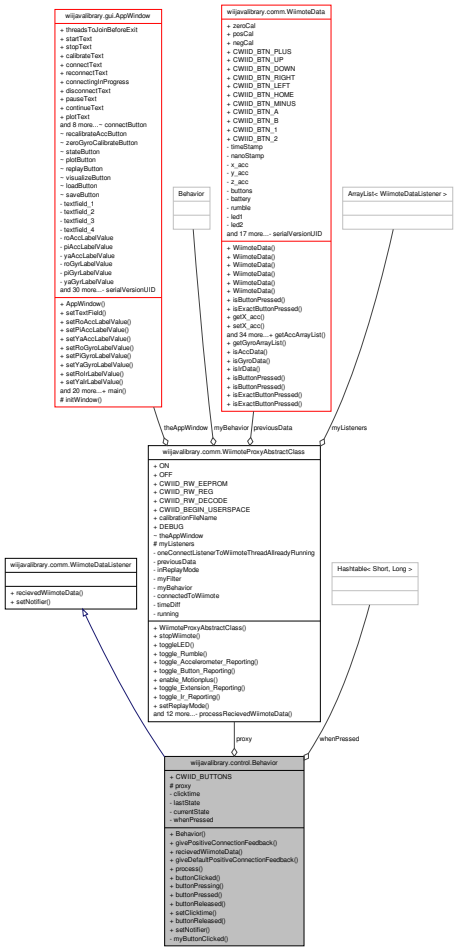


Abbildung A.1: Kollaborationsdiagramm der Behavior-Klasse

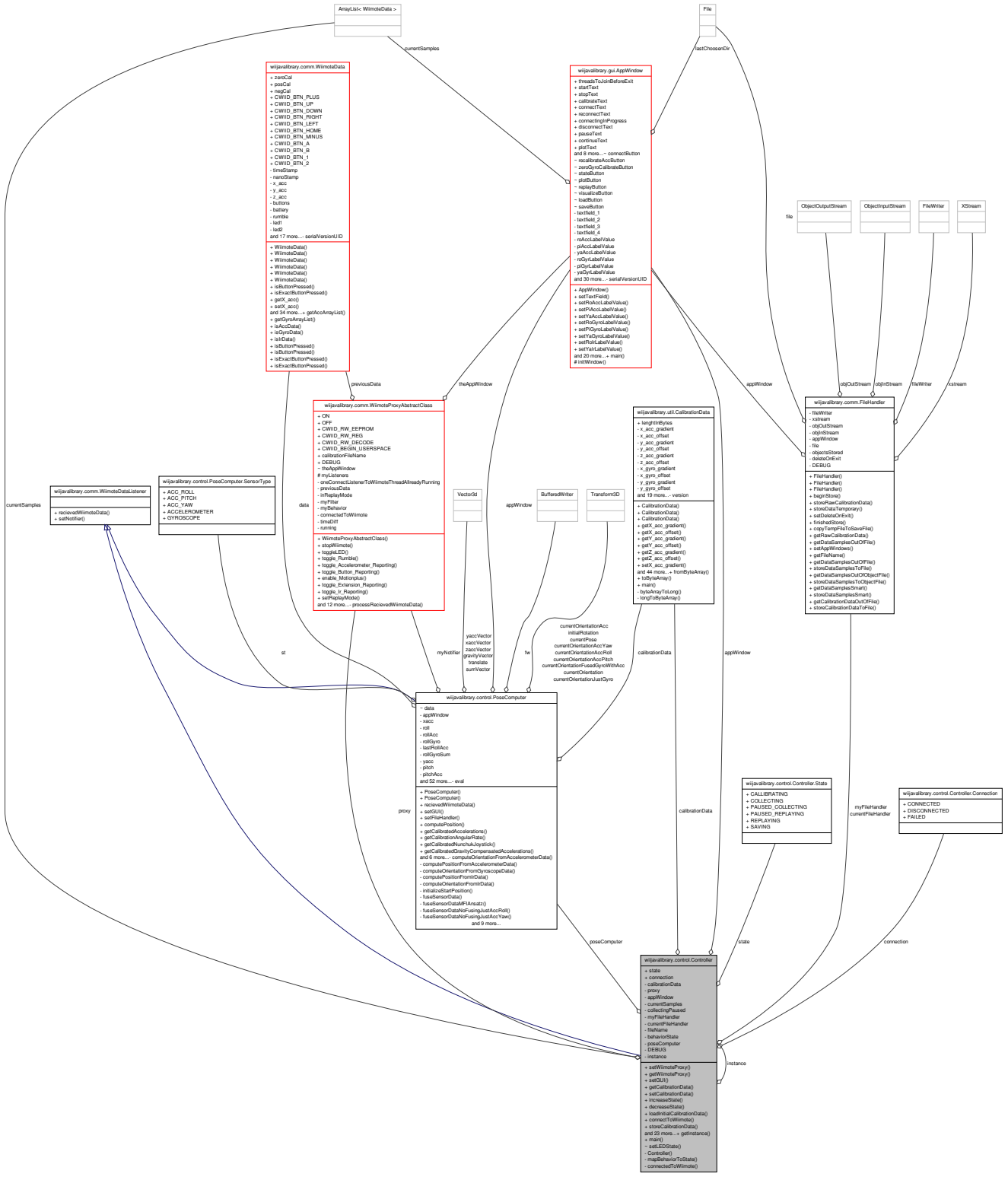


Abbildung A.2: Kollaborationsdiagramm der Contoller-Klasse

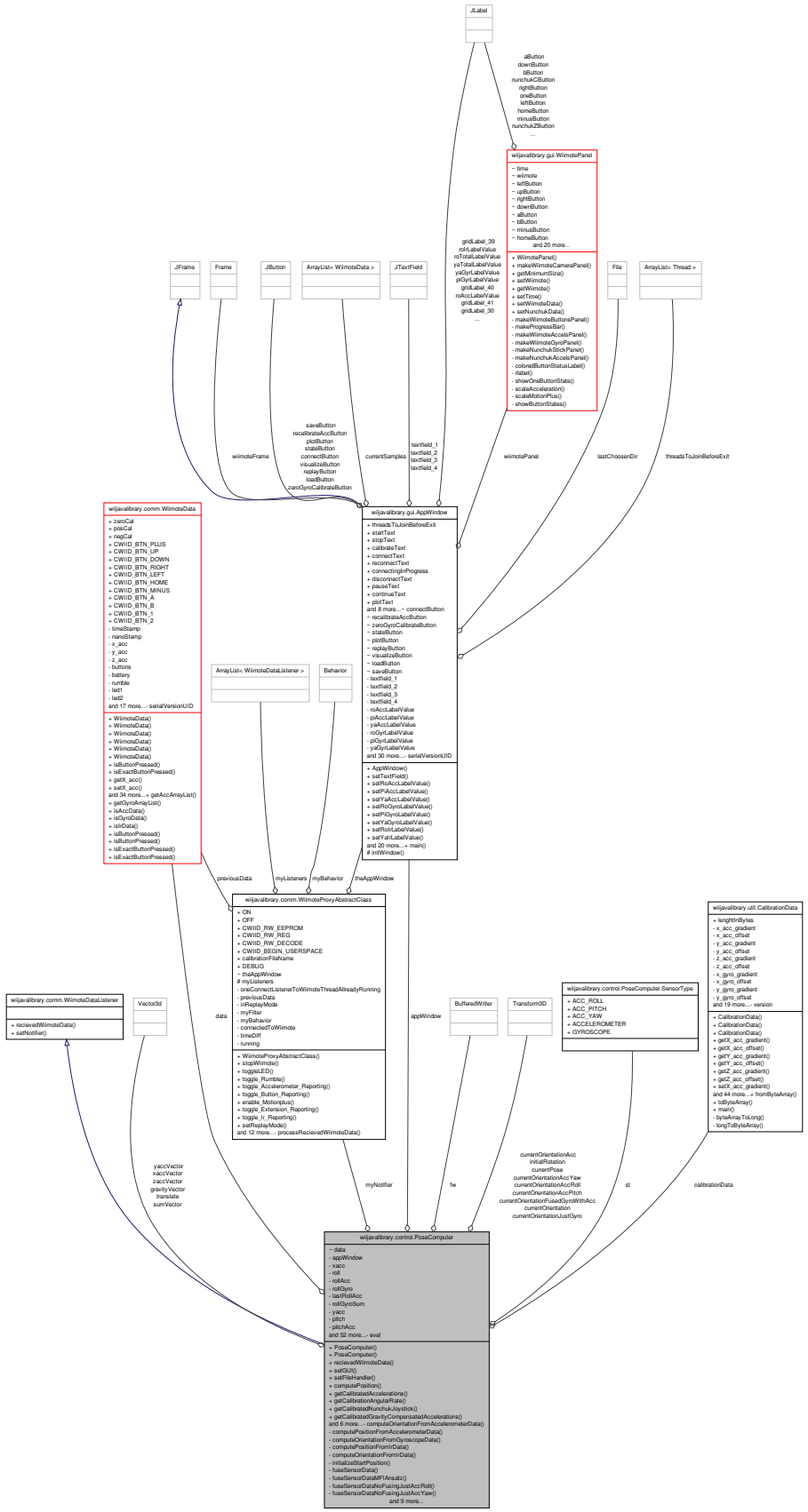


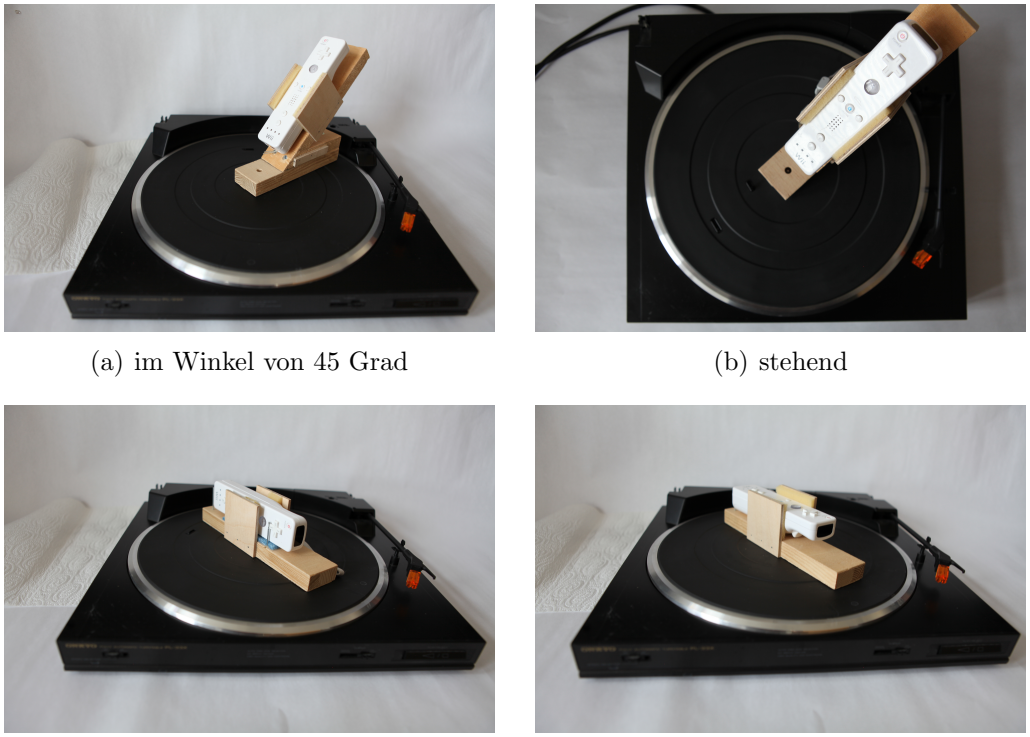
Abbildung A.3: Kollaborationsdiagramm der PoseComputer-Klasse

A.3 Datenblätter der Accelerometer- und Gyroskopchips der Wiimote, Nunchuk und MotionPlus

Die Datenblätter der lis3l02al, adxl330 IDG-600 InvenSense wurden ihrer Größe wegen aus dem Anhang entfernt und befinden sich auf dem beiliegenden Datenträger (im Verzeichnis *data_sheets* bzw. können im Internet eingesehen werden).

A.4 Bilder der Haltevorrichtungen für die Wiimote

Diese Haltevorrichtungen, gebaut durch Herrn Florstedt, wurden für die Kalibrierungsexperimente genutzt.



(a) im Winkel von 45 Grad

(b) stehend

(c) liegend, seitlich

(d) liegend

Abbildung A.4: Haltevorrichtungen auf dem Plattenspieler zur Kalibrierung der Gyroskope