Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Gesten-basierte Steuerung von Desktop-Applikationen durch Smartphones als Steuer- und Eingabegerät

Studiengang:	Bachelor of Science Informatik
Verfasser:	Kolja Bruns
E-Mail:	8Bruns@informatik.uni-hamburg.de
	Kolja-Bruns@gmx.de
Matrikel-Nummer.:	6053568

Datum: 18.03.2014

Erstgutachter: Prof. Dr.-Ing. Züllighoven

Zweitgutachter: Dipl.-Inform. Pechau

Danksagung

Ich möchte Prof. Heinz Züllighoven und Jörg Pechau danken, dass sie mir die Möglichkeit gegeben haben diese Bachelorarbeit bei ihnen zu schreiben.

Ganz besonders möchte ich Aileen und Sascha danken, die mich aufgebaut haben wann immer es mir nicht gut ging und mich daran erinnert haben, dass es immer einen Ausweg gibt.

Zusätzlich möchte ich Alex danken, der sich die Zeit genommen hat meine Arbeit noch einmal Korrektur zu lesen.

Natürlich danke ich allen Ereignissen in meinem Leben die mir bis heute widerfahren sind, denn ohne sie wäre ich nicht der, der ich jetzt bin und wäre nicht da, wo ich jetzt bin.

Inhaltsverzeichnis

1 Einleitung

Durch die technologische Weiterentwicklung im Bereich der Mensch-Computer-Interaktion stehen heutzutage Hardware und Technologien bereit, welche neue Bedienkonzepte ermöglichen, deren Implementation die Analyse und Entwicklung entsprechender Verfahren erforderlich macht.

Schon lange ist die Bedienung von Computern nicht mehr auf Tastatur oder Maus beschränkt. Touchkonzepte werden nunmehr ergänzt durch Entwicklungen wie Gestensteuerung über Kamera oder Beschleunigungssensor aufgezeichnete Bewegungen. Während die Gestensteuerung ihren Ursprung in der Verwendung in Spielkonsolen wie der Nintento WII hat, wo die Steuerung über einen mitgelieferten Controller erfolgt, sind neue Konzepte gefragt, die die Möglichkeiten dieser Technologie auch breiten Anwendergruppen erschließen können, deren Anwendungsbereiche sich auch fernab der Unterhaltungsindustrie bewegen können.

Denn der Erfolg einer Technologie geht auch mit ihrer Verbreitung einher.
Daher wird in dieser Arbeit aufgezeigt, wie mit Hilfe von handelsüblichen Smartphones ein Gestensteuerungssystem zur Interaktion mit Desktop-PCs geschaffen werden kann, ohne dabei auf spezielle Hardware zurückgreifen zu müssen.
Durch die enorme Verbreitung von entsprechenden Smartphones kann eine Plattform geschaffen werden, die einem sehr großen Benutzerkreis neue Anwendungsmöglichkeiten zur Verfügung stellt.

1.1 Zielsetzung

Ziel der Bachelorarbeit ist es einen Computer mit Hilfe von Gesten-basierter Steuerung zu bedienen. Hierbei wird eine Android-Applikation entwickelt mit deren Hilfe sich Sensordaten auslesen und speichern lassen. Mit diesen Daten sollen Gesten trainiert und wiedererkannt werden. Dazu wird ein Serverprogramm entwickelt, welches auf einem herkömmlichen Personal Computer läuft. Wenn eine Geste erkannt wurde soll der Server die entsprechenden Befehle ausführen.

Es sollen folgende Fragen beantwortet werden.

- **Lässt sich ein Smartphone als Eingabegerät für einen Computer nutzen?**
- **Welche Möglichkeiten gibt es ein Smartphone mit einem Computer zu verbinden?**
- **Welche Sensoren lassen sich nutzen, um Gesten zu erkennen?**
- **Welche Gesten lassen sich erkennen?**
- **Welche Problematik tritt bei mehreren Geräten auf, die mit dem gleichen Computer verbunden sind und die Gestensteuerung benutzen wollen?**

1.2 Aufbau der Arbeit

In Kapitel 2 wird erläutert was Gesten sind, wie diese klassifiziert werden können und wie diese zur Kommunikation benutzt werden. Danach werden einige Projekte vorgestellt in denen Methoden und Algorithmen zur Gestenerkennung verwendet wurden. Auf dieser Basis wird entschieden welche Methode für die Gestenerkennung bei der Implementierung verwendet wird.

In Kapitel 3 wird das Smartphone vorgestellt und dessen Entwicklung hin zum zunehmend alltäglichen Alltagsgegenstand beschrieben. Danach werden die Sensoren erläutert die für die Gestenerkennung genutzt werden können. Am Ende erfolgt die Entscheidung für welches Betriebssystem entwickelt wird.

In Kapitel 4 werden die Anforderung und Architektur der Client- und Serverapplikation aufgezeigt. Hierbei wird die Sensoren API und die Verbindungsmöglichkeiten des Betriebssystem vorgestellt.
Danach wird die Umsetzung und Implementierung der einzelnen Module erläutert.

In Kapitel 5 wird die Evaluierung der Programme beschrieben und am Ende ein Ausblick über mögliche Erweiterung und Verwendung der Software gegeben.

2 Gesten

2.1 Was sind Gesten

Gesten zählen zu der nonverbalen Kommunikation und werden von Kurtenbach und Hulteen definiert, als eine Bewegung des Körpers die Information enthält. Zum Beispiel ist zum Abschied winken eine Geste, aber eine Taste auf der Tastatur drücken nicht. Denn die Bewegung vom Finger zur Taste enthält keine Information, sondern es ist nur wichtig welche Taste gedrückt wurde. [(vgl. (Laurel, 1990), S. 309-317)]

Eine Geste, nach George H. Mead, ist eine Handlung eines Senders, die eine Verhaltensanpassung des Empfängers zur Folge hat [(Mead, 1973).]
Damit es nicht zu Missverständnissen kommt, ist dabei wichtig, dass Sender und Empfänger die Information der Geste gleich interpretieren. Das ist gegeben, wenn diese einen gleichen kulturellen Hintergrund oder ähnliche Erfahrungen gemacht haben. Zum Beispiel bedeutet in Albanien ein Kopfnicken „nein“, wobei in den meisten Teilen der Welt es „ja“ bedeutet [(Ludwig, 2007)].

Gesten können in unterschiedliche Arten Klassifiziert werden [(McNeill, 1992)]:

- **Ikonisch:**
  Gesten die auf ein bestimmtes Objekt referenzieren. Zum Beispiel die Form einer Person in der Luft zeichnen um sie zu beschreiben.

- **Metaphorisch:**
  Sind im Grunde wie die Ikonischen, nur beziehen sich auf abstrakte Objekte. Dabei muss das abstrakte Konzept bekannt sein.

- **Zeigen:**
  Hierbei wird mit einem Körperteil oder Gegenstand in der Hand auf ein Objekt gezeigt. Es können aber auch auf abstrakte Objekte oder Ideen gezeigt werden, indem Objekte vorgestellt werden und zum Beispiel in der Luft gezeigt werden.

- **Lexikalische:**
  Hierbei werden Gesten wie Wörter einer Sprache benutzt. Zum Beispiel das Wort „nein“ durch die Geste „Kopfnicken“.

2.2 Gestenerkennung

Damit eine Geste von einem Computer erkannt werden kann, müssen zuvor Vergleichsdaten der Geste (Prototypen) gesammelt und diese dann mit der zu klassifizierenden Aufnahme bzw. den aus ihr gewonnenen Daten verglichen werden. Eine Bewegung kann mit sehr geringer Wahrscheinlichkeit zweimal exakt identisch ausgeführt werden. Deswegen müssen Methoden entwickelt werden, die Gesten trotz abweichender Bewegungen erkennen können.
Im Folgenden Abschnitt werden einige exemplarische Verfahren aufgeführt, in denen unterschiedliche Methoden zur Gestenerfassung und -Erkennung erfolgreich angewendet werden.

2.2.1 Das Hidden Markov Model

Das Hidden Markov Model (im Folgenden HMM abgekürzt) ist ein stochastisches Model und kann durch ein 5-Tupel (HMM = {O, **Ω**, A, B, π}) beschrieben werden.

O = { o_1, o_2, o_3, …, o_N } ist ein Alphabet von Ausgabesymbolen.

Ω = {1, 2, 3,..., N} sind die Zustände des Models.

A = { a_{ij} } ist eine Matrix mit Übergangswahrscheinlichkeiten vom Zustand i zu Zustand j. $0 < i, j < N+1$

B = { $b_j(k)$ } ist eine Wahrscheinlichkeitsmatrix bei der $b_j(k)$ die Wahrscheinlichkeit angibt, dass ein bestimmtes Symbol k aus dem Ausgabealphabet ausgegeben wird, wenn in den Zustand j gewechselt wird.

π = { π_i } ist die Wahrscheinlichkeitsverteilung für den Anfangszustand i.

Bei dem HMM wird ein System durch eine Markow-Kette modelliert in welchem die Zustände verborgen (Englisch: *Hidden*) sind und nicht direkt beobachtet werden können. Jeder Zustand hat eine Wahrscheinlichkeit ein bestimmtes Ausgabesymbol zu emittieren. Die Wahrscheinlichkeit ist nur vom vorherigen Zustand abhängig. Hat man eine Sequenz von Ausgabesymbolen und ein HMM, kann die Wahrscheinlichkeit berechnet werden, dass diese Sequenz von dem HMM emittiert wurde. Das lässt sich für die Gestenerkennung nutzen, indem die Wahrscheinlichkeit berechnet wird, dass eine Sequenz von Sensordaten von einer HMM emittiert wird. Es wird pro Geste eine HMM benötigt und die HMM mit der höchsten Wahrscheinlichkeit gilt als erkannte Geste. [(Rabiner, Hidden Markov Model, 1989)]

Das Model wurde an der Universität Oldenburg benutzt, um mit Hilfe einer Wii-Fernbedienung (Wiimote) Gesten zu erkennen.[(Schlomer, Poppinga, Henze, & Boll, 2008)]
Die Wiimote (Abbildung 1) ist das Eingabegerät der Spiele-Konsole Wii die 2006 von Nintendo veröffentlicht wurde. Im Gegensatz zu herkömmlichen Eingabegeräten von Konsolen, setzte die Wiimote auf eine neuartige Möglichkeit der Spielsteuerung.

Abbildung 1: Die Wii-Fernbedienung (Wiimote)
(Schlomer, Poppinga, Henze, & Boll, 2008), S. 1

Durch eine Infrarotkamera an ihrer Vorderseite, erkennt die Wiimote ihre Position im Raum relativ zu einer Sensorleiste die auf dem Fernseher platziert wird.
Anhand von drei Beschleunigungssensoren können Bewegungen erkannt werden. Da die Sensoren konstant die Position des Controllers zur Gravitationsachse mit messen, kann die Ausrichtung der Wiimote im Koordinatenraum ermittelt werden. In anderen Projekten

mussten extra Geräte mit entsprechenden Sensoren gebaut werden, durch die Wiimote konnten sich Entwicklungskosten und Aufwand einsparen.
Über Bluetooth können die Daten an die Konsole übermittelt werden, aber es gibt auch Bibliotheken für unterschiedliche Programmiersprachen um die Daten auch auf den Computer zu senden.
Ein Beschleunigungssensor sendet viele Daten die sich nur im geringen Maße unterscheiden. Um den Rechenaufwand des HMM möglichst gering zu halten, werden die Sensordaten vorher quantifiziert. Es wurde mit Hilfe des k-mean Algorithmus eine Vektorquantifizierung durchgeführt.
Das Ziel des k-mean Algorithmus ist, dass die Daten in k unterschiedliche Gruppen unterteilt werden die sich innerhalb der Gruppe möglichst ähnlich sind.
Der k-mean Algorithmus funktioniert so, dass im ersten Schritt k Vektoren ausgesucht werden die als Startzentrum gelten. Danach werden die Vektoren der Rohdaten dem Zentrum zugewiesen dem sie am nächsten sind, das kann zum Beispiel durch den euklidischen Abstand berechnet werden. Nun wird für jede Gruppe das neue Zentrum berechnet, so dass von allen Punkten der möglichst kleinste Abstand gilt. Diese Schritte werden solange iterativ durchlaufen, bis keine Änderung der Zentren mehr stattfindet oder die neue Zuordnung mit einer Zuordnung aus einem vorherigen Iterationsschritt übereinstimmt. In Abbildung 2 sind die ersten vier Schritte des Algorithmus im 2-Dimensionalem Raum mit k = 3 aufgezeigt. Die hohlen Symbole sind die jeweiligen Zentren

Um das HMM für jede Geste zu trainieren wurde der Baum-Welch-Algorithmus benutzt [((Rabiner & Juang, An Introduction to Hidden Markov Models, 1986) S. 11)]. Hierfür benötigt man mehrere Sequenzen seiner Geste auf die das HMM trainiert werden soll. Die anfänglichen Übergangswahrscheinlichkeiten des HMM sind hierbei beliebig und werden bei jedem Iterationsschritt neu berechnet. Bei jedem Schritt wird ermittelt wie oft Übergänge und Ausgabesymbole benutzt wurden und häufiger benutze bekommen eine höhere und weniger benutze eine niedrigere Wahrscheinlichkeit. Diese Schritte werden solange wiederholt bis sich nichts bzw. kaum noch etwas ändert.

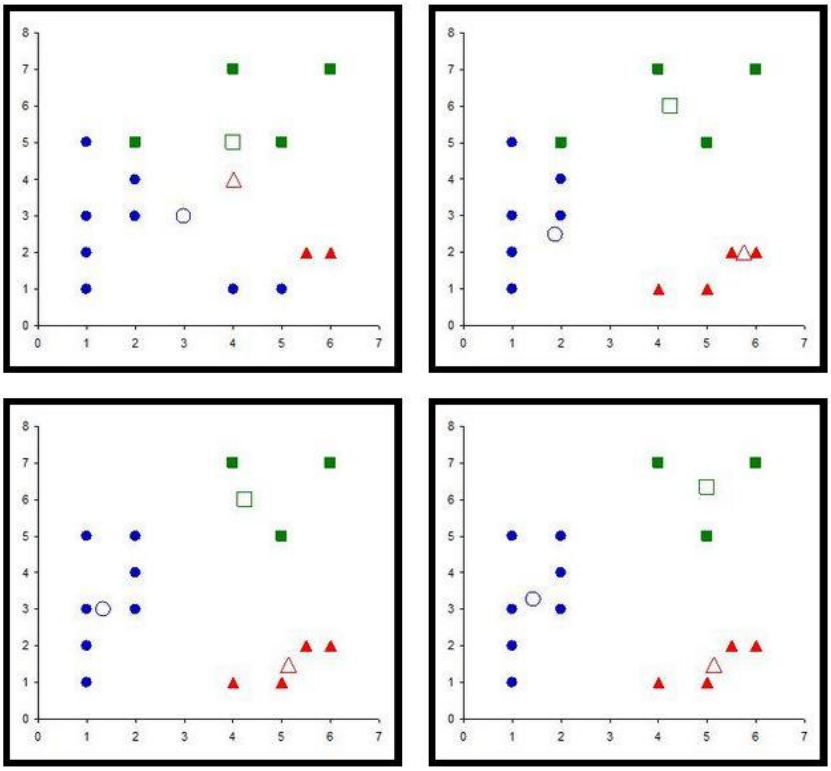

Abbildung 2: Ein Beispiel der ersten vier Schritte für einen k-mean Algorithmus. (Hammerstein, Ellßel, Tröbs, Schrottenloher, Zach, & Hupp, 2009) Seite 1.

Bei der Evaluierung wurden sechs Probanden und fünf unterschiedliche Gesten (Abbildung 3) getestet. Die Gesten sind ein Viereck, ein Kreis, eine Drehung um 90°, ein Z und eine Simulierung eines Aufschlags beim Tennis. Bei der Tennis Geste wird die Wiimote nach oben gehalten und schnell in einer gebogenen Bewegung nach unten bewegt.

Durchschnittlich wurden Tennis und Z (mit 94,5% und 94,3%) am besten und die Drehung mit 84,3% am schlechtesten erkannt (Abbildung 3).
Mit einer Wahrscheinlichkeit zwischen 84% und 93,4% wurde bei den Probanden eine Geste erkannt (Abbildung 4).

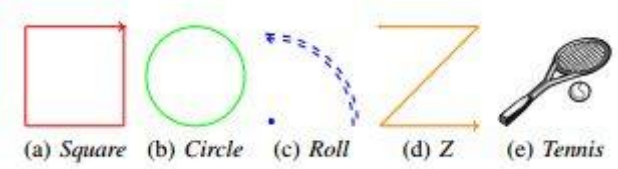

Abbildung 3: Die unterschiedlichen Gesten. (a) ein Viereck, (b) ein Kreis, (c) eine Drehung um 90°, (d) ein Z und (c) ein Tennis Simulierung. ((Schlomer, Poppinga, Henze, & Boll, 2008), S.2)

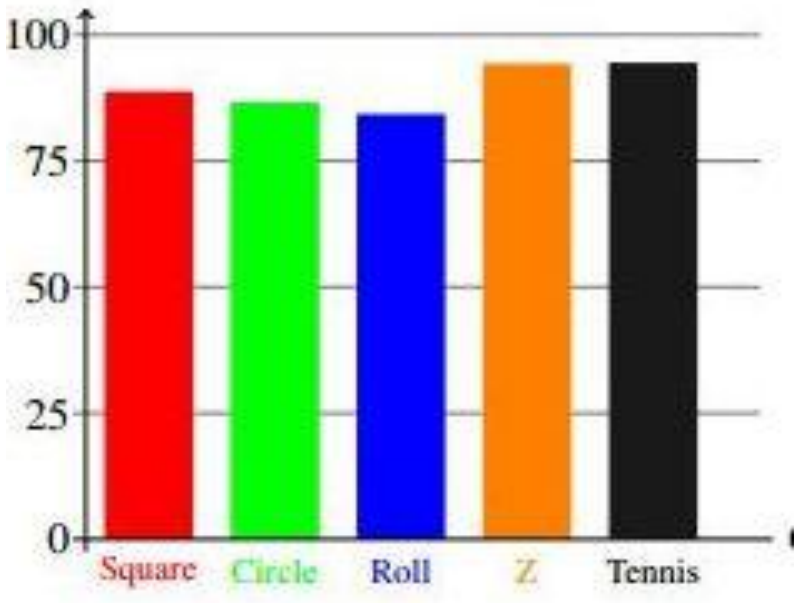

Abbildung 4: Die durchschnittliche Erkennungsrate der fünf Gesten. Viereck = 88,8%, Kreis = 86,6%, Drehung = 84,3%, Z = 94,3% und Tennis = 94,5%.((Schlomer, Poppinga, Henze, & Boll, 2008), S. 4)

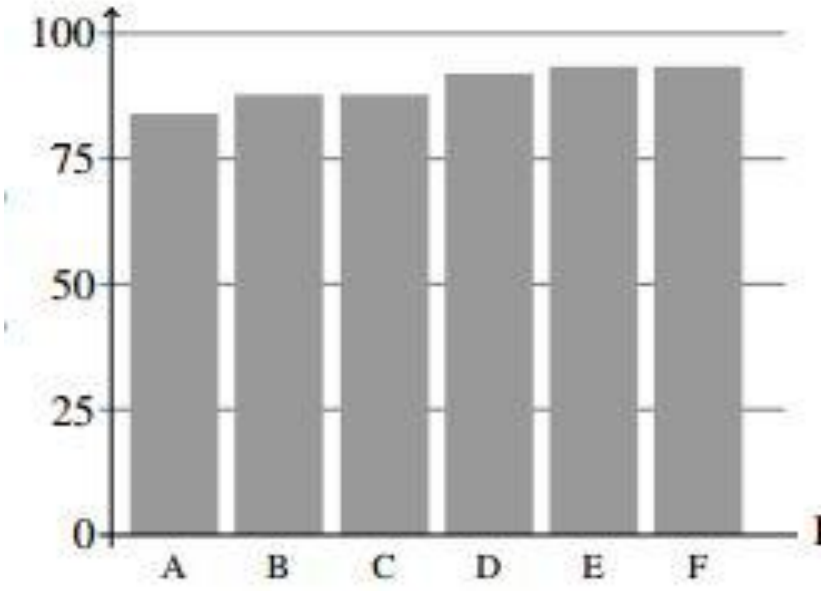

Abbildung 5: Die durchschnittlichen Erkennungsraten der Probanden. A = 84,0%, B = 87.8%, C = 87,7%, D = 92,0%, E = 93,4 und F = 93,4%((Schlomer, Poppinga, Henze, & Boll, 2008), S 4)

2.2.2 Linear Time Warping & Dynamic Time Warping

Die Xwand (Abbildung 6)ist eine Fernbedienung die mit Sensoren ausgestattet ist. Diese wurde von Daniel Wilson und Andy Wilson entwickelt, um alltägliche Elektrogeräte durch natürliche Gesten zu bedienen. [(Wilson & Wilson, Gesture Recognition Using the XWand, 2004)]

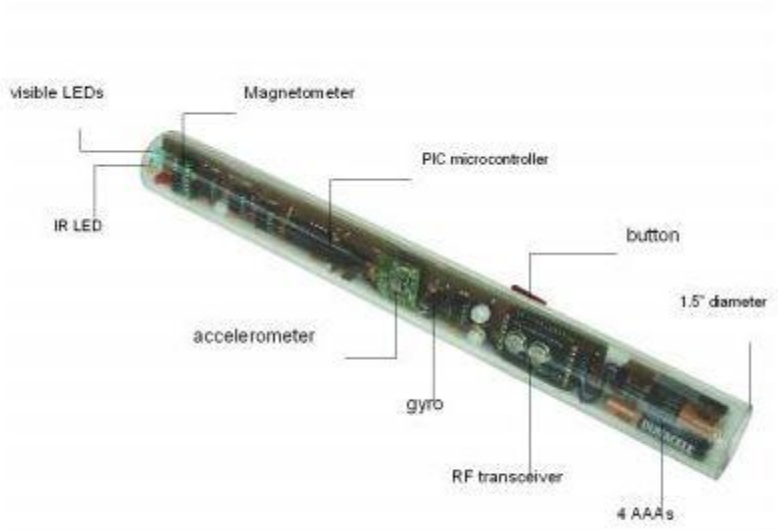

Abbildung 6: Die XWand und ihre Hardware (Wilson & Wilson, Gesture Recognition Using the XWand, 2004), S.3

Ausgestattet ist sie mit einem 2-Achsen Beschleunigungssensor. Wie bei der Wiimote wird auch hier die Gravitation mit gemessen. Er ist so angebracht, dass man kippen und drehen erkennen kann. Mittels eines 3-Achsen Magnetometer und 1-Achsen Gyroskope wird die momentane Ausrichtung festgestellt.

Um eine Geste zu erkennen muss ein an der Seite angebrachter Knopf gehalten werden während man die Bewegung der Geste vollführt. Nachdem der Knopf losgelassen wurde, werden die Daten über einen *RF transceiver* an eine Basisstation gesendet, bei der diese ausgewertet werden.

In dem Projekt wurden unterschiedliche Algorithmen verwendet, die auch schon bei Spracherkennung zum Einsatz gekommen sind.

„Linear Time Warping“-Algoritmus

In einer Trainingsphase werden Sensordaten für mehrere Prototypen einer Geste gesammelt und als Sequenz gespeichert. Wenn eine Geste erkannt werden soll, wird die aufgenommene Geste mit den Prototypen verglichen.
Die Bewegung des Benutzers, der die Geste vollführt, unterscheidet sich naturgemäß jedes Mal in ihrer Dauer und Intensität. Der Algorithmus vergleicht nun dieses Muster mit den Prototypen, die in Relation zum Eingabemuster des Users in Dauer und Amplitude gestreckt bzw. gestaucht sein können.
Der am besten passende Prototyp wird mittels der quadratischen euklidischen Distanz ermittelt. Der Prototyp mit der geringsten Distanz zum Eingabemuster gilt dann als erkannte Geste.

„Dynamic Time Warping“-Algoritmus

Der „Dynamic Time Warping“-Algorithmus hat den gleichen Ansatz wie der „Linear Time Warping“-Algoritmus. Nur berücksichtigt er bei Streckung bzw. Stauchung der Prototypen, dass unterschiedliche Teile der Geste, aber nicht die gesamte Geste, unterschiedliche Geschwindigkeiten haben. Man könnte eine Geste langsam beginnen und dann bei ihrer Durchführung die Bewegung zunehmend beschleunigen und schneller durchführen, und dennoch wäre sie als dieselbe Geste zu klassifizieren, wie wenn die Gestendurchführung mit einer konstanten Geschwindigkeit durchgeführt worden wäre. Würde aber diese Geste im gesamten gestaucht werden, so würden auch die Teile die die richtige Geschwindigkeit haben gestaucht und würden somit die Erkennung verhindern.
Dieses Problem kann mit der Dynamischen Programmierung gelöst werden.
[(Culjat, 1999)]

Hidden Markov Model

Auch in diesem Projekt wurde das HMM (Kapitel 2.2.1) benutzt, dafür wurde das HTK Toolkit Version 3.0 verwendet [(HTK Toolkit, 2009)].

2.2.3 Atomic Gesture Recognition Algorithmus

Am MIT Media Laboratory der Universität von Massachusetts haben Ari Y. Benbasat und Joseph A. Paradiso zusammen an einer anderen Lösung gearbeitet. Sie haben atomare Gesten definiert, die sich nicht weiter unterteilen lassen. [(Benbasat & Paradiso, 2002)]
Es konnten zwei atomare Gesten erkannt werden, die geradlinige und die hin-und-zurück Bewegung.
Bei der geradlinigen Bewegung gibt es zuerst eine Beschleunigung und dann eine Bremsung. In Abbildung 7(Links) erkennt man die Beschleunigung und Bremsung an ihren zwei Höchstwerten. Die hin-und-zurück Bewegung (Abbildung 7 Rechts) hat drei Höchstwerte. Hier erfolgt zusätzlich nach der ersten Abbremsung eine weitere Beschleunigung.

Um eine Geste zu erkennen, muss die Aufnahme nach der Anzahl von Peaks und ihren Integralen untersuchen. Dadurch und unter Berücksichtigung der Länge können komplexe Gesten definiert und erkannt werden.

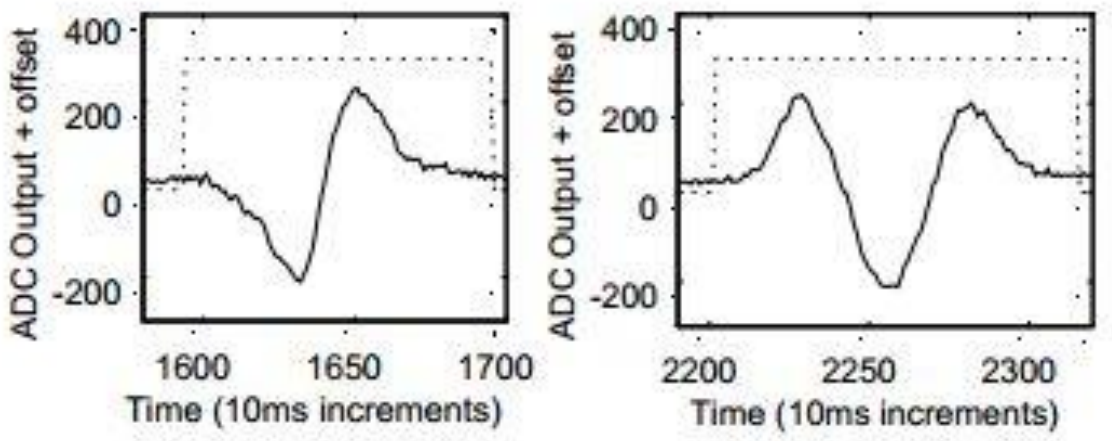

Abbildung 7: Links die geradlinige Bewegung und Rechts die Hin-und-Zurück.
((Benbasat & Paradiso, 2002), Kapitel 5.1)

3 Smartphone

Ein Smartphone ist ein „Mobiltelefon mit erweitertem Funktionsumfang. Dazu zählen neben der Telefonie und Short Message Service (SMS) üblicherweise Zusatzdienste wie Electronic Mail (E-Mail), World Wide Web (WWW), Terminkalender, Navigation sowie Aufnahme und Wiedergabe audiovisueller Inhalte. Auf Smartphones laufen gegenüber herkömmlichen Mobiltelefonen komplexere Betriebssysteme wie etwa Symbian OS, Blackberry OS oder das iPhone OS. Die hierdurch geschaffene Möglichkeit zur Installation weiterer Applikationen durch den Endnutzer verleiht Smartphones einen erweiterbaren und individualisierbaren Funktionsumfang. [vgl. (Gabler Wirtschaftslexikon, 2011)]
Im Weiteren, wenn von Mobiltelefonen die Rede ist, sind keine Smartphones gemeint.

An Abbildung 8 erkennt man, dass seit 2011 deutlich mehr Smartphones als Mobiltelefone verkauft werden. In Abbildung 9 sind die Nutzer von Smartphones in Deutschland ab 13 Jahren angegeben. Auch hier erkennt man einen deutlichen Anstieg. Das Smartphone wird somit immer mehr zu einem Alltagsgegenstand und könnte somit als immer vorhandene Universal-Fernbedienung dienen.

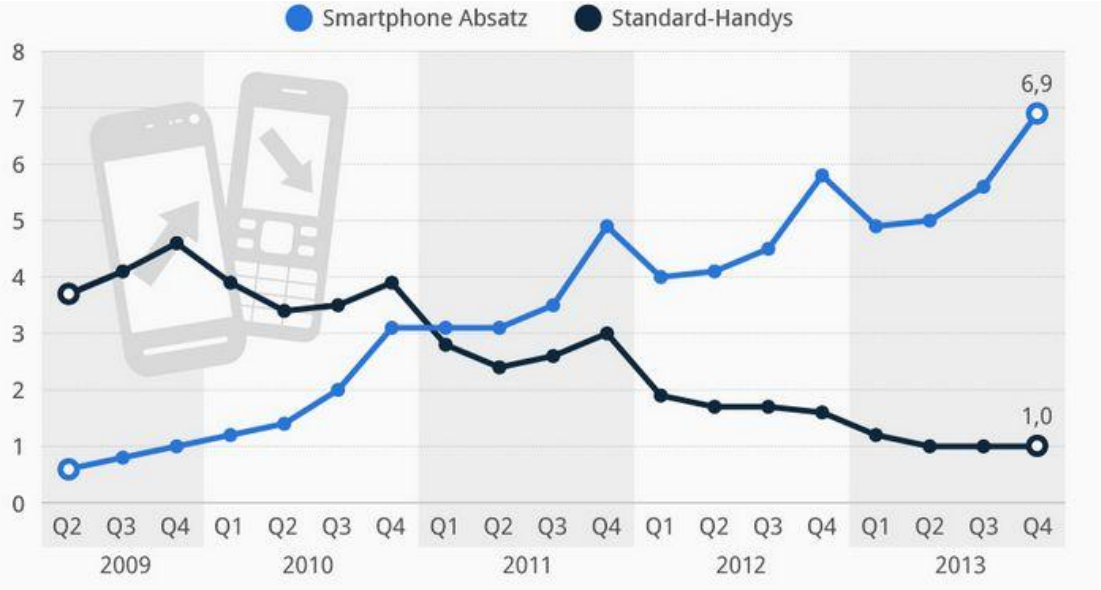

Abbildung 8: Die Absatzzahlen von Smartphone und Mobiltelefone im Vergleich zwischen den Jahren 2009 und 2013 in Deutschland (in Millionen). (Brandt, Statista, 2013)

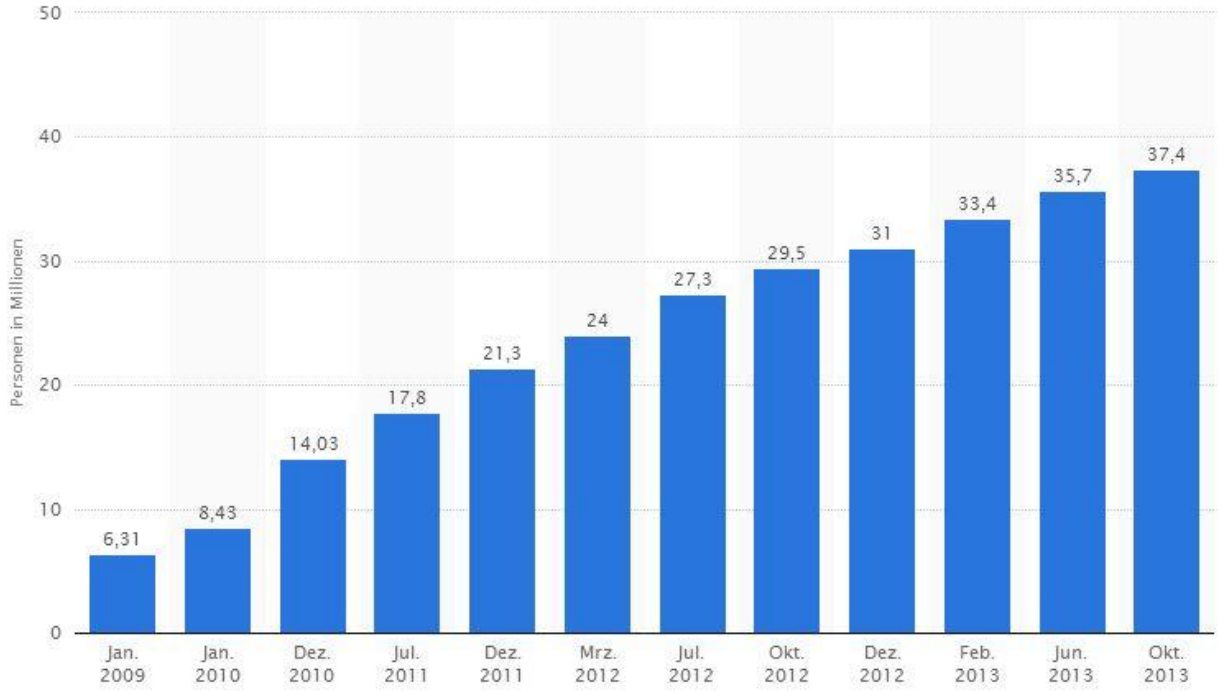

Abbildung 9: Die Anzahl der Smartphone-Besitzer in Deutschland (in Millionen) (Statiska.com, 2013)

3.1 Sensoren in Smartphones

3.1.1 Beschleunigungssensoren

In einem Smartphone sind meistens drei Beschleunigungssensoren eingebaut die so ausgerichtet sind, dass man in jede Richtung die Beschleunigung messen kann.
Das System kann mit diesen Sensoren erkennen, ob sich das Smartphone gedreht hat und kann dem entsprechend die Bildschirmdarstellung anpassen.

Die Applikation *Free Pedometer* benutzt den Beschleunigungssensor um einen Schrittzähler zu simulieren. [(Free Pedometer & Step Counter, 2013)]
In dem Spiel *Los, Ziege, Los! w*ird der Beschleunigungssensor dazu genutzt, zuerkennen wenn das Smartphone zur Seite geneigt wird. Dadurch wird die Spielfigur in die entsprechende Richtung gelenkt.[(Los, Ziege, Los!, 2013)]

3.1.2 Gyroskop

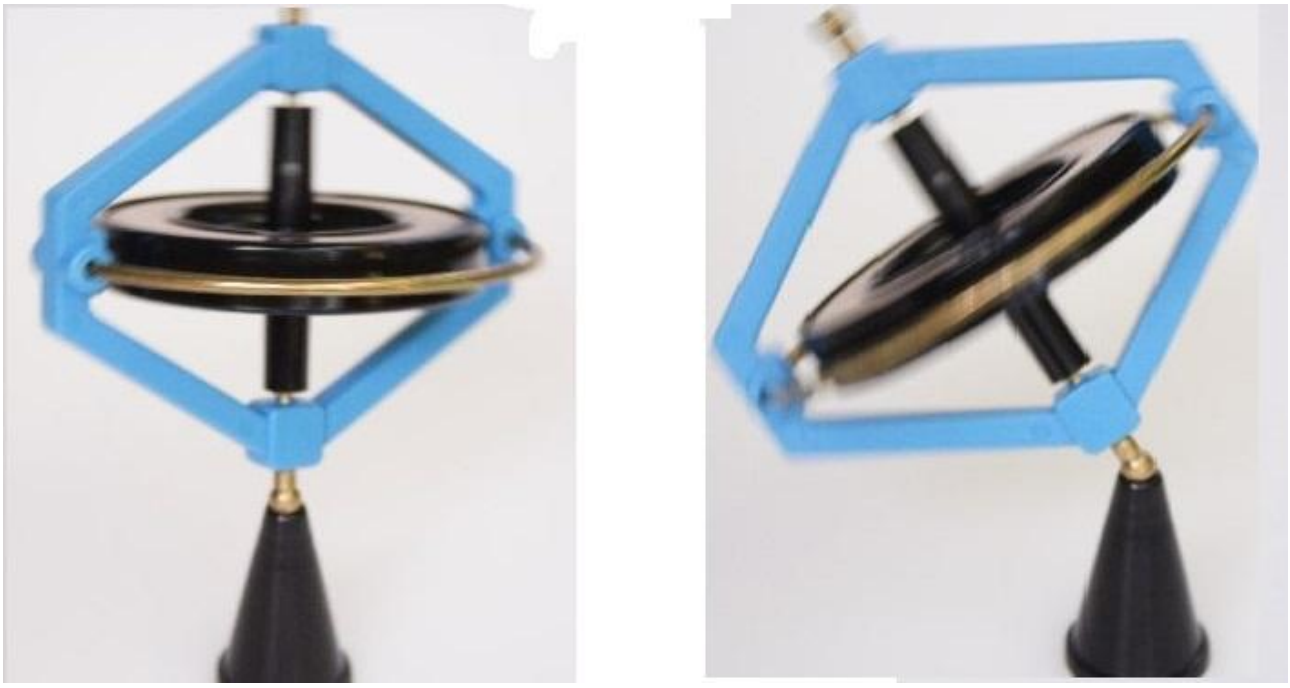

Abbildung 10: Ein Gyroskop. Links balanciert es senkrecht auf einem Ständer und Rechts bleibt es trotz Neigung an seiner Position. (Gyroskop Abbildung, 2013)

Ein Gyroskop ist eine schnell rotierende Scheibe in einem beweglichen Lager.
Durch die Drehimpulserhaltung bleibt das Gyroskop in seiner Lage im Raum.
Das bedeutet, wenn eine äußere Kraft versucht die Achse des Gyroskope zu neigen, entsteht ein Drehmoment durch den das Gyroskop sich entgegengesetzt der äußeren Kraft neigt, um den Gesamtimpuls zu bewahren (Abbildung 10)

Gyroskope wurden viel in der Luftfahrt eingesetzt, um die Position des Flugzeugs in der Luft zu bestimmen. In einem Smartphone ist ein sogenannter *vibrating structure Gyroscop* eingebaut, in dem anstatt eines rotierenden Kreisels eine vibrierende Scheibe im Mikrochip eingebaut ist. [(Apostolyuk, 2006)]

So wie bei einem Flugzeug lässt sich auch bei einem Smartphone die Rotation um die drei Achsen bestimmen. So kann bestimmt werden in welcher Ausrichtung sich das Smartphone während einer Bewegung befindet.

3.1.3 Magnetometer

Sollten keine anderen Magnetfelder den Magnetometer beeinflussen, lässt sich mit Hilfe von ihm das Erdmagnetfeld messen. Das kann dazu genutzt werden eine Kompass-Funktion in Applikationen zu integrieren. Wie bei der Xwand(Kapitel 2.2.2) kann der Magnetometer helfen die Orientierung im Raum, in Relation zum Erdmagnetfeld, festzustellen.

3.2 Betriebssysteme

Es gibt unterschiedliche Betriebssysteme für Smartphones. In Abbildung 11 wird sichtbar, dass das Android Betriebssystem seit Ende 2010 gemessen an seiner Verbreitung auf Smartphones zum Marktführer aufgestiegen ist, und dessen Verbreitung seitdem ungebrochen weiter ansteigt. Anfang 2013 waren 75% der verkauften Smartphones mit dem Betriebssystem Android.

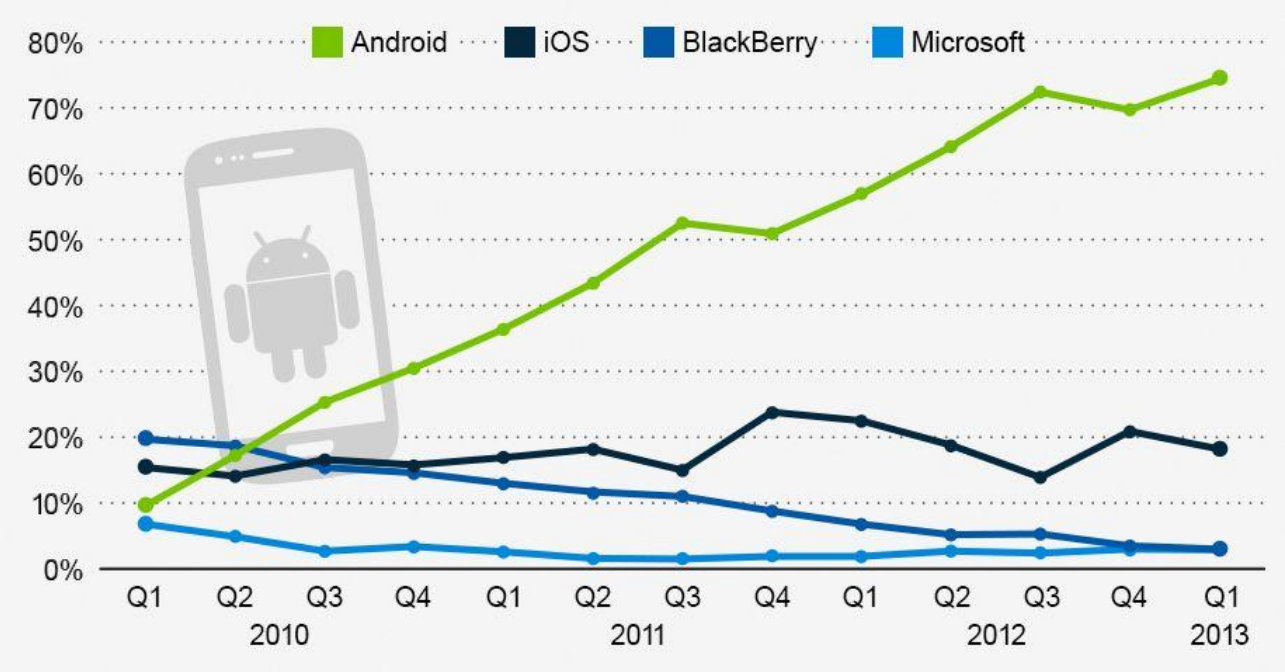

Abbildung 11: Marktanteile der Betriebssysteme (Brandt, Statista.com, 2013)

Der prozentuale Anteil von Apple's iOS unter den Betriebssystemen ist im gleichen Zeitraum fast auf demselben Niveau geblieben, circa 20%. Die anderen Betriebssysteme

liegen bei circa 2-5% und werden im Folgenden der Betrachtung entzogen.

Als Grund für die weite Verbreitung von Android gilt im Allgemeinen, dass es als quelloffenes und lizenzkostenfreies Betriebssystem bei den Smartphone Herstellern einfach verwendet und aufgrund seiner Architektur einfach an deren jeweilige Bedürfnisse angepasst werden kann. Hersteller ersparen sich dadurch Entwicklungskosten für ein eigenes Betriebssystem und müssen nur noch die Gerätetreiber an Android anpassen. Android wird hauptsächlich von der Open Handset Alliance entwickelt. Diese ist ein Zusammenschluss von 84 unterschiedlichen Unternehmen und wurde 2007 von Google und 33 Partnern gegründet [(Open Handset Alliance)]. Android ist unter der *Apache v2 open source* Lizenz veröffentlicht [(Apache License)]. Diese Lizenz erlaubt es Smartphone Hersteller Android Funktionen hinzuzufügen oder zu entfernen. Wenn eine neue Android-Version veröffentlicht wird, müssen Smartphone Hersteller ihre Anpassungen an die neue Version anpassen. Dadurch ergibt sich das Problem der Fragmentierung (Abbildung 12) da alte Geräte nicht durch Updates unterstützt werden, bleiben diese auf einer alten Android-Version.

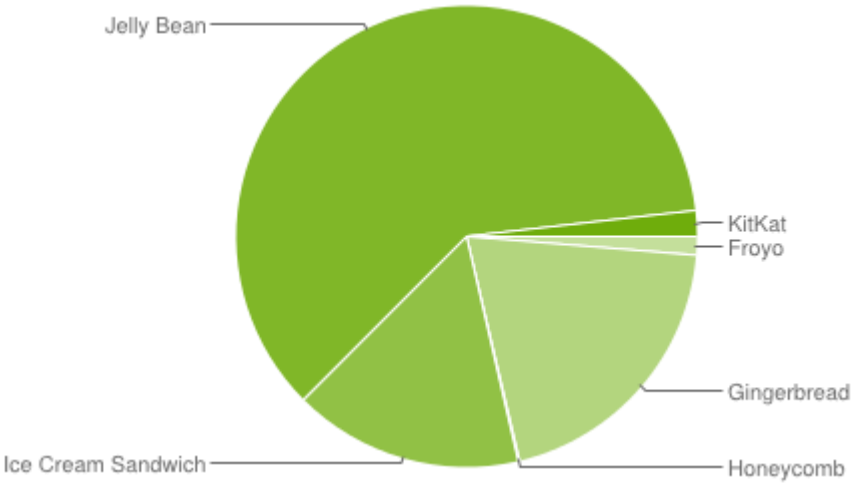

Abbildung 12: Verteilung der Android Versionen Jelly-Bean: 60,7%, Ice Cream Sandwich 16,1%, Gingerbread 20,0%, KitKat 1,8% (dashboard, 2014)

Entwickler für Applikationen sollten für die Version entwickeln, bei der die gewünschte Funktionalität enthalten ist, aber die möglichst niedrig ist, damit die Applikation auf

möglichst vielen Geräten funktioniert. Es kann aber auch auf *Support Librarys* zurückgegriffen werden, die Funktionalität von höheren Android Versionen zur Verfügung stellen.
Zusätzlich haben die vielen Geräte unterschiedliche Auflösungen und Pixeldichten [(dashboard, 2014)], das erhöht den Aufwand für Entwickler ihre Applikation universell für alle Geräte zu programmieren. Zum Beispiel hat das Samsung Galaxy S4 eine Bildschirmdiagonale von 5,0 Zoll mit einer Auflösung von 1920 x 1080 Pixeln und somit eine Pixeldichte von 441 *ppi*[1]. Im Vergleich hat das Samsung Galaxy Pocket eine Auflösung von 240 x 320 Pixeln und mit einer Bildschirmdiagonalen von 2,8 Zoll eine Pixeldichte von 143 ppi.

Das Betriebssystem iOS von Apple wird nur auf Geräten benutzt die von Apple hergestellt wurden. Neue Versionen können somit leichter und schneller auf den Geräten installiert werden, aber auch hier werden zu alte Geräte nicht mehr unterstützt.

Im Rahmen dieser Arbeit wurde die Entscheidung für ein Betriebssystem zu Gunsten von Android getroffen, da die Verbreitung, Anzahl der Anwender und somit die Erreichbarkeit von Usern sich als erheblich größer darstellt.

[1] Pixel pro Zoll (Englisch: pixel per inch)

4 Anforderungen, Design und Implementierung

Für die Aufgabe müssen zwei Programme geschrieben werden. Ein Client der auf einem Smartphone läuft und ein Serverprogramm auf einem Computer zu dem der Client sich verbinden kann. Es muss möglich sein, dass

- Sensordaten aufgezeichnet und in passender Form gespeichert werden.
- Gesten trainiert und wiedererkannt werden.
- der Client sich mit dem Server verbinden und Daten hin und her schicken kann
- der Client vom Server eindeutig identifiziert werden kann.
- der Computer auf die erkannten Gesten entsprechend reagiert.
- die trainierten Gesten so zu speichern und laden, dass sie beim Neustart ohne Training erkannt werden können.

Der Server ist nach einem MVP-Muster entwickelt (Osmann, 2012). Das MVP-Muster basiert auf drei Komponenten, dem Modell, der Ansicht (English: *View)* und dem Präsentator.

Das Modell enthält die gesamten Daten und wie diese manipuliert werden können, aber steuert sich nicht selber. Die Ansicht ist für die Darstellung der Daten und die Eingabe des Benutzers zuständig.
Der Präsentator ist das Bindeglied zwischen Modell und Ansicht. Die Eingaben werden von der Ansicht an den Präsentator übermittelt und von diesem weiterverarbeitet, indem das Modell entsprechend manipuliert wird. Die Geschäftslogik kann hierbei im Präsentator oder im Modell selber liegen. Die Validierung von einfachen Eingaben, wie Textformatierungen, können im View realisiert werden. Der Präsentator kann die Benutzerinteraktionen einschränken. So kann ein normaler Benutzer keine Aktionen benutzen die nur ein Mitarbeiter benutzen kann. Änderungen von Daten werden dem Präsentator mitgeteilt, der entsprechend die Ansicht oder das Modell aktualisiert. Dadurch, dass Modell und Ansicht nicht direkt miteinander in Berührung kommen, können diese ausgetauscht werden ohne das jeweils andere Element neu entwickeln zu müssen. Dasselbe Modell kann für unterschiedliche Anwendungen benutzt werden.

Der Server ist in Java geschrieben, läuft auf einem Computer und ist in fünf unterschiedliche Pakete aufgeteilt (Abbildung 13).

1. **GestureManagement**

Das Paket verwaltet alles was mit den Gesten zu tun hat. Es ist dafür zuständig die Gesten zu trainieren, erkennen und zu speichern. Im MVP-Muster entspricht dies dem Modell.

2. **GUI**

Das Paket entspricht der Ansicht.

3. **Controller**

Der Controller entspricht dem Präsentator und verbindet die einzelnen Komponenten. Durch das Beobachter-Muster werden in den Komponenten Ereignisse(Englisch: *events)* an den Controller gesendet und diese entsprechend behandelt. Im Weiteren wird von dem Eventsystem gesprochen wenn Ereignisse für den Controller ausgelöst werden.

4. **Network**

Die Behandlung der Netzwerkaktivität wird in einem separaten Paket ausgelagert. Das hat den Vorteil, dass unterschiedliche Möglichkeiten der Datenübertragung genutzt werden können, ohne andere Bereiche im System zu ändern.

5. **ComputerInput**

Nachdem die Geste erkannt wurde, wird in diesem Paket die jeweils passende Aktion ausgelöst. Auch hier kann die Art und Weise, wie der Computer auf die Gesten reagiert, verändert werden, ohne dass andere Bereiche geändert werden müssen.

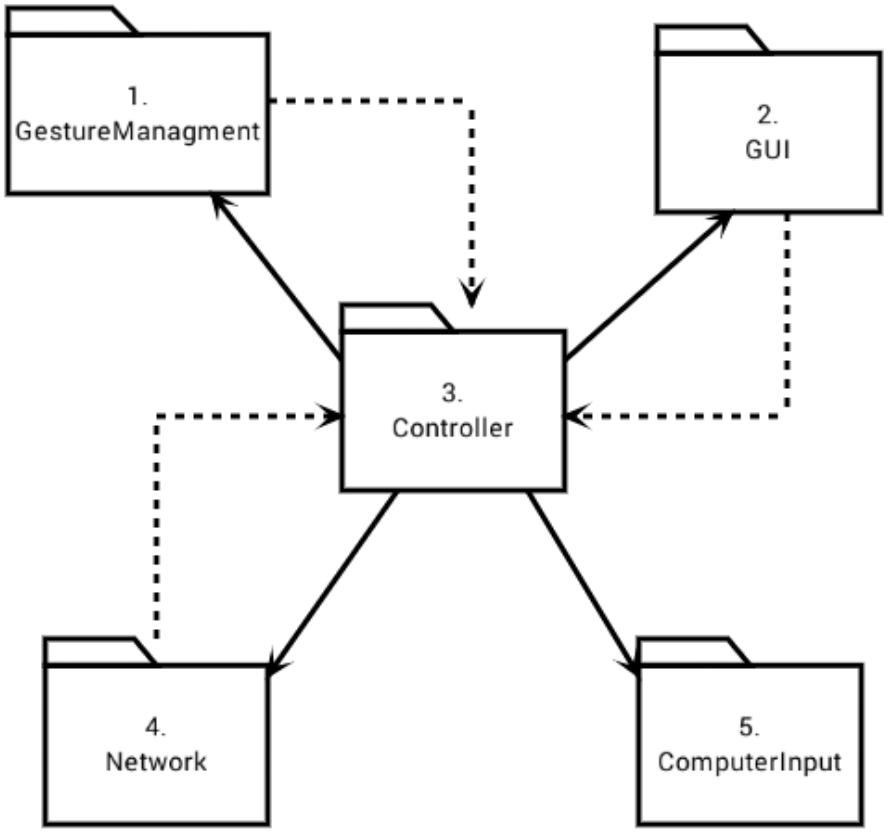

Abbildung 13: Das Paketdiagramm für den Server.

Der Client ist eine Android-Applikation und ist in vier Pakete eingeteilt (Abbildung 14).

1. **GestureManagement**

Hier werden die Daten von den Sensoren eingelesen und in ein passendes Format gebracht.

2. **GUI**

Das Paket übernimmt die Funktion der Ansicht.

Anders als bei einem Java Programm auf einem Computer ist der Eintrittspunkt nicht die **main()**[2]-Methode, sondern die Methode **onCreate()** in der Klasse **Activity** [(Acitivitys, 2014)]. Die **Activity** ist für die Darstellungen zuständig und verwaltet seinen Lebens-Zyklus. Das heißt die **Activity** kann sich im wesentlichen in drei Zuständen befinden,

[2] Methoden und Klassen werden, zur besseren Erkennbarkeit, in Fett und in Lila Farbe dargestellt.

resumed, *paused* und *stopped*. Wenn die **Activity** im Vordergrund ist, ist sie im Zustand *resumed* und hat den Fokus des Benutzers. Im Zustand *paused* ist die **Activity** transparent oder nur teilweise Sichtbar und eine andere **Activity** hat den Fokus. Wenn das System zu stark ausgelastet ist, kann es **Activitys** im Zustand *paused* zerstören, um Speicher freizubekommen. Zuerst werden aber die **Activitys** geschlossen, die im Zustand *stopped* sind. In diesem Zustand ist die **Activity** komplett im Hintergrund.
Damit die Applikation entsprechend reagieren kann, gibt es Methoden die bei einem Zustandswechsel aufgerufen werden. Wenn man zum Beispiel den **SensorEventlistener** (Kapitel 4.1.1) beim Wechsel vom *resumed*-Zustand in einen der anderen Zustände abmeldet, kann Batterie gespart werden.

3. Controller

Dieses Paket hat dieselbe Funktion, wie das entsprechende in der Serverimplementierung.

4. Network

Auch beim Client sind die Netzwerkaktivitäten in einem extra Paket ausgelagert.

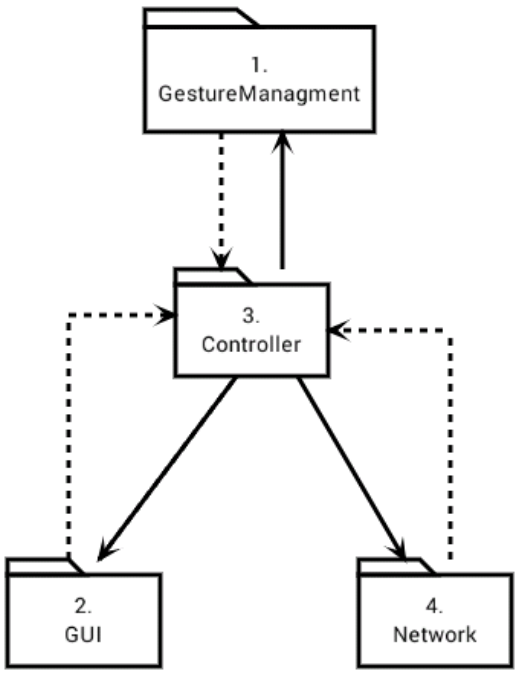

Abbildung 14: Das Paketdiagramm für den Client.

4.1 Gesten Management

4.1.1 Android Sensor API

Um die Gesten erkennen zu können, müssen die Sensoren des Smartphones ausgelesen werden. Android ermöglicht dieses durch die Klassen **SensorManager, Sensor, SensorEvent** und **SensorEventListener**. [(Sensors, 2014)]
Mit dem **SensorManager** bekommt man Zugriff auf die einzelnen Sensoren und es lassen sich **SensorEventListener** an- und abmelden.
Wenn sich die Werte des Sensors ändern wird die Methode **onSensorChanged()** des **SensorEventListeners** aufgerufen. Dabei wird ein **SensorEvent** mitgegeben, welches die Informationen enthält welcher Sensor das Event ausgelöst hat, dazu die rohen Sensordaten, Angaben über die Genauigkeit der Daten und ein Zeitstempel.
Wenn die Sensoren nicht mehr benötigt werden, sollten sie abgemeldet werden, da sie sonst die Batterie und Arbeitsspeicher zu sehr belasten und das Smartphone in wenigen Stunden kein Strom mehr hat. Die Sensoren sollten auch abgemeldet werden, wenn die **Activity** pausiert, da das System diese nicht automatisch deaktiviert.

Wenn ein Sensor angemeldet wird, kann angegeben werden in welchen Intervallen SensorEvents gesendet werden. Hierbei gibt es vier mögliche Einstellungen mit einer Verzögerung von 200.000, 60.000, 20.000 oder 0 Millisekunden.
Da die Änderungsrate der Sensoren ziemlich hoch sein kann, wird die **onSensorChanged()**-Methode entsprechend oft aufgerufen. Es sollte somit möglichst wenig in der Methode berechnet werden.
Deswegen wurde bei der Implementierung eine Verzögerung von 20.000 Millisekunden gewählt.
Die Sensoren aus Kapitel 3.1 benutzen ein 3-Achsen Koordinatensystem um Daten darzustellen. Bei Smartphones ist die Standard Orientierung des Bildschirms hochkant, wobei sie bei Tablets quer ist. Relativ zum Bildschirm ist das Koordinatensystem so definiert, dass die X-Achse nach rechts, die Y-Achse nach oben und die Z-Achse aus den Bildschirmheraus zeigen (Abbildung 15) Wenn sich die Bildschirmorientierung zwischen hochkant und quer ändert, ändert sich nicht das Koordinatensystem.

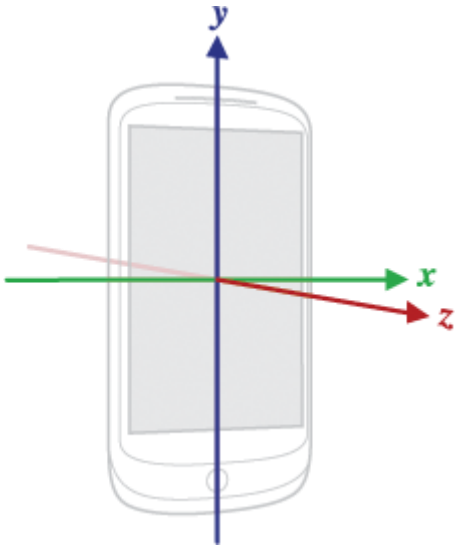

Abbildung 15: Koordinatensystem relativ zu einem Smartphone

Android wird auf vielen unterschiedlichen Smartphones benutzt, die jeweils nicht immer alle Sensoren integriert haben. Es gibt zwei Möglichkeiten dieses zu berücksichtigen. Zum Einen kann man im Androidmanifest [3] der Applikation angeben, dass ein bestimmter Sensor benötigt wird. Der Google Play Store filtert für jedes Gerät die angebotenen Applikationen, so dass die Applikation im Store gar nicht erst angezeigt wird, sollte das Gerät die Spezifikationen nicht besitzen.
Zum Anderen ist es möglich zur Laufzeit Funktionalität auszuschalten welche bestimmte Sensoren benutzen. Die Applikation kann somit trotzdem noch benutzt werden und wird auch beim Google Play Store angezeigt.

4.1.2 Implementierung der Gestenerkennung

Wie in Kapitel 2.2 erläutert gibt es unterschiedliche Methoden um Gesten zuerkennen.
In dem Projekt XWand(Kapitel 2.2.2) erreichte das HMM mit circa 90% Genauigkeit die besten Ergebnisse.
Auch bei dem Projekt von der Universität Oldenburg (Kapitel 2.2.1) wurde bei dem HMM eine Genauigkeit von durchschnittlich 90% erreicht.

Aufgrund der Genauigkeit wird das HMM für die Gestenerkennung implementiert.
Dazu wird die Bibliothek *Jahmm* benutzt. *Jahmm* ist eine Implementierung des HMM. Da

[3] Eine XML-Datei die Informationen über die Applikation für das Androidsystem bereitstellt. (AndroidManifest)

das HMM komplex ist, wurde bei der Entscheidung, ob der Programmcode von Jahmm lesbar oder effizient sein soll, die Lesbarkeit gewählt [(Jahmm, 2009)]. Es implementiert dazu Algorithmen, wie den Baum-Welch und k-mean, die für das HMM notwendig sind. Eine Alternativ wäre die Bibliothek HMMWeka [(HMMWeka, 2010)].

Da eine Implementierung eines HMM sehr speicherintensiv ist, erfolgt die Gestenerkennung auf dem Server. Das hat den Vorteil, dass Leistungsschwache Smartphones die Applikation auch benutzen können. Somit wird die Anzahl an potenziellen Benutzern möglichst hoch gehalten

Für die Gestenerkennung und Verwaltung von dem HMM ist das Paket GestureManagment zuständig (Abbildung 16).

Mit der Klasse **GestureInstance** wird eine Bewegungssequenz repräsentiert. Diese hält mehrere Instanzen von **GestureData**, die jeweils die Sensordaten zu einem bestimmten Zeitpunkt repräsentieren. Die Geste kann mit der Methode **getTrainingsSequence()** als ein 2-Dimensionales Array ausgegeben werden.
Mit **GestureDataSet** können mehrere **GestureInstance** Instanzen gespeichert werden und zum trainieren einer Geste im HMM genutzt werden.

Der **GestureRecognizer** ist für das Trainieren, Erkennen, Speichern und Laden der Gesten zuständig. Dabei referenziert es Instanzen von **Gesture** für jede Geste. **Gesture** hält das HMM für die jeweilige Geste. Mit der Methode **probability()** kann eine **GestureInstance** mit dem HMM verglichen werden. Als Ergebnis wird die Wahrscheinlichkeit zurückgegeben mit der diese **GestureInstance** von dieser HMM repräsentiert wird.
Mit der Klasse **JahmmAdapter** kann auf die Jahmm Bibliothek zugegriffen werden.
In der Methode **buildTrainedHMM()** wird ein HMM erstellt. Dabei wird ein **GestureDataSet** übergeben und dieses wird mit der Klasse **KMeansLearner** quantifiziert. Danach wird die Klasse **BaumWelchScaledLearner** benutzt um das HMM auf das **GestureDataSet** zu trainieren.
Mit **trainHMM()** kann ein HMM zusammen mit einem **GestureDataSet** übergeben werden um das HMM zusätzlich zu trainieren.

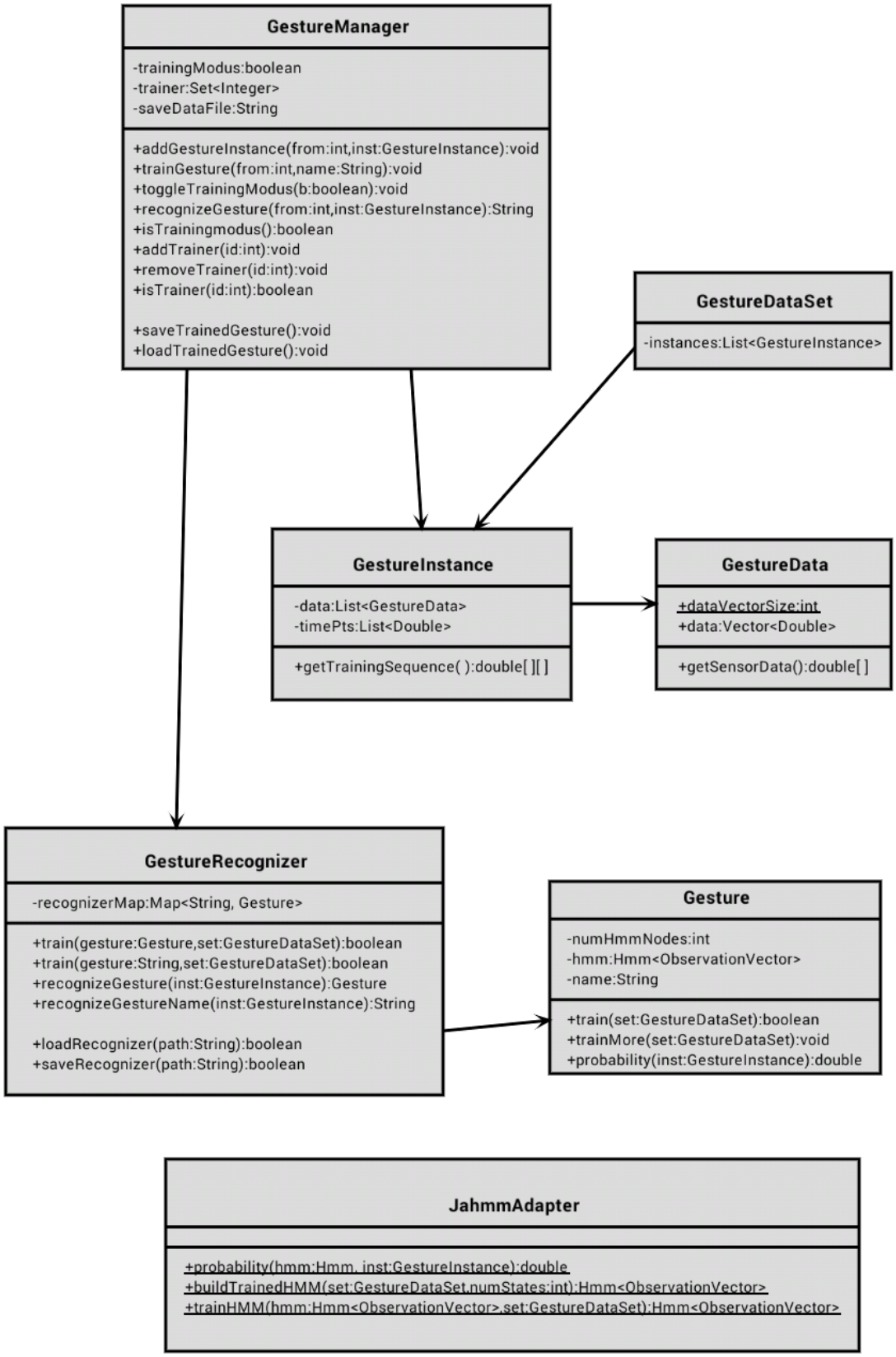

Abbildung 16: Das Klassendiagram für das Paket GestureManagement des Servers

Der **GestureManager** verwaltet den inneren Zustand des Modells und ermöglicht den Zugriff darauf. In dem Sequenzdiagramm (Abbildung 17) wird aufgezeigt wie sich das Modell verhält wenn eine Geste trainiert werden soll.

1. Mit **toggleTrainingModus()** wird in den Trainingszustand gewechselt in dem man Gesten trainieren kann.
2. Danach werden mehrere Instanzen von **GestureInstance** mit **addGestureInstance()** für den Benutzer hinterlegt.
3. Beim Aufruf von **trainGesture()** werden die hinterlegten **GestureInstance** zur Erstellung eines **GestureDataSet** benutzt.
4. Der **GestureManager** übergibt das **GestureDataSet** mit dem Namen der Geste an den **GestureRecognizer**.
5. Der **GestureRecognizer** übergibt mit **train()** das **GestureDataSet** dem **Gesture** Objekt.
6. Wenn noch kein HMM angelegt ist, wird die Methode **buildTrainedHMM()** des **JahmmAdapter** aufgerufen und ein trainiertes HMM zurückgegeben.
7. Alternativ kann ein vorhandenes HMM mit dem **GesturDataSet** zusätzlich trainiert werdern.

Das Trainieren kann nur von Benutzern durchgeführt werden, die als Trainier registriert sind. Das Erkennen von Gesten wird solange blockiert, bis der Trainingsmodus deaktiviert wird.

Damit das Modell bei jedem Neustart nicht neu trainiert werden muss, gibt es die Methoden **saveTrainedGesture()** und **loadTrainedGesture()**. Jahmm besitzt die Möglichkeit die trainierten Gesten in eine Datei zu speichern und wieder zu laden. Leider ist diese Funktion in der Bibliothek fehlerhaft. Als Übergangslösung werden die **Gesture** Objekte als JSON(mehr zu JSON in Kapitel 4.2) in eine Datei geschrieben und aus dieser wieder geladen.

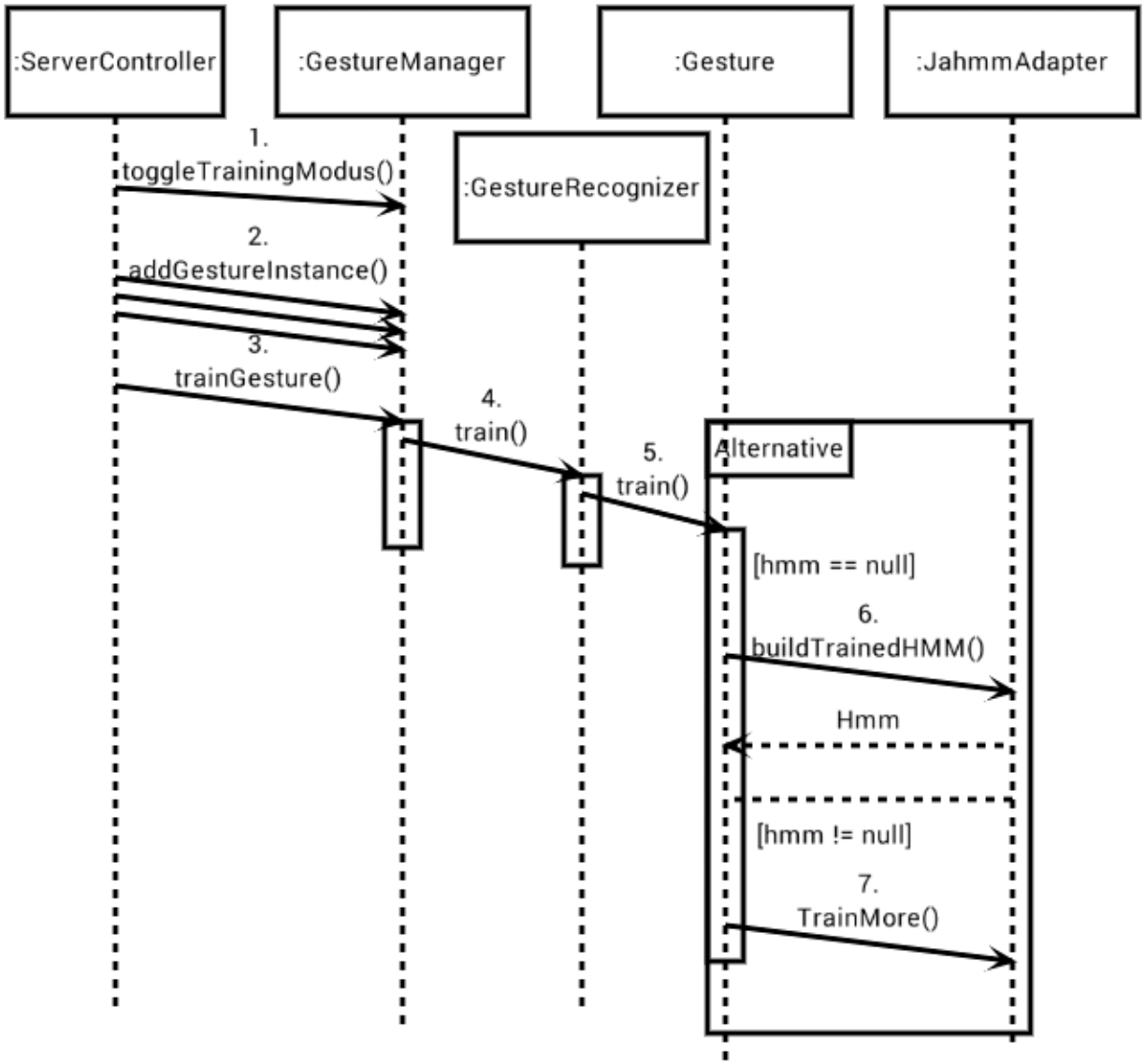

Abbildung 17: Das Sequenzdiagramm wenn eine Geste trainiert wird.

Das Sequenzdiagramm in Abbildung 18 zeigt was passiert, wenn eine Geste erkannt werden soll.

1. Mit **recognizeGestureName()** soll der Name der Geste ermittelt werden, die zu der **GestureInstance** passt. Wenn die Geste nicht erkannt wird, wird null zurückgegeben.
2. In **recognizeGesture()** wird in einer Schleife für jede Geste die Wahrscheinlichkeiten berechnet, dass die **GestureInstance** dieser entspricht. Die Geste mit der höchsten Wahrscheinlichkeit wird zurückgegeben. Wenn keine erkannt wird, ist der Rückgabewert null.

3. In der Klasse **Gesture** wird zunächst getestet, ob das HMM null ist. Wenn ja wird sofort 0.0 zurückgegeben.
4. Jahmm kann die Wahrscheinlichkeit ausrechnen, dass die Sequenz von dem HMM emittiert wird.

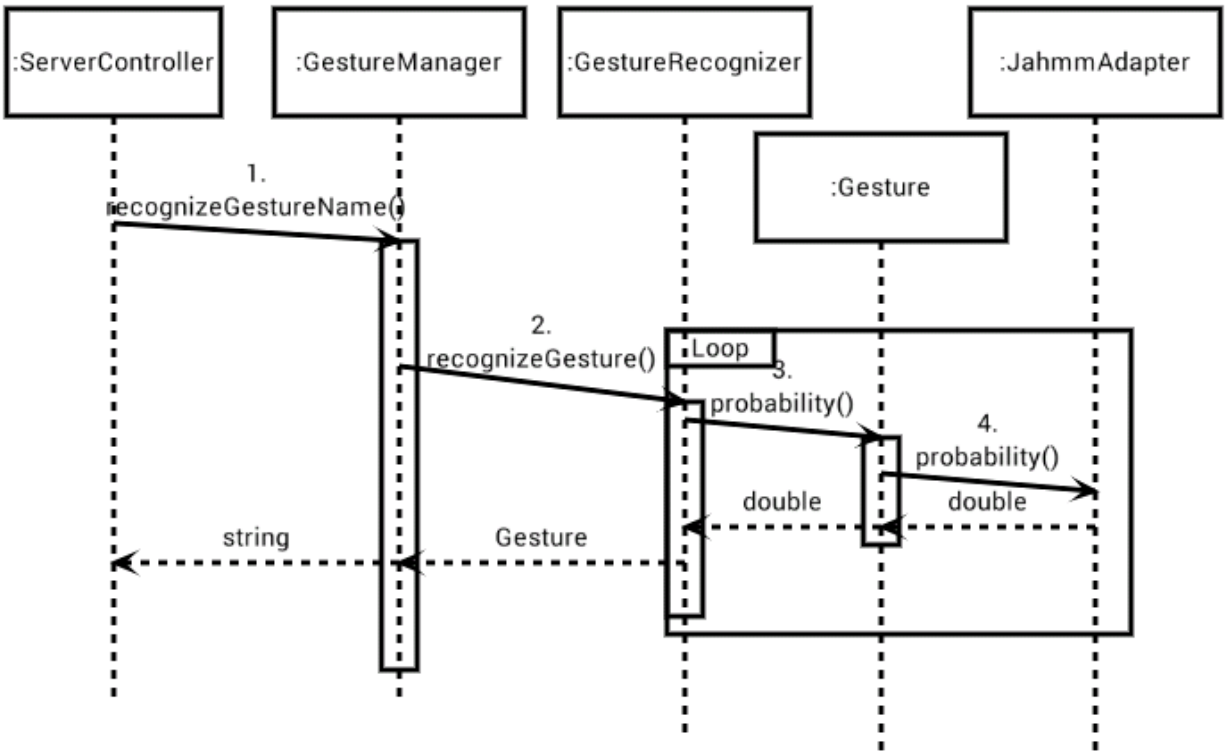

Abbildung 18: Das Sequenzdiagramm wenn eine Geste erkannt werden soll.

Auf der Clientseite müssen die Sensordaten ausgelesen und gespeichert werden. Der **GestureManager** (Abbildung 19) implementiert das Interface **SensorEventListener** (Kapitel 4.1.1). Wenn die Methode **startRecord()** aufgerufen wird, werden die Sensoren beim SensorManager angemeldet. In der Methode **onSensorChanged()** werden die Sensordaten in einem **GestureData** Objekt gespeichert. Sobald **stopRecord()** aufgerufen wird werden die Sensoren abgemeldet und ein **GestureInstance** Objekt mit den **GestureData** Objekten erstellt und über das Eventsystem an den Controller übergeben.

In der **Activity** ist ein Knopf implementiert der **startRecord()** aufruft wenn er gedrückt wird und **stopRecord()** sobald er losgelassen wird.

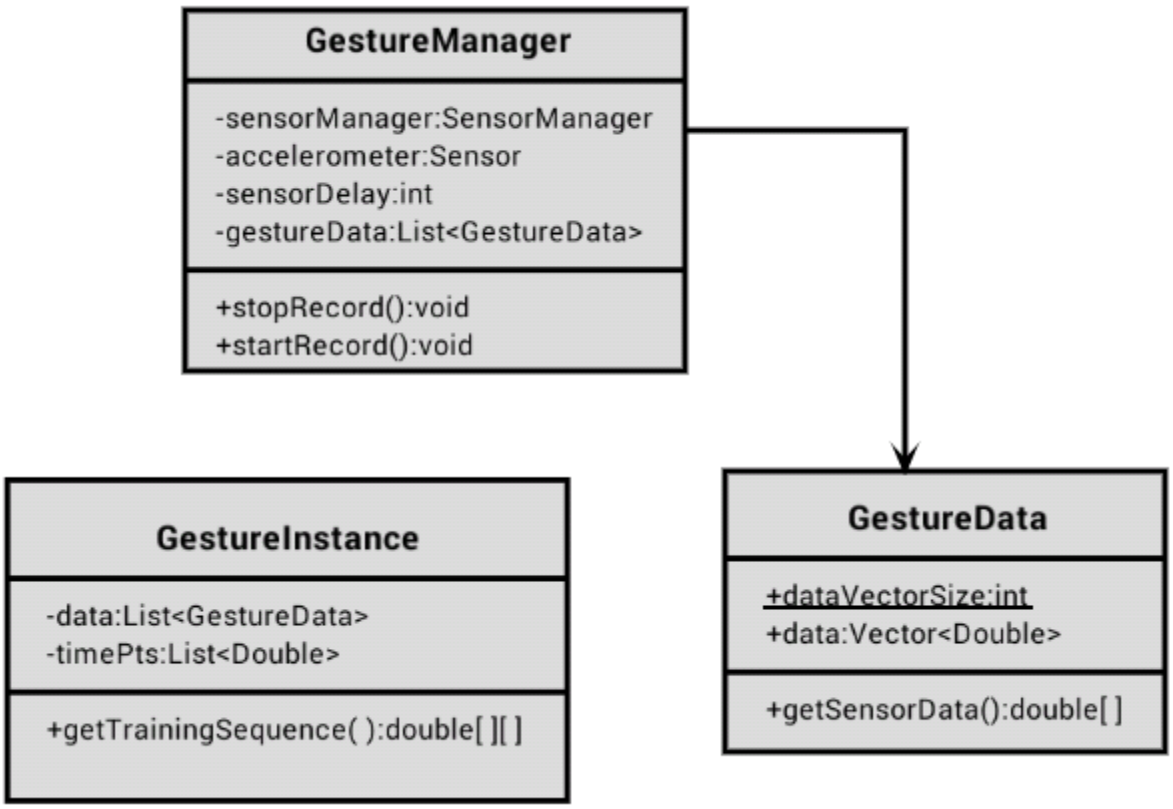

Abbildung 19: Das Klassendiagramm für das Paket GestureManagement des Clients

4.2 Client-Server Kommunikation

Android bietet unterschiedliche Möglichkeiten um Verbindungen zu anderen Geräten aufzubauen.[(Connectivity, 2014)]

Near-Field-Communication (NFC) hat eine Reichweite von wenigen Zentimetern. Für die Aufgabe lässt sich das nicht verwenden, denn die Bewegung der Geste müsste zu nah an einem Empfänger ausgeführt werden.[(Near Field Communication)]

USB könnte benutzt werden, wenn das Kabel lang genug wäre. Das Kabel könnte aber bei der Bewegung der Geste stören und wird deswegen auch nicht implementiert.

Bluetooth gibt es in drei unterschiedlichen Klassen. Je nach Klasse erreicht es eine Reichweite von 1-100 Metern. In den meisten Geräten ist die Klasse 2 eingebaut die eine Reichweite von 10 Metern hat [(Bluetooth Reichweite, 2013)]. Bluetooth 4.0 mit einer

Reichweite von bis zu 100 Metern wird von Android erst seit 4.3 unterstützt [(Bluetooth 4.0, 2014)].

Wireless LAN wird von jedem Smartphone unterstützt. Wenn der Server an das Netzwerk angeschlossen ist kann er darüber erreicht werden.

Bluetooth und Wireless LAN eignen sich beide für die Aufgabe und bieten keine Vor- oder Nachteile gegenüber dem anderen. Es wurde sich für eine Implementierung für Wireless LAN entschieden.

4.2.1 Implementierung der Netzwerkverbindung

Das Netzwerkmodul (Abbildung 20) wird durch die Schnittstelle **Server** definiert.
Das Interface **GestureServer** erweitert diese um spezifische Methoden für die Gestensteuerung. Die Klasse **AbsGestureServer** implementiert **GestureServer** und benutzt in den Methoden das Eventsystem.

Die Klasse **TCPServer** erweitert **AbsGestureServer** und implementiert die Methoden **addCommandToOutput(), startServer()** und **stopServer()**. Java bietet die Möglichkeit über die Klassen **Socket** und **ServerSocket** eine Verbindung über TCP aufzubauen. Ein Server erstellt ein **ServerSocket** Objekt, welches durch die Methode **accept()** auf eingehende Verbindungen wartet. Wenn eine Verbindung aufgebaut wird, wird ein Thread erstellt der sich um die Verbindung kümmert, damit der Server nicht blockiert wird und weiterhin auf Verbindungen warten kann.
TCPServer verwendet dafür die Klasse **ClientData** die in einem Thread die Verbindung aufrecht erhält. Über die Socket Objekte erhält man einen **InputStream** und **OutputStream**.
Über die Methode **readInput()** werden die Daten vom Client verarbeitet und die entsprechenden Methoden von **GestureServer** aufgerufen.
Daten zum Client können mit der Methode **addCommandToClient()** gesendet werden. Hierbei wird ein einfacher String im JSON Format übermittelt (im Weiteren als JSON-String bezeichnet) [(Einführung in JSON)].
JSON ist für den Austausch von Daten entwickelt worden. Es sollte dabei für Menschen einfach zu lesen und durch Maschinen leicht analysiert werden können.

Alternativ hierzu gibt es das XML-Format [(Extensible Markup Language (XML))].
JSON ist schneller und benötigt weniger Ressourcen als XML [(Nurseitov, Paulson, Reynolds, & Izurieta, 2009)]
Auch wird JSON häufiger als XML benutzt (Abbildung 21).

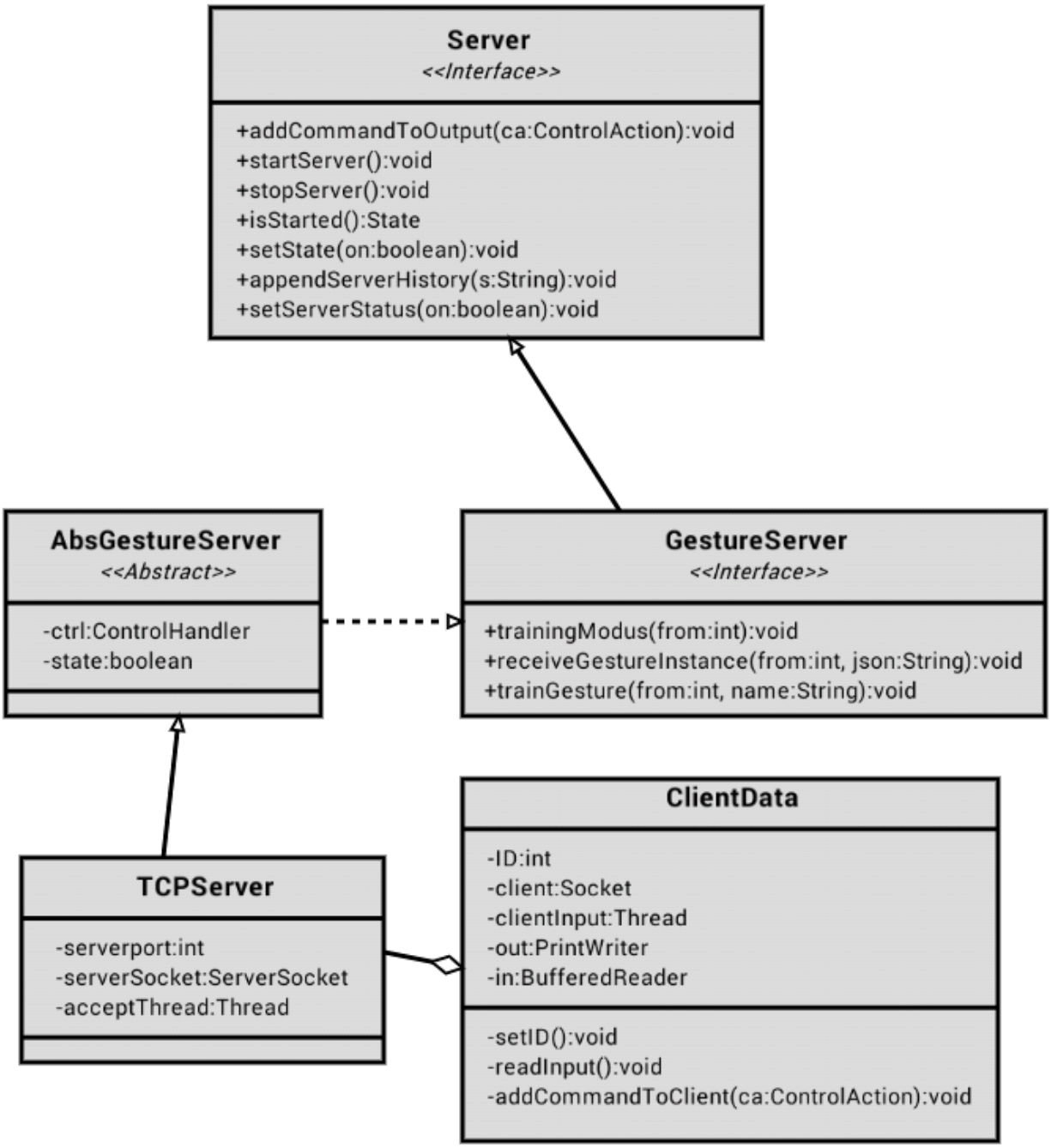

Abbildung 20: Das Klassendiagramm vom Netzwerk Paket

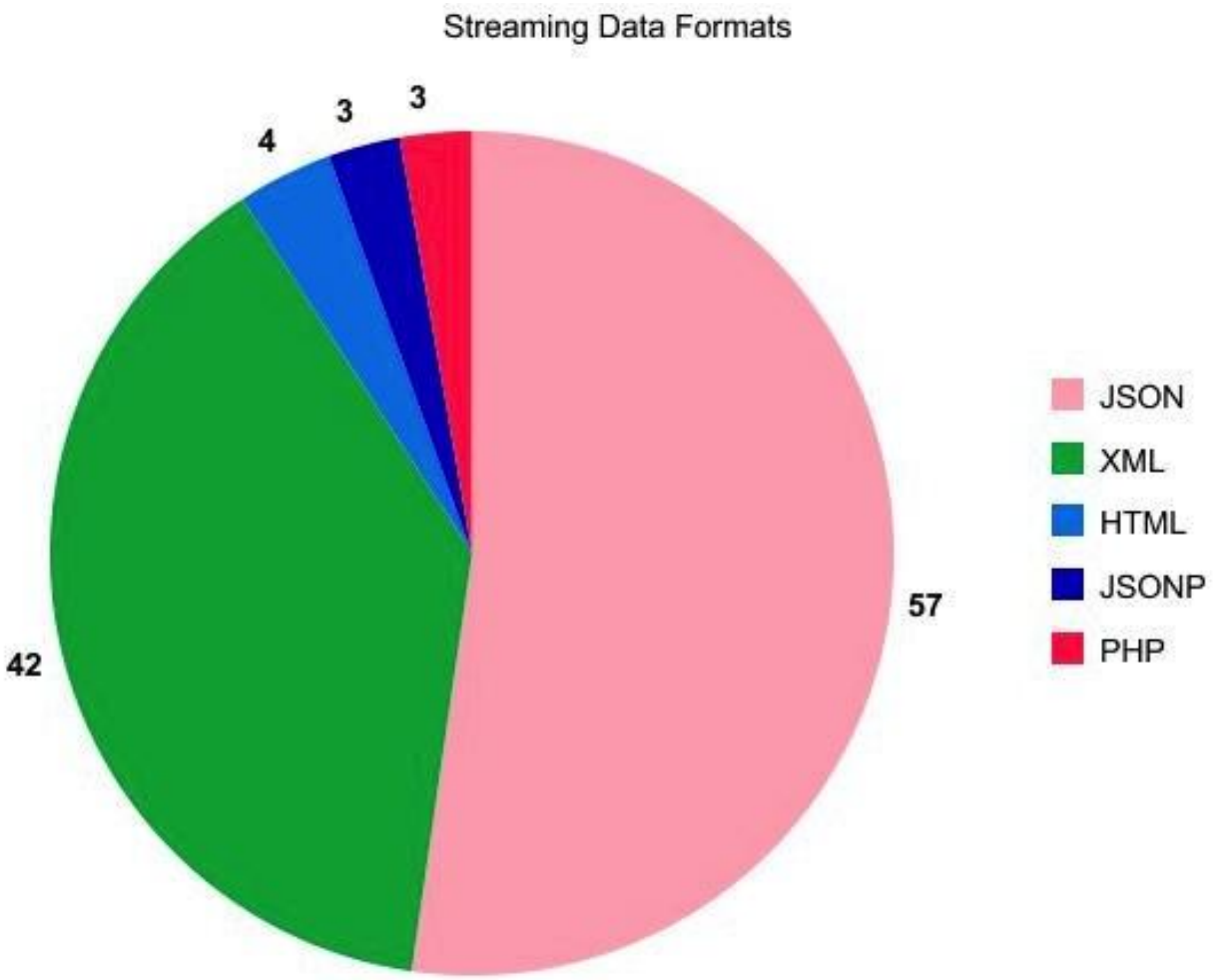

Abbildung 21: Hier wurde das Repository von APIhub.com untersucht welche Formate für die Datenübertragung benutzt wurden. (Mitch, 2013)

Mit der Bibliothek Gson [(Gson)] können Java Objekte in einen String umgewandelt werden der ein JSON repräsentiert. Ein JSON-String kann dann zurück in ein Java-Objekt gewandelt werden.

Die Klasse **ControlAction** (siehe Kapitel 4.3) wird als JSON-String versendet und empfangen. In der Klasse kann der Befehl in der Variablen **action** gesetzt werden. Die möglichen Werte für die Befehle sind als Konstanten in dem Interface **ControlHandler** (siehe Kapitel 4.3). Zusätzlich können Strings und Double in einer Map gespeichert werden. Die Schlüssel hierfür sind als Konstanten in der Klasse **ControlAction**. Dadurch können auch andere Objekte als JSON-String mit übermittelt werden.

4.3 Steuerung

Das Interface **ControlHandler** ist die Schnittstelle für das Eventsystem (Abbildung 22). Die Klassen die das Interface **ControlListener** implementieren können ein **ControlHandler** bei sich anmelden und berichten ihm mit der Methode **tryAction()** über Ereignisse.
Als Ereignis wird die Klasse **ControlAction** benutzt.
Die Klasse **ServerController** implementiert **ControlHandler** und verarbeitet die Ereignisse in der Methode **scheduleAction()**.

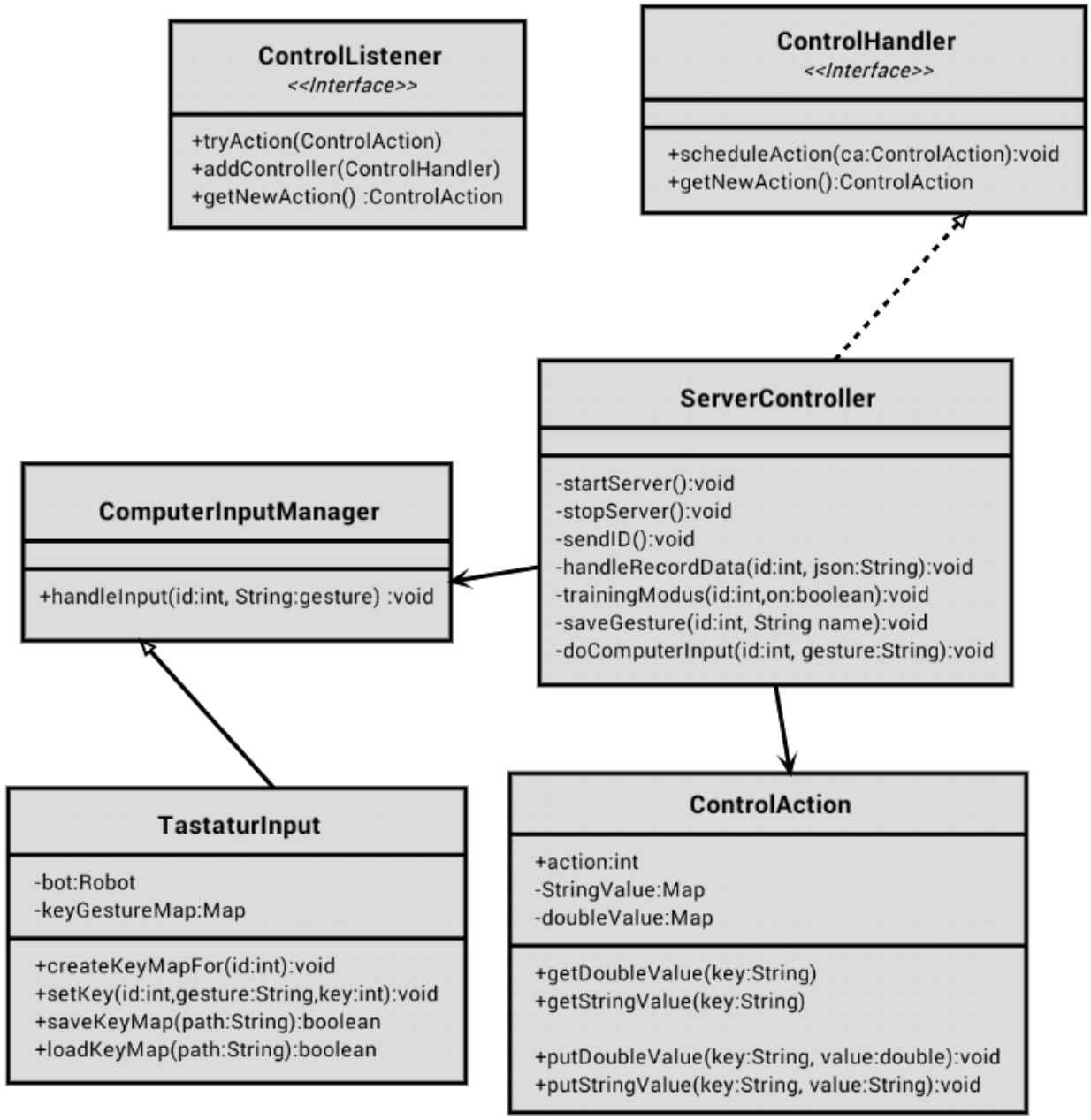

Abbildung 22: Das Klassendiagramm zum Eventsystem, zuzüglich der Klasse für die Computersteuerung

Mit der Methode **getNewAction()** können **ControlAction** Objekte aus einem Pool geholt werden. Das ist auf Android wichtig, da bei Verwendung sehr vieler Objekte durch den *Garbage-Collectors*[4] die Applikation verzögert wird (Performance Android).
In der Methode **handleRecordData()** werden die Sensordaten, die vom Client gesendet wurden, an den **GestureManager** (siehe Kapitel 4.1.2) übergeben. Abhängig davon, ob der Trainingsmodus aktiv ist, wird dafür die **addGestureInstance()** oder **recognizeGesture()** Methode benutzt. Wenn der Trainingsmodus deaktiviert ist, wird der Name der erkannten Geste mit der Methode **doComputerInput()** an das Interface **ComputerInputManager** übergeben.

Die Klasse **TastaturInput** implementiert **ComputerInputManager**. Es hält eine Map die Gesten auf Tasten der Tastatur abbildet.
Mit Hilfe von der Klasse **Robot** (Robot) in Java können Tastatur und Maus Eingaben simuliert werden. In der Methode **handleInput()** wird für die Geste der Knopf aus der Map an die Methode **keyPress()** von der Klasse Robot aufgerufen.
Mit der Methode **createKeyMapFor()** können für jeden Benutzer eine eigene Map angelegt werden. So können zwei unterschiedliche Benutzer zum Beispiel das Spiel Pong (Abbildung 23) spielen und jeder einen unterschiedlichen Balken steuern, obwohl sie dieselben Gesten benutzen.

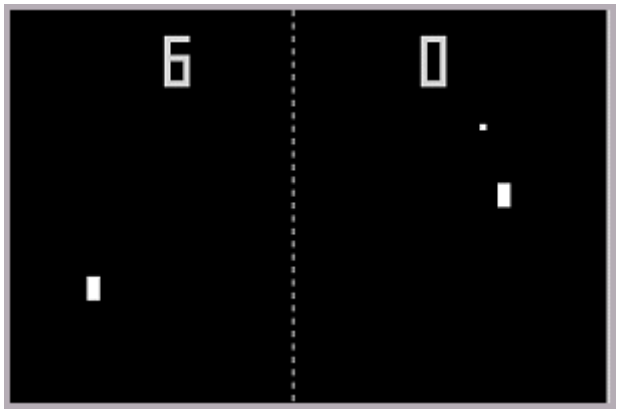

Abbildung 23:Das Computerspiel Pong. (Pong)

[4] Der Garbage-Collector entfernt automatisch unbenutzte Objekte aus dem Speicher [(Garbage Collection)]

5 Evaluierung und Ausblick

5.1 Ziel und Ablauf

Bei der Evaluierung soll getestet werden, wie gut das HMM die Gesten bei der Benutzung eines Smartphones erkennt. Hierfür wurden nur die Beschleunigungssensoren benutzt.
Es wurden zwei verschiedene Smartphones benutzt, das Samsung Galaxy S2 und S4 (Tabelle 1). Dabei soll geprüft werden, ob die Berechnung auf dem Server ausreicht, damit auch ältere leistungsschwächere Smartphones die Steuerung benutzen können.
Zusätzlich haben sie unterschiedliche Android-Versionen.

	Samasung Galaxy S2	Samsung Galaxy S4
Prozessor Anzahl	2	4
Prozessor Taktrate	Jeweils 1200MHz	Jeweils 1900MHz
Arbeitsspeicher	1024 mb	2048 mb
Android-Version	2.3	4.2
Erscheinungsjahr	2011	2013

Tabelle 1:Samsung Galaxy S2 und S4 Daten im Vergleich ((Galaxy S2) & (Galaxy S4))

Dabei wurden neun Probanden im Alter von 20 bis 85 geprüft. Keine Erfahrung mit einem Smartphone hatten dabei nur drei Probanden, die Senioren bzw. über 60 jährigen.
Es soll dabei ermittelt werden ob Alter und Vorerfahrung signifikante Unterschiede ausmachen.
Das Smartphone wurde bei der Bewegung im Hochformat in der Hand gehalten.
Es sollten dabei sechs Gesten erkannt werden (Abbildung 24). Die Gesten wurden so gewählt, dass die Verwechslungsgefahr vermutlich hoch (1. – 4) und sehr niedrig ist (5. und 6). Der Startpunkt der Geste ist der Punkt und der Endpunkt der Pfeil.

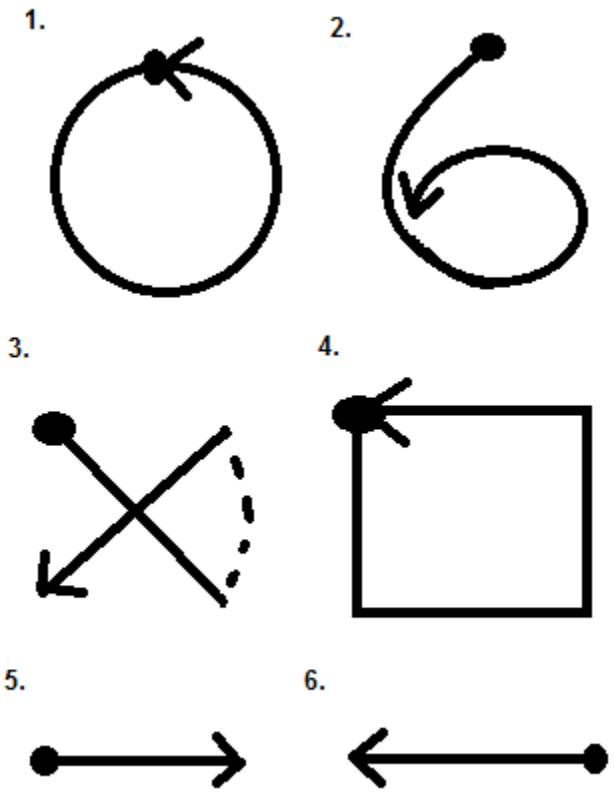

Abbildung 24: Das Gestenalphabet für die Evaluierung. 1. Kreis, 2.Sechs, 3.Kreuz, 4. Viereck, 5.Rechtsbewegung, 6.Linksbewegung.

Zunächst mussten die Gesten trainiert werden. Dabei wurde jede Geste fünf Mal von jedem Probanden ausgeführt. Es konnten dabei zwei Probanden gleichzeitig ohne Probleme trainieren.

Beim Versuchsablauf führte jeder Proband jede Geste 10 Mal pro Smartphone aus. Dabei war ein Texteditor geöffnet und die Gesten wurden als Tastendruck ausgegeben.

5.2 Ergebnisse und Auswertung

In Tabelle 1 sind die Probanden und Gesten nach aufsteigendem Alter aufgelistet. In jeder Zelle stehen zwei Werte, der Erste für das S2 und der Zweite für das S4. Der Wert ist die Wahrscheinlichkeit mit der die Geste erkannt wurde.

	Kreis	Sechs	Kreuz	Viereck	Rechts	Links	Gesamt
A	80 % 70 %	70 % 80 %	70 % 70 %	90 % 100 %	70 % 80 %	60 % 70 %	73,3 % 78,3 %
B	80 % 80 %	60 % 70 %	60 % 70 %	70 % 90 %	60 % 70 %	80 % 80 %	68,3 % 76,6 %
C	70 % 80 %	60 % 80 %	80 % 70 %	70 % 80 %	70 % 70 %	60 % 70 %	68,3 % 75 %
D	80 % 80 %	100 % 90 %	90 % 90 %	80 % 90 %	70 % 70 %	70 % 70 %	81,6 % 81,6 %
E	90 % 80 %	70 % 70 %	70 % 70 %	70 % 70 %	80 % 80 %	60 % 70 %	73,3 % 73,3%
F	80 % 90 %	70 % 80 %	70 % 80 %	70 % 60 %	60 % 90 %	80 % 80 %	71,6 % 80 %
G	70 % 70 %	80 % 80 %	80 % 80 %	80 % 60 %	50 % 60 %	80 % 70 %	73,3 % 73,3 %
H	80 % 90 %	40 % 60 %	70 % 70 %	80 % 80 %	70 % 70 %	60 % 90 %	66,6 % 76,6 %
I	70 % 60 %	50 % 60 %	70 % 80 %	70 % 80 %	60 % 70 %	70 % 80 %	65 % 71,6 %
Gesamt	77,7 % 77,7 %	66,6 % 74,4 %	73,3 % 75,5 %	75,5 % 78,8 %	65,5 % 75,5 %	68,8 % 75,5 %	71,2% 76,2%

Tabelle 2: Die Erkennung von Gesten pro Proband und Geste. In jeder Zelle steht als erste Zahl der Wert für das S2 und der Zweite für das S4

Der Unterschied der Genauigkeit zwischen S2 und S4 beträgt im Durchschnitt 5%.
Pro Proband schwankt dieser Wert zwischen 0 und 10%.
Das liegt vermutlich daran, dass die eingebauten Sensoren beim S2 ungenauer sind und nicht, dass die Leistung des S2 zu schwach wäre.

Im Weiteren werden nur noch die Werte vom S4 benutzt.

Die Senioren haben eine durchschnittliche Genauigkeit von 73% und die anderen von 77%. Das ist, dafür dass sie keine Erfahrungen mit einem Smartphone hatten, relativ gut.

In Tabelle 3 werden alle Gesten mit ihren absoluten Häufigkeiten aufgelistet. Dabei sind die Zeilen die ausgeführte Geste und die Spalten die erkannte Geste.

	Kreis	Sechs	Kreuz	Viereck	Rechts	Links
Kreis	70 / 70	10/10	2/0	8/10	0/0	0/0
Sechs	15/11	60/67	0/0	15/12	0/0	0/0
Kreuz	13/12	11/10	66/68	0/0	0/0	0/0
Viereck	10/9	12/10	0/0	68/71	0/0	0/0
Rechts	0/0	0/0	6/3	25/19	59/68	0/0
Links	0/0	0/0	1/3	27/19	0/0	62/68

Tabelle 3: Die absoluten Häufigkeiten der Gesten, die Zeilen sind die ausgeführten Gesten und die Spalten die jeweils erkannten. Die linken Werte sind vom S2 und die Rechten vom S4

Die einfachen Gesten Links und Rechts wurden häufiger falsch erkannt als erwartet.
Das Liegt vermutlich daran, dass die Bewegungen unterschiedlich ausgeführt wurden, da nicht gesagt wurde wie genau die Bewegung auszusehen hatte. Dabei wurden die Bewegungen entweder mit dem gesamten Arm vollführt, oder nur aus dem Handgelenk. Beide Bewegungen entsprechen dem Vorgegebenen Muster (Abbildung 24) führen aber zu unterschiedlichen Bewegungssequenzen.
Die Genauigkeit könnte hierbei erhöht werden, wenn man die Probanden besser einweisen würde, wie genau sie die Bewegung zu vollführen haben.

Bei den Gesten Sechs, Kreis und Viereck ist das Problem, dass wenn die Gesten nicht exakt ausgeführt werden es leicht zu Fehler führt.
Der Kreis und die Sechs fangen beide an derselben Stelle an und unterscheiden sich nur daran, dass die Sechs ein bisschen früher einknickt
Das Viereck kann bei zu rund ausgeführten Bewegungen bei den Ecken als Kreis identifiziert werden. Das Selbe bei der Sechs.

Interessant hierbei ist, dass die Kreuz Geste zwar als Kreis und Sechs, aber Sechs keinmal und Kreis nur zweimal(und das nur beim S2) als Kreuz erkannt wurden. Das liegt vermutlich daran, dass die Wahrscheinlichkeit für Viereck jedes Mal höher war als für Kreuz.

5.3 Ausblick

Die hier entwickelte Anwendung und die durchgeführten Versuche zeigen, dass das Smartphone als Eingabegerät zur Gestenerkennung durchaus gut geeignet ist. Zur Beurteilung der weiteren Verwendung stellen sich mehrere Fragen, denn u.a. zeigten die Versuche, dass die Ergonomie bzw. Bedienbarkeit des Systems noch optimiert werden kann. Gerade die flüssige Bedienung wurde durch Fehlerkennungen teilweise noch eingeschränkt. Hier bieten sich aber noch vielfältige Korrekturmöglichkeiten an, wie z.B. eine Werteglättung zur Erkennung von Ausreißern, wofür vielfältige bestehende Algorithmen verwendet werden können. Auch weiter Sensoren könnten die Genauigkeit erhöhen.
Zudem ist das derartige System darauf ausgelegt, immer die wahrscheinlichste Geste zurückzugeben, auch wenn der Grad der Erkennung sehr gering ist. So wird derzeit immer eine Geste erkannt, auch wenn die Eingabedaten nur eine schwache Erkennungssignifikanz aufweist. Dabei könnte bei einer zukünftigen Version ein Schwellenwert im höheren Prozentbereich als Mindesterkennungswert Abhilfe schaffen, so dass insgesamt von einer sehr zufriedenstellenden Erkennungsrate und Bedienbarkeit ausgegangen werden kann.

Mit einem solchen System wären einige Anwendungsszenarien denkbar, die teilweise schon von den derzeit vorliegenden Prototypen bedient werden könnten. Folgend soll ein kleiner Ausblick auf entsprechende Implementierungen gegeben werden:

- **Spieleanwendungen in Schaufenstern:** Der Anwender findet einen Spielaufbau in einem Schaufenster vor, zum Beispiel auf einem Flatscreen. Über eine Bluetooth oder WLAN Verbindung nimmt eine Client-Applikation, die mit einem QR Code-Link im Schaufenster beworben werden könnte, Verbindung mit dem Spielsystem auf.

- **Kataloganwendungen:** Der Anwender blättert mit seinem Smartphone-Controller in einem Katalog oder wählt zwischen Fotos oder Musiktiteln.

- **Smartphone als Spielcontroller**: Zur Erweiterung von Spielekonsolen oder sonstigen Spielfähigen Geräten könnten Smartphones eingebunden werden. Plattformübergreifende Hersteller wie z.B. Sony, die Smartphones aber auch TVs herstellen, könnten hier Verbundeffekte für die Konsumentenbindung an die Geräte des Herstellers erzeugen.

- **Controller für spielferne Zwecke, wie z.B. Fernsehgeräten**: Das Smartphone könnte über die Gestensteuerung auch zum fernbedienen des TVs genutzt werden. Zum Beispiel links-rechts Bewegung zum Durchschalten von Programmen.

- **Home-Automation**: Kleine einfache Anwendungen könnten auch für Nischenbereiche entwickelt werden, wie z.B. eine Steuerung, die beim nächtlichen Umdrehen oder Bewegen des Smartphones eine voreingestellte Helligkeitsstufe für die Raumbeleuchtung aktiviert.

Weitere Verbesserungen im Steuerungskonzept der Kernanwendung könnten die Vielfältigkeit der jeweiligen Anwendungsbreite noch erhöhen – unbeschränkt auf einzelne Anwendungsbeispiele wären zum Beispiel:

- Realisierung kontinuierlicher Gesten, bei denen die Erkennung die ganze Zeit in Echtzeit stattfindet, anstatt dass im aktuellen Modell ein Knopf zur Gestenerkennung gedrückt werden muss

- Den Inputbereich wechseln, zum Beispiel Umschalten zwischen einer Anwendungssteuerung und Makros oder Hotkeys.

- Komplexere Anwendungssteuerungen anstatt einfach nur eine Geste-zu-Taste Zuordnung vorzunehmen

Insgesamt konnte gezeigt werden, dass die entwickelte Anwendung funktional ist und die verfügbaren Algorithmen zur Umsetzung gut geeignet waren. Durch weitere Optimierung kann die Bedienbarkeit soweit verfeinert werden, dass in der Praxis vielfältige Einsatzmöglichkeiten für eine derartige Umsetzung denkbar sind. Die zunehmende Verbreitung von Smartphones und das immer stärker erforschte Gebiet der Mensch-Maschine-Interaktion machen entsprechende Anwendungen hier für Anbieter und Anwender gleichermaßen interessant und werden sicherlich in Zukunft mehr und mehr Beispiele für praktische Anwendungen finden.

Abbildungsverzeichnis

Literaturverzeichnis

Acitivitys. (2014). Abgerufen am 2014. März 07 von http://developer.android.com/guide/components/activities.html

Android SDK. (2014). Abgerufen am 13. März 2014 von developer.android.com: http://developer.android.com/sdk/index.html

AndroidManifest. (kein Datum). Abgerufen am 10. März 2014 von http://developer.android.com/guide/topics/manifest/manifest-intro.html

Apache License. (kein Datum). Abgerufen am 12. März 2014 von apache.org: http://www.apache.org/licenses/LICENSE-2.0.html

Apostolyuk, V. (2006). *Theory and design of micromechanical vibratory gyroscopes.* Abgerufen am 03. 03 2014 von http://www.astrise.com/research/library/memsgyro.pdf

Apple Konto. (2014). Abgerufen am 13. März 2014 von developer.apple.com: https://developer.apple.com/programs/ios/

Benbasat, A. Y., & Paradiso, J. A. (2002). Atomic Gesture Recognition Algorithmus. *An Intertial Measurement Framework for Gesture Recognition and Applications* . Cambridge, Massachusetts, USA.

Bluetooth 4.0. (2014). Abgerufen am 2014. März 07 von developer.android.com: http://developer.android.com/guide/topics/connectivity/bluetooth-le.html

Bluetooth Reichweite. (2013). Abgerufen am 10. März 2014 von http://www.bluetooth.com/Pages/Basics.aspx

Brandt, M. (25. November 2013). *Statista*. Abgerufen am 2014. 03 03 von http://de.statista.com/infografik/736/absatz-von-mobiltelefonen-und-smartphones-in-deutschand/

Brandt, M. (15. Mai 2013). *Statista.com*. Abgerufen am 2014. März 03 von http://de.statista.com/themen/581/smartphones/infografik/1097/marktanteile-der-smartphone-betriebssysteme/

Connectivity. (2014). Abgerufen am 07. März 2014 von http://developer.android.com/guide/topics/connectivity/index.html

Culjat, D. (23. Februar 1999). Abgerufen am 2014. 03 03 von http://www.inf.fu-berlin.de/lehre/WS98/SprachSem/culjat/node4.html

dashboard. (04. Februar 2014). Abgerufen am 03. März 2014 von developer.android.com: https://developer.android.com/about/dashboards/index.html

Einführung in JSON. (kein Datum). Abgerufen am 11. März 2014 von json.org: http://json.org/json-de.html

Extensible Markup Language (XML). (kein Datum). Abgerufen am 13. März 2014 von www.w3.org: http://www.w3.org/TR/2006/REC-xml-20060816/

Free Pedometer & Step Counter. (12. Juni 2013). Abgerufen am 10. März 2014 von play.google.com: https://play.google.com/store/apps/details?id=com.gzone.pedometer

Gabler Wirtschaftslexikon. (2011). Abgerufen am 10. März 2014 von Springer Gabler Verlag: http://wirtschaftslexikon.gabler.de/Archiv/569824/smartphone-v1.html

Galaxy S2. (kein Datum). Abgerufen am 15. März 2014 von samsung.com: http://www.samsung.com/de/consumer/mobile-device/mobilephones/archive-mobile-phones/GT-I9100LKADBT-spec

Galaxy S4. (kein Datum). Abgerufen am 15. März 2014 von samsung.com: http://www.samsung.com/de/consumer/mobile-device/mobilephones/smartphones/GT-I9505ZKADBT-spec

Garbage Collection. (kein Datum). Abgerufen am 12. März 2014 von oracle.com: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Google Konto. (2014). Abgerufen am 13. März 2014 von developer.android.com: http://developer.android.com/distribute/googleplay/publish/register.html

Gson. (kein Datum). Abgerufen am 13. März 2014 von code.google.com: https://code.google.com/p/google-gson/

Gyroskop Abbildung. (2013). Abgerufen am 10. März 2014 von perpetuum-mobile.ch: http://www.perpetuum-mobile.ch/de/produkte/gyroskop-1076.html

Hammerstein, L., Ellßel, S., Tröbs, V., Schrottenloher, A., Zach, M., & Hupp, P. (2009). *Clusteranalyse.* Abgerufen am 03. 03 2014 von http://www-m9.ma.tum.de/material/felix-klein/clustering/Methoden/K-Means.php

HMMWeka. (2010). Abgerufen am 2014. März 07 von http://www.doc.gold.ac.uk/~mas02mg/software/hmmweka/index.html

HTK Toolkit. (2009). Abgerufen am 10. März 2014 von htk.eng.cam.arc.uk: http://htk.eng.cam.ac.uk/

Jahmm. (2009). Abgerufen am 01. März 2014 von code.google.com: https://code.google.com/p/jahmm/

Laurel, B. (1990). *The Art of Human-Computer Interface Design.* Addison-Wesley Professional.

Los, Ziege, Los! (2013). Abgerufen am 14. März 2014 von play.google.com: https://play.google.com/store/apps/details?id=com.bestcoolfungamesfreegamecreation.gogogoat&hl=de

Ludwig, H. (09. November 2007). *Albanien*. Abgerufen am 15. März 2014 von uni-jena.de: http://www.uni-jena.de/PM071109_LNDW_Albanien.html

McNeill, D. (1992). *Hand and mind: What gestures reveal about thought.* Chicago: TheUniversity of Chicago Press.

Mead, G. H. (1973). *Geist, Identität und Gesellschaft aus der Sicht des Sozialbehaviorismus.* Suhrkamp Verlag.

Mitch, P. (23. Januar 2013). *Streaming APIs JSON vs XML*. Abgerufen am 13. März 2014 von java.dzone.com: http://java.dzone.com/articles/streaming-apis-json-vs-xml-and

Near Field Communication. (kein Datum). Abgerufen am 10. März 2014 von developer.android.com: http://developer.android.com/guide/topics/connectivity/nfc/index.html

Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Comparison of JSON and XML Data In terchange Formats: A Case Study. *ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE 2009.* San Francisco.

Open Handset Alliance. (kein Datum). Abgerufen am 2014. März 10 von http://www.openhandsetalliance.com/
Osmann, A. (2012). *Learning JavaScript Design Patterns.* United States of America: O'Reilly Media. Inc.

Performance Android. (kein Datum). Abgerufen am 12. März 2014 von developer.android.com: http://developer.android.com/training/articles/perf-tips.html

Pong. (kein Datum). Abgerufen am 12. März 2014 von 8bit-museum.de: http://8bit-museum.de/videospiele/stage-1-die-steinzeit/level-2-atari-steigt-auf/

Rabiner, L. R. (1989). Hidden Markov Model. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition , VOL.77 , No. 2*, 257-286. Fellow, IEEE.
Rabiner, L. R., & Juang, B.-H. (Januar 1986). An Introduction to Hidden Markov Models. *IEEE. ASSP Magazine , Band. 3* (Nr. 1), S. 4-16.

Robot. (kein Datum). Abgerufen am 12. März 2014 von docs.oracle.com: http://docs.oracle.com/javase/1.5.0/docs/api/java/awt/Robot.html

Schlomer, T., Poppinga, B., Henze, N., & Boll, S. (2008). Gesture Recognition with a Wii Controller. *Proceedings of the 2nd international conference on Tangible and embedded interaction* , 11-14. Bonn, Deutschland.

Sensors. (2014). Abgerufen am 13. März 2014 von developer.android.com: http://developer.android.com/guide/topics/sensors/sensors_overview.html

Statiska.com. (2013). Abgerufen am 2014. März 03 von http://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/

Wilson, D., & Wilson, A. (2004). *Gesture Recognition Using the XWand.* Robotics Institute, Pittsburgh, PA.

Wilson, D., & Wilson, A. (2004). *Gesture Recognition Using the XWand.* Pittsburgh, PA: Robotics Institute.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und unter Benutzung keiner anderen Quellen als der genannten verfasst habe. Alle aus solchen Quellen wörtlich oder sinngemäß übernommenen Passagen habe ich im Einzelnen unter Angabe des Fundortes gekennzeichnet.Die schriftliche Fassung entspricht derjenigen auf dem elektronischen Speichermedium. Die vorliegende Arbeit habe ich vorher nicht in einem anderen Prüfungsverfahren eingereicht.

Datum/ Unterschrift

Kolja Bruns