Masterarbeit

Machbarkeitsstudie zur Erkennung von kindgerechten Pilatesübungen anhand von Smartphone-Sensordaten

Manuel Parfant

7. Mai 2014

Betreuer: Univ.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner

Technische Universität Graz
Institut für Informationssysteme und Computer Medien

Kurzfassung

Immer mehr Kinder leiden heutzutage an Rückenproblemen und Wirbelsäulenfehlstellungen. Verschiedene Studien widmen sich diesem Thema und unterschiedliche Bewegungsprogramme werden zur Prävention entworfen. Generell bleibt ein Therapieerfolg jedoch oft aus, da die Patientinnen und Patienten zuhause die vorgeschriebenen Übungen nicht regelmäßig oder falsch absolvieren und den Therapeutinnen und Therapeuten eine Kontrollmöglichkeit fehlt.

In dieser Forschungsarbeit wird auf die Frage eingegangen, ob es möglich ist, durch den Einsatz der immer weiter verbreiteten Smartphones und den darin eingebauten Sensoren, die Übungen der Probandinnen und Probanden automatisch aufzuzeichnen und dadurch falsche beziehungsweise unvollständige Bewegungsprotokolle zu ersetzen.

Es wird ein System zur Aufzeichnung und Auswertung solcher Bewegungsprogramme entwickelt. Dieses besteht aus einer Client-Applikation für die iOS-Plattform, welche Daten anhand der im iPhone eingebauten Sensoren sammelt, einem Webservice zur Auswertung dieser Daten und einer Webapplikation, welche die absolvierten Programme für Therapeutin beziehungsweise Therapeut und Teilnehmerinnen und Teilnehmer grafisch aufbereitet.

Anhand einer Evaluierung, mit vier Probandinnen beziehungsweise Probanden mit unterschiedlichen Voraussetzungen, wird gezeigt, dass die Bewegungen zwar nicht exakt, aber für die Praxis ausreichend genau erkannt und damit gewertet werden können. Man sieht, dass die gesammelten Bewegungsdaten eine gute Datengrundlage für weitere Analysen liefern. Mithilfe von verschiedenen Machine-Learning-Methoden können typische Fehler gefunden und Verbesserungsvorschläge geliefert werden.

Abstract

Nowadays more and more children are suffering from back pain and postural defects. Different studies dedicate to this issue and various workouts are specially designed for prevention. Generally therapeutic success remains low, because patients do not accomplish the required exercises at home or perform them wrong and the therapist has no possibility to monitor their activities.

This master thesis tries to answer the question, if it is possible to monitor the activities of people, just with the data provided by sensors included in smartphones, which are on the increase. Furthermore the implemented system should substitute incomplete and wrong protocols from patients.

For this purpose a system is developed. It consists out of a client application for the iOS platform, which collect the available sensor information, a webservice for evaluating this data and a webapplication, which represents accomplished workouts graphically.

An evaluation, with four subjects with different prerequisites, shows that it is possible to recognise and count the movements, even though not exactly. But the results are sufficient for practical applications. It also shows that the collected motion data is well-suited for further analyses and evaluations. With the help of various machine learning techniques, it is possible to distinguish between typical errors of execution and provide suggestions for improvement of the accomplished exercises.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Formelverzeichnis

1 Einleitung und Motivation

Immer mehr Kinder und Jugendliche leiden in der heutigen Zeit an Rückenproblemen, aus diesem Grunde ist das Thema *Prävention von Wirbelsäulenfehlstellungen* ein wesentliches. Oft bleibt ein notwendiger Therapieerfolg auch relativ bescheiden, da speziell Kinder, die vom Therapeuten vorgeschriebenen Übungen zuhause meist nicht Ordnungsgemäß durchführen können. (Deutschmann, 2014)

Diese Arbeit widmet sich der Forschungsfrage, ob es möglich ist, auf einfachste Weise durch den Einsatz von neuen Medien, wie Smartphones und deren eingebauten Sensoren, die Übungen der Probanden und Probandinnen automatisch aufzuzeichnen. Ziel ist es, zumeist unvollständige beziehungsweise falsche Bewegungsprotokolle zu ersetzen oder auch zuhause absolvierte Bewegungsprogramme zu bewerten und Verbesserungsvorschläge zu liefern, um den bestmöglichen Fortschritt zu gewährleisten.

Umfangreichere tragbare Datenverarbeitungssysteme (engl. Wearable Computing) sind hier unpraktisch und somit unbrauchbar, da, abgesehen vom finanziellen Aufwand, Kinder und Jugendliche bei zu aufwendigen Vorbereitungsmaßnahmen diese nicht selbstständig verwenden würden. Somit wird eine einfachere Methode benötigt. Da immer mehr Kinder und Jugendliche ein Smartphone besitzen (Grimus u. Ebner, 2014), soll versucht werden, die dadurch zur Verfügung stehenden Informationen zur Klassifizierung zu verwenden.
Die entwickelte MotionTracker Applikation soll eine Studie im Rahmen einer Dissertation mit dem Titel *Kindgerechtes Pilates als Intervention zur Haltungsstabilisierung und Verbesserung von Wirbelsäulenfehlhaltungen bei SchülerInnen im Alter von 10-12 Jahren*, von Dietlind Deutschmann (Deutschmann, 2014), unterstützen und ein Bewegungsprotokoll ersetzen.
Anhand von Bewegungsdaten, der im iPhone zur Verfügung stehenden Sensoren, sollen sechs verschiedene Übungen erkannt und gezählt werden. In einer Webapplikation stehen der Studienbetreuerin dann detaillierte Informationen zu absolvierten Trainingsprogrammen, sowie den einzelnen Übungswiederholungen zur Verfügung.
Durch den Einsatz von iPhones und der entwickelten iOS-Applikation soll den Kindern ein Anreiz gegeben werden, regelmäßig das vorgeschriebene Bewegungsprogramm

zuhause zu absolvieren. Die Studienbetreuerin hingegen erhält ein Kontrollinstrument, um die Teilnahme, sowie Verbesserungen und Fortschritte, kontrollieren zu können und gezielt zu intervenieren.
Für jede dieser Übungen wurde ein Referenzpunkt definiert, an dem das iPhone mit einer handelsüblichen Tasche befestigt werden muss. Diese Position entspricht den Teilen des Körpers, die die charakteristischste Bewegung für die jeweilige Übung ausführen. Dadurch können auch typische Fehler klassifiziert und somit Verbesserungsvorschläge geliefert werden. In Abbildung 1.1 sieht man ein Beispiel für die Befestigung des iPhones in der dafür vorgesehenen Tasche am Oberarm. Dies ist der Referenzpunkt für die Übungen *Über Kreuz*, *Wirbelsäulendrehung* und *Zirkuspferd*.

Abbildung 1.1: Anbringung eines iPhones an einem Referenzpunkt am Oberarm (eigener Entwurf)

2 State of the Art

2.1 Einleitung

Dank der stetigen Weiterentwicklung von digitalen Technologien, nehmen diese einen immer größer werdenden Stellenwert in unserem Alltag ein. Wir leben im Zeitalter der oft so genannten *Netzgeneration* (Arnold u. Weber, 2011), das heißt, dass bereits Kinder und Jugendliche einen weitgehend homogenen Zugang zu neuen Medien besitzen. Da sie bereits damit aufgewachsen sind, können sie auch selbstverständlich und kompetent mit diesen Technologien umgehen. Dieser Umgang mit neuen Medien der *digitalen Eingeborenen* (engl. digital natives) (Arnold u. Weber, 2011) unterscheidet sich damit grundlegend von dem der älteren Generation, welche allerdings dadurch auch zur Verwendung getrimmt wird und so zu *digitalen Einwanderern und Einwanderinnen* (engl. digital immigrants) (Arnold u. Weber, 2011) werden. (Ebner u. Schön, 2011)

Durch die höhere Bereitschaft digitale mobile Endgeräte in den Alltag zu integrieren steigt auch die Anwendungszahl. Vom E-Learning über E-Health, mit seinen Teilgebieten des Telemonitorings und Ambient Assisted Living, der Computerspielindustrie, mit den immer populärer werdenden, bewegungsgesteuerten Spielen, bis hin zum Gesundheits-, Sport- und Wellness-Sektor werden immer mehr Systeme, die auf digitale mobile Endgeräte bauen, in den Umlauf gebracht. [1,2]
Im Profisport wird die Echtzeitüberwachung der SportlerInnen bereits weitgehend eingesetzt, Vitalwerte sowie Bewegungen können aufgezeichnet und in Echtzeit ausgewertet werden. Dadurch wird der Trainingsablauf und somit auch deren Erfolg optimiert. Doch den immer billiger werdenden Elektronikbauteilen ist es zu verdanken, dass diese Vorteile massentauglich und somit auch Hobbysportlern und Hobbysportlerinnen zugänglich gemacht werden. Verschiedenste Daten können gemessen und über Smartphones oder Sportuhren interpretiert und sogar im Internet auf diversen sozialen Medien geteilt und

[1] Spiegel Online. Messwut im Hobbysport: Sensor unter der Sohle, 2014. http://www.spiegel.de/gesundheit/ernaehrung/sportler-messen-ihre-leistung-geringer-nutzen-fuer-das-training-a-827799.html, 12.03.2014.

[2] Wirtschafts Woche. Intelligente Kleidung - Der vernetzte Laufschuh, 2014. http://www.wiwo.de/unternehmen/industrie/intelligente-kleidung-der-vernetzte-laufschuh/5832276.html, 12.03.2014.

diskutiert werden. Der eigene Körper wird genau vermessen und alles interpretiert um möglichst Effizient zu trainieren. [1,2]
Auch beim Telemonitoring, einem Teilbereich des E-Health-Bereiches, werden tragbare Datenverarbeitungssysteme (engl. Wearable Computing) eingesetzt, um Patientendaten zu beobachten. Verschiedenste Vitalwerte können, je nach Krankheit oder Anwendung, überwacht werden und im Notfall wird an eine Notrufzentrale ein Alarm abgesetzt. Vor allem für ältere Menschen bringt das eine enorme Erleichterung, da sie ohne Angst und trotz Beeinträchtigung ihr gewohntes Leben leben können. (Deutscher Bundestag, 2014)
Die Computerspielindustrie baut auf die Spieler und Spielerinnen als Controller und Motion-Gaming-Systeme messen die Bewegungen anhand von Sensoren, um so das Spiel zu steuern. [3]
Dieser Bereich ist teilweise überlappend mit dem Forschungsgebiet des E-Learnings beziehungsweise speziell dem Begriff des Game-Based-Learning. Werden bei Computerspielen meist Fähigkeiten erlernt, die rein der Freizeitgestaltung dienen, legt das Game-Based Learning das Hauptaugenmerk auf das spielerische Erlernen spezieller Lerninhalte. Möchte man also im Spiel vorankommen, muss man zum Beispiel richtig Rechnen oder richtige Antworten zu bestimmten Themengebieten geben. Dies prägt den Begriff der Gamification, dabei steht „nicht die Produktion eines Spiels im Vordergrund, sondern vielmehr verschiedenste Applikationen mit spielerischen Elementen gezielt zu ergänzen“ (Le u. a., 2011, S. 8). (Ebner u. Schön, 2011)
In den folgenden Kapiteln wird auf einige dieser Bereiche genauer eingegangen, weiters werden praktische Beispiele zu den einzelnen Anwendungsgebieten geliefert.

2.2 E-Health

2.2.1 Einleitung

E-Health ist ein immer häufiger verwendeter Begriff und wird immer bedeutender, doch was bedeutet es eigentlich? E-Health ist mehr als nur die bloße Unterstützung von Medizin durch Technologie. Eysenbach versucht diesen weitläufigen Begriff in seinem Artikel „What is e-health?“ im Journal of Medical Internet Research wie folgt zu definieren (Eysenbach, 2001)

[1] Spiegel Online. Messwut im Hobbysport: Sensor unter der Sohle, 2014. http://www.spiegel.de/gesundheit/ernaehrung/sportler-messen-ihre-leistung-geringer-nutzen-fuer-das-training-a-827799.html, 12.03.2014.

[2] Wirtschafts Woche. Intelligente Kleidung - Der vernetzte Laufschuh, 2014. http://www.wiwo.de/unternehmen/industrie/intelligente-kleidung-der-vernetzte-laufschuh/5832276.html, 12.03.2014.

[3] Whatis.com. motion gaming, 2014. http://whatis.techtarget.com/definition/motion-gaming-motion-controlled-gaming, 15.02.2014.

„e-health is an emerging field in the intersection of medical informatics, public health and business, referring to health services and information delivered or enhanced through the Internet and related technologies. In a broader sense, the term characterizes not only a technical development, but also a state-of-mind, a way of thinking, an attitude, and a commitment for networked, global thinking, to improve health care locally, regionally, and worldwide by using information and communication technology.“ (Eysenbach, 2001, S. 1)

Laut Eysenbach kann das „E“ nicht nur für „electronic“ stehen, sondern vielmehr folgende Bedeutungen haben: (Eysenbach, 2001)

- **Efficiency:** E-Health soll also die Effizienz der Gesundheitsfürsorge verbessern und die Kosten reduzieren. Als Möglichkeit wird die Vermeidung von redundanten Untersuchungen genannt. (Eysenbach, 2001)
- **Enhancing quality:** Die Steigerung der Effizienz bedeutet nicht nur Reduzierung der Kosten, es soll auch die Qualität gesteigert werden. Dies könnte zum Beispiel durch Vergleiche von Anbieter bzw. Anbieterinnen und Zuweisung zum besten Arzt bzw. der besten Ärztin gewährleistet werden. (Eysenbach, 2001)
- **Evidence based:** Die Effizienz der Eingriffe in die Gesundheitsfürsorge durch E-Health sollten wissenschaftlich belegt werden. (Eysenbach, 2001)
- **Empowerment:** Durch die Ermächtigung von Anbieterinnen und Anbieter und Patientinnen und Patienten, Zugang zu Wissensgrundlagen der Medizin und persönlichen Patientendaten über das Internet zu bekommen, ergeben sich neue Möglichkeiten. (Eysenbach, 2001)
- **Encouragement:** Die Verbindung zwischen Patientinnen und Patienten und Gesundheitsexpertinnen und Gesundheitsexperten wird eine richtige Partnerschaft, in der Entscheidungen zusammen getroffen werden. (Eysenbach, 2001)
- **Education:** Dies bedeutet Zugang zu mehr Informationen für Gesundheitsexpertinnen und Gesundheitsexperten sowie zum Beispiel Informationen zu Vorsorgemöglichkeiten für Patientinnen und Patienten. (Eysenbach, 2001)
- **Enabling:** Ermöglichung von standardisiertem Informationsaustausch und Kommunikation zwischen Gesundheitseinrichtungen. (Eysenbach, 2001)
- **Extending:** Ausdehnung der bisher üblichen Betreuungsgrenzen, sowohl geographisch als auch konzeptuell. E-Health ermöglicht medizinische Services online zu erwerben. Dies können Informationen sowie Medikamente sein. (Eysenbach, 2001)

- **Ethics:** Durch die neuen Interaktionen zwischen Patientinnen und Patienten und Gesundheitseinrichtungen kommen natürlich neue Herausforderungen auf das System zu. Dies sind vor allem Themen die den Datenschutz und die Privatsphäre betreffen. (Eysenbach, 2001)

- **Equity:** E-Health kann die Behandlungen gerechter machen, allerdings könnte es auch in die entgegengesetzte Richtung gehen. Menschen die zum Beispiel keinen Zugang zu neuen Medien, Computern oder dem Internet haben, können die neuen Services nicht nutzen und erhalten deshalb eine schlechtere medizinische Versorgung. (Eysenbach, 2001)

Auch die Punkte *easy-to-use, entertainment und exciting* werden genannt und gehen mit dem Einsatz von digitalen mobilen Endgeräten einher. (Eysenbach, 2001)

Ähnlich sieht es auch die Weltgesundheitsorganisation (WHO), laut der E-Health als „Transfer zwischen Gesundheitsressourcen und Gesundheitsversorgung auf elektronischen Wege“ [1] definiert ist. Diese definiert des weiteren drei Hauptbereiche:

- Die Zustellung von Gesundheitsinformationen an Gesundheitsexpertinnen und Gesundheitsexperten, Gesundheitseinrichtungen und Patientinnen und Patienten durch das Internet,

- die Benutzung von Internet und E-Commerce zur Verbesserung von öffentlichen Gesundheitsservices sowie,

- die Benutzung von E-Commerce und E-Business Praktiken im Management des Gesundheitssystems. [1]

Grundsätzlich kann man folgende Formen des E-Health unterschieden: [2]

- **Information:** Informationen werden für Patientinnen bzw. Patienten und Ärztinnen bzw. Ärztinnen über Informationsportale bereitgestellt. [2]

- **Kommunikation:** Informationen können ohne zeitnahem Mitwirken der Kommunikationsparteien abgerufen werden. Als Beispiel könnte hier ein online Diabetestagebuch genannt werden. [2]

- **Interaktion:** Daten und Informationen können ausgetauscht werden und die

[1] World Health Organisation. E-Health, 2014. http://www.who.int/trade/glossary/story021/en/index.html, 13.02.2014.

[2] Wikipedia. E-Health, 2014. http://de.wikipedia.org/wiki/EHealth, 14.02.2014.

Kommunikationsparteien können zeitnah reagieren. Ein wichtiges Beispiel diesbezüglich wäre das *Home-Monitoring*, also die elektronisch unterstützte Überwachung von Patientinnen und Patienten zuhause. [1]
In dieses Gebiet fällt auch die in dieser Arbeit näher erklärte MotionTracker-Applikation, bei der automatisch ein Bewegungsprotokoll erstellt wird.

- **Transaktion:** Gezielter Datenaustausch mit dem Ziel, alle Leistungen elektronisch erfassen und so abwickeln zu können. [1] Praktisches Beispiel für diese Form ist die in Österreich eingeführte eCard.

- **Integration:** Darunter versteht man die lebenslange Aufzeichnung aller medizinisch relevanten Daten eines Patienten bzw. einer Patientin in Form von elektronischen Patientenakten. [1]
  Als aktuelles Beispiel kann das ELGA System, welches in Österreich Anfang 2014 eingeführt wurde, genannt werden. [2]

Speziell der Teilbereich Telemonitoring ist für diese Arbeit interessant und wird im folgenden Unterkapitel näher vorgestellt. Weiters werden einige praktische Anwendungen aufgelistet.

2.2.2 Telemonitoring

Unter Telemonitoring versteht man das Überwachen von Vitaldaten von Patientinnen und Patienten, wie zum Beispiel Herzfrequenz oder Blutdruck, die ihr gewöhnliches Leben im vertrauten Umfeld genießen können. Im Notfall kann ein überwachendes Institut beziehungsweise der überwachende Arzt oder die überwachende Ärztin eingreifen und zum Beispiel die Rettung alarmieren oder sonstige Schritte einleiten. Neben der Überwachung von Vitaldaten können auch andere Szenarien, wie zum Beispiel Sturzerkennung bei älteren Patienten, erkannt und notwendige Schritte eingeleitet werden. [3] (Almer, 2011)
Vor allem für chronisch kranke Menschen ist dies eine enorme Erleichterung, da eventuell überflüssige Kontrolltermine beim behandelnden Arzt bzw. der behandelnden Ärztin entfallen. Auch Menschen die in ländlicheren Gegenden leben, in deren Umfeld sich eventuell nicht unmittelbar ein Arzt bzw. eine Ärztin befindet oder die aus diversen Gründen Schwierigkeiten mit der Mobilität haben, profitieren speziell von derartigen Systemen. (Deutscher Bundestag, 2014)

[1] Wikipedia. E-Health, 2014. http://de.wikipedia.org/wiki/EHealth, 14.02.2014.

[2] ELGA GmbH. Homepage der ELGA GmbH, 2014. http://www.elga.gv.at/, 14.02.2014.

[3] Wikipedia. Telemonitoring, 2014. http://de.wikipedia.org/wiki/Home_Monitoring, 15.02.2014.

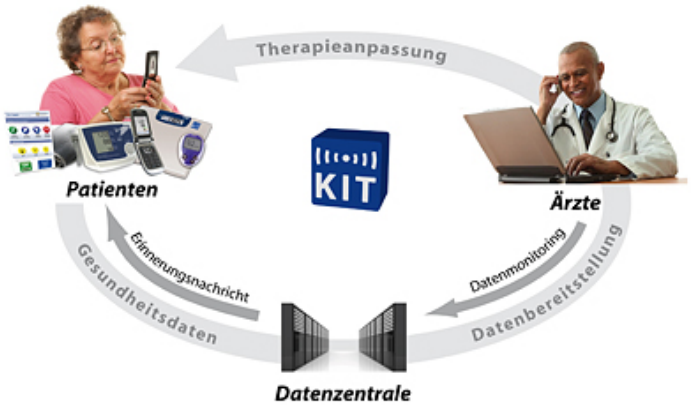


Abbildung 2.1: Veranschaulichung des Telemonitoring Systems des Austrian Institute of Technologie

Beispiel: **Telemonitoring des Austrian Institute of Technologie (AIT)**

Das AIT bietet Telemonitoring-Systeme für verschiedene Einsatzbereiche, wie zum Beispiel Herzinsuffizienz, Diabetes oder Bluthochdruck. In Abbildung 2.1 [1] sieht man eine vereinfachte Darstellung der Telemonitoring Systeme des AIT. [1]
Hervorzuheben ist hier die aktuelle Entwicklung des DIABMEMORY Systems. Dieses sammelt, berührungs- und kabellos über ein Mobiltelefon, selbst gemessene Daten von den jeweiligen Geräten (Blutzucker, Blutdruck und Gewicht) und sendet diese an eine zentrale Datenbank. Ärzte und Ärztinnen können dann über eine Webapplikation den Therapieerfolg und ähnliches kontrollieren, die gesammelten Daten werden hier übersichtlich aufbereitet. [2]
Dieses Diabetestagebuch erhielt auch den eT-Award für innovative E-Health Lösungen. [3]

Beispiel: **AutoAlert-System von Phillips zur Sturzerkennung**

Phillips bietet mit dem AutoAlert-System eine Sturzerkennungslösung an, die bei einem Sturz einen Alarmanruf an eine Zentrale auslöst. Dort können die MitarbeiterInnen

[1] Austrian Insitute of Technologie. Telemonitoring and Therapy Management, 2014. http://www.ait.ac.at/research-services/research-services-safety-security/health-information-systems/telemonitoring-and-therapy-management, 15.02.2014.

[2] Austrian Insitute of Technologie. Diabetestagebuch DiabMemory, 2014. http://www.ait.ac.at/research-services/research-services-safety-security/health-information-systems/telemonitoring-and-therapy-management/diabetes-mellitus/diabmemory-gesundheitsdialog-diabetes/?L=mggveicevxqku, 15.02.2014.

[3] eHealth 2011. eHealth eT-Award, 2011. http://www.eHealth2011.at, 15.02.2014.

dann die Situation klären und im Ernstfall einen Notruf absetzen. [1]

Beispiel: **FARSEEING-Projekt**

Im FARSEEING-Projekt wurde eine Applikation, die uTUG-Applikation, entwickelt, die anhand von Smartphone-Sensoren Stürze erkennen und Alarm schlagen kann. (Mellone u. a., 2012)

2.3 Motion-Gaming

2.3.1 Einführung

Motion-Gaming-Systeme bezeichnen Spielsysteme die es dem Spieler und der Spielerin erlauben das Spiel mit Bewegungen des eigenen Körpers zu steuern. [2]
Führt also zum Beispiel ein Spieler oder eine Spielerin in einem Box-Spiel einen Schlag aus, schlägt der Avatar im Spiel ebenfalls.
Entstanden ist dieser Trend im November 2006, als der Spielkonsolen Hersteller Nintendo, mit seiner neuen Konsole Nintendo Wii, das erste Motion-Gaming-System auf den Markt brachte. Mittlerweile konkurrieren Microsoft, mit dem Kincect-System für die X-Box 360, sowie Sony, mit seinem Move Add-on zur Playstation, mit Nintendo auf diesem Gebiet. [2]
Die Funktionsweise ist dabei unterschiedlich, die Nintendo Wii sowie die Playstation Move verwenden Fernbedienungen, die der Spieler bzw. die Spielerin in der Hand hält, welche mit Bewegungssensoren ausgestattet ist. Microsoft's Kinect hingegen ist Controller frei und benutzt eine Tiefenkamera zur Identifizierung der Spieler bzw. Spielerinnen und dessen Gesten. Die Spiele können so mit unterschiedlichen Bewegungen gesteuert werden. [2]

Ähnliche Bewegungssensoren sind heutzutage in fast allen Smartphones vorhanden, und werden von vielen Spieleherstellern zur Steuerung der Spiele verwendet. Dabei kann das Smartphone als Fernbedienung für Spielkonsolen dienen oder das Spiel am Gerät selbst steuern.

[1] Phillips Inc. AutoAlert System for fall detection, 2014. http://www.lifelinesys.com/content/lifeline-products/auto-alert, 15.02.2014.

[2] Whatis.com. motion gaming, 2014. http://whatis.techtarget.com/definition/motion-gaming-motion-controlled-gaming, 15.02.2014.

(a) Die Nintendo Wii Konsole (b) Der Nintendo Wii Plus Controller

Abbildung 2.2: Das Nintendo Wii Motion-Gaming-System

2.3.2 Spielkonsolen

In diesem Kapitel werden die drei wichtigsten Motion-Gaming-Systeme, das Nintendo Wii, das PlayStation Move sowie das Microsoft-Kinect-System, näher erläutert.

2.3.2.1 Nintendo Wii

Die Nintendo Wii Spielkonsole ermöglicht es dem Spieler oder der Spielerin mit dem Spiel anhand der eigenen Bewegungen zu interagieren. Herzstück dieser Konsole ist der Wii Remote Plus Controller, der dabei in der Hand gehalten werden muss. Dieser reagiert auf Beschleunigungen und Rotationen und kann dadurch Bewegungen wie Schwingen, Durchziehen, Stoßen oder Drehungen erkennen, anhand welcher zum Beispiel der Avatar bewegt werden kann. Dies funktioniert unter anderem mithilfe der im Controller eingebauten Accelerometer und Gyroscope-Sensoren, welche die Beschleunigung und Rotation im dreidimensionalen Raum messen. Die Kommunikation mit der Konsole selbst erfolgt kabellos bis zu zehn Meter via Bluetooth. Abbildung 2.2 [1] zeigt die Konsole sowie den Wii Remote Plus Controller. [1]

[1] Nintendo Inc. What is Wii?, 2014. https://www.nintendo.com/wii/what-is-wii/, 05.03.2014.

2.3.2.2 Playstation Move

Auch das PlayStation-Move-System setzt auf einen Controller, den Move Controller, der von den Spielerinnen und Spielern in der Hand gehalten werden muss. Er wird als Zubehör zur Spielkonsole PlayStation3 angeboten. [1]
Auch dieser besteht aus einem dreidimensionalen Gyroscope-Sensor zur Rotationsmessung, einem dreidimensionalen Accelerometer-Sensor zur Messung der Beschleunigung, einem Erdmagnetfeldsensor und einer leuchtenden Kugel, welche zum Tracking durch die PlayStation Eye Camera benötigt wird. Die Kommunikation erfolgt ebenfalls kabellos über Bluetooth. [2]
Abbildung 2.3 [1] zeigt die Konsole, die Eye Camera sowie den Move Controller.

2.3.2.3 Microsoft Kinect

Die Microsoft Kinect kommt komplett ohne zusätzlichen Controller aus, eine Tiefenkamera filmt den Spieler bzw. die Spielerin und das System erkennt dessen Gesten. So wird man selbst zum Controller und steuert die Spiele mit dem eigenen Körper. Die Kinect-Tiefenkamera wird in Kombination mit der Microsoft Xbox 360 Spielkonsole angeboten. Technisch wird dies mit Methoden aus dem Forschungsbereich Computer Vision umgesetzt. Beim Spielen werden alle Körperbewegungen erfasst und in Steuerbewegungen umgewandelt. Kinect merkt sich sogar die Gesichter der Menschen, die vor der Tiefenkamera stehen und man kann über Kinect ID einstellen, dass man automatisch mit seinem Profil angemeldet wird, wenn man vor die Kamera tritt. [3]
Abbildung 2.4 [3] zeigt diese Tiefenkamera.

2.3.3 Mobile Spiele

Viele Smartphones sind mittlerweile mit gängigen Bewegungssensoren ausgestattet, die ähnlich den in Kapitel 2.3 beschriebenen Technologien arbeiten. Diese können dann einen zusätzlichen Controller ersetzen. Das Apple iPhone zum Beispiel besitzt einen Accelerometer-Sensor, welcher die Beschleunigung und einen Gyroscope-Sensor, der die Rotation des Gerätes im dreidimensionalen Raum misst. Die Beschleunigungsdaten werden in der Einheit Meter pro Sekunde und die Rotationsdaten in Radiant pro Sekunde

[1] Sony Inc. This Is How I Move, 2014. http://us.playstation.com/ps3/playstation-move/, 05.03.2014.

[2] develop-online.net. Full Tech Specs: PlayStation Move, 2014. http://www.develop-online.net/tools-and-tech/full-tech-specs-playstation-move/0116722, 05.03.2014.

[3] Microsoft Inc. Kinect für Xbox 360, 2014. http://www.xbox.com/de-DE/Kinect/, 05.03.2014.

vom vorhergehenden Abtastwert gemessen. (Apple Inc., 2014)
Die Abbildung 2.5 veranschaulicht die Funktionsweise der beiden genannten Sensoren.

Beispiel: **Motion Tennis**
Beim Spiel Motion Tennis wird das iPhone zum Controller. Man kann es zum Beispiel in der Hand halten und wie im realen Tennis Vorhand- und Rückhandschläge ausführen. Diese werden mittels der eingebauten Bewegungssensoren erkannt und auf den Game-Charakter übertragen, der diese dann am Bildschirm ausführt. Dieses Prinzip wird in der Abbildung 2.6 dargestellt. Man sieht einen Spieler mit dem iPhone in der Hand, wie er den Avatar am Bildschirm steuert. (ROLOCULE GAMES PRIVATE LIMITED, 2014)

Beispiel: **Real Racing 2**
Dies ist ein sehr umfangreiches Autorennspiel, in dem man ebenfalls das iPhone als Lenkrad benutzen kann. Anhand der eingebauten Sensoren wird so der Grad der Lenkung bestimmt und auf das Fahrzeug übertragen. Anders als beim Spiel Motion Tennis, läuft dieses direkt am Gerät. (Firemint Pty Ltd, 2014)

2.4 Tragbare Datenverarbeitungssysteme im Sport

Sowohl im Profisport, als auch im Hobbybereich werden tragbare Datenverarbeitungssysteme (engl. Wearable Computing) immer mehr eingesetzt. Während es beim Profisport vor allem darum geht, die Trainingsabläufe zu verbessern und bei Wettkämpfen die Fans und Zuschauer mit Echtzeitinformationen zu versorgen, wird im Hobbybereich das Hauptaugenmerk auf die eigenen Werte im Vergleich zu anderen Sportlern gelegt. Man möchte wissen wer schneller und besser ist, sowie einen Vergleich zu Spitzensportlern ziehen. [1,2]
Diesen Megatrend der Selbsterfassung (engl. self tracking) verfolgen alle bekannten Anbieter, darunter Nike mit seinen *Nike+* Produkten, Adidas mit seinem Fußballschuh *adizero f50* und der miCoach-App oder auch SAP mit der *HANA*-Plattform.
Auch bei der diesjährigen *Consumer Electronics Show* in Las Vegas waren diese neuen Geräte sehr gefragt. Expertinnen und Experten schätzen, dass der Markt für Fitness- und mobile Geräte zur Selbstoptimierung, in Zukunft der Markt mit dem größten Wachs-

[1] Wirtschafts Woche. Intelligente Kleidung - Der vernetzte Laufschuh, 2014. http://www.wiwo.de/unternehmen/industrie/intelligente-kleidung-der-vernetzte-laufschuh/5832276.html, 12.03.2014.

[2] Spiegel Online. Messwut im Hobbysport: Sensor unter der Sohle, 2014. http://www.spiegel.de/gesundheit/ernaehrung/sportler-messen-ihre-leistung-geringer-nutzen-fuer-das-training-a-827799.html, 12.03.2014.

tumspotential sein wird. [1]
Abbildung 2.7 [2] zeigt den Adidas Speed_Cell Sensor, der z.B. im Fußballschuh adizero f50 eingebaut ist, den X_Cell Sensor und die Auswertung der miCoach App [3] als Beispiel für ein solches System. [2]

Die Funktionsweise ist ähnlich, die SportlerInnen tragen Sensoren in Kleidung, Pulsgurten, Schienbeinschoner oder Schuhen welche Daten an eine Basisstation liefern. Auch die Spielbälle sind oft mit Sensoren ausgestattet. Im Fußball wird zum Beispiel immer wieder über eine Tortechnologie diskutiert, bei welcher ein Sensor im Ball Auskunft darüber geben kann, ob sich dieser über der Torlinie befunden hat oder nicht. Über ein Smartphone oder Tablet, welches mit der jeweiligen App diese Daten grafisch für Trainer und Trainerinnen oder Zuschauer und Zuschauerinnen aufbereitet, bekommt man Informationen über Zustand, Einsatz und ähnliches in Echtzeit geliefert. Anwendung finden diese Systeme bereits sehr weitläufig, unter anderem wird das Adidas System mit der miCoach App von diversen Mannschaften der Major Soccer League verwendet. [4]

Diese intelligenten Sensoren kommunizieren miteinander via Funk und bilden ein sogenanntes Wireless-Body-Area-Network, also ein räumlich begrenztes drahtloses Netzwerk. Die Daten werden über dieses Netzwerk an eine Basisstation gesendet, welche dann eine weitere Verarbeitung vornimmt. Oft ist diese Basisstation ein Smartphone, dass dann die Daten auswertet und grafisch darstellt oder auch für einen späteren Gebrauch auf einen Server speichert. [5]
Abbildung 2.8 [5] veranschaulicht ein solches System, man sieht eine Sportlerin mit verschiedenen Sensoren, sowie das Smartphone als zentralen Knoten und mögliche Applikationen im Hintergrund.

[1] Klaus Dr. Krüger. SELF-TRACKING, SELBSTOPTIMIERUNG UND PSYCHOSOMATIK?, 2014. http://psychosomatik-info.com/self-tracking-selbstoptimierung-psychosomatik/, 04.04.2014.

[2] Adidas. miCoach Produkte, 2014. http://www.adidas.at/micoach?grid=true, 18.03.2014.

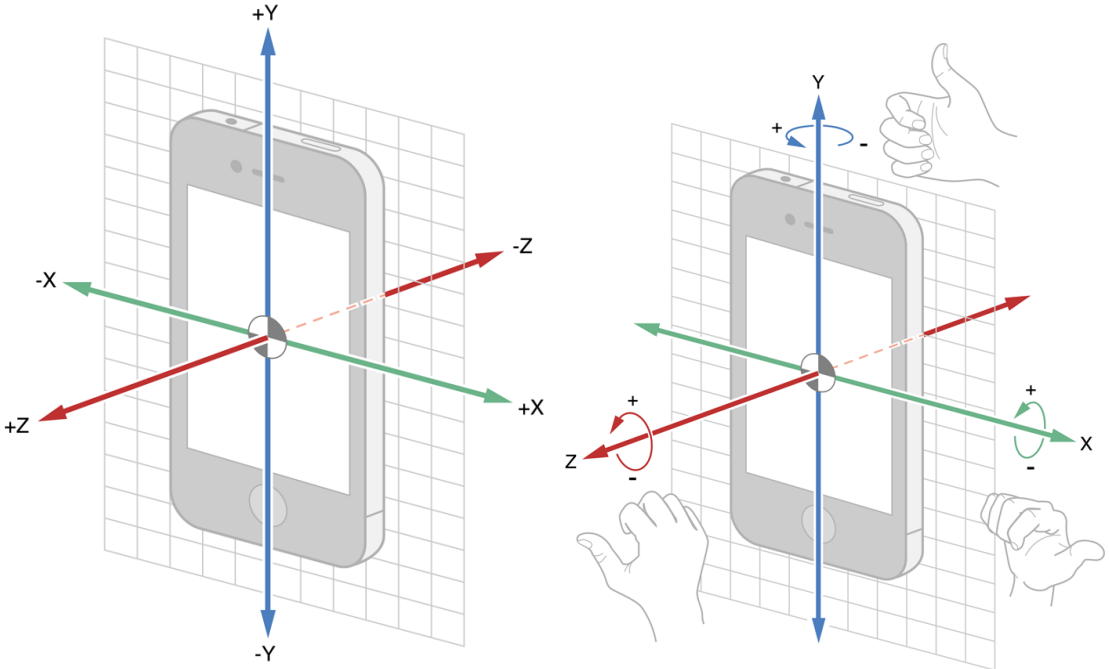
[3] Adidas. miCoach Multisport. iTunes, 2014. https://itunes.apple.com/us/app/micoach-multi-sport/id649765045?mt=8, 15.03.2014.

[4] Huffington Post. Data Tracking Devices Are Changing The Game For Sports, 2014. http://www.huffingtonpost.com/2013/10/22/athlete-data-tracking_n_4141956.html, 12.03.2014.

[5] Friedrich-Alexander-Universität Erlangen-Nürnberg. Applikationen für tragbare Sensor-Netzwerke, 2014. http://www5.cs.fau.de/lectures/ss-10/applikationen-fuer-tragbare-sensor-netzwerke-sem-app-sn/, 15.03.2014.

Abbildung 2.3: Das PlayStation-Move-Motion-Gaming-System

Abbildung 2.4: Die Microsoft-Kinect-Tiefenkamera zur Motion-Steuerung der Xbox 360



(a) Der Accelerometer-Sensor zum Messen der Beschleunigung in den 3D-Achsen (Apple Inc., 2014)

(b) Der Gyroscope-Sensor zum Messen der Rotation in den 3D-Achsen (Apple Inc., 2014)

Abbildung 2.5: Die Sensoren im Apple iPhone (Apple Inc., 2014, S. 61/63)

Abbildung 2.6: Beim Spiel Motion Tennis steuert der Spieler den Avatar über das iPhone (ROLOCULE GAMES PRIVATE LIMITED, 2014)

(a) Der Adidas miCoach-Speed_Cell-Sensor

(b) Der Adidas miCoach-X_Cell-Sensor

(c) Die Auswertung der Adidas miCoach-App

Abbildung 2.7: Bestandteile des Adidas miCoach-Systems

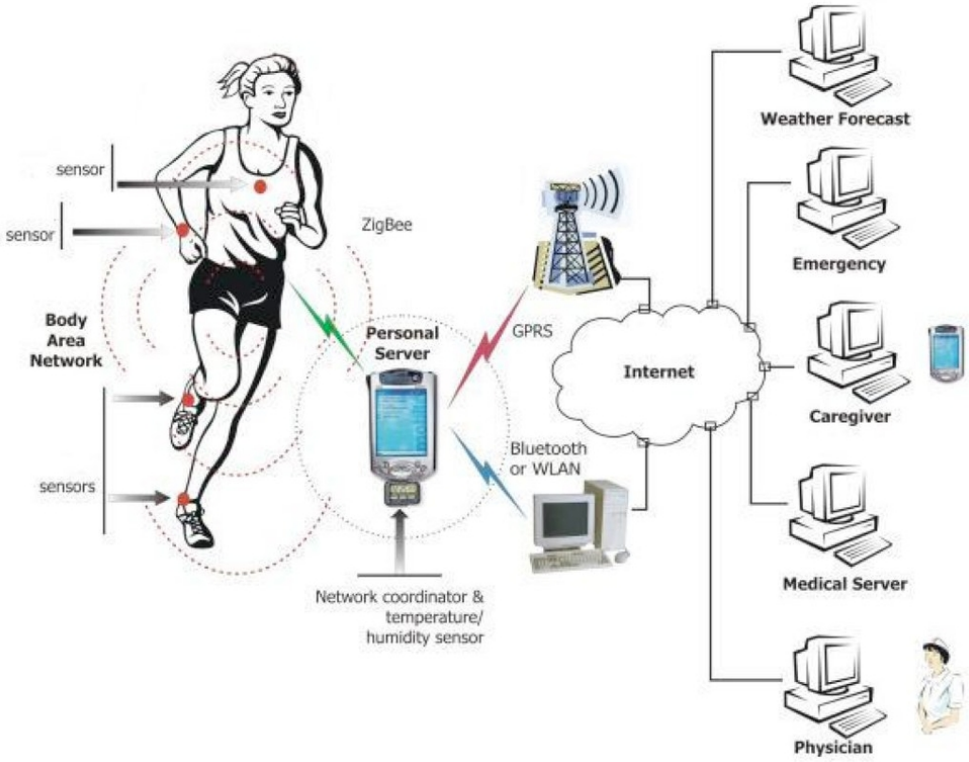


Abbildung 2.8: Beispiel für ein tragbares Sensor-Netzwerk

3 Umsetzung

3.1 Softwarearchitektur

Das MotionTracker-System besteht aus einem iOS-Client, einem Webservice zur Auswertung der Bewegungsdaten, sowie einer Webanwendung zur Veranschaulichung der Ergebnisse. Der iOS-Client, die MotionTracker-Applikation, ist für die Generierung der Bewegungsdaten während der Trainingseinheit, die Verbindung zum Server und die grafische Aufarbeitung der Rückmeldung des Servers verantwortlich. Das Webservice nimmt diese Daten entgegen, teilt sie dem jeweiligen Benutzer bzw. der jeweiligen Benutzerin zu und wertet sie aus. Die Rückmeldung wird dann wiederum im iOS-Client grafisch aufbereitet und liefert eine Übersicht über die absolvierten Übungswiederholungen sowie Verbesserungsvorschläge bei typischen Fehlern.

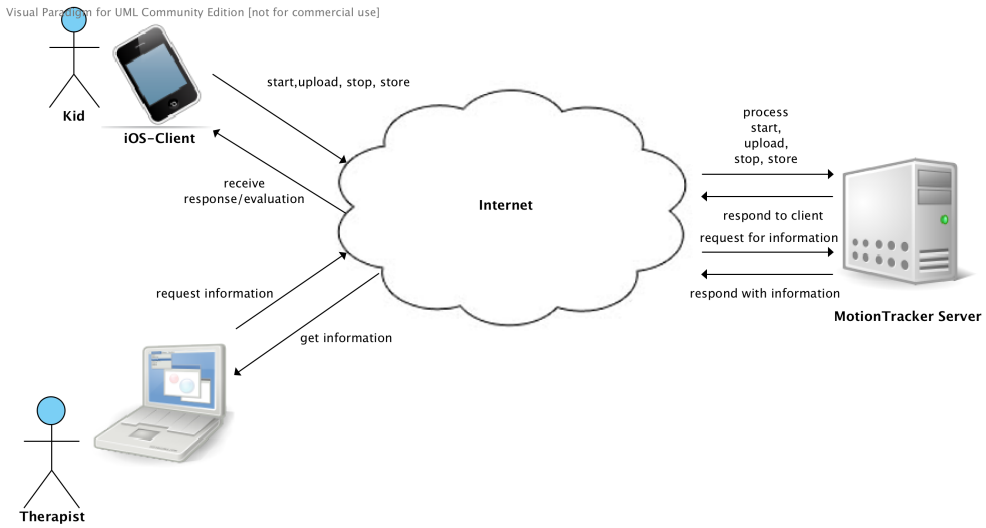


Abbildung 3.1: Übersicht des MotionTracker-Systems mit seinen Komponenten (eigener Entwurf)

Abbildung 3.1 zeigt diesen grundsätzlichen Aufbau, man sieht die Kommunikation des iOS-Clients mit dem MotionTracker Server, welcher die Rückmeldung an den Benutzer

bzw. die Benutzerin liefert. Des weiteren ist die Möglichkeit, Trainingseinheitsdetails und Trainingsvergangenheit über die MotionTracker-Webanwendung anzusehen, dargestellt.

3.1.1 Webserver

Der Server besteht aus einem Webservice, welches auf Auswertungsanfragen des Clients reagiert, sowie einer Webanwendung, welche erfasste Daten grafisch aufbereitet und dem Benutzer bzw. der Benutzerin zur Verfügung stellt.

3.1.1.1 Webservice

Das MotionTracker Webservice ist in der Programmiersprache Java implementiert und besteht aus folgenden Klassen und Objekten:

- **mobile_login.jsp**
  Diese JSP-Seite wird verwendet, um sich über die iOS-Applikation mithilfe des Google UserService Interfaces einzuloggen.

- **TestSVMServlet**
  Dieses Servlet reagiert auf Client-Anfragen, übernimmt die Authentifizierung der Benutzer und Benutzerinnen und regelt die Verarbeitung der Daten der Trainingseinheit.

- **SessionHandler**
  Singleton-Klasse (Gamma u. a., 1995), welche neue Sitzungen erstellt und diese verwaltet.

- **Session**
  Speichert die Bewegungsdaten und Auswertungsergebnisse eines Benutzers bzw. einer Benutzerin.

- **ClassificationStrategy**
  Interface welches die konkrete Implementierung der Klassifikationsmethode abkapselt. Diese sind mithilfe des Strategy-Entwurfsmusters (Gamma u. a., 1995) an die Session-Klasse gekoppelt.

- **Verschiedene NCCMethodStrategy Klassen**
  Implementierung der Klassifizierung anhand der Template-Matching-Methode, wie

in Kapitel 3.2.1.6 erklärt. Um verschieden große Ausschnitte miteinander vergleichen zu können, sind die Varianten *NCCLongStrategy*, *NCCShortStrategy* und NCCMultiTemplStrategy implementiert.

- **SVDSVMStrategy**
  Implementierung der Klassifizierung mithilfe von Support Vector Machines (siehe Kapitel 3.2.1.5), wobei als Merkmale nur die Singulärwerte der Sensordatenmatrix verwendet werden.

- **STDSVMStrategy**
  Implementierung der Klassifizierung mithilfe von Support Vector Machines (siehe Kapitel 3.2.1.5), wobei neben den Singulärwerten der Sensordatenmatrix zusätzlich noch weitere Merkmale verwendet werden.

- **SVMFactory und SVMFullFactory**
  Diese Klassen erzeugen die *SVDSVMStrategy- und STDSVMStrategy-* Objekte und sind im Erzeugungsmuster Fabrikmethode (Gamma u. a., 1995) implementiert. Die *TrainData*-Klasse liefert die dafür benötigten Trainingsdaten.

- **DatastoreUtils**
  Diese Klasse beinhaltet alle Methoden, die benötigt werden, den Datenbankzugriff zu regeln. Zu diesem Zweck wird *JDO* verwendet.

- Für die Implementierung wurden noch viele weitere Klassen verwendet, vor allem Datenklassen und Hilfsklassen. Diese sind allerdings für das Verständnis nicht notwendig und werden daher nur am Rande erwähnt:

  - **Exercise**
  - **ExerciseTemplate**
  - **FoundRecord**
  - **MotionExercise**
  - **MotionWorkout**
  - **MotionGroup**
  - **CalenderUtils, ClassificationUtils, DatatypeUtils, DifferentUtils, PMF**

Die Abbildung 3.2 zeigt ein UML-Diagramm, welches den Aufbau beschreibt, wobei nicht essentielle Klassen aus Gründen der Übersichtlichkeit nicht dargestellt sind. Das Servlet *TestSVMServlet* nimmt die Http-Post-Anfragen des iOS-Clients entgegen und leitet die Koordinierung der Auswertung. Da sich die Benutzer und Benutzerinnen am iOS-Client mit ihrem Google-Konto anmelden müssen, wird zu deren Identifizierung am Server das von Google bereitgestellte *UserService*-Interface[1] verwendet. Beim Start einer neuen Trainingseinheit wird im *SessionHandler* eine neue Sitzung erstellt, in welche dann die Daten gespeichert und dabei ausgewertet werden. Um diese Sitzungen zentral verwalten zu können, ist diese Klasse als Singleton-Entwurfsmuster implementiert (Gamma u. a., 1995). Die Auswertung übernehmen die unterschiedlichen Strategy-Klassen, welche aus Gründen der Austauschbarkeit mithilfe des Strategy-Entwurfsmuster an die *Session*-Klasse gekoppelt sind (Gamma u. a., 1995). In den konkreten Implementierungen der Klassifikationsmethoden *NCCTypeStrategy*, *SVDSVMStrategy* und *STDSVMStrategy* erfolgt die eigentliche Auswertung der Bewegungsdaten. Die Erzeugungsklassen *SVMFactory* und *SVMFullFactory* erzeugen die Strategien, welche auf Support Vector Machines beruhen und sind als Entwurfsmuster Fabrikmethode (Gamma u. a., 1995) implementiert.

3.1.1.2 Webanwendung

Die Webanwendung besteht aus mehreren JSP-Seiten, welche über die DatastoreUtils-Klasse verschiedene Informationen aus der Datenbank auslesen und grafisch aufbereiten. In Kapitel 3.4 wird auf diese Darstellungen näher eingegangen. Zur Authentifizierung der Benutzer wird hier ebenfalls das von Google bereitgestellte *UserService*-Interface [1] verwendet.
Zusätzlich zu den bereits in Kapitel 3.1.1.1 erwähnten Klassen umfasst die Webanwendung noch folgende JSP-Seiten:

- **checks.jsp**
  Authentifiziert den Benutzer bzw. die Benutzerin und erteilt die Autorisierung.

- **index.jsp**
  Landeseite, erzeugt mithilfe des Google *UserService*-Interface[1] eine Login-Seite oder leitet auf die *list_users.jsp* (Therapeut bzw. Therapeutin) beziehungsweise auf *list_user_workouts.jsp* Seite (normale Benutzer und Benutzerinnen sehen nur die eigenen Bewegungsprogramme) weiter.

- **list_users.jsp**

[1] Google Inc. Interface UserService, 2014. https://developers.google.com/appengine/docs/java/javadoc/com/google/appengine/api/users/UserService, 17.03.2014.

Listet alle BenutzerInnen, die bereits ein Bewegungsprogramm absolviert haben, auf.

- **list_user_workouts.jsp**
  Listet alle Bewegungsprogramme zu einem Benutzer bzw. einer Benutzerin auf.

- **show_workout.jsp**
  Zeigt die Detaildarstellung zu einem absolvierten Bewegungsprogramm.

- **show_data.jsp**
  Zeigt die Bewegungsdaten zu einer Wiederholung.

- **show_full_data.jsp**
  Zeigt die kompletten Bewegungsdaten.

Abbildung 3.3 zeigt ein UML-Diagramm dieses Teils des MotionTracker-Systems. Man sieht die oben angeführten JSP-Seiten und deren Verbindungen, wobei auch in diesem Diagramm, aus Gründen der Übersicht, für das Verständnis unwesentliche Klassen nur per Notiz erwähnt sind.

3.1.2 Client

Die iOS-Applikation besteht neben weiteren iOS spezifischen Klassen, welche hier aus Gründen der besseren Übersichtlichkeit ausgelassen werden, hauptsächlich aus dem *TrainingsViewController*, den verschiedenen Operationsklassen (*StartWorkoutOperation, StopWorkoutOperation, UploadPartOperation und StoreWorkoutOperation*) und verschiedenen Hilfs- und Datenklassen (*ContextUtils, NetworkUtils, Tracking und TrackingEntry*). Die Operationsklassen führen die jeweilige Operation aus, indem sie eine Http-Anfrage an den Server senden und den *TrainingsViewController* über die Rückmeldung informieren. Mehr Details zu diesen Punkten folgen in Kapitel 3.3.
Abbildung 3.4 veranschaulicht den Aufbau des wichtigen Teils der Applikation in einem UML-Diagramm. Man sieht die verschiedenen Komponenten und ihren Zusammenhang.

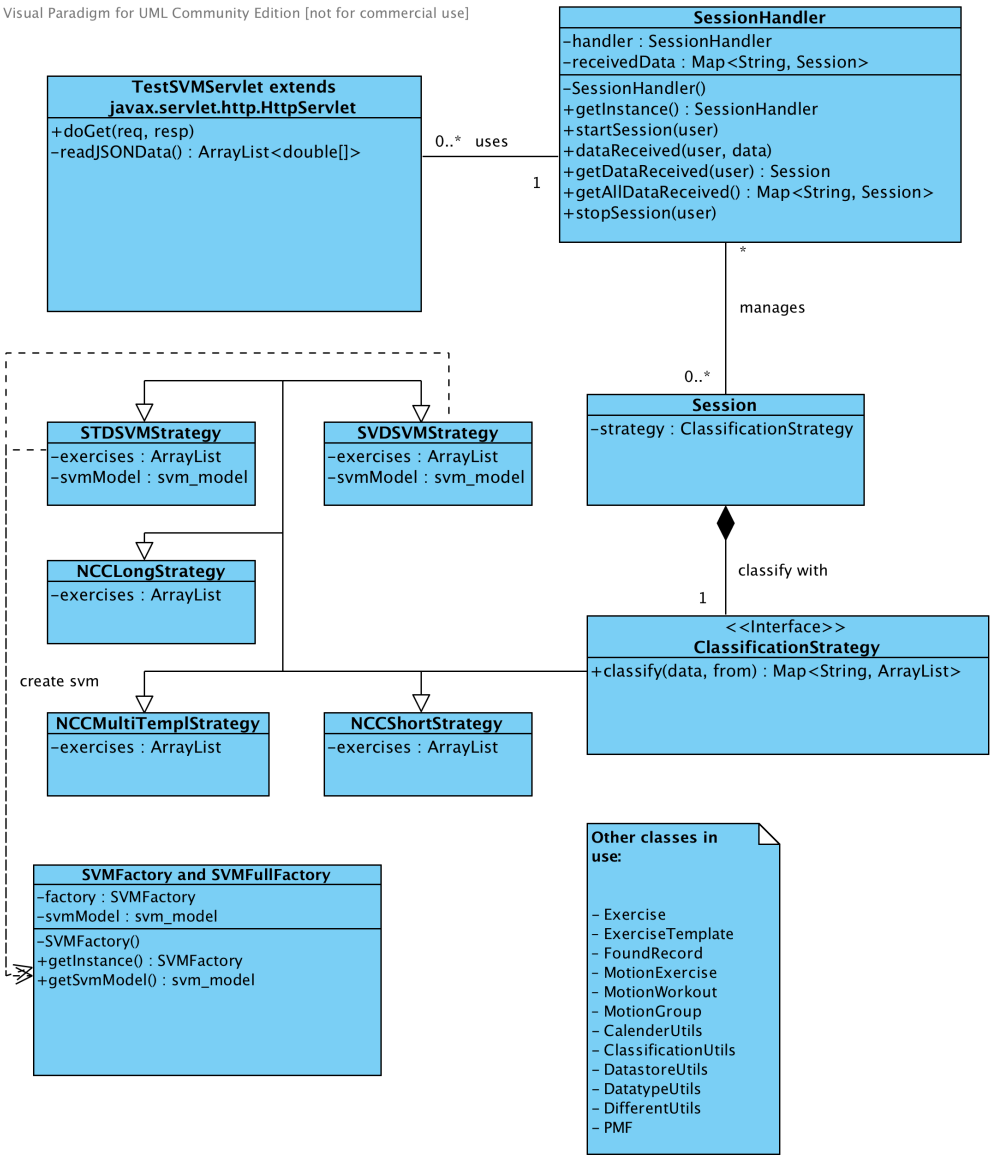


Abbildung 3.2: UML-Diagramm mit den relevanten Klassen des Webservices (eigener Entwurf)

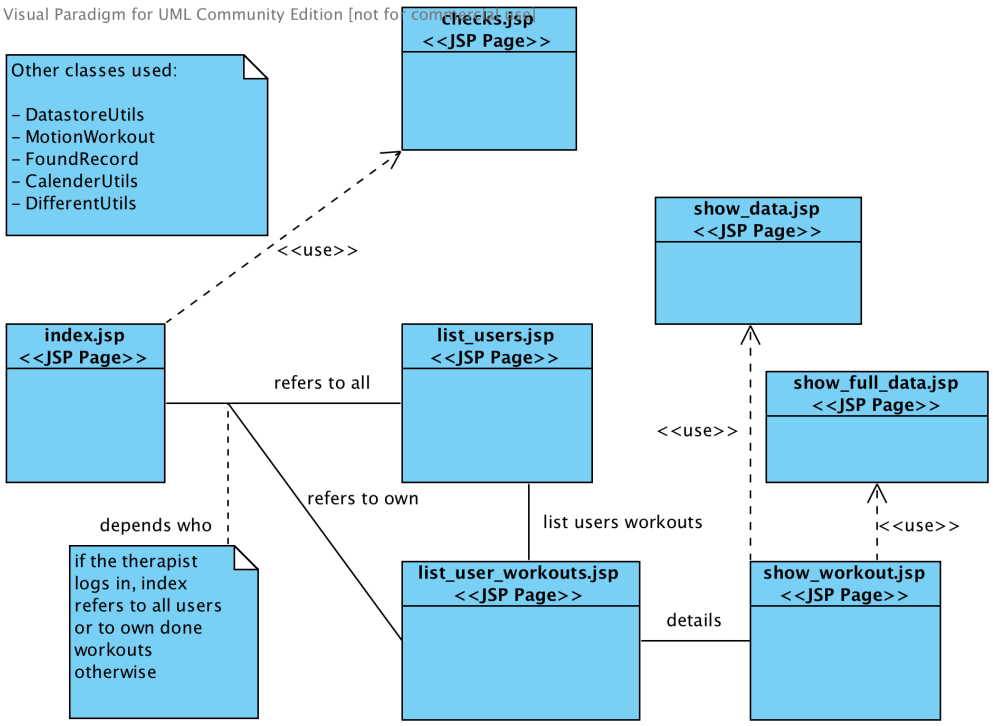


Abbildung 3.3: UML-Diagramm mit den relevanten JSP-Seiten der Webapplikation (eigener Entwurf)

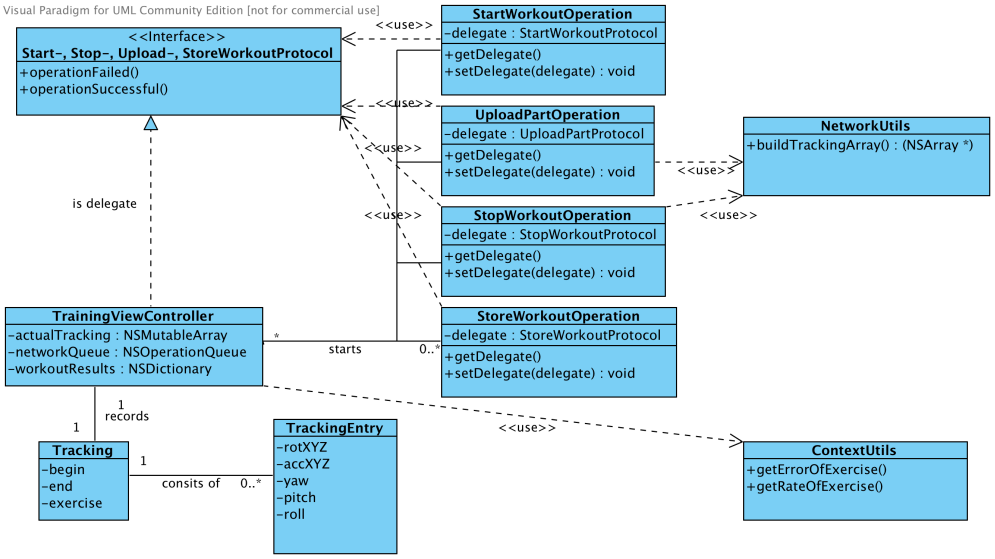


Abbildung 3.4: UML-Diagramm mit den relevanten Klassen des iOS-Clients (eigener Entwurf)

3.2 Der Klassifikationsprozess

In diesem Kapitel werden die Grundlagen der verwendeten Klassifikationsmethoden, sowie der eigentliche Klassifikationsprozess und andere untersuchte Methoden näher erläutert.

3.2.1 Einleitung

Die grundsätzliche Idee um verschiedene Bewegungen anhand eines iPhones erkennen zu können, ist Sensordaten, welche das Gerät liefert, zu analysieren und auszuwerten. In den folgenden Kapiteln werden die verfügbaren Daten sowie die verwendeten Methoden zur besseren Verständlichkeit kurz erklärt.

3.2.1.1 Das Bewegungsprogramm

Die MotionTracker-Applikation wurde entwickelt um die Übungen des Wiraculix Bewegungsprogrammes, welches im Rahmen einer Studie der Dissertation von Dietlind Deutschmann, *Kindgerechtes Pilates als Intervention zur Haltungsstabilisierung und Verbesserung von Wirbelsäulenfehlhaltungen bei SchülerInnen im Alter von 10-12 Jahren*, entworfen wurde, zu erkennen und zu protokollieren. Die teilnehmenden Schüler und Schülerinnen sollen dieses Bewegungsprogramm täglich zehn Minuten zuhause durchführen. (Deutschmann, 2014)

- **Über Kreuz**
  Bei dieser Übung legt man sich mit dem Rücken auf den Boden, wobei die Knie angewinkelt (90 Grad) werden. Weiters gibt man die Finger zu den Schläfen und drückt die Ellbogen auseinander. Nun hebt man den Kopf vom Boden ab und führt abwechselnd jeweils einen Ellbogen zum gegenüberliegenden Knie, das gleichzeitig ein wenig angezogen wird. Das zweite Bein wird zugleich ausgestreckt. (Deutschmann, 2014)

- **Wirbelsäulendrehung**
  Hier muss man sich im Langsitz oder im Schneidersitz ganz aufrecht hinsetzen, sodass der Oberkörper und die Beine einen rechten Winkel bilden. Nun hebt man die Arme gestreckt seitlich ab bis sie waagrecht sind. Nun wird der Oberkörper langsam in eine Richtung gedreht, wobei die Arme in dieser Position belassen werden müssen. Dabei wird der hinteren Hand nachgeschaut. Wenn man am äußersten Punkt angekommen ist, dreht man sich wieder zurück zur Mitte und

anschließend auf die andere Seite. Der Rücken muss dabei immer gerade bleiben. (Deutschmann, 2014)

- **Zirkuspferd**
  Beim Zirkuspferd muss man sich kniend und auf gestreckte Arme gestützt, die parallel zu hüftbreit geöffneten Oberschenkeln sind, positionieren. Mit Blick zum Boden wird nun immer ein Arm und das gegenüberliegende Bein gestreckt, bis sie parallel zum Boden sind. Diese Position muss kurz gehalten werden und anschließend wird der zweite Arm und wieder das gegenüberliegende Bein gehoben. (Deutschmann, 2014)

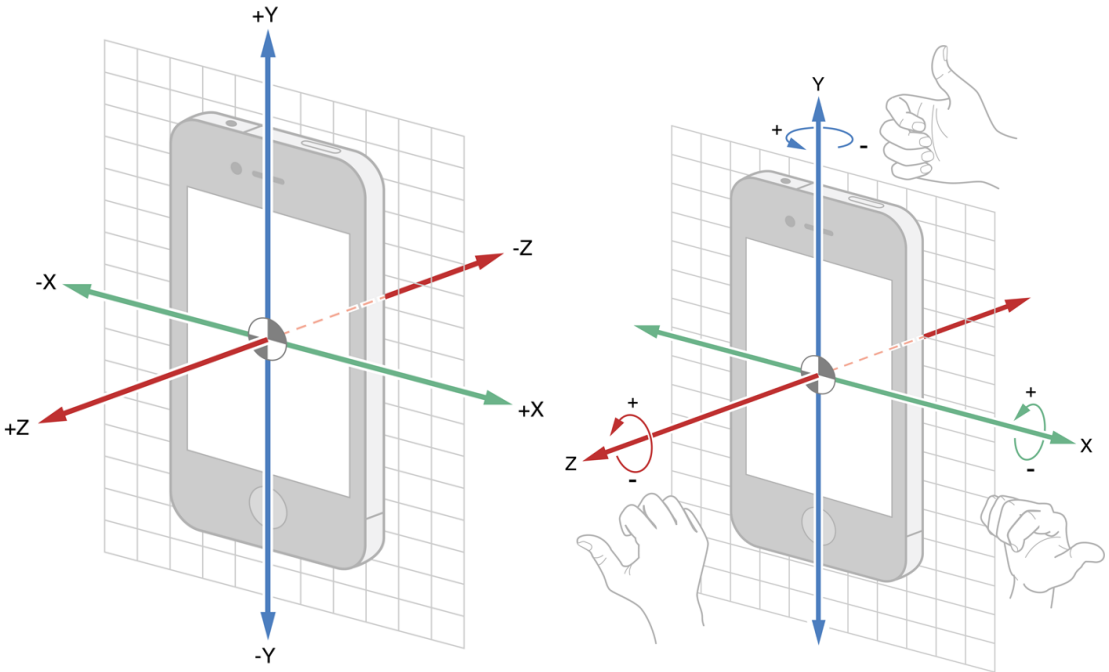
- **Rollender Ball**
  Der Rollende Ball beginnt am Boden sitzend und mit angezogenen Beinen, welche noch zusätzlich mit den Händen angezogen werden. Man zieht das Kinn zur Brust und blickt auf den Bauch, wodurch ein runder Rücken entsteht. Diese Position beibehaltend, lässt man sich nun kontrolliert nach hinten und wieder nach vorne rollen. Beim Vorrollen ist darauf zu achten, dass die Beine nicht abgesetzt werden. (Deutschmann, 2014)

- **Luftmatratze**
  Bei dieser Übung legt man sich auf den Bauch und streckt die Beine ganz lang aus, die Hände werden abgewinkelt und übereinander gelegt. Die Stirn muss auf den Händen abgelegt werden. Nun hebt man die Beine ein wenig hoch und bewegt sie mit schnellen, kurzen Bewegungen auf und ab, wobei diese ganz gestreckt bleiben müssen. Zusätzlich sollte der Kopf leicht angehoben und die Schultern nach hinten unten gezogen werden. (Deutschmann, 2014)

- **Raupe**
  Die Ausgangsposition der Raupe ist am Rücken liegend, mit aufgestellten Beinen und seitlich neben dem Körper gestreckten Armen. Nun wird das Gesäß so nach oben gehoben, dass Oberschenkel und Oberkörper eine gerade Linie bilden. Diese Position muss kurz gehalten werden und anschließend wird das Gesäß wieder abgesetzt. (Deutschmann, 2014)

3.2.1.2 Sensordaten im iPhone

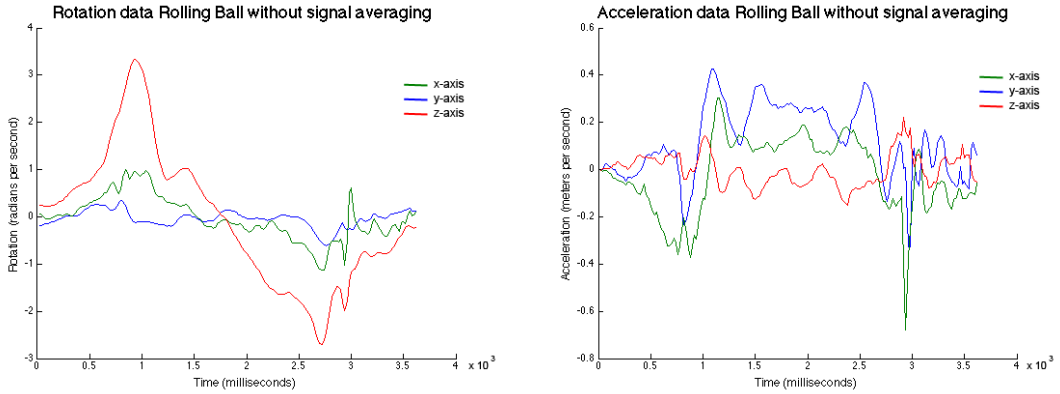
Die Grundlage für die Datengenerierung liefern die im iPhone eingebauten Bewegungssensoren, der Accelerometer-Sensor und der Gyroscope-Sensor. Es werden diese Sensordaten in einem bestimmten Zeitzyklus abgetastet und erzeugen so einen Datenstrom, der dann anhand von Mustererkennungsmethoden analysiert wird.



(a) Der Accelerometer-Sensor zum Messen der Beschleunigung in den 3D-Achsen (Apple Inc., 2014)

(b) Der Gyroscope-Sensor zum Messen der Rotation in den 3D-Achsen (Apple Inc., 2014)

Abbildung 3.5: Die Sensoren im Apple iPhone (Apple Inc., 2014, S. 61/63)



(a) Rotationsdaten

(b) Beschleunigungsdaten

Abbildung 3.6: Rotations- und Beschleunigungsdaten der Übung *Rollender Ball* (eigener Entwurf)

Accelerometer: Das Accelerometer ist ein Beschleunigungssensor, der die Beschleunigung des Gerätes, in den dreidimensionalen Achsen x, y und z in Meter pro Sekunde, liefert. (Apple Inc., 2014, S. 59-68)
Abbildung 3.5 zeigt die Auslegung der Achsen, sowie in welche Richtung positive und in welche negative Beschleunigung gemessen wird.

Gyroscope: Dieser Sensor liefert die Rotation des Gerätes, in den dreidimensionalen Achsen x, y und z in Radiant pro Sekunde. (Apple Inc., 2014, S. 59-68) Abbildung 2.5 zeigt die Auslegung der Achsen, sowie in welche Richtung positive und in welche negative Rotation gemessen wird.

Als Beispiel sieht man in Abbildung 3.6 die Rotations- bzw. Beschleunigungsdaten der Übung *Rollender Ball* (siehe Kapitel 3.2.1.1). Die Amplituden geben die Rotationen und Beschleunigungen in den verschiedenen Achsen an. Die bei dieser Bewegung typisch signifikante, rote Rotationsamplitude gibt die Rotationsbewegung in der z-Dimension an.

3.2.1.3 Datenvorverarbeitung

Um den generierten Datenstrom von möglichen Ausreißern und anderem Rauschen zu befreien, wird dieser geglättet. Anschließend erfolgt eine Segmentierung anhand einer einfachen Spitzenwerterfassung (engl. peak detection), welche dann die Eingangsbeispiele für die Mustererkennung liefert.

Signalglättung: Eine Methode zur Glättung eines Datenstromes ist die Faltung mit einem Rechteck-Filter. Dabei wird einfach jede Stelle des Datenstromes mit den Mittelwert ihrer Nachbarn ersetzt. [1]

Segmentierung: Um mögliche Übungen zu erkennen wird der Datenstrom anhand von Extrempunkten segmentiert. Diese möglichen Übungen liefern dann die Eingangsbeispiele für die Mustererkennung.

3.2.1.4 Mustererkennung

Unter Mustererkennung versteht man den Prozess, unverarbeitete Eingangsdaten verschiedenen Musterkategorien zuzuweisen und aufgrund dieser Zuweisung verschiedene Aktionen abzuleiten. In diesen Bereich fallen speziell Probleme, die für Menschen selbstverständlich, jedoch für Maschinen sehr schwer nachzubilden und zu automatisieren sind. Klassische Beispiele sind das Erkennen von Gesichtern und Menschen (Face recognition) oder das Verstehen von handgeschriebenen oder gesprochenen Wörtern (Optical character recognition bzw. automated speech recognition). Es wird versucht, einzelne Merkmale aus Daten, von denen die zugehörige Musterkategorie bekannt ist, zu extrahieren und anhand dieser Modelle zu erstellen, welche für künftige, unbekannte Daten die zugehörige Kategorie liefern. (Duda u. a., 2000, S. 1ff)
Um diese Aufgabe ausführen zu können, benötigt man als erstes einen Klassifikator, der die gewünschten Ergebnisse liefern soll. Abbildung 3.7 zeigt den grundsätzlichen Ablauf bei der Entwicklung eines solchen Klassifikators, auf die einzelnen Phasen wird in den kommenden Absätzen genauer eingegangen.

Datenbeschaffung: Bei der Datenbeschaffung wird versucht typische und repräsentative Beispiele für die Kategorien, welche später klassifiziert werden sollen, zu finden. Diese Datenbeispiele mit bekannter Musterkategorie werden als *Trainingsdaten* bzw. *Testdaten* bezeichnet. Ein gutes Mustererkennungssystem ist abhängig von der Qualität dieser Trainingsdaten und die Beschaffung nimmt oft einen großen Teil im Entwickeln eines solchen Systems ein. Die Anzahl sowie die repräsentative Fähigkeit ist oft ausschlaggebend für die Qualität der Klassifizierung. (Duda u. a., 2000, S. 14)

Selektion der Merkmale: Aus diesen Trainingsdaten können dann verschiedene Merkmale extrahiert werden, welche zur Klassifizierung herangezogen werden. Die Auswahl

[1] Wikipedia. Glätten (Mathematik), 2014. http://de.wikipedia.org/wiki/Glätten_(Mathematik), 17.02.2014.

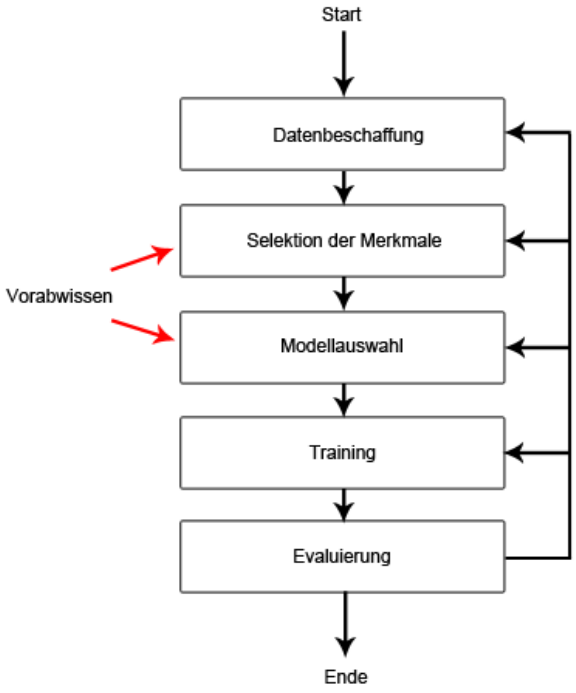


Abbildung 3.7: Der klassische Entwicklungszyklus eines Mustererkennungssystems (in Anlehnung an (Duda u. a., 2000, S. 14))

der trennfähigsten aus der Menge aller möglichen Merkmale ist oft ein kritischer Entwicklungsschritt und ist abhängig von den Charakteristiken der Problemdomäne. Das Vorabwissen zur Problemdomäne spielt also eine entscheidende Rolle. Bei der Gesichtserkennung könnten zum Beispiel die Anzahl der Augen sowie die Positionen von Nase und Mund aussagekräftige Merkmale bilden. (Duda u. a., 2000, S. 14f)

Modellauswahl: Bei der Modellauswahl geht es darum, ein geeignetes mathematisches Modell, zur Vorhersage der Kategorie von künftigen Daten, zu finden. Dieses sollte möglichst das richtige, dem Muster zugrundeliegende Modell treffen, um gute Ergebnisse erzielen zu können. (Duda u. a., 2000, S. 15)
Verschiedene Modelle haben unterschiedliche Parameter, die die Komplexität beeinflussen. In praktischen Applikationen müssen die Parameterwerte so optimiert werden, dass die besten Resultate bei der Vorhersage auf neue Daten erzielt werden. (Bishop, 2006, S. 32f)
Für diese Arbeit wurden *Support Vector Machines*, wie in Kapitel 3.2.1.5 beschrieben, als mathematisches Modell gewählt.

Training: Mithilfe der von den Trainingsdaten extrahierten Merkmale wird das ausgewählte mathematische Modell erstellt, welches dann anhand der Testdaten evaluiert wird. Dieser Schritt wird auch als Training des Klassifizierers bezeichnet. (Duda u. a., 2000, S. 15)

Evaluierung: In diesem Schritt wird der Klassifizierer anhand der Testdaten evaluiert. Dies ist wichtig, um nötige Verbesserungen, wie zum Beispiel die Verwendung einer anderen Auswahl an Merkmalen oder eines anderen Modells, zu erkennen und umzusetzen. (Duda u. a., 2000, S. 15f)
Die einfachste Methode zur Bemessung der Qualität eines Klassifizierers ist die *Fehlerrate*, welche die prozentuale Zuordnung von neuen Mustern zu falschen Kategorien angibt. (Duda u. a., 2000, S. 13)
Weiters ist dieser Schritt wichtig um das bekannte Problem der Überanpassung (Overfitting) zu vermeiden. Dieses tritt auf, wenn zu viele Merkmale zur Klassifizierung, und somit ein zu komplexes System, verwendet werden und äußert sich durch sehr gute Ergebnisse auf die Trainingsdaten, aber schlechte auf neue Muster. Die richtige Komplexität des Systems ist wesentlich für die Qualität des Klassifizierungssystems. Ist sie zu einfach, kann sie die verschiedenen Kategorien nicht unterscheiden bzw. bedeutet eine zu hohe Komplexität schlechte Ergebnisse auf neue Muster. (Duda u. a., 2000, S. 16)

Der so gewonnene Klassifikator wird im Klassifizierungssystem zur automatischen Entscheidung verwendet. Abbildung 3.8 zeigt den typischen Aufbau eines Klassifizierungssystems, welches in die Komponenten *Erfassung, Segmentierung, Merkmalsgewinnung, Klassifizierung und Nachbearbeitung* aufgeteilt werden kann. (Duda u. a., 2000, S. 9ff)

Erfassung: Unter dem Erfassungsschritt wird die Datengewinnung verstanden. Die Eingabedaten zu einem Mustererkennungssystem sind meist verschiedene Arten von Signalgebern, wie zum Beispiel Sensoren, Kameras oder Mikrophone. Es werden also je nach System die Sensorinformationen ausgelesen, Bilddaten erzeugt etc. (Duda u. a., 2000, S. 9)

Segmentierung: Bei der Segmentierung werden die einzelnen Muster aus den Eingangsdaten extrahiert. Man versucht also Eingangsbilder oder Sensordaten etc. so zu segmentieren, dass man die einzelnen Beispiele erhält, um sie weiterverarbeiten zu können. (Duda u. a., 2000, S. 9f)

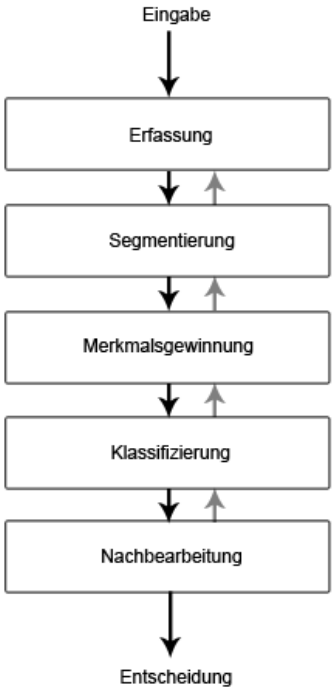


Abbildung 3.8: Die klassischen Komponenten eines Mustererkennungssystems (in Anlehnung an (Duda u. a., 2000, S. 10))

Merkmalsgewinnung: In diesem Schritt werden aus den segmentierten Beispielen die zur Klassifizierung verwendeten Merkmale extrahiert. Diese sollten möglichst gleich für alle in einer Kategorie befindlichen Muster, sowie möglichst unterschiedlich zu anderen Kategorien sein, um die Klassifizierung durch den Klassifikator so einfach wie möglich zu gestalten. Die Ausgabe der Merkmalsgewinnung ist der sogenannte Merkmalsvektor (feature vector), der die extrahierten Merkmale (features) enthält. (Duda u. a., 2000, S. 11f)

Klassifizierung: Der trainierte Klassifizierer hat in diesem Schritt die Aufgabe, die Muster anhand des Mermalsvektors den verschiedenen Kategorien zuzuordnen. Da eine perfekte Klassifikation von Muster in der Praxis oft nicht möglich ist, wird meist die Wahrscheinlichkeit aller möglichen Kategorien bestimmt und die höchste ausgewählt. (Duda u. a., 2000, S. 12f)

Nachbearbeitung: Meist wird das Klassifizierungssystem nicht isoliert benötigt, sondern sollen aus dem Ergebnis gewisse Aktionen abgeleitet werden. Eine solche könnte

zum Beispiel bei einem Sortierungssystem für Fischarten das Befördern eines Fisches in den jeweiligen Korb seiner Art, von einem Fließband, sein. Diese Ableitung wird in diesem Schritt durchgeführt. (Duda u. a., 2000, S. 13f)

3.2.1.5 Support Vector Machines

Support Vector Machines (SVM) sind eine Supervised-Learning-Methode zur Klassifizierung von Daten. Anhand von Trainingsdaten und den dazugehörigen Zielwerten wird ein Modell produziert, welches die Zielwerte für Testdaten vorhersagen kann. Die Trainingsdaten bestehen aus Paaren von Merkmalen (features), was verschiedene Kennzahlen der Daten sein können, sowie deren Zielwerten $(x_i, y_i), i = 1, ..., l \; where \; x_i \in R^n \; and \; y \in \{1, -1\}^l$. (Hsu u. a., 2010, S. 1)
Dabei ergibt sich das in Formel 3.1 dargestellte Optimierungsproblem (Boser u. a., 1992; Cortes u. Vapnik, 1995) (Bishop, 2006, S. 340):

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ subject\,to \quad & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{3.1}$$

Die Trainingsvektoren x_i werden durch die Funktion ϕ in eine höhere Dimension (kann auch unendlich sein) abgebildet. Die Support Vector Machine findet nun eine sogenannte *linear separierende Hyperplane*, welche den Abstand zwischen den verschiedenen Gruppen der Trainingsdaten, im höher dimensionalen Raum, maximiert. Parameter C ist der Strafparameter des Fehlerterms und die Funktion $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ wird Kernelfunktion genannt. (Hsu u. a., 2010, S. 2)
Meistens werden folgende vier Basis-Kernelfunktionen verwendet (Hsu u. a., 2010, S. 2):

- **linear:** $K(x_i, x_j) = x_i^T x_j$.
- **polynomial:** $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$.
- **radial basis function (RBF):** $K(x_i, x_j) = exp(-\gamma ||x_i - x_j||^2), \gamma > 0$.
- **sigmoid:** $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

In der Praxis begegnet man oft dem Problem, nicht nur zwei Klassen klassifizieren zu müssen, sondern mehrere. In diesem Fall kombiniert man mehrere Zwei-Klassen-

Klassifikatoren K_n, wobei immer die jeweiligen Trainingsbeispiele der aktuellen Klasse K_i als positive und die der restlichen Klassen $K_0 \cdots K_{i-1}, K_{i+1} \cdots K_n$ als negative Zielwerte verwendet werden. (Bishop, 2006, S. 338ff)

3.2.1.6 Template Matching

Template Matching stammt aus dem Fachbereich *Computer Vision* und dient dazu, ähnliche Bereiche in Bildern zu erkennen. Es ist also eine Technik, die in einem Bild Bereiche findet, welche einem Template (Beispielbild) ähneln. [1]
Als Ähnlichkeitsmaß können verschiedene Funktionen dienen, in dieser Arbeit wurde die, von Richard Szeliski in seinem Computer Vision Book vorgeschlagene, Normalized Cross-Correlation verwendet. (Szeliski, 2010, S. 386f)

$$E_{NCC}(u) = \frac{\sum_i [I_0(x_i) - \bar{I_0}][I_1(x_i + u) - \bar{I_1}]}{\sqrt{\sum_i [I_0(x_i) - \bar{I_0}]^2} \sqrt{\sum_i [I_1(x_i + u) - \bar{I_1}]^2}} \tag{3.2}$$

where

$$\bar{I_0} = \frac{1}{N} \sum_i I_0(x_i), \qquad \bar{I_1} = \frac{1}{N} \sum_i I_1(x_i + u) \tag{3.3}$$

In der Formel 3.2 entspricht I_0 dem Template und es wird im Bildausschnitt I_1 an der Stelle u das Ähnlichkeitsmaß, die Normalized Cross-Correlation (NCC), berechnet. $\bar{I_0}$ bzw. $\bar{I_1}$, aus der Formel 3.3, entsprechen den Mittelwerten des Templates bzw. des Bildausschnittes. (Szeliski, 2010, S. 386f)

[1] OpenCV. Template Matching, 2014. http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html, 19.02.2014.

3.2.2 Mustererkennung

In diesem Kapitel wird auf die Implementierung der Mustererkennung eingegangen und die einzelnen Schritte genauer erklärt.

3.2.2.1 Trainingsdaten Generierung

Bevor mit der Implementierung eines Klassifizierungssystems, zur Erkennung der Übungen des Bewegungsprogrammes laut Kapitel 3.2.1.1, begonnen werden konnte, mussten möglichst viele Trainingsbeispiele generiert werden. Dazu wurden freiwillige Probanden und Probandinnen an verschiedenen Schulen und anderen Umfeldern gesucht, welche dann mit einem iPhone, auf dem ein Prototyp der MotionTracker-Applikation (siehe Kapitel 3.3) installiert war, diverse Übungen absolvieren mussten. Mithilfe dieser Daten konnten erste Analysen der Datenkurven durchgeführt werden.
In Abbildung 3.9 sieht man einen Probanden, der im Rahmen eines Schulbesuches die Übungen absolvierte.

Abbildung 3.9: Proband mit angeschnalltem iPhone am rechten Oberarm

3.2.2.2 Vorverarbeitung

Um stabilere Resultate zu bekommen, wurden die Datenkurven verschiedenen Vorverarbeitungsschritten unterzogen. Diverses Rauschen wurde durch die Implementierung einer Signalglättung verringert. Da die verschiedenen Übungen in unterschiedlichen Geschwindigkeiten ausgeführt werden und ruckartige Bewegungen oder längere Pausen zu unterschiedlichen Amplituden führen, wurden zur besseren Segmentierung teilweise verschiedene Stärken der Glättung benutzt. Dadurch konnte eine grobe Kurve, die nur noch die signifikanten Amplituden aufwies, erzeugt werden. Die Verwendung dieser Kurve zur Segmentierung liefert stabilere und genauere Ergebnisse.
Die Kurve wurde auf wie in Gleichung 3.4 ersichtlich geglättet.

$$x(i) = \frac{1}{2n+1} \sum_{j=i-n}^{i+n} x(j), \qquad i > n \, and \, i < N - n. \tag{3.4}$$

Der Wert für einen Datenpunkt ergibt sich also durch den Durchschnitt der benachbarten Punkte und dem Datenpunkt selbst. Für die Merkmalsgewinnung (feature extraction, Vergleich Kapitel 3.2.2.4) werden vier benachbarte Punkte, also zwei zeitlich vor dem betrachteten Punkt und zwei danach, berücksichtigt und die Glättung fünfmal wiederholt. Die Wiederholung des Vorganges dient der besseren Glättung des Rauschens. Dieser Vorgang wird in Abbildung 3.10 veranschaulicht. Man sieht den betrachteten Datenpunkt zum Zeitpunkt *t*, und seine vier berücksichtigten, benachbarten Punkte *t-2* - *t+2*, welche in grün dargestellt sind. Die Punkte *t-3* bzw. *t+3* werden bei der Mittelwertberechnung nicht berücksichtigt und sind in rot gekennzeichnet.

Durch diverse Tests und Versuche wurden folgende Glättungsparameter für die verschiedenen Übungen festgelegt:

- **Über Kreuz:** Zehn benachbarte Punkte, zehnmal ausgeführt
- **Wirbelsäulendrehung:** Zehn benachbarte Punkte, 30 mal ausgeführt
- **Zirkuspferd:** Zehn benachbarte Punkte, zehnmal ausgeführt
- **Rollender Ball** vier benachbarte Punkte, 15 mal ausgeführt
- **Luftmatratze** zwei benachbarte Punkte, fünfmal ausgeführt
- **Raupe** Zehn benachbarte Punkte, zehnmal ausgeführt

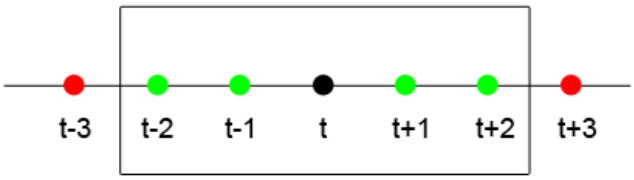


Abbildung 3.10: Signalglättung aller Beispiele, es werden der betrachtete und seine vier benachbarten Punkte berücksichtigt (eigener Entwurf)

Die unterschiedliche Anzahl an betrachteten Punkten sowie der Wiederholungen ergeben sich dadurch, dass beispielsweise langsam auszuführende Übungen mehr Rauschen durch zittrige Bewegungen oder Pausen enthalten als schnell auszuführende. Wie bereits erwähnt dienen die unterschiedlich geglätteten Kurven nur der Segmentierung anhand der Amplituden (Peaks). Für die Merkmalsgewinnung werden alle Beispiele der gleichen Glättung, von vier benachbarten Punkten und fünfmaliger Wiederholung, unterzogen. Quellcode 3.1 zeigt die Implementierung dieses Vorganges in Java.

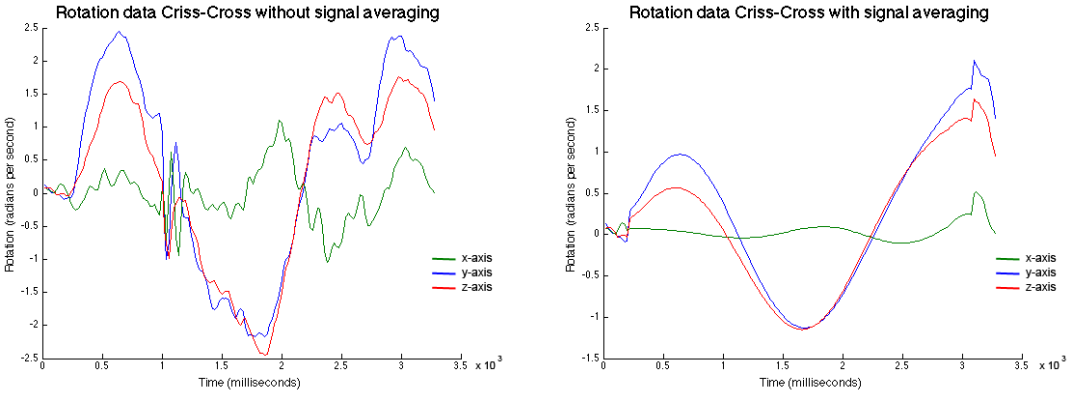
```
public static ArrayList<double[]> averageSignal(
                                            List<double[]> data,
                                            int times,
                                            int points) {

        ArrayList<double[]> averagedSig = new ArrayList<double[]>(data);
        int dataLength = data.size();
        int rowLength = 0;

        if (dataLength > 0)
                rowLength = data.get(0).length;

        for (int cntTimes = 0; cntTimes < times; cntTimes ++) {
                ArrayList<double[]> dataTMP =
                        new ArrayList<double[]> (averagedSig.size());
                dataTMP.addAll(averagedSig.subList(0, points));

                for (int i=points; i < dataLength - points; i++) {
                        List<double[]> actPart =
```

```
                        averagedSig.subList(i-points, i + points + 1);

                double[] data_avg = new double[rowLength];

                Iterator<double[]> actPartIt = actPart.iterator();
                while (actPartIt.hasNext()) {
                        double[] actRow = actPartIt.next();
                        for (int row = 0; row < actRow.length; row ++) {
                                data_avg[row] += actRow[row];
                        }
                }

                for (int row = 0; row < data_avg.length; row ++) {
                        data_avg[row] /= (2*points+1);
                }

                dataTMP.add(data_avg);
        }

        dataTMP.addAll(averagedSig.subList(
                        dataLength-points,
                        dataLength));
        averagedSig = dataTMP;
    }
    return averagedSig;
}
```

Quellcode 3.1: Implementierung der Signalglättung in Java

3.2.2.3 Segmentierung

Um das Problem, den Anfang und das Ende einer Übung im Datenstrom zu finden, zu lösen, wird der Datenstrom einer groben Klassifizierung anhand der Extrempunkte unterzogen. Damit lokale Maximal- und Minimalpunkte das Ergebnis nicht verfälschen, wurde eine starke Glättung, wie in Kapitel 3.2.2.2 beschrieben, angewandt. Die Segmentierung erfolgt aufgrund der Rotationsdaten, da diese stabilere Kurven liefern.
Bei einer näheren Analyse der Bewegungsdaten der Übungen ergeben sich verschiedene grobe Strukturen. Jede Übung weist eine signifikante Rotationskurve in einer bestimmten Dimension auf, was zur groben Segmentierung verwendet wird. Die Abbildungen 3.11, 3.12, 3.13, 3.14, 3.15 und 3.16 zeigen die Rotationsdaten der verschiedenen Übungen mit und ohne Vorverarbeitung.
In Quellcode 3.2 wird beispielhaft die Funktion *findSpineTwistSamples* gezeigt, welche mögliche Versuche der Übung *Wirbelsäulendrehung* in einem Datenstrom findet. Diese



(a) Rotationsdaten ohne Signalglättung (b) Rotationsdaten mit Signalglättung

Abbildung 3.11: Rotationsdaten mit und ohne Signalglättung der Übung Über Kreuz (eigener Entwurf)

Vorgehensweise ist für die restlichen Übungen weitestgehend gleich, nur die jeweiligen Dimensionen und Musterabfragen unterscheiden sich.

- **Über Kreuz:** Die Übung *Über Kreuz* weist vor allem in der x-Dimension eine signifikante Rotation auf. Die geglättete Rotationskurve, wie in Abbildung 3.11 (b) ersichtlich, wird durch einen Hochpunkt von mehr als *+0.2 rad/sec*, gefolgt von einem Tiefpunkt von weniger als *-0.2 rad/sec* und erneut einem Hochpunkt von mehr als *-0.1 rad/sec* gekennzeichnet. Man kann hier erkennen, dass diese Klassifikation sehr grob ist, da die Werte weit über den Schwellwerten liegen, um alle möglichen Versuche erkennen zu können.

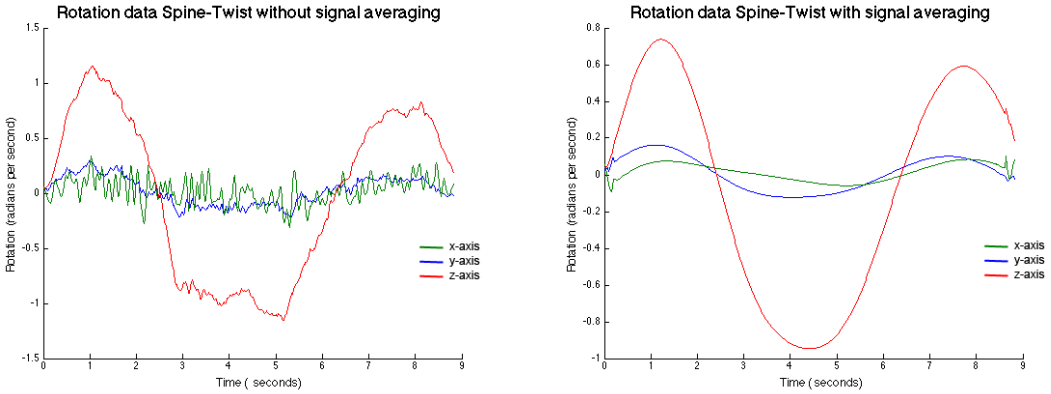
- **Wirbelsäulendrehung:** Die Übung *Wirbelsäulendrehung* hingegen, weist vor allem in der z-Dimension eine signifikante Rotation in der geglätteten Kurve auf. Abbildung 3.12 (b) zeigt diese Struktur. Man sieht, dass sie durch einen Hochpunkt von mehr als *+0.2 rad/sec*, gefolgt von einem Tiefpunkt von weniger als *-0.2 rad/sec* und erneut einem Hochpunkt von mehr als *0.2 rad/sec* gekennzeichnet ist. Da die Endposition der Armrotation kurz gehalten werden soll, schleicht sich hier nach dem Tiefpunkt oft ein weiterer Hochpunkt und Tiefpunkt ein, welche in der Segmentierung als eigener Fall berücksichtigt wurden.
  Quellcode 3.2 zeigt die Implementierung dieses Vorganges in Java. Die Funktion *findSpineTwistSamples* nimmt die Datenkurve entgegen und liefert eine *ArrayList* mit allen Anfangs- und Endzeitpunkten der erkannten möglichen Versuche.

```
private ArrayList<Integer[]> findSpineTwistSamples(List<double[]> data) {
        ArrayList<Double> zRotEntrys = new ArrayList<Double>();
        // Skipped: Extract rotation values of z-axis into zRotEntrys
```

```
        int step = 30;
        ArrayList<Integer> maxVals = getMaxValuesOfStream(zRotEntrys,
                step);
        ArrayList<Integer> minVals = getMinValuesOfStream(zRotEntrys,
                step);

        SortedMap<Integer, Boolean> bothValues = new TreeMap<Integer,
                Boolean>();
        // Skipped: Insert minVals and maxVals into bothValues with the
                form [value, isHigh]

        ArrayList<Integer[]> posSamples = new ArrayList<Integer[]>();

        Integer[] points = bothValues.keySet().toArray(new Integer[0]);
        for (int i = 0; i < points.length-2; i ++) {
                if (bothValues.get(points[i]) == true &&
                        bothValues.get(points[i+1]) == false &&
                    bothValues.get(points[i+2]) == true) {

                        // check for general structure
                        if (zRotEntrys.get(points[i]) > 0.2 &&
                            zRotEntrys.get(points[i+1]) < -0.2 &&
                            zRotEntrys.get(points[i+2]) > 0.2) {
                                int from = points[i] - 50;
                                if (from < 0) { from = 0; }

                                int to = points[i+2]+50;
                                if (to >= zRotEntrys.size())
                                        to = zRotEntrys.size() - 1;

                                posSamples.add(new Integer[] {from, to});
                        }
                }
                // Skipped: check for extra case when rotation was stopped
                        at end
        }
        return posSamples;
}
```

Quellcode 3.2: Implementierung der Segmentierung des Datenstroms nach Wirbelsäulendrehungs-Versuchen

- **Zirkuspferd:** Die Übung *Zirkuspferd* weist wieder in der x-Dimension eine signifikante Rotation auf. In Abbildung 3.13 (b) sieht man, dass die geglättete Rotationskurve durch einen Tiefpunkt von weniger als *-0.5 rad/sec*, gefolgt von



(a) Rotationsdaten ohne Signalglättung

(b) Rotationsdaten mit Signalglättung

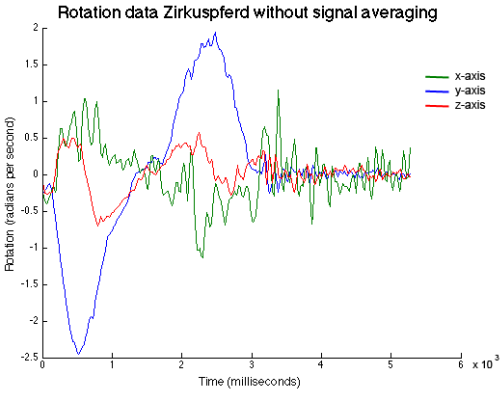
Abbildung 3.12: Rotationsdaten mit und ohne Signalglättung der Übung Wirbelsäulendrehung (eigener Entwurf)

einem Hochpunkt von mehr als *+0.5 rad/sec* gekennzeichnet ist.

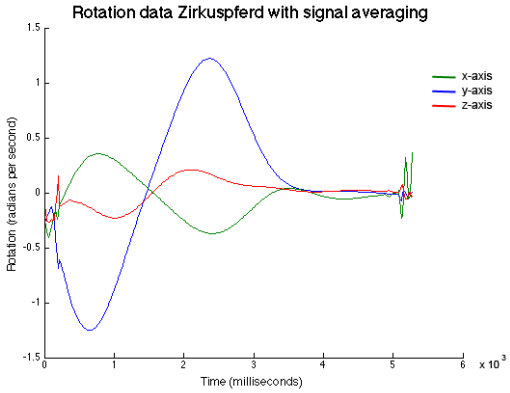
- **Rollender Ball** Die Übung *Rollender Ball* ist wiederum durch die signifikante Rotation in der z-Dimension gekennzeichnet. Die geglättete Rotationskurve , welche in Abbildung 3.14 (b) ersichtlich ist, zeigt, dass sie aus einem Hochpunkt von mehr als *+1 rad/sec*, gefolgt von einem Tiefpunkt von weniger als *-1 rad/sec* besteht.

- **Luftmatratze** Die Übung *Luftmatratze* wird ebenfalls durch eine signifikante Rotation in der z-Dimension am besten erkannt. Hier wurden, um bessere Ergebnisse zu erhalten immer zwei Wiederholungen gesucht, da diese Übung sehr kurz ist und dadurch weniger falsche Versuche als richtig klassifiziert werden. Dies sieht man auch in Abbildung 3.15 (b), da zwei Hoch- und zwei Tiefpunkte zu sehen sind. Beide müssen jeweils über *0.1 rad/sec* beziehungsweise unter *0 rad/sec* liegen.

- **Raupe** Die *Raupe* Übung weist wieder in der x-Dimension eine signifikante Rotation auf. Anhand der geglätteten Rotationskurve in Abbildung 3.16 (b) sieht man, dass sie durch einen Tiefpunkt von weniger als *-0.1 rad/sec*, gefolgt von einem Hochpunkt von mehr als *+0.1 rad/sec* gekennzeichnet ist. Des weiteren übersteigt der Amplitudenwert nie *1 rad/sec* beziehungsweise unterschreitet *-1 rad/sec*. Dies wird genutzt um überflüssige Versuche, die zufällig diese Kriterien aufweisen, auszuschließen.

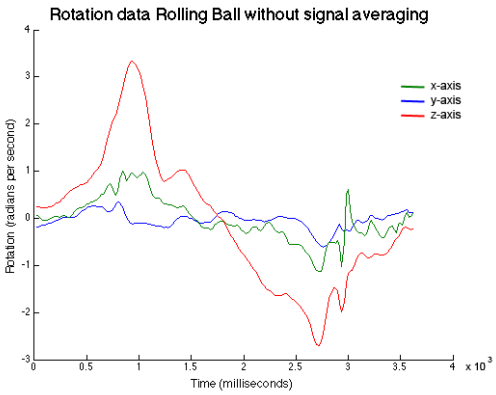


(a) Rotationsdaten ohne Signalglättung

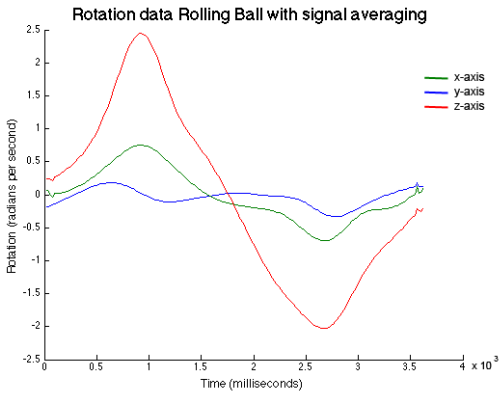


(b) Rotationsdaten mit Signalglättung

Abbildung 3.13: Rotationsdaten mit und ohne Signalglättung der Übung Zirkuspferd (eigener Entwurf)

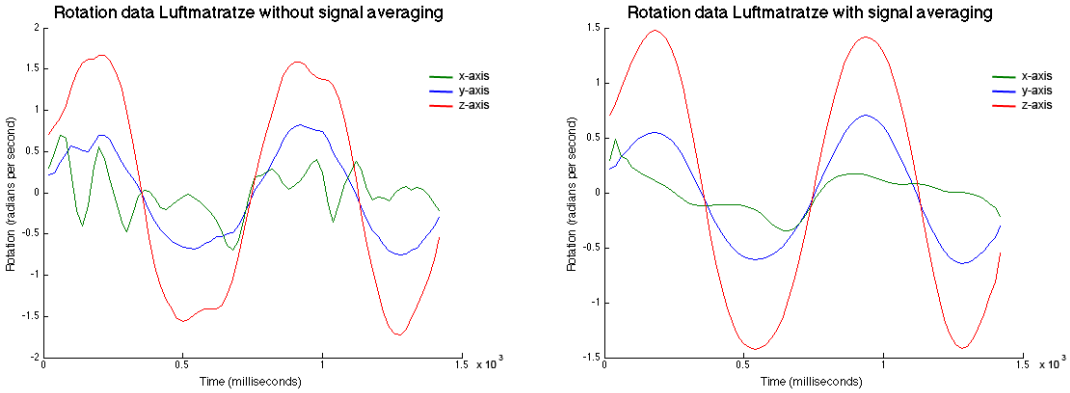


(a) Rotationsdaten ohne Signalglättung



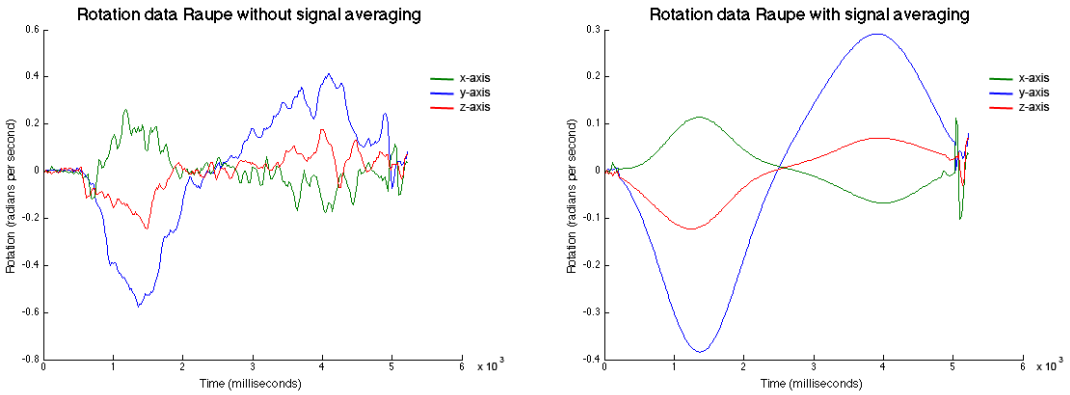
(b) Rotationsdaten mit Signalglättung

Abbildung 3.14: Rotationsdaten mit und ohne Signalglättung der Übung Rollender Ball (eigener Entwurf)



(a) Rotationsdaten ohne Signalglättung (b) Rotationsdaten mit Signalglättung

Abbildung 3.15: Rotationsdaten mit und ohne Signalglättung der Übung Luftmatratze(eigener Entwurf)



(a) Rotationsdaten ohne Signalglättung (b) Rotationsdaten mit Signalglättung

Abbildung 3.16: Rotationsdaten mit und ohne Signalglättung der Übung Raupe (eigener Entwurf)

3.2.2.4 Merkmalsgewinnung

Da man einen Klassifikator nicht direkt auf Bewegungsdaten anwenden kann, werden als erstes verschiedene Merkmale (Features) berechnet (Li u. a., 2007, S. 2).
Um Merkmale mit einer möglichst hohen Trennfähigkeit zu finden, wurden verschiedene Kennzahlen herangezogen und bei Feldversuchen evaluiert. Li, Kulkarni und Prabhakaran schlagen, in ihrer wissenschaftlichen Veröffentlichung (Li u. a., 2007, S. 2ff), die Singulärwerte, welche durch die Singulärwertzerlegung gewonnen werden, vor. Singulärwerte sind dazu besonders gut geeignet, da sie die geometrische Struktur einer linearen Abbildung besonders gut wiedergeben. (Li u. a., 2007, S. 2ff)
Dies wird auch SVD-Methode (Singular Value Decomposition) genannt. Dabei werden die beobachteten Sensorwerte als Abbildungsmatrix einer linearen Abbildung interpretiert, wobei die verschiedenen Variablen in den Spalten und die Zeitintervalle in den Reihen eingetragen werden. Bei einem dreidimensionalen Rotations- (Gyroscope) und Beschleunigungssensor (Accelerometer) erhält man dadurch sechs Spalten, je drei Spalten für die *x, y und z* Werte. Betrachtet man das Abtastintervall von einer Sekunde, erhält man 50 Zeileneinträge, da die Abtastrate 50 Hertz beträgt. Abbildung 3.17 veranschaulicht diese Konstruktion.

Die Singulärwertzerlegung zerlegt eine Matrix $A(m\,x\,n)$ in drei weitere Matrizen, die orthogonalen Matrizen $U(m\,x\,m)$ und $V(n\,x\,n)$, sowie die Diagonalmatrix $\Sigma(m\,x\,n)$.

$$A = U\Sigma V^t \tag{3.5}$$

bzw.

$$A = \begin{pmatrix} u_1 & \cdots & u_k & u_{k+1} & \cdots & u_m \end{pmatrix} \begin{pmatrix} \sigma_1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_k & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} v_1^t \\ \vdots \\ v_k^t \\ v_{k+1}^t \\ \vdots \\ v_n \end{pmatrix} \tag{3.6}$$

wobei

$$k < \min(m, n) \tag{3.7}$$

Die Skalare $\sigma_1, \cdots, \sigma_k$ werden als Singulärwerte bezeichnet, die Vektoren $u_1, \cdots, u_m$ als linke und $v_1, \cdots, v_n$ als rechte Singulärvektoren. Die Singulärwerte werden absteigend sortiert gelistet. (Lange, 2010, S. 129ff)

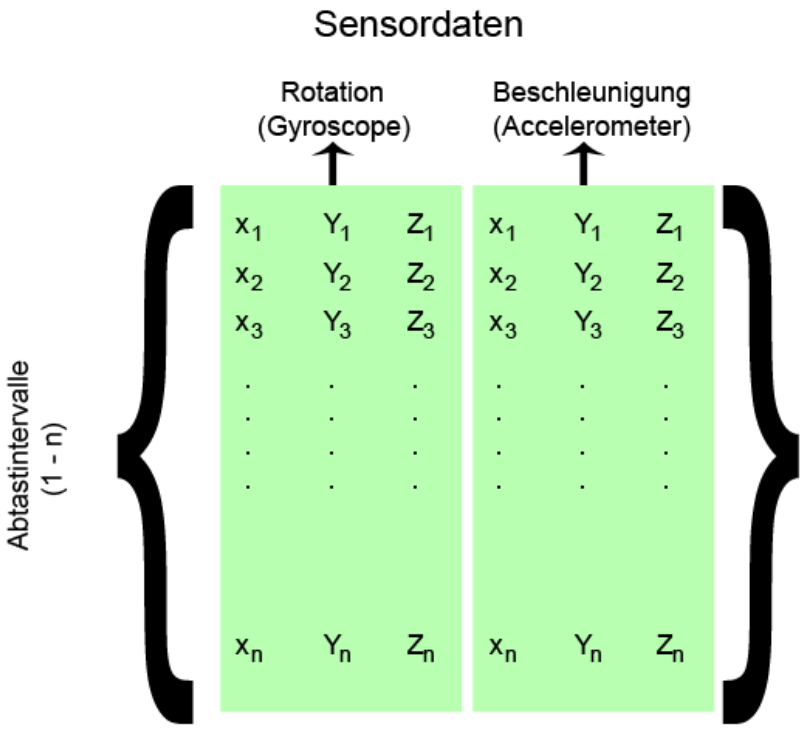


Abbildung 3.17: Die Sensordaten werden in Form einer Matrix dargestellt (eigener Entwurf)

Wendet man dies nun auf die beobachtete Matrix an, erhält man sechs Singulärwerte, welche als Merkmale (Features) verwendet werden.

Des weiteren werden folgende Merkmale verwendet:

- **Mittelwert** $\bar{X}$
  Der Mittelwert einer Menge an Punkten X wird wie folgt in Formel 3.8 berechnet:

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{3.8}$$

  Verwendet werden der Mittelwert der einzelnen Dimensionen beider Sensoren.

- **Standardabweichung** ϕ
  Die Standardabweichung ist ein Maß für die Streuung um den Mittelwert und

wird für eine Punktmenge X wie folgt in Formel 3.9 berechnet:

$$\phi^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{X})^2 \tag{3.9}$$

- **Normalized Cross-Correlation (*NCC*)**
  Diese Kreuzkorrelation wurde bereits in Kapitel 3.2.1.6 beschrieben. Es wird die Ähnlichkeit zwischen den verschiedenen Dimensionen der Rotations- und der Beschleunigungswerte berechnet und als Merkmal verwendet. Außerdem stellt auch die gesamte *NCC*-Korrelation der Beschleunigungs- zu den Rotationsdaten ein Merkmal dar.

Um bessere Ergebnisse bei der Klassifizierung zu erreichen, wurden die so gewonnen Merkmale noch skaliert. Diesen Vorgang empfehlen Hsu, Chang und Lin (Hsu u. a., 2010, S. 4), welche eine signifikante Verbesserung versprechen. Diese Annahme konnte durch einen erfolgreichen Test bestätigt werden. Dabei werden die Daten um den Mittelwert, wie in Formel 3.10 ersichtlich, normalisiert.

$$x_{norm} = \frac{x_{original} - \bar{X}}{\phi} \tag{3.10}$$

3.2.2.5 Klassifizierung

Zur Klassifizierung werden Support Vector Machines, wie in Kapitel 3.2.1.5 beschrieben, benutzt. Diese erfolgt in zwei Ebenen, zum einen wird eine SVM trainiert, welche Auskunft über die generelle Klasse der Übung liefert, zum anderen sollen weitere SVMs mögliche Fehlerquellen erkennen. Erstere gibt also an, um welche der sechs Übungen des Wiraculix Workouts, laut Kapitel 3.2.1.1, es sich handelt. Im zweiten Schritt wird der mögliche Versuch einer Unterklasse zugeordnet. Dieser Versuch wird so noch enger in Fehlerklassen beziehungsweise Güteklassen eingeteilt. Jede dieser Fehlerklassen wird anhand einer eigenen SVM klassifiziert, um so Verbesserungsvorschläge liefern zu können.

Abbildung 3.18 zeigt beispielhaft die Menge der Fehler- bzw. Güteklassen der Übung *Über Kreuz*. Man sieht unter anderem Überschneidungen verschiedener Klassen, was daher kommt, dass zwei Fehler auch gleichzeitig begangen beziehungsweise erkannt werden können. Folgende Unterscheidungen bei den verschiedenen Übungen sind möglich:

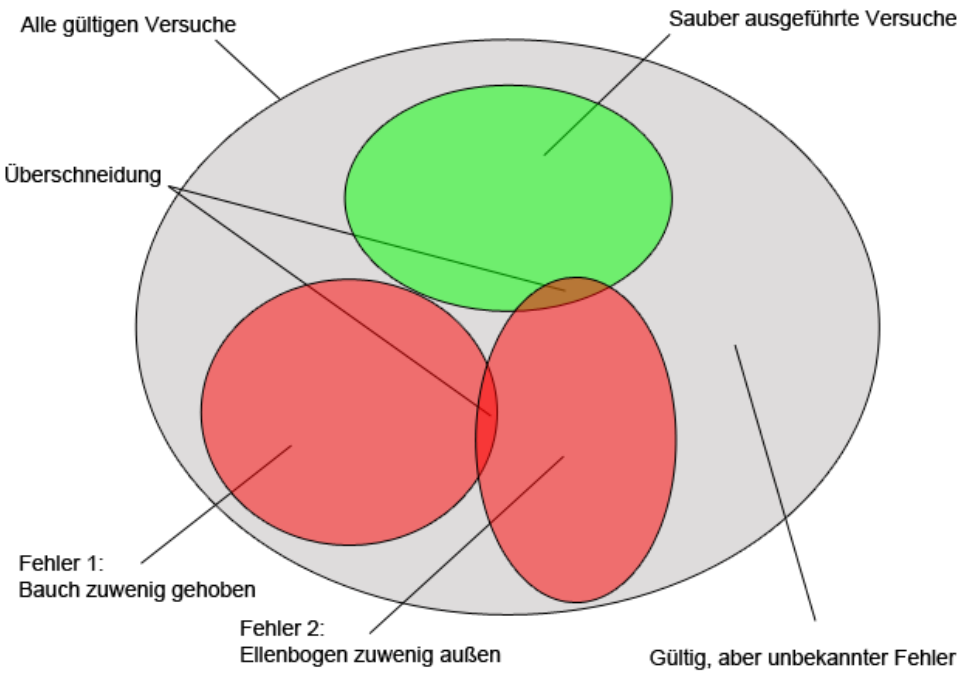


Abbildung 3.18: Die Fehlerklassen die bei der *Über Kreuz* Übung erkannt werden können (eigener Entwurf)

- **Über Kreuz:**
  - Übung wurde sauber ausgeführt
  - Bauch wurde zu wenig angehoben
  - Ellenbogen wurden zu wenig nach außen gestreckt
- **Wirbelsäulendrehung:**
  - Übung wurde sauber ausgeführt
  - Arme wurden nicht waagrecht gehalten
- **Zirkuspferd:**
  - Übung wurde sauber ausgeführt
  - Arme wurden nicht ausreichend gehoben

- **Rollender Ball**
  - Übung wurde sauber ausgeführt
- **Luftmatratze**
  - Übung wurde sauber ausgeführt
  - Knie wurden nicht gestreckt gehalten
- **Raupe**
  - Übung wurde sauber ausgeführt
  - Höchster Punkt wurde nicht kurz gehalten

Klassifizierung nur anhand der Singulärwerte

Als Merkmale (Features) wurden zunächst, wie in (Li u. a., 2007, S. 4ff) beschrieben, die Singulärwerte der beobachteten Variablen, interpretiert als lineare Abbildung, verwendet. Die Java-Implementierung verwendet die Support Vector Machines Library LIBSVM [1] zur Klassifizierung. Für diese Variante wurden SVMs mit einer RBF-Kernelfunktion und skalierten Features verwendet. Um die Genauigkeit zu verbessern, wurde wie in (Hsu u. a., 2010, S. 5) empfohlen, eine Parameteroptimierung in Matlab [2] vorgenommen.
Quellcode 3.3 zeigt den Trainingsvorgang der Support Vector Machines. Man sieht die Funktion *createSVMModels*, die je nach Übung unterschiedlich viele Support Vector Machines erzeugt. Dies wird im Parameter *cntModels* bestimmt. Die weiteren Funktionsparameter enthalten die optimierten Kernel-Parameter für die Trainingsdaten, sowie diese selbst. Zur Erzeugung einer Support Vector Machine wird die Funktion *buildSVMModel* aufgerufen, welche anhand der übergebenen Parameter und Trainingsdaten ein Klassifizierungsproblem erzeugt und dazu diese trainiert und zurückliefert.

```
private static svm_model[] createSVMModels (
        int cntModels,
        double[][] params,
        double[][] features,
        ArrayList<int[]> labels) throws Exception {
```

[1] Chih-Chung Chang and Chih-Jen Lin. LIBSVM – A Library for Support Vector Machines, 2013. http://www.csie.ntu.edu.tw/~cjlin/libsvm/, 24.02.2014.

[2] MathWorks. Matlab - Die Sprache für technische Berechnungen, 2014. http://www.mathworks.de/products/matlab/, 24.02.2014.

```
        // Skipped: Some checks for sizes etc.

        svm_model[] models = new svm_model[cntModels];

        Iterator<int[]> labelsIt = labels.iterator();

        for (int i = 0; i < cntModels; i ++) {
                double[] actParam = params[i];
                int[] actLabels = labelsIt.next();

                // Skipped: Some checks for same sizes etc.

                svm_parameter param = new svm_parameter();
                param.svm_type = svm_parameter.C_SVC;
                param.kernel_type = svm_parameter.RBF;
                param.degree = 3;
                param.gamma = actParam[0];
                param.coef0 = 0;
                param.nu = 0.5;
                param.cache_size = 40;
                param.C = actParam[1];
                param.eps = 1e-3;
                param.p = 0.1;
                param.shrinking = 1;
                param.probability = 0;
                param.nr_weight = 0;
                param.weight_label = new int[0];
                param.weight = new double[0];

                models[i] = buildSVMModel(features, actLabels, param);

        }
        return models;
}

private static svm_model buildSVMModel (
                double[][] features,
                int[] labels,
                svm_parameter param) throws Exception {
        // build problem
        svm_problem prob = new svm_problem();
        prob.l = labels.length;
        prob.y = new double[prob.l];
        prob.x = new svm_node[prob.l][6];

        for (int i = 0; i < labels.length; i++) {
```

```
                double[] actFeature = features[i];

                // Skipped: Some checks for same sizes etc.
                for (int j = 0; j < features.length) {
                        prob.x[i][j] = new svm_node();
                        prob.x[i][j].index = j + 1;
                        prob.x[i][j].value = actFeature[j];
                }
                prob.y[i] = labels[i];
        }
        // build model
        svm_model model = svm.svm_train(prob, param);

        return model;
}
```

Quellcode 3.3: Implementierung des Trainingsvorganges der Support Vector Machines zur Klassifizierung eines Versuchs in Java

Mithilfe der Library *Commons Math* [1] kann nun für jeden erkannten möglichen Versuch die Singulärwertzerlegung berechnet und anhand der Singulärwerte klassifiziert werden. Dieser Vorgang wird in Quellcode 3.4 dargestellt. Am Anfang der Funktion *classify* erfolgt die bereits beschriebene Vorverarbeitung (siehe Kapitel 3.2.2.2), danach werden für jede der sechs Übungen (siehe Kapitel 3.2.1.1) die möglichen Versuche, unter Anwendung der ebenfalls bereits beschriebenen Segmentierung (siehe Kapitel 3.2.2.3), extrahiert. Anschließend sieht man die Berechnung der Singulärwerte und die Klassifizierung der möglichen Versuche. Im positiven Fall wird ein gefundener Versuch mit allen wichtigen Daten in eine *Map* gespeichert, welche dann zurückgeliefert wird.

```
public Map<String, ArrayList<FoundRecord>> classify(List<double[]> inData,
        int from) {
        Map<String, ArrayList<FoundRecord>> returnList =
                new HashMap<String, ArrayList<FoundRecord>>();

        Iterator<Exercise> exerciseIt = exercises.iterator();
        while (exerciseIt.hasNext()) {
                Exercise exercise = exerciseIt.next();

                ArrayList<double[]> data =
                        ClassificationUtils.averageSignal(inData, 5, 2);
                ArrayList<double[]> data_soft =
                        ClassificationUtils.averageSignal(
```

[1] Apache Commons. Commons Math: The Apache Commons Mathematics Library, 2013. http://commons.apache.org/proper/commons-math/, 24.02.2014.

```
                inData,
                exercise.getAverageFactor(),
                exercise.getAveragePoints());

ArrayList<FoundRecord> founds = new ArrayList<FoundRecord>();

List<double[]> actualPart = null;
List<double[]> actualPart_soft = null;
if ((from != 0) && (from > 500)) {
        from = from - 500;
        actualPart = data.subList(from, data.size());
        actualPart_soft = data_soft.subList(from,
                data_soft.size());
} else {
        actualPart = data;
        actualPart_soft = data_soft;
}

ArrayList<Integer[]> possibleSamples = null;
switch(exercise.getType()) {
case CRISS_CROSS:
        possibleSamples = findCrissCrossSamples(actualPart_soft);
        possibleSamples =
                deleteDuplicateCrissCrossSamples(possibleSamples);
        break;
case SPINE_TWIST:
        possibleSamples = findSpineTwistSamples(actualPart_soft);
        break;
case ZIRKUSPFERD:
        possibleSamples =
                findZirkuspferdSamples(actualPart_soft);
        break;
case ROLLING_BALL:
        possibleSamples =
                findRollingBallSamples(actualPart_soft);
        break;
case LUFTMATRATZE:
        possibleSamples =
                findLuftmatratzeSamples(actualPart_soft);
        possibleSamples =
                deleteDuplicateLuftmatratzeSamples(possibleSamples);
        break;
case RAUPE:
        possibleSamples = findRaupeSamples(actualPart_soft);
        break;
}
```

```
Iterator<Integer[]> posSamplesIt = possibleSamples.iterator();

while (posSamplesIt.hasNext()) {
        Integer[] actSample = posSamplesIt.next();

        int sampleStart = actSample[0];
        int sampleEnd = actSample[1];
        if (sampleEnd - sampleStart > 10) {
                List<double[]> actPart =
                        actualPart.subList(sampleStart,
                        sampleEnd);
                double[][] actPartA =
                        DatatypeUtils.convertListToArray(actPart);

                // Calculation of the SVD for classification
                RealMatrix mat = new BlockRealMatrix(actPartA);

                SingularValueDecomposition svd =
                        new SingularValueDecompositionImpl(mat);
                double[] actFeature = svd.getSingularValues();

                svm_node[] x = new svm_node[6];
                for (int j = 0; j < features.length) {
                        prob.x[j] = new svm_node();
                        prob.x[j].index = j + 1;
                        prob.x[j].value = actFeature[j];
                }

                svm_model[] svmModels = exercise.getSvmModels();
                double[] classification = new
                        double[svmModels.length];

                for (int i = 0; i < svmModels.length; i++) {
                        classification[i] =
                                svm.svm_predict(svmModels[i], x);
                }

                if (classification[0] == 1) {
                        // store found
                        FoundRecord found =
                                new FoundRecord(
                                        from + sampleStart,
                                        from + sampleEnd,
                                        0, 0, 0,
                                        classification,
                                        exercise.getName());
```

```
                                    founds.add(found);
                                }
                            }
                        }
                        returnList.put(exercise.getName(), founds);
                }
                return returnList;
        }
```

Quellcode 3.4: Implementierung der Klassifizierung eines möglichen Versuchs mithilfe der trainierten Support Vector Machines in Java

Klassifizierung anhand aller Features

Durch die Anwendung verschiedener Features zur Klassifizierung kann eine signifikante Verbesserung des Ergebnisses erreicht werden. Zusätzlich zu den Singulärwerten werden die Mittelwerte und Standardabweichungen der drei Dimensionen von Rotations- und Beschleunigungsdaten, sowie deren Korrelationen (NCC) zueinander, verwendet. Auch die Korrelation (ebenfalls NCC) zwischen den gesamten Rotations- und Beschleunigungsdaten wird als Kennzahl herangezogen. Dadurch ergeben sich insgesamt 25 Features, welche wieder skaliert und normalisiert werden. In (Hsu u. a., 2010, S. 12ff) sind verschiedene Gründe zur Verwendung eines einfachen linearen Kernels angegeben, einer ist eine hohe Anzahl an Merkmalen (Features). Aus diesem Grund wird für diese Variante eine Support Vector Machine mit linearem Kernel verwendet, was auch die gewünschten, zufriedenstellenden Ergebnisse liefert.
Großteils unterscheidet sich diese Variante nur in der Berechnung der Merkmale, Quellcode 3.5 zeigt diese Berechnung in der *classify*-Methode dieser Klassifizierungsstrategie.

```
...
// Skipped: rest of classify method very similiar to previous strategy

RealMatrix mat = new BlockRealMatrix(actPartA);

SingularValueDecomposition svd = new SingularValueDecompositionImpl(mat);
double[] actFeature = svd.getSingularValues();

svm_node[] x = new svm_node[25];

// add SVD values
for (int actfeatCnt = 0; actfeatCnt < 6; actfeatCnt++) {
        x[actfeatCnt] = new svm_node();
        x[actfeatCnt].index = actfeatCnt+1;
        x[actfeatCnt].value = actFeature[actfeatCnt];
}
```

```
// calculate and add covariances
double[] covRot = ClassificationUtils.covarianz(actRot);
for (int covCnt = 0; covCnt < 9; covCnt ++) {
       x[covCnt + 6] = new svm_node();
       x[covCnt + 6].index = covCnt + 7;
       x[covCnt + 6].value = covRot[covCnt];
}

double[] covAcc = ClassificationUtils.covarianz(actAcc);
for (int covCnt = 0; covCnt < 9; covCnt ++) {
       x[covCnt + 15] = new svm_node();
       x[covCnt + 15].index = covCnt + 16;
       x[covCnt + 15].value = covAcc[covCnt];
}

double actRotMue = ClassificationUtils.calculateMue(actRot);
double actRotPhi = ClassificationUtils.calculatePhi(actRot, actRotMue);

double actAccMue = ClassificationUtils.calculateMue(actAcc);
double actAccPhi = ClassificationUtils.calculatePhi(actAcc, actAccMue);

double ncc = ClassificationUtils.calculateNCC(
              actRot, actAcc,
              actRotMue, actRotPhi,
              actAccMue, actAccPhi);

x[24] = new svm_node();
x[24].index = 25;
x[24].value = ncc;

// get trained trained svms
svm_model[] svmModels = exercise.getSvmModels();
double[] classification = new double[svmModels.length];

// normalize data
svm_node[] x_norm = new svm_node[25];
for (int cC = 0; cC < 25; cC++) {
       x_norm[cC] = new svm_node();
       x_norm[cC].value = (x[cC].value + mue[cC]) * phi[cC];
       x_norm[cC].index = x[cC].index;
}

// classify
for (int i = 0; i < svmModels.length; i++) {
       classification[i] = svm.svm_predict(svmModels[i], x_norm);
}
```

```
if (classification[0] == 1) {
        // store found
        FoundRecord found = new FoundRecord(
                        from + sampleStart,
                        from + sampleEnd,
                        0, 0, 0,
                        classification,
                        exercise.getName());
        founds.add(found);
}
...
```

Quellcode 3.5: Berechnung aller Features zur Klassifizierung mit einer linearen SVM in Java

3.2.2.6 Nachbearbeitung

Aus verschiedenen Gründen kann es dazu kommen, dass der gleiche Versuch mehrmals erkannt wird. Dies kann folgende Ursachen haben:

- Da während des Workouts im Minutenintervall Daten von der MotionTracker-Applikation, wie in Kapitel 3.3 beschrieben, an den Server zur Auswertung geschickt werden, kann es passieren, dass der Anfang dieser Übung in einem alten Intervall liegt bzw. das Ende knapp im neuen. Aus diesem Grunde, wird nicht genau eine Minute betrachtet, sondern zusätzlich die letzten zehn Sekunden des alten Intervalls. Wenn in diesem Zeitraum ein Versuch absolviert wurde, wird er doppelt erkannt.

- Einige Übungen, wie zum Beispiel die *Luftmatratze*, können in zwei verschiedene Richtungen gestartet werden. Fängt der Proband bzw. die Probandin mit der Fußbewegung nach unten an, ist der erste Spitzenwert (Peak, Amplituden Extrempunkt) negativ, umgekehrt positiv. Werden mehrere Bewegungen hintereinander ausgeführt wird von Peak eins bis Peak vier beziehungsweise von Peak zwei bis Peak fünf etc. ein möglicher Versuch erkannt. Somit erhält man zu viele Wiederholungen, welche sich überlappen.
  Abbildung 3.19 veranschaulicht dieses Szenario.

- Wenn der Proband bzw. die Probandin zwischen der Übungsausführung eine Position kurz haltet, schleichen sich Spitzenwerte (Peaks) ein, welche zur doppelten Erkennung von Wiederholungen führen.

Um im Endergebnis diese doppelten Einträge nicht inkludiert zu haben, wird ein weiterer Schritt zur Nachbearbeitung durchgeführt. Alle Einträge werden nochmals durchlaufen und die prozentuale Überlappung, wie in Formel 3.11 dargestellt, berechnet.

$$p = \frac{l_B}{l_A} \tag{3.11}$$

wobei

$$l_A = end_A - start_A, \qquad l_B = end_B - start_B \tag{3.12}$$

die Längen der jeweiligen Versuche sind.

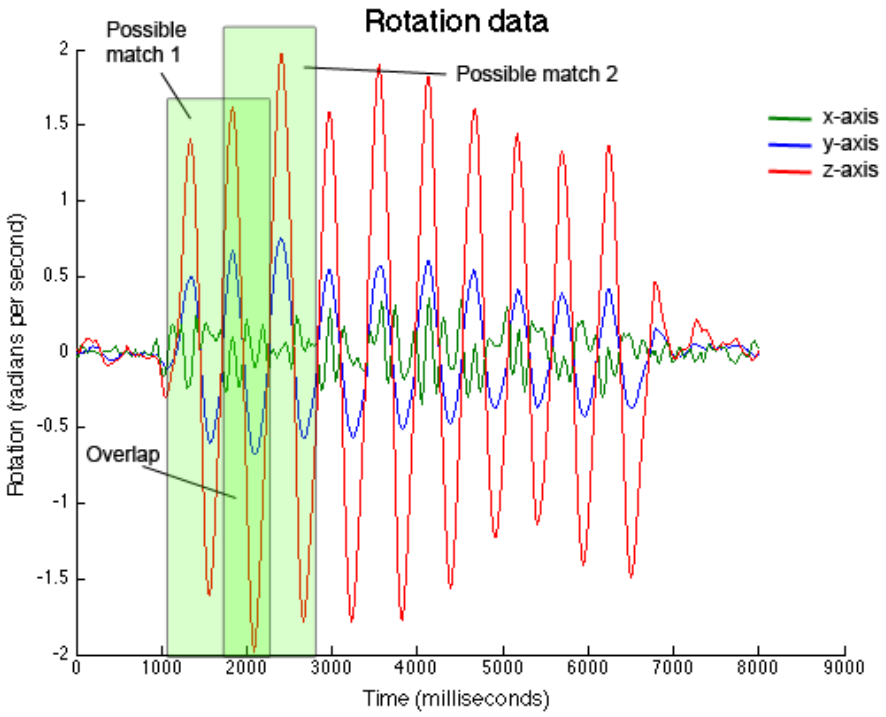


Abbildung 3.19: Doppelt erkannte Versuche der Übung Luftmatratze (eigener Entwurf)

Ist die Überschneidung größer als ein gewisser Prozentwert, wird ein Eintrag gelöscht. Dies ist abhängig von der Ursache der doppelten Eintragung. Quellcode 3.6 zeigt die Implementierung dieses Vorganges in Java. In der Funktion *deleteDuplicateLuftmatratze-Samples* werden doppelt gefundene Versuche der Übung *Luftmatratze* aussortiert. Übersteigt die Überschneidung zehn Prozent, wird der zeitlich später beginnende Versuch als ungültig deklariert und verworfen.

```
private ArrayList<Integer[]> deleteDuplicateLuftmatratzeSamples(
                ArrayList<Integer[]> possibleSamples) {
        // delete duplicate entries
        ArrayDeque<Integer[]> validFoundRecords = new ArrayDeque<Integer[]>();

        // if sth is found
        if (!possibleSamples.isEmpty()) {
                ListIterator<Integer[]> itFoundRecs =
                        possibleSamples.listIterator();
                validFoundRecords.add(itFoundRecs.next());

                while (itFoundRecs.hasNext()) {
                        Integer[] actFound = itFoundRecs.next();
                        Integer[] lastValidFound = validFoundRecords.peek();

                        // if next found record is too near to the first
                        int hundred = lastValidFound[1] - lastValidFound[0];
                        int both = lastValidFound[1] - actFound[0];
                        double percent = ((double)both) / ((double)hundred);

                        if (percent >= ((double)0.1)) {
                                // Do nothing, just take the first one
                        } else {
                                // or take it into the valids otherwise
                                validFoundRecords.push(actFound);
                        }
                }
        }

        return new ArrayList<Integer[]>(validFoundRecords);
}
```

Quellcode 3.6: Nachbearbeitung und Verwerfung der doppelten gefundenen Einträge in Java

3.2.3 Template Matching

Als Alternative zur Mustererkennung, wie in Kapitel 3.2.2 beschrieben, wurde versucht das Mitzählen von Wiederholungen der verschiedenen Übungen mithilfe von Template Matching zu implementieren. Ein Template besteht im Anwendungsfall aus einer Datenmatrix mit sechs Zeilen, für die je drei Rotations- bzw. Beschleunigungsdimensionen, und unterschiedlich vielen Spalten. Die Spaltenanzahl ist abhängig von der Dauer des betrachteten Templates, wobei eine Spalte einem Abtastpunkt entspricht. In diesem Kapitel wird diese Strategie näher beleuchtet und auf die einzelnen Teilbereiche eingegangen.

3.2.3.1 Vorverarbeitung

Da Template Matching auf der Berechnung von Korrelationen von Datenausschnitten und statischen Templates, also Musterkurven, beruht, müssen diese immer gleich groß sein. Dies bedeutet für das Anwendungsbeispiel, es können nur zeitlich gleich lange Abschnitte verglichen werden. Da die Wiederholungen aber beinahe nie genau in einer bestimmten Zeit ausgeführt werden, braucht man bei dieser Methode für jede Übung eine Mindest- und Maximaldauer, also einen Zeitraum in dem die Wiederholung absolviert werden muss.
Um nun gleich große Ausschnitte miteinander vergleichen zu können, wurden verschiedene Herangehensweisen implementiert.

Subsampling auf Mindestgröße
Beim Subsampling auf die Mindestgröße wird das Template sowie der betroffene Datenausschnitt auf die Mindestgröße reduziert. Dies erfolgt indem man solange zwei Punkte zu einem zusammenfügt, bis man die gewünschte Länge des Ausschnittes erreicht hat. Beim Zusammenfügen wird ganz einfach der Mittelwert zweier benachbarter Abtastpunkte berechnet und diese beiden durch das Resultat ersetzt.

$$x_i = \frac{x_i + x_{i+1}}{2} \tag{3.13}$$

Quellcode 3.7 zeigt die Implementierung dieser Methode, bei der zufällig benachbarte Indizes ausgewählt und anschließend zusammengefügt werden.

```
public static ArrayList<double[]> compressArrayToSize(
```

```
                List<double[]> original, int minTime) {
        ArrayList<double[]> tmpSample = new ArrayList<double[]>(original);
        Random random = new Random();
        while (tmpSample.size() > minTime) {
                int randIndex = random.nextInt(tmpSample.size() - 1);
                double[] before = tmpSample.get(randIndex);
                double[] after = tmpSample.remove(randIndex + 1);
                double[] mean = new double[before.length];
                // calculate mean
                for (int i = 0; i < before.length; i ++) {
                        mean[i] = ( before[i] + after[i] ) / 2.0;
                }
                tmpSample.set(randIndex, mean);
        }
        return tmpSample;
}
```

Quellcode 3.7: Subsampling eines Datenarrays zu einer bestimmten Größe

Upsampling auf Maximalgröße

Beim Upsampling auf die Maximalgröße wird das Template sowie der betroffene Datenausschnitt auf die Maximalgröße vergrößert. Dies erfolgt, in dem man solange einen Punkt, der aus dem Mittelwert der zwei benachbarten Punkte besteht, einfügt, bis man die gewünschte Länge des Ausschnittes erreicht hat.

$$x_{neu} = \frac{x_i + x_{i+1}}{2} \tag{3.14}$$

Quellcode 3.8 zeigt die Implementierung dieser Methode, bei der in regelmäßigen Abständen neue Punkte eingefügt werden.

```
public static ArrayList<double[]> stretchArrayToSize(
                List<double[]> original, int maxTime) {
        ArrayList<double[]> tmpSample = new ArrayList<double[]>(original);
        boolean repeat = false;
        do {
                int countElements = tmpSample.size();
                int inserted = 0;
                int difference = maxTime - countElements;
                if (difference != 0) {
                        int stepsize = countElements / difference;
                        int rest = countElements % difference;
```

```
                        if (stepsize == 0) {
                                stepsize = 1;
                                rest = 0;
                                repeat = true;
                        } else
                                repeat = false;

                        for (int i=0; i < (countElements - rest); i += stepsize)
                                {
                                double[] before = tmpSample.get(i+inserted);
                                double[] after = tmpSample.get(i+inserted+1);
                                double[] mean = new double[before.length];
                                // calculate mean
                                for (int j = 0; j < before.length; j ++) {
                                        mean[j] = ( before[j] + after[j] ) / 2.0;
                                }

                                tmpSample.add(i+inserted+1, mean);
                        }
                }
        } while (repeat);

        return tmpSample;
}
```

Quellcode 3.8: Upsampling eines Datenarrays zu einer bestimmten Größe

Verwendung mehrerer Templates in verschiedenen Größen

Hier werden in verschiedenen Schrittgrößen, innerhalb der definierten Grenzen, Templates angefertigt und jedes einzeln verglichen. Durch die, dafür notwendige, mehrmalige Berechnung der Kreuz-Korrelation (NCC) wird zwar mehr Zeit benötigt, allerdings liefert diese Variante die genauesten Ergebnisse.

3.2.3.2 Segmentierung

Die Segmentierung wird mit einem sich verschiebenden Fenster (Sliding Window) implementiert. Dabei werden Teilausschnitte des gesamten Datenstroms betrachtet und dieser Ausschnitt wie eine Schablone weiter verschoben. Wie in Abbildung 3.20 veranschaulicht, wird in einem ersten Schritt vom Abtastpunkt t_i die minimale Dauer der Übung betrachtet. Dieser Ausschnitt vergrößert sich immer um einen bestimmten Wert (*stepSize*), bis die maximale Dauer erreicht wird. Jeder dieser Ausschnitte wird

anschließend ausgewertet und klassifiziert. Im zweiten Schritt wird die Schablone um einen bestimmten Wert (*windowStepSize*, meist die halbe Minimalzeit) weiter zum Abtastpunkt t_{i+1} verschoben und beginnt wieder mit der Minimalgröße. Dieser Vorgang wiederholt sich, bis der komplette Datenstrom abgearbeitet wurde.
Quellcode 3.9 zeigt die Implementierung dieser Segmentierung in Java. Dabei wird eine Subsampling-Strategie zum Vergleichen von Bereichen von verschiedener Größe verwendet.

```
public static ArrayList<FoundRecord> analyzeStreamPartForExcerciseShort(
                    List<double[]> data, MotionExercise actExcercise, int
                            offset) {
        int actStart = 0;
        int windowStepSize = actExcercise.getMinTime() / 2;
        int stepSize = 5;

        int dataSize = data.size();
        int minTime = actExcercise.getMinTime();
        int maxTime = actExcercise.getMaxTime();
        ArrayList<FoundRecord> posFoundRecords = new ArrayList<FoundRecord>();

        while ((actStart + minTime) < dataSize) {
                int actFrameEnd = actStart + minTime;
                int frameEnd =
                        ((actStart + maxTime) > dataSize) ? dataSize : (actStart
                                + maxTime);

                ArrayList<Double> nccValues = new ArrayList<Double>();

                while (actFrameEnd < frameEnd) {
                        // Subsampling of the data
                        ArrayList<double[]> actSampled =
                                ClassificationUtils.compressArrayToSize(
                                        data.subList(actStart, actFrameEnd),
                                                minTime);

                        double actMue =
                                ClassificationUtils.calculateMue(actSampled);
                        double actPhi =
                                ClassificationUtils.calculatePhi(actSampled,
                                actMue);
                        double actNCC =
                                ClassificationUtils.calculateNCC(
                                        actSampled, actMue, actPhi,
                                                actExcercise.getTemplate());
```

```
                    nccValues.add(actNCC);

                    actFrameEnd += stepSize;
                }

                // Skipped: Classification

                actStart += windowStepSize;
            }

            return posFoundRecords;
        }
```

Quellcode 3.9: Sliding-Window-Implementierung zur Segmentierung der Inputs für das Template Matching in Java

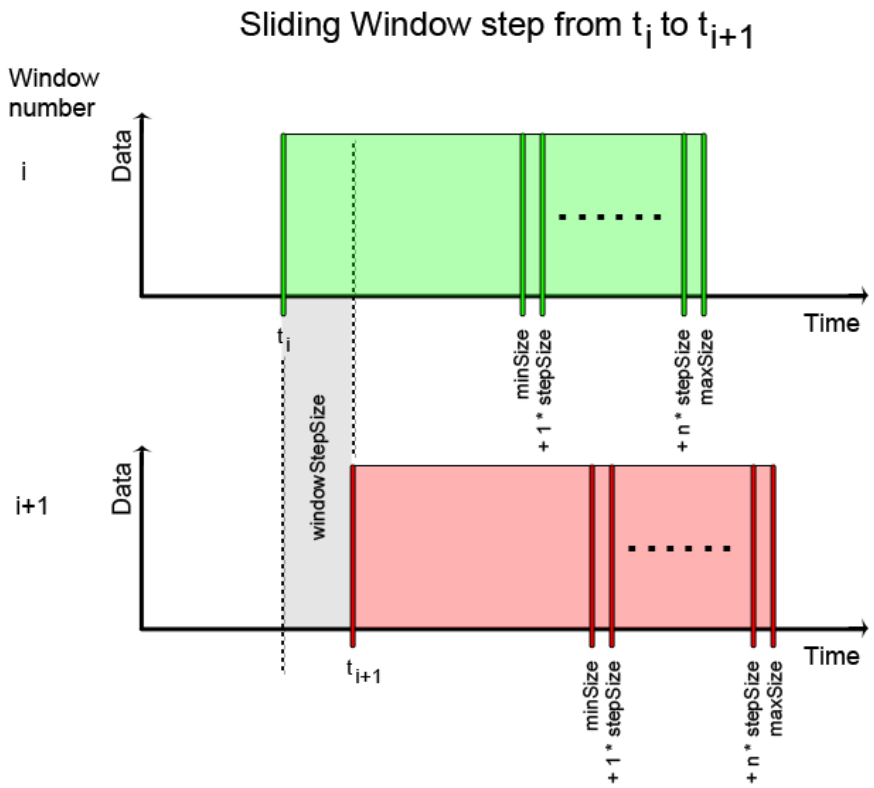


Abbildung 3.20: Veranschaulichung der Sliding-Window-Segmentierung (eigener Entwurf)

3.2.3.3 Klassifizierung

Zur Klassifizierung der gewonnen Datenausschnitte wird ein Ähnlichkeitsmaß berechnet und anhand diesem entschieden, ob es sich um eine gültige Wiederholung einer Übung handelt oder nicht. Man geht davon aus, dass ein Ausschnitt der die Übung genau trifft ein höheres Ähnlichkeitsmaß besitzt, als solche, die noch zusätzliche Daten enthalten. Daher wird von allen Ausschnitten, die den gleichen Startabtastpunkt besitzen, nur jener mit dem maximalen Ähnlichkeitsmaß gespeichert.
Als Ähnlichkeitsmaß wird der normalisierte Kreuzkorrelationskoeffizient (NCC), wie in Kapitel 3.2.1.6 beschrieben, verwendet und überprüft ob dieser einen definierten Schwellwert (Threshold) übersteigt. Ist dies der Fall, wird das aktuelle Fenster als gültiger Versuch abgespeichert.
Dieser Vorgang wird in Quellcode 3.10 veranschaulicht. Man sieht die Funktion *calculateNCC*, welche, die in Kapitel 3.2.1.6 erklärte, NCC-Berechnung zeigt, sowie den Codeteil, der die Ausschnitte klassifiziert.

```
public static double calculateNCC(ArrayList<double[]> actSampled, double
        actMue,
        double actPhi, ExcerciseTemplate template) {

        // Skipped: Some Range checks etc.

        double phiDivisor = Math.sqrt(actPhi * template.getPhi());
        double ncc = 0;
        int elementCount = 0;

        ListIterator<double[]> templDataIt = template.getData().listIterator();
        ListIterator<double[]> actSampledIt = actSampled.listIterator();

        while(templDataIt.hasNext()) {
                double[] nextTemplRow = templDataIt.next();
                double[] nextActSampledRow = actSampledIt.next();

                for (int col = 0; col < nextTemplRow.length; col ++) {
                        ncc += ((nextActSampledRow[col] - actMue) *
                                (nextTemplRow[col] - template.getMue()) /
                                phiDivisor);
                        elementCount ++;
                }
        }

        ncc /= elementCount;

        return ncc;
```

```
}

        ...
        // Skipped: Sliding Window implementation
        ...

        // get maximum of all segments with actual starting point
        double maxNCC = Collections.max(nccValues);

        // get specific threshold of actual exercise
        double threshold = actExcercise.getThreshold();

        // check if ncc value exceeds threshold
        if (Double.compare(threshold, maxNCC) <= 0) {
                // if yes, store new found segment
                FoundRecord foundElement = new FoundRecord(
                                actStart + offset,
                                actStart + offset + minTime +
                                        (nccValues.indexOf(maxNCC) * stepSize),
                                maxNCC,
                                0, 0, null,
                                actExcercise.getName());
                posFoundRecords.add(foundElement);
        }

        ...
```

Quellcode 3.10: Klassifizierung von Datenausschnitten anhand des NCC-Wertes in Java

3.2.3.4 Nachbearbeitung

Bei der beschriebenen Variante der Segmentierung und Klassifikation kommt es zu doppelten Einträgen, da auch Ausschnitte, die die Übung nicht hundertprozentig treffen, oft den Schwellwert übersteigen und somit als richtig klassifiziert werden. Aus diesem Grund ist eine Nachbearbeitung notwendig, bei der diese doppelt gefundenen Wiederholungen verworfen werden. Man setzt wieder die Annahme voraus, dass Ausschnitte, die die Übung genau treffen, einen höheren Korrelationskoeffizienten aufweisen, als solche, die sie ungenau treffen. Überschneiden sich nun gefundene Übungswiederholungen über einen gewissen Prozentwert, werden Treffer mit geringerem Korrelationskoeffizienten ausgeschieden. Somit sind nur noch die richtigen Treffer enthalten.
Abbildung 3.21 zeigt Rotationsdaten, in denen unter anderen eine Wiederholung der Übung *Über Kreuz* zu finden ist. Man sieht zwei Treffer die zusätzliche Daten vor bzw. nach der eigentlichen Übung enthalten, welche rot markiert sind. Der grün markierte

Treffer besitzt den höchsten NCC-Wert und wird somit behalten. Die beiden anderen Treffer werden ausgeschieden.
Der Java-Quellcode 3.11 zeigt diesen Vorgang. Man sieht wie alle gefundenen Einträge nochmals durchlaufen und bei Überschneidungen nur die Treffer mit dem höchsten NCC-Wert behalten werden.

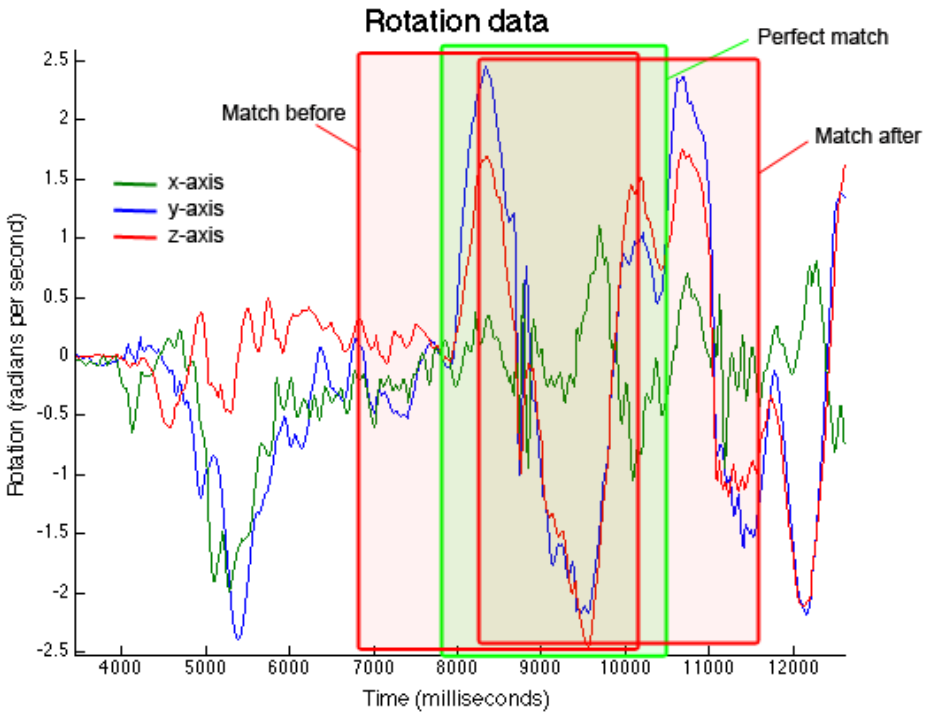


Abbildung 3.21: Doppelte Template-Matching Einträge der Übung Über Kreuz, zwei ungenauere und einen genauen Treffer (eigener Entwurf)

```
public static ArrayList<FoundRecord> deleteDuplicateEntries(
                ArrayList<FoundRecord> posFoundRecords, int minTime) {
        ArrayDeque<FoundRecord> validFoundRecords = new
                ArrayDeque<FoundRecord>();
        Collections.sort(posFoundRecords, FoundRecord.START_COMPARATOR);

        // if something is found
        if (!posFoundRecords.isEmpty()) {
                ListIterator<FoundRecord> itFoundRecs =
                        posFoundRecords.listIterator();
                validFoundRecords.add(itFoundRecs.next());

                while (itFoundRecs.hasNext()) {
                        FoundRecord actFound = itFoundRecs.next();
```

```
                        FoundRecord lastValidFound = validFoundRecords.peek();

                        // if next found record is too near to the first
                        int hundred = lastValidFound.getEnd() -
                                lastValidFound.getStart();
                        int both = lastValidFound.getEnd() - actFound.getStart();
                        double percent = ((double)both) / ((double)hundred);
                        if (percent >= ((double)0.60)) {

                                // take the one with the higher ncc value
                                if (actFound.getNcc() > lastValidFound.getNcc()) {
                                        validFoundRecords.pop();
                                        validFoundRecords.push(actFound);
                                }
                        } else {
                                // or take it into the valids otherwise
                                validFoundRecords.push(actFound);
                        }
                }
        }

        return new ArrayList<FoundRecord>(validFoundRecords);
}
```

Quellcode 3.11: Verwerfen von doppelten Template-Matching-Einträgen anhand des NCC-Wertes in Java

3.3 Die MotionTracker Applikation

Die MotionTracker-Applikation wurde für das iPhone implementiert und ist ein wesentlicher Bestandteil dieser Aufgabe. Das iPhone wurde als Client gewählt, da eine einfache Alternative zu umfangreichen tragbaren Datenverarbeitungssystemen gesucht wurde und immer mehr Kinder und Jugendliche ein solches Smartphone besitzen (Grimus u. Ebner, 2014).
In den kommenden Kapiteln wird auf die technischen Details des iOS-Clients, sowie die Handhabung aus der Sicht des Benutzers bzw. der Benutzerin näher eingegangen.

3.3.1 Datengewinnung

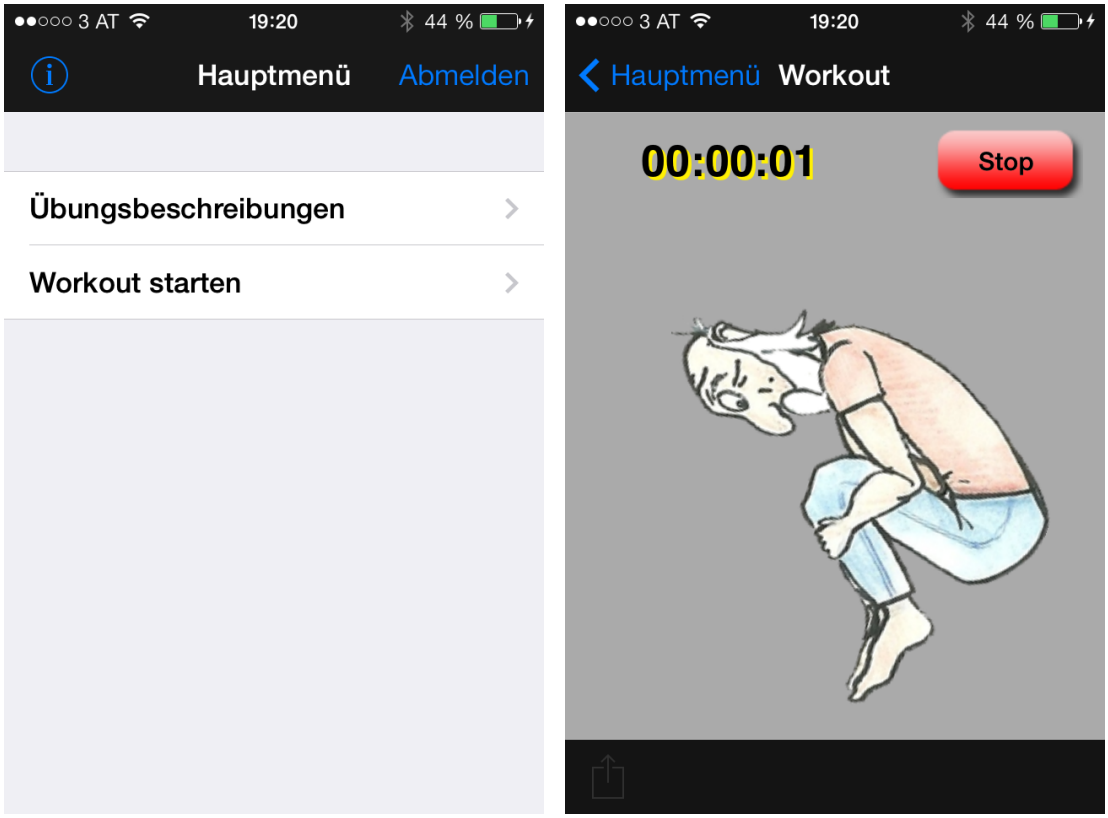
Technische Sicht

Um auf die Bewegungsdaten, der im iPhone eingebauten Sensoren (wie in Kapitel 3.2.1.2 beschrieben), zugreifen zu können, wird die von Apple bereitgestellte *CMMotionManager*-Klasse verwendet. Für Daten sammelnde Anwendungen schlägt Apple eine sogenannte *Push-Approach* vor, welche auch implementiert wurde. (Apple Inc., 2014, S. 65ff)
Dabei wird in der gewählten Abtastfrequenz immer ein bestimmter Codeblock ausgeführt, welcher die aktuellen Sensordaten verarbeitet. Diese Blöcke werden an eine *NSOperationQueue* übergeben, welche diese dann nacheinander, in einem eigenen Thread, ausführt. Nach mehreren Versuchen wurde eine Abtastfrequenz von 50 Hertz gewählt, welche auch für derartige Anwendungen empfohlen wird. (Apple Inc., 2014, S. 60)
Quellcode 3.12 zeigt die Implementierung des Startes dieser *Motion-Updates* und den Codeblock, welcher die Sensordaten in einem *NSMutableArray* speichert.

```
- (void)processStart {
    StartWorkoutOperation *swop = [[StartWorkoutOperation alloc] init];
    [self.networkQueue addOperation:swop];

    lastUploaded = 0;
    self.secondsTimer = [NSTimer scheduledTimerWithTimeInterval:1
                                                         target:self
                                                       selector:@selector(updateTime:)
                                                       userInfo:nil
                                                        repeats:YES];


    CMMotionManager *mManager = [(AppDelegate *)[[UIApplication
            sharedApplication] delegate] sharedManager];
    double updateInterval = 1.0 / 50.0;
```

```
    TrainingViewController *__weak weakSelf = self;
    if ([mManager isDeviceMotionAvailable] == YES) {
        self.actualTracking = [[NSMutableArray alloc] init];
        self.actBegin = [[NSDate alloc] init];

        NSTimeInterval tmp_timeStamp = [[NSDate date] timeIntervalSinceNow];
        NSNumber *timeStamp = [NSNumber numberWithDouble:tmp_timeStamp];

        NSDateFormatter *formatter;
        NSString *dateString;

        formatter = [[NSDateFormatter alloc] init];
        [formatter setDateFormat:@"mm:ss"];
        dateString = [formatter stringFromDate:[NSDate date]];
        [mManager setDeviceMotionUpdateInterval:updateInterval];

        self.updateQueue = [[NSOperationQueue alloc] init];

        [mManager startDeviceMotionUpdatesToQueue:self.updateQueue
                withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

            TrackingEntry *actEntry = [[TrackingEntry alloc] init];
            actEntry.yaw = deviceMotion.attitude.yaw;
            actEntry.pitch = deviceMotion.attitude.pitch;
            actEntry.roll = deviceMotion.attitude.roll;

            actEntry.rotX = deviceMotion.rotationRate.x;
            actEntry.rotY = deviceMotion.rotationRate.y;
            actEntry.rotZ = deviceMotion.rotationRate.z;

            actEntry.accX = deviceMotion.userAcceleration.x;
            actEntry.accY = deviceMotion.userAcceleration.y;
            actEntry.accZ = deviceMotion.userAcceleration.z;

            actEntry.timestamp = timeStamp;

            [weakSelf.actualTracking addObject:actEntry];
        }];

    }
}
```

Quellcode 3.12: Start der Motion-Updates mithilfe der CMMotionManager-Klasse



(a) Das Hauptmenü (b) Der Workout-Modus

Abbildung 3.22: Screenshots des hierarchisch aufgebauten Hauptmenüs und des gestarteten Workout-Modus (eigener Entwurf)

Bedienung aus Sicht des Benutzers bzw. der Benutzerin

Wurde die Applikation erst einmal gestartet, kann der Benutzer bzw. die Benutzerin ganz einfach über die hierarchische Darstellung zum *Workout-Modus* navigieren und sein bzw. ihr Bewegungsprogramm mit der Betätigung des grünen *Start*-Buttons starten. Abbildung 3.22 zeigt das hierarchisch aufgebaute Hauptmenü, über welches man zum *Workout-Modus* navigieren kann, sowie diesen selbst.

Im *Workout-Modus* sieht man einen grünen Start-Button, sowie das Studienmaskottchen Wiraculix. Wird der Start-Button gedrückt, beginnt sich Wiraculix zu drehen und der Benutzer bzw. die Benutzerin kann mit dem Bewegungsprogramm beginnen. Die Reihenfolge, mit welcher die verschiedenen Übungen absolviert werden, ist für das System nicht von Bedeutung, da die erfassten Daten automatisch auf alle überprüft werden.

3.3.2 Streaming zum Server

Die Auswertung der Daten übernimmt ein Webservice, das auf der Google App Engine [1] läuft. Da der Upload der Daten, sowie die Auswertung etwas Zeit in Anspruch nimmt, werden bereits während des Absolvieren des Bewegungsprogrammes Daten an den Server gesendet und ausgewertet. Dies geschieht im Intervall von einer Minute und bietet den großen Vorteil, dass dem Benutzer bzw. der Benutzerin eine schnellere Rückmeldung gegeben werden kann. Nach der Betätigung des Stop-Buttons, muss nur noch die Zeitüberschreitung der letzten vollen Minute ausgewertet werden. Damit keine Wiederholungen, die genau über die Minutenübergänge gehen, übersehen werden, wird immer ein Teil des letzten Intervalls mitberücksichtigt.

Quellcode 3.13 zeigt die Implementierung dieser Vorgehensweise, die Methode *updateTime* wird im Sekundenintervall aufgerufen und aktualisiert die abgelaufene Zeit des Bewegungsprogrammes. Dies wurde mithilfe eines *NSTimers*-Objektes umgesetzt. Des weiteren wird überprüft, ob das Minutenintervall vergangen ist und im positiven Fall muss der aktuelle Datenbereich berechnet und extrahiert werden. Nun wird eine neue *UploadPartOperation* erzeugt und der eigens für die Serverkommunikation erzeugten *networkQueue* hinzugefügt. Diese führt die Operationen via FIFO-Prinzip (First In First Out) sequentiell aus.

```
- (IBAction)updateTime:(id)sender {
    NSTimeInterval interval = -[self.actBegin timeIntervalSinceNow];
    int seconds = ((int) interval) % 60;
    int minutes = ((int) (interval - seconds) / 60) % 60;
    int hours = ((int) interval - seconds - 60 * minutes) % 3600;

    [self.timeLabel setText:
        [NSString stringWithFormat:@"%.2d:%.2d:%.2d", hours, minutes,
                seconds]];

    if (seconds == 0 && minutes != 0) {
        NSRange range;
        NSMutableArray *act = nil;
        @synchronized(_actualTracking) {
            lastCount = self.actualTracking.count;
            range.location = lastUploaded;
            range.length = lastCount - lastUploaded;
            self.trackingRange = range;
```

[1] Google Inc. Google App Engine: Platform as a Service, 2014. https://developers.google.com/appengine/, 03.03.2014.

```
            act = [NSMutableArray arrayWithArray:
                        [self.actualTracking subarrayWithRange:range]];
        }
        UploadPartOperation *upop =
            [[UploadPartOperation alloc] initWithTracking:act];
        [upop setDelegate:self];
        [self.networkQueue addOperation:upop];
    }
}
```

Quellcode 3.13: Aktualisierung der abgelaufenen Zeit und Initiierung des Datenuploads zum Server

Die Hauptfunktion der *UploadPartOperation* serialisiert die Daten im JSON-Format (ECMA International, 2013), sendet diese an den Server und verarbeitet dessen Rückmeldung. Quellcode 3.14 zeigt diesen Vorgang im Detail. Die *UploadPartOperation* informiert den *TrainingsViewController*, welcher für das Sammeln und Verwalten der Daten zuständig ist, über den Erfolg des Uploads. War er erfolgreich, wird der letzte Index in der Variable *lastUploaded* gespeichert, um beim nächsten Intervall wieder den aktuellen Bereich extrahieren zu können. Im Fehlerfall bleibt diese *lastUploaded*-Variable unverändert und so wird beim nächsten Upload-Versuch dieser Bereich ebenfalls berücksichtigt. Schlagen alle Versuche fehl, werden beim Stoppen des Bewegungsprogrammes die kompletten Bewegungsdaten hochgeladen und ausgewertet. Der Benutzer bzw. die Benutzerin erhält erst beim Fehlschlagen dieses Versuches eine Fehlermeldung, mit der Möglichkeit den Upload zu wiederholen. Praktisch kann dies vorkommen, wenn am Trainingsort kein Empfang für mobile Daten und keine WLAN-Verbindung vorliegt, oder andere Szenarien einen Upload verhindern.

```
-(void)main {
    if (self.isCancelled)
        return;

    NSMutableDictionary *jsonObject = [[NSMutableDictionary alloc] init];

    NSMutableArray *trackingEntries =
        [NetworkUtils buildTrackingArray:self.actualTracking];

    if (self.isCancelled)
        return;

    [jsonObject setValue:trackingEntries forKey:@"entries"];
    NSData *data = [NSJSONSerialization dataWithJSONObject:jsonObject
                                                   options:nil
                                                     error:nil];
```

```
    NSMutableURLRequest *request =
        [NSMutableURLRequest requestWithURL:[NSURL URLWithString:serverURL]];

    [request setHTTPMethod:@"POST"];
    [request addValue:[NSString stringWithFormat:@"%d", data.length]
        forHTTPHeaderField:@"Content-Length"];
    [request setValue:@"text/json" forHTTPHeaderField:@"Content-Type"];
    [request setHTTPBody:data];
    [request setTimeoutInterval:50];

    [request setHTTPShouldUsePipelining:YES];

    if (self.isCancelled)
        return;

    NSURLResponse* response;
    NSError* error = nil;

    NSData *receivedData = [NSURLConnection sendSynchronousRequest:request
                                                 returningResponse:&response
                                                             error:&error];
    NSString *theResponse = [[NSString alloc] initWithData:receivedData
                                                  encoding:NSUTF8StringEncoding];

    // Skipped: Inform delegate about error or success
}
```

Quellcode 3.14: Vorbereitung und Upload der Daten in der UploadPartOperation

3.3.3 Verarbeitung am Server

Am Server wird als Erstes eine Authentifizierung des Benutzers bzw. der Benutzerin durchgeführt. Je nach Aktion werden anschließend verschiedene Aufgaben erledigt.

- **Starten des Bewegungsprogrammes**
  Beim Start des Bewegungsprogrammes wird für den Benutzer bzw. die Benutzerin ein *Session*-Objekt erzeugt und in einer *Map* gespeichert. Die Verwaltung dieses *User-to-Session-Mappings* übernimmt der *SessionHandler*.

- **Upload der Daten**
  Die Datenbereiche, die vom Client während des Bewegungsprogrammes an den

Server gesendet werden, werden der aktuellen Session des Benutzers bzw. der Benutzerin hinzugefügt und gleichzeitig wird der Auswertungsvorgang gestartet. Um die verschiedenen Varianten des Klassifikationsprozesses einfach austauschen zu können, wurden diese als verschiedene Strategien des *Strategy Patterns* (Gamma u. a., 1995) implementiert. Jedes *Session*-Objekt hat also ein *Strategy*-Objekt zur Klassifikation, welches wiederum die gewünschte Variante in seiner *classify*-Methode implementiert. Der Klassifkationsprozess wird in Kapitel 3.2 näher erklärt, sowie auf die verschiedenen Methoden eingegangen. Anschließend wird dem Client eine Erfolgsmeldung zurückgeliefert, um die fehlerlose Verarbeitung zu signalisieren.

- **Stoppen des Bewegungsprogrammes**
  Beim Stoppen des Bewegungsprogrammes wird zuerst die Auswertung des letzten unausgewerteten Teils, was dem Datenbereich der letzten angefangenen Minute entspricht, initiiert.
  Da bei bei der Auswertung, wie in den Kapiteln 3.2.2.6 und 3.2.3.4 erläutert, doppelt erkannte Einträge vorkommen können, muss anschließend ein Nachtbearbeitungsschritt, der diese Einträge löscht, ausgeführt werden.
  Das endgültige Ergebnis wird dann kompakt im JSON-Format (ECMA International, 2013) serialisiert und dem Client zurückgeliefert.

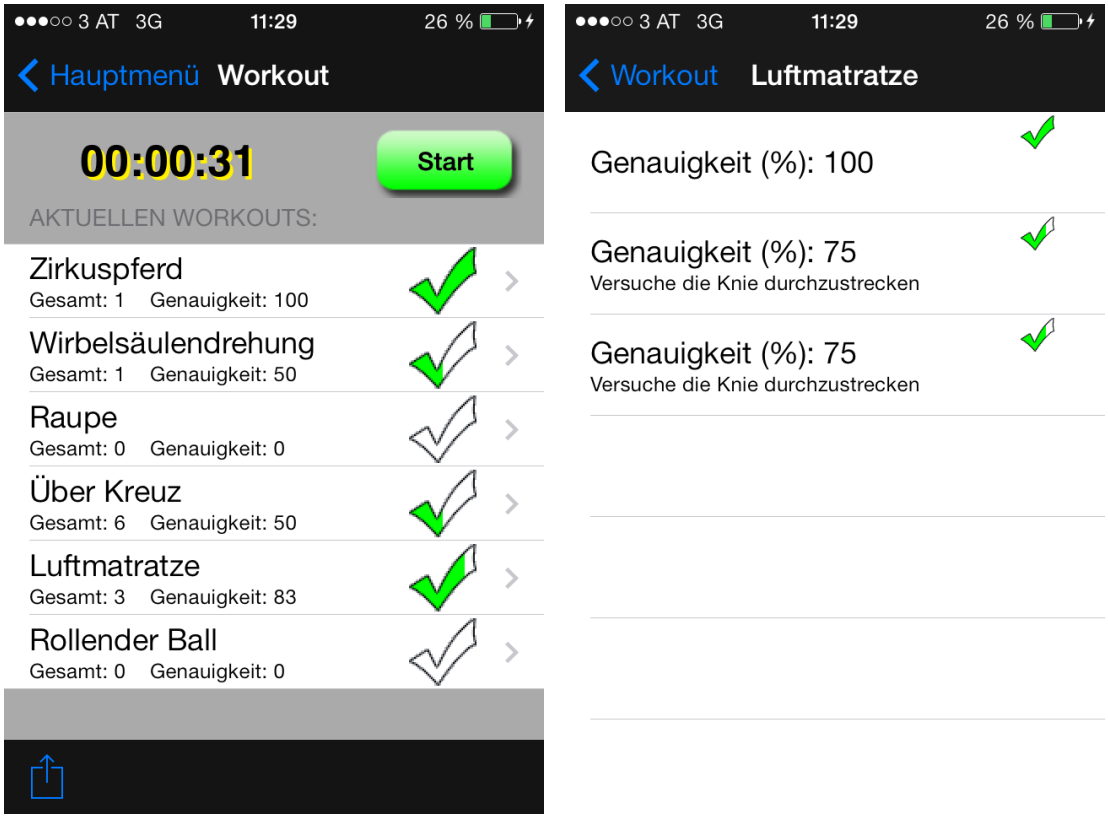
- **Speichern des Bewegungsprogrammes**
  Nachdem der Benutzer bzw. die Benutzerin die Rückmeldung der Auswertung erhalten hat, kann das absolvierte Bewegungsprogramm gespeichert werden. Diese Option speichert die gefundenen Übungswiederholungen, Datums- und Zeitinformationen sowie die Daten selbst mittels JDO [1] im Google Datastore der Google App Engine [2] um später darauf zugreifen zu können.

3.3.4 Rückmeldung an den User

Nach der Betätigung des Stop-Buttons und der Auswertung am Server, erhält der Benutzer bzw. die Benutzerin eine genaue Aufstellung der erkannten Übungen und deren Genauigkeit. Abbildung 3.23 zeigt die grafische Darstellung der Rückmeldung. Man sieht die sechs Übungen und die Anzahl der erkannten Wiederholungen. Im rechten Teil der jeweiligen Spalte wird in einem prozentual grün gefülltem Häkchen die Genauigkeit veranschaulicht. Diese setzt sich, je nach Klassifikation zusammen.
Eine als sehr gut klassifizierte Wiederholung hat 100 Prozent, wird ein Fehler trotz-

[1] Apache DB. JDO - Java Data Objects, 2014. https://db.apache.org/jdo/, 08.04.2013.

[2] Google Inc. Google App Engine: Platform as a Service, 2014. https://developers.google.com/appengine/, 03.03.2014.



(a) Die gesamte Auswertung

(b) Detaillierte Auswertung einer Übung

Abbildung 3.23: Screenshots der Auswertungen eines beendeten Bewegungsprogrammes (eigener Entwurf)

dem erkannt, erhält sie 90 Prozent. Eine erkannte, aber als nicht sehr gut klassifizierte Wiederholung erhält 50 Prozent. Kann diese Wiederholung einem typischen Fehler zugeordnet werden, werden daraus 75 Prozent. Wiederholungen die zwar als gültig erkannt werden, aber weder einem typischen Fehler, noch als sehr gute zugeordnet werden können, erhalten 50 Prozent. In diesem Fall ist der begangene Fehler unbekannt.
Der Benutzer bzw. die Benutzerin hat anschließend die Möglichkeit dieses absolvierte Bewegungsprogramm zu speichern oder zu verwerfen. Wird die erstere Variante gewählt, erscheint noch ein positives Feedback mit der Anzahl der bereits absolvierten Programme der aktuellen Woche.

3.4 Die MotionTracker-Webapplikation

Die MotionTracker-Webapplikation hat die Aufgabe, die gesammelten Daten anschaulich darzustellen. Die folgenden Kapitel beleuchten die einzelnen Darstellungsmodi im Detail.

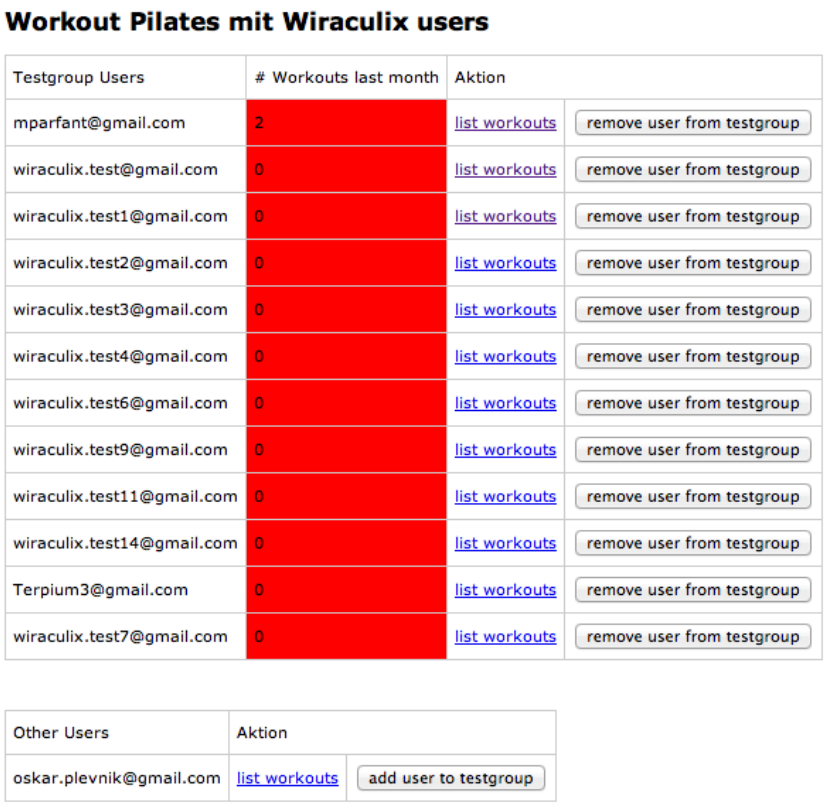
3.4.1 Darstellung der TeilnehmerInnen

Diese Darstellung kann nur von der Studienbegleiterin aufgerufen werden, alle anderen BenutzerInnen können nur ihre eigenen absolvierten Bewegungsprogramme sehen. Nach dem erfolgreichen Login, erscheint eine Darstellung aller TeilnehmerInnen, welche zumindest ein Bewegungsprogramm absolviert haben. Die Studienbegleiterin hat die Möglichkeit, bestimmte Probanden und Probandinnen auszuwählen und zur Testgruppe hinzuzufügen, diese Auswahl erscheint immer ganz oben. In Abbildung 3.24 sieht man die Auflistung der TeilnehmerInnen. Probanden und Probandinnen aus der Testgruppe, welche die gewünschte Anzahl an gültigen Bewegungsprogrammen im letzten Monat nicht absolviert haben, erscheinen rot hervorgehoben. Durch einen Klick auf die *list workouts*-Aktion gelangt man zu einer Ansicht, welche alle absolvierten Bewegungsprogramme eines bestimmten Benutzers bzw. einer bestimmten Benutzerin darstellt.

3.4.2 Darstellung der absolvierten Bewegungsprogramme

Hat sich ein bestimmter Benutzer bzw. eine bestimmte Benutzerin eingeloggt, oder die Studienbegleiterin einen Probanden bzw. eine Probandin ausgewählt, erscheint die Darstellung der absolvierten Bewegungsprogramme, welche in Abbildung 3.25 dargestellt ist. Man sieht eine Tab-Darstellung, bei der man zwischen einer grafischen Aufbereitung, einer detaillierten tabellarischen Ansicht und einer Zusammenfassung dieses Bewegungsprogrammes wählen kann.

- **Grafische Aufbereitung**
  Die grafische Aufbereitung zeigt ein Diagramm, welches die Anzahl der absolvierten Programme pro Monat des vergangenen Jahres zeigt.
  Interessiert man sich für einen speziellen Monat, kann man diesen auswählen und man erhält eine detaillierte Monatsansicht. Diese ist in Abbildung 3.26 ersichtlich, man sieht die Anzahl der absolvierten Bewegungsprogramme pro Tag des Monats. Auch hier kann zeitlich noch weiter hineingezoomt werden, durch die Auswahl eines Tages erhält man alle an diesem absolvierten Programme in der tabellarischen



Workout Pilates mit Wiraculix users

Testgroup Users	# Workouts last month	Aktion	
mparfant@gmail.com	2	list workouts	remove user from testgroup
wiraculix.test@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test1@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test2@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test3@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test4@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test6@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test9@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test11@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test14@gmail.com	0	list workouts	remove user from testgroup
Terpium3@gmail.com	0	list workouts	remove user from testgroup
wiraculix.test7@gmail.com	0	list workouts	remove user from testgroup

Other Users	Aktion	
oskar.plevnik@gmail.com	list workouts	add user to testgroup

Abbildung 3.24: Detaillierte Auflistung aller BenutzerInnen, die zumindest ein Bewegungsprogramm absolviert haben (eigener Entwurf)

Ansicht.

- **Tabellarische Ansicht**
  Die detaillierte tabellarische Ansicht zeigt den Start- und Endzeitpunkt sowie die Dauer der einzelnen Bewegungsprogramme. Wählt man eines davon aus, gelangt man zur nächsten Darstellungsebene, bei der alle Informationen eines speziellen Bewegungsprogrammes ersichtlich sind.

- **Zusammenfassung**
  Die Zusammenfassung berechnet die insgesamt absolvierten Bewegungsprogramme, die gesamte Anzahl der Wiederholungen pro Übung sowie die durchschnittliche Anzahl der Übungswiederholungen pro Programm und stellt diese Kennzahlen tabellarisch dar. Dies soll der Studienbegleiterin als Übersicht dienen, ob in etwa die benötigten Werte erreicht wurden.

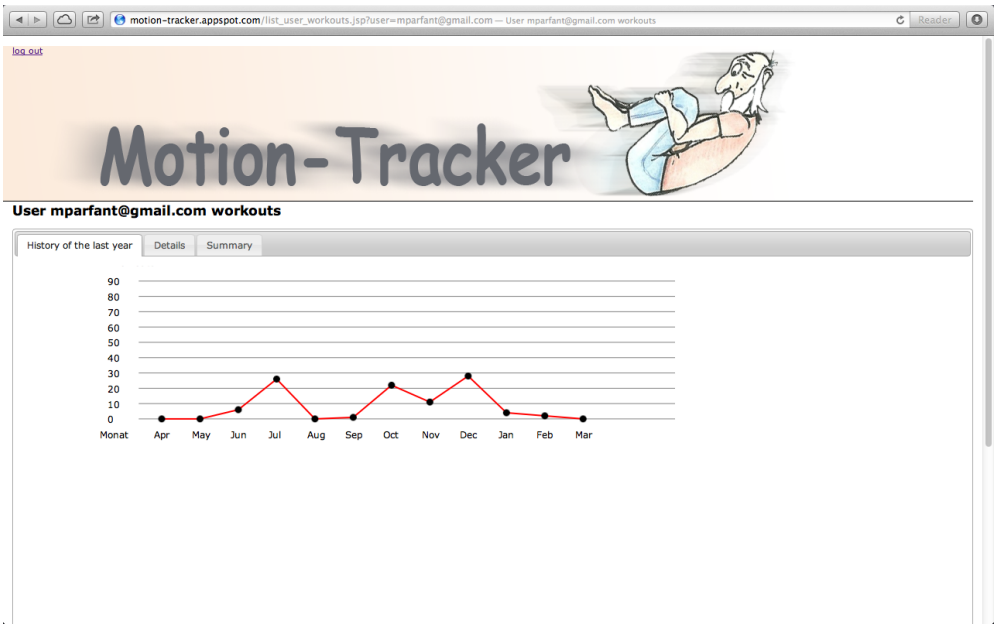


Abbildung 3.25: Grafische Aufbereitung der im letzten Jahr absolvierten Bewegungsprogramme eines Benutzers bzw. einer Benutzerin (eigener Entwurf)

3.4.3 Detaildarstellung eines Bewegungsprogrammes

Ähnlich wie bei der Darstellung der absolvierten Bewegungsprogramme (Kapitel 3.4.2) kann man auch bei der Detaildarstellung, mithilfe von Tabs, zwischen verschiedenen Modi wechseln. Es stehen wieder eine grafische Aufbereitung, eine Zusammenfassung, Details zu den einzelnen Wiederholungen und die Daten selbst zur Verfügung.
Abbildung 3.27 zeigt diese Tab-Darstellung sowie die grafische Aufbereitung.

- **Grafische Aufbereitung**
  Bei der grafischen Aufbereitung des absolvierten Bewegungsprogrammes, werden die erkannten Wiederholungen pro Übung, in zeitlicher Reihenfolge, symbolisch angezeigt. Dadurch kann die Studienbegleiterin relativ schnell nachvollziehen, ob die Versuche in der richtigen Reihenfolge gemacht bzw. fälschlicherweise erkannt wurden. Dies würde sich in gleichzeitig erkannten Wiederholungen verschiedener Übungen äußern.

- **Zusammenfassung**
  Der Zusammenfassung können Start- und Endzeitpunkt sowie die Dauer des absolvierten Bewegungsprogrammes entnommen werden.

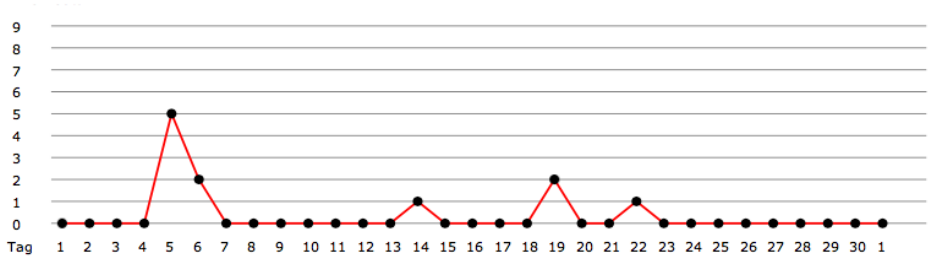


Abbildung 3.26: Detaillierte grafische Aufbereitung der in einem bestimmten Monat absolvierten Bewegungsprogramme eines Benutzers bzw. einer Benutzerin (eigener Entwurf)

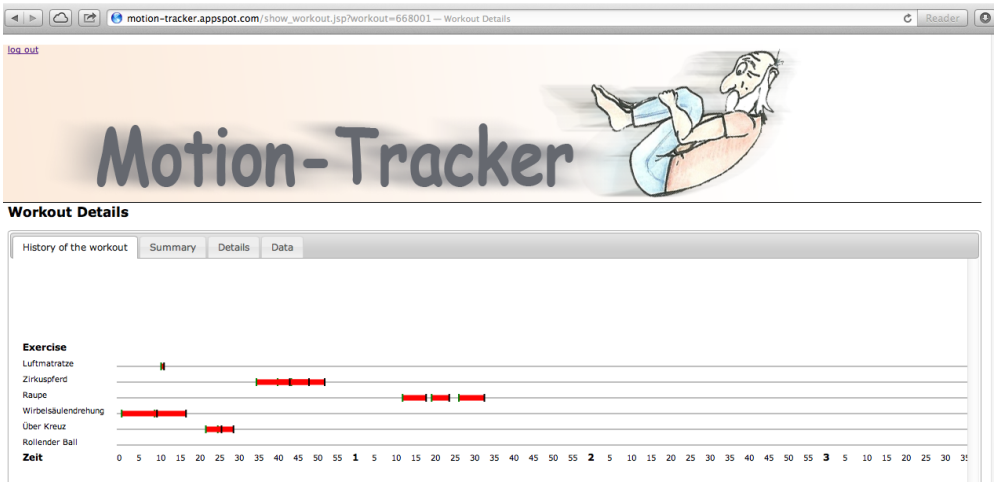


Abbildung 3.27: Grafische Aufbereitung eines absolvierten Bewegungsprogrammes eines Benutzers bzw. einer Benutzerin (eigener Entwurf)

- **Detailansicht**
  In der Detailansicht werden Informationen zu den einzelnen Übungswiederholungen dargestellt, dies ist in Abbildung 3.28 ersichtlich. Man sieht die verschiedenen Übungen und die einzelnen Wiederholungen. Für jede erkannte Wiederholung werden der Start- bzw. Endzeitpunkt, die Präzision (NCC) und die Dauer in Sekunden angezeigt. Des weiteren erhält man die Dauer aller Wiederholungen pro Übung. Wird eine Wiederholung ausgewählt, erhält man die Bewegungsdaten des extrahierten Abschnitts dieser Wiederholung.

- **Vollständige Daten**
  Die vollständigen Bewegungsdaten sind unter dem Daten-Tab ersichtlich, in dem auch eine Unterteilung in Rotations- und Beschleunigungsdaten möglich ist.

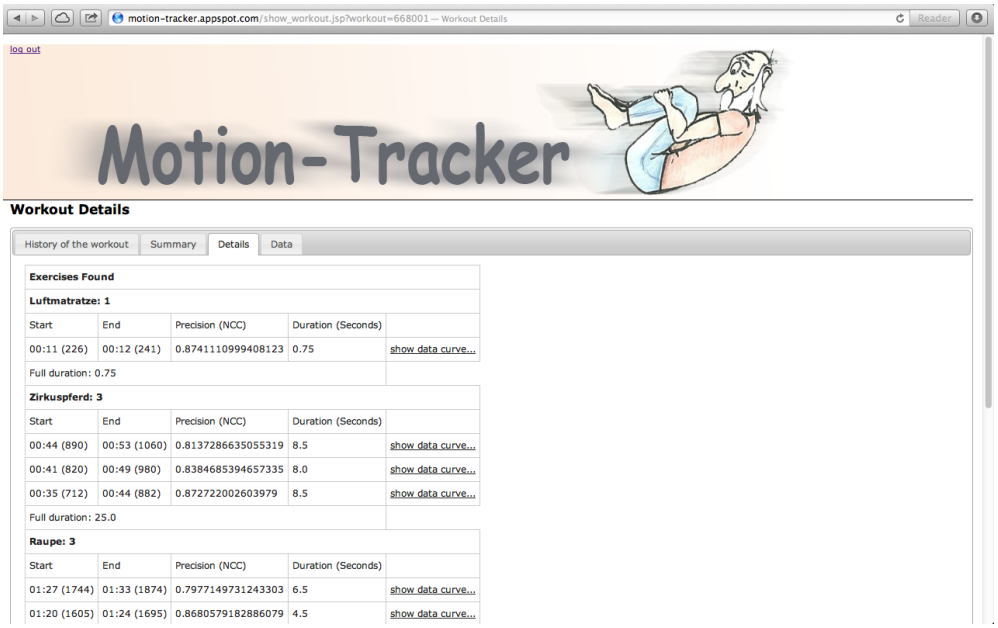


Abbildung 3.28: Tabellarische Darstellung eines absolvierten Bewegungsprogrammes eines Benutzers bzw. einer Benutzerin (eigener Entwurf)

4 Evaluierung, Diskussion, Zusammenfassung und Ausblick

4.1 Evaluation

Das MotionTracker-System mit seinen verschiedenen Methoden, welches in Kapitel 3 erläutert wird, wurde anhand von vier verschiedenen Probanden bzw. Probandinnen mit unterschiedlichen Voraussetzungen getestet und evaluiert. Dazu wurden Kinder unterschiedlichen Alters und Fitnesslevels dazu aufgefordert, das betroffene Trainingsprogramm an einem geeigneten Ort zu absolvieren. Jedes dieser Kinder bekam ein Gerät mit der dafür vorgesehenen Tasche an den verschiedenen Referenzpunkten angeschnallt und den Auftrag jeweils zehn Wiederholungen der sechs Übungen des Wiraculix Bewegungsprogrammes (laut Kapitel 3.2.1.1) zu absolvieren.
Als Gerät wurde ein iPhone 4, auf dem das Betriebssystem iOS 7 installiert war, und eine 3G-Internetverbindung zur Datenkommunikation verwendet. Wie bereits in Kapitel 3.3.1 erwähnt, beträgt die Abtastfrequenz 50 Hertz. Tabelle 4.1 fasst diese Testmerkmale nochmals zusammen.

Merkmal	Wert
Teilnehmer	4
Alter	9-16 Jahre
Gerät	iPhone 4
Betriebssystem	iOS 7
Datenverbindung	3G
Abtastfrequenz	50 Hz

Tabelle 4.1: Testmerkmale für die Evaluation des MotionTracker-Systems (eigener Entwurf)

Nach einer Einführung in das Bewegungsprogramm und Erklärung der verschiedenen Übungen, wurden die Tests für jedes der Kinder in folgenden Schritten durchgeführt:

- Anschnallen des Gerätes auf den Referenzpunkt am rechten Oberarm und Aus-

führung der davon betroffenen Übungen. Dies sind die drei Übungen *Über Kreuz, Wirbelsäulendrehung und Zirkuspferd.*

- Umschnallen des Geräts auf den Referenzpunkt am rechten Knöchel und Ausführung der davon betroffenen Übungen. Dies sind die zwei Übungen *Rollender Ball und Luftmatratze.*

- Als letztes wurde die Übung Raupe absolviert, da dies die einzige Übung mit dem Referenzpunkt am rechten Oberschenkel ist. Auch hier musste natürlich das Gerät umgeschnallt werden.

- Auswertung des Trainings mithilfe des MotionTracker-Systems

- Analyse der Auswertung

Die Kinder wurden angehalten jeweils zehn Wiederholungen pro Übung durchzuführen, dies gelang jedoch nicht exakt. Die Hauptgründe dafür waren, dass sich teilweise Probewiederholungen eingeschlichen haben, oder falsch gezählt wurde. Die genaue Auflistung der absolvierten Wiederholungen wird in Tabelle 4.2 dargestellt.

Übung	**Wiederholungen**				
	Proband				Gesamt
	A	B	C	D	
Über Kreuz	10	12	5	8	35
Wirbelsäulendrehung	12	10	4	7	33
Zirkuspferd	11	11	5	9	36
Rollender Ball	9	10	2	10	31
Luftmatratze	14	14	14	11	53
Raupe	10	10	5	10	35

Tabelle 4.2: Absolvierte Wiederholungen der Kinder für die Evaluation (eigener Entwurf)

Zur Analyse und Evaluation bzw. grafischen Veranschaulichung wurde das Programm Matlab [1] verwendet.
Die Gyroscope- und Accelerometer-Sensordaten liefern eine gute Grundlage für die weitere Bewegungsanalyse.

[1] MathWorks. Matlab - Die Sprache für technische Berechnungen, 2014. http://www.mathworks.de/products/matlab/, 24.02.2014.

Die absolvierten Trainings wurden zusätzlich gefilmt um anhand der Aufnahmen eine Einschätzung vornehmen zu können.

Tabelle 4.3 zeigt die Evaluierung der absolvierten Bewegungsprogramme der Probanden und Probandinnen, man sieht die einzelnen Klassifikationsraten der sechs Übungen pro Propand bzw. Probandin der Auswertung des MotionTracker-Systems sowie die aggregierten Raten der Auswertung.
Die **Trefferquote** beziehungsweise die Richtig-Positiv-Rate gibt dabei die richtigen Wiederholungen, die auch als solche klassifiziert wurden, an. Das Gegenstück dazu stellt die **Ausfallrate** beziehungsweise Falsch-Positiv-Rate dar, welche den Prozentwert, der falschen Wiederholungen die als richtig eingestuft wurden, angibt. Zu falschen Wiederholungen kann es kommen, wenn Datenabschnitte zufällig die Segmentierungskriterien erfüllen und somit als mögliche Wiederholungen identifiziert werden. Zusammenfassend gibt die **Korrektklassifikationsrate** prozentual die richtig klassifizierten Wiederholungen an, das heißt die richtigen Wiederholungen die auch als richtig klassifiziert wurden als auch die falschen Wiederholungen die richtigerweise als falsch erkannt wurden. Insgesamt wurden bei den Daten der Probanden 633 mögliche Wiederholungen aller Übungen gefunden, davon waren 223 richtige Ausführungen und 410 zufällige Segmente. Mit dem implementierten System konnten 189 der 223 richtigen Wiederholungen erkannt werden, was einer Trefferquote von *84,75 Prozent* entspricht. 73 der 410 falschen Wiederholungen wurden als richtig eingestuft, dies entspricht einer Ausfallrate von *17,8 Prozent*. Insgesamt ergab die Evaluierung eine Korrektklassifikationsrate von *83,1 Prozent*.
Anhand der Ergebnisse sieht man, dass beinahe alle Wiederholungen der Übungen erkannt werden. Dabei fallen vor allem Zusammenhänge zwischen Bewegungskomplexität bzw. Menge an Bewegung und der Trefferquote auf. Komplexere Bewegungen, wie zum Beispiel die Übung *Über Kreuz*, sowie Bewegungen mit wenig Bewegung, wie zum Beispiel die Übung *Raupe*, weisen die geringsten Trefferquoten auf. Alle anderen erreichen eine für die Praxis ausreichende Rate von über 95 Prozent.
Problematischer ist aber, dass oft zu viele Wiederholungen erkannt werden, wobei auch hier Zusammenhänge zwischen Komplexität und Grad der Bewegung erkennbar sind. Durch die vielen unterschiedlichen Kurven der selben Bewegungsmuster werden, zum Beispiel mit *31,52 Prozent* Ausfallrate bei der Übung *Über Kreuz*, viele zufällige Bewegungen als richtig erkannt. Auch sehr einfache Bewegungsmuster, wie zum Beispiel die der Übung *Rollender Ball*, weisen hohe Ausfallraten auf, da diese zum Beispiel beim schwunghaften Hinsetzen oder Aufstehen zustande kommen.
Man kann auch deutlich Unterschiede im Zusammenhang mit dem Alter und dem Fitnessgrad der Probanden und Probandinnen erkennen. Dabei haben die Jüngeren vor allem bei komplexen Übungen oft Probleme diese auch exakt auszuführen und weisen daher geringere Raten auf. Ähnlich ist der Zusammenhang auch zwischen Fitnessgrad und Genauigkeit, da die Bewegungen von Fitteren leichter und dadurch genauer ausgeführt werden können. Dies zeigt sich vor allem mit steigender Wiederholungszahl.

Die verschiedenen Prozentwerte sind natürlich auch abhängig von der Qualität der Segmentierung, da die verschiedenen Kriterien unterschiedlich viele Segmente überhaupt erkennen.

Tabelle 4.4 zeigt die Zusammenfassung der Fehlerzuordnung bei der Ausführung der Übungen durch die Probanden und Probandinnen. Es konnten insgesamt *74,32 Prozent* der gemachten Fehler richtig zugeordnet werden, wobei richtig im Sinne einer Übereinstimmung mit der Einschätzung anhand der Videoaufnahmen verstanden wird. Auch bei der Fehlerzuordnung können Zusammenhänge zwischen Komplexität der Bewegung und individuelle Eigenschaften der Probanden (Alter, Fitnessgrad etc.) erkannt werden. Für einfachere Übungen können Fehler besser zugeordnet werden als für komplexere und genauere Ausführungen (Bedingt durch Koordinationsfähigkeit bzw. Beweglichkeit abhängig von Alter bzw- Fitnessgrad) liefern ebenfalls stabilere Ergebnisse.

4.1.1 Template Matching vs. Mustererkennung

Im Laufe der Arbeit wurde neben der Mustererkennung (siehe Kapitel 3.2.2) auch anfänglich Template Matching (siehe Kapitel 3.2.3) eingesetzt. Dies führte aber zu mehreren Problemen, die zu einer neuen Implementierung der Klassifikation führten. Folgende Hindernisse sind aufgetreten:

- **Zeit als wesentlicher Faktor**
  Da beim Template Matching immer nur gleich lange Ausschnitte miteinander verglichen werden können, ist man immer von der Ausführungszeit, die für eine Wiederholung benötigt wird, abhängig und muss also Mindest- und Maximalzeiten definieren. Vor allem bei Kindern ist die Einhaltung eines speziellen Zeitfensters aber schwierig durchzusetzen beziehungsweise nur bedingt einsatzfähig. Versuche die zeitabhängigen Daten zu komprimieren beziehungsweise zu erweitern bringen zwar Verbesserungen, diese halten sich aber in Grenzen.

- **Genauigkeit korreliert zur Musterausführung**
  Die Bewegungsdaten einer Übungswiederholung sind unter anderem abhängig von der Ausführungszeit, Geschwindigkeit, Pausen und den körperlichen Proportionen der ausführenden Person. Dadurch werden Bewegungen von Menschen, die ähnlich dem Muster sind als genauer und besser eingestuft als andere. Dies muss aber nicht immer stimmen, da zum Beispiel langsamere Ausführungen oft wünschenswerter sind. Versuche, mehrere Templates zu verwenden, führen zwar zu besseren Ergebnissen, allerdings sind diese für die Praxis zu ungenau.

- **Genaue Fehler können nicht identifiziert werden**
  Da unterschiedliche Ausführungen, die nicht zwangsläufig besser oder schlechter sind, verschiedene Korrelationen liefern, ist es praktisch unmöglich mit dieser statischen Methode genaue Fehler in der Übungsausführung zu klassifizieren. Oft haben schlechtere zufällig höhere Korrelationen zum Muster als bessere Ausführungen.

- **Richtiges Mittelmaß des Schwellwertes**
  Durch einen geringen Schwellwert werden zwar alle Übungen erkannt aber zu viele falsche Wiederholungen als richtig klassifiziert. Umgekehrt werden zu viele richtige Übungen nicht erkannt wenn der Schwellwert zu hoch gewählt wurde.

Die Erkennung der Übungen ist aber auch mit dieser Variante möglich, Tabelle 4.5 zeigt einen Vergleich zwischen dem Template-Matching-Verfahren und der final implementierten Mustererkennung. Zu den Übungen *Über Kreuz* und *Wirbelsäulendrehung* zeigt Tabelle 4.6 zusätzlich eine detailliertere Auflistung.
Man sieht, dass die Auswertung deutlich abhängig vom Probanden bzw. der Probandin ist, umso ähnlicher die Übungen der Musterausführung ausgeführt werden, umso besser die Klassifizierung. Dies führt teilweise sogar zu genaueren Ergebnissen, vor allem bei Übungen die weniger Fehler zulassen, wie zum Beispiel die *Wirbelsäulendrehung*. Umso komplexer die Übung, umso ungenauer ist meist die Ausführung und damit auch das Template Matching. Dies ist vor allem bei der Übung *Über Kreuz* deutlich bemerkbar. Aus diesem Grunde wurden bei allen Probanden und Probandinnen, ausgenommen A, welcher diese Übung sehr sauber und somit ähnlich zum Template ausgeführt hat, zu wenig beziehungsweise gar keine Wiederholungen erkannt.
Auch die großen Zeitfenster die für die Ausführung einer Wiederholung definiert wurden, stellen für die Template-Matching-Methode Probleme dar. Dies ist zum Beispiel bei der Übung *Zirkuspferd* zu erkennen, bei der die Wiederholungen der Kinder überhaupt nicht erkannt wurden.
Die dargestellten Prozentwerte unterscheiden sich leicht von jenen in Tabelle 4.3, da die durch die Segmentierung gefundenen, aber falschen Wiederholungen nicht in der Berechnung berücksichtigt werden.

4.1.2 Auswahl verschiedener Merkmale zur Mustererkennung

Neben verschiedenen Verfahren wurden auch unterschiedliche Merkmale zur Mustererkennung verwendet. Die Auswahl der Merkmale und der weiteren Parameter wie Kernel und dessen Parameter (Hsu u. a., 2010), oder generell das Modell, das verwendet wird (Duda u. a., 2000), beeinflusst die Qualität des Klassifikators maßgeblich. Im Rahmen

der Entwicklung wurden verschiedene Merkmale getestet und evaluiert, dabei erwiesen sich die wie in (Li u. a., 2007) vorgeschlagenen Singulärwerte der Sensordatenmatrix als besonders geeignet.
Evaluiert wurden zwei Varianten, wobei erstens nur die Singulärwerte als Merkmale und eine RBF-Kernelfunktion und zweitens neben diesen auch noch folgende Merkmale mit linearem Kernel 3.2.2.4 verwendet wurden:

- Mittelwerte der einzelnen Variablen
- Standardabweichung der einzelnen Variablen
- Korrelation der einzelnen Rotationsdimensionen zueinander (NCC)
- Korrelation der einzelnen Beschleunigungsdimensionen zueinander (NCC)
- Korrelation der Rotationsdaten zu den Beschleunigungsdaten (NCC)

Tabelle 4.7 zeigt den Vergleich zwischen diesen beiden Mustererkennungsstrategien. Man sieht vor allem, dass die Anwendung mehrerer Merkmale zu stabileren Ergebnissen führt, was sich auch in den Korrektklassifikationsraten zeigt. Es konnten zwar bei beiden Varianten gute Trefferquoten erzielt werden, allerdings besitzen die Singulärwerte alleine zu wenig Trennfähigkeit, um richtige von falschen Segmenten ausreichend unterscheiden zu können. Aus diesem Grunde wurden zu viele falsche Wiederholungen als richtig erkannt, was sich in der hohen Ausfallrate widerspiegelt.
Für die Übung *Raupe* konnte mit dieser kleineren Auswahl an Merkmalen allerdings sowohl eine deutlich bessere Trefferquote, als auch Ausfallrate und somit eine höhere Korrektklassifikationsrate erreicht werden. Dies deutet darauf hin, dass für diese Übung die Verwendung aller vorhandenen Merkmale zu einer Überanpassung (engl. *Overfitting*), möglicherweise aufgrund der geringen Bewegung in der Ausführung, führt. Eine Kombination aus beiden Methoden wäre hier empfehlenswert, da die Trefferquote, mit *+51,43 Prozent*, deutlich gesteigert werden kann.
Durch weitere Optimierung der Parameter und Verwendung von qualitativ und quantitativ besseren Trainingsdaten könnten beide Varianten wahrscheinlich noch weiter verbessert werden.

Übung	**Klassifikationsraten in %**				
	Proband				Gesamt
	A	B	C	D	
Über Kreuz					
Trefferquote (Richtig-Positiv-Rate)	100	91,67	80	100	94,29
Ausfallrate (Falsch-Positiv-Rate)	30,43	45,83	30,77	21,88	31,52
Korrektklassifikationsrate	78,79	63,64	72,22	83,33	75,59
Wirbelsäulendrehung					
Trefferquote (Richtig-Positiv-Rate)	100	100	100	85,71	96,97
Ausfallrate (Falsch-Positiv-Rate)	10,53	0	0	0	3,77
Korrektklassifikationsrate	93,55	100	100	94,74	96,51
Zirkuspferd					
Trefferquote (Richtig-Positiv-Rate)	100	100	100	100	100
Ausfallrate (Falsch-Positiv-Rate)	0	41,18	33,33	11,11	22,5
Korrektklassifikationsrate	100	75	87,5	94,44	88,16
Rollender Ball					
Trefferquote (Richtig-Positiv-Rate)	100	90	100	100	96,77
Ausfallrate (Falsch-Positiv-Rate)	30	40	40	0	31,03
Korrektklassifikationsrate	84,21	75	71,42	100	83,33
Luftmatratze					
Trefferquote (Richtig-Positiv-Rate)	100	78,57	64,29	90,91	98,11
Ausfallrate (Falsch-Positiv-Rate)	7,41	19,23	16,67	36,84	12,22
Korrektklassifikationsrate	95,12	80	75	73,33	91,61
Raupe					
Trefferquote (Richtig-Positiv-Rate)	10	0	100	80	40
Ausfallrate (Falsch-Positiv-Rate)	0	0	5,56	16,22	6,6
Korrektklassifikationsrate	71,88	74,36	95,65	82,98	80,14
Gesamt					
Trefferquote (Richtig-Positiv-Rate)	86,36	77,61	82,86	92,73	84,75
Ausfallrate (Falsch-Positiv-Rate)	12,5	23,08	16,18	18,58	17,8
Korrektklassifikationsrate	87,07	77,17	83,5	85,12	83,1

Tabelle 4.3: Die Klassifikationsraten der im Rahmen der Evaluation absolvierten Bewegungsprogramme (eigener Entwurf)

Übung	**Erkannte Fehler in Prozent**				
	Proband				Gesamt
	A	B	C	D	
Über Kreuz	76,67	69,7	33,33	87,5	66,80
Wirbelsäulendrehung	70,83	60	75	100	76,46
Zirkuspferd	45,45	72,73	80	77,78	68,99
Luftmatratze	67,86	18,18	55,56	80	55,40
Raupe	100	0 (keine erkannt)	100	68,75	89,58
Gesamt	72,16	73,54	68,75	82,81	74,32

Tabelle 4.4: Die Klassifikationsraten der zugeordneten Fehler zur jeweiligen Fehlerkategorie (eigener Entwurf)

Verfahren	**Erkannte Wiederholungen** Absolviert	Treffer	Ausfall	Korrektklassifikationsrate (%)
	Über Kreuz			
Template-Matching	35	7	0	20
Mustererkennung	35	33	29	51,56
	Wirbelsäulendrehung			
Template-Matching	34	22	0	64,71
Mustererkennung	34	32	2	88,89
	Zirkuspferd			
Template-Matching	36	0	0	0
Mustererkennung	36	36	9	80
	Rollender Ball			
Template-Matching	34	9	2	25
Mustererkennung	34	30	9	69,77
	Luftmatratze			
Template-Matching	53	11	5	18,97
Mustererkennung	53	44	17	62,86
	Raupe			
Template-Matching	35	4	1	11,11
Mustererkennung	35	14	7	33,33
	Gesamt			
Template-Matching	227	53	8	22,55
Mustererkennung	227	189	73	63

Tabelle 4.5: Vergleich des Template-Matching-Verfahrens und der Mustererkennungsmethode (eigener Entwurf)

Verfahren	**Erkannte Wiederholungen**			
	Absolviert	Treffer	Ausfall	Korrektklassifikationsrate (%)
	Über Kreuz			
Proband A				
Template-Matching	10	7	0	70
Mustererkennung	10	10	7	58,82
Proband B				
Template-Matching	12	0	0	0
Mustererkennung	12	11	11	47,82
Proband C				
Template-Matching	5	0	0	0
Mustererkennung	5	4	4	44,44
Proband D				
Template-Matching	8	0	0	0
Mustererkennung	8	8	7	53,33
	Wirbelsäulendrehung			
Proband A				
Template-Matching	12	10	0	83,33
Mustererkennung	12	12	2	85,71
Proband B				
Template-Matching	10	2	0	20
Mustererkennung	10	10	0	100
Proband C				
Template-Matching	5	4	0	80
Mustererkennung	5	4	0	80
Proband D				
Template-Matching	7	6	0	85,71
Mustererkennung	7	6	0	85,71

Tabelle 4.6: Vergleich des Template-Matching-Verfahrens und und der Mustererkennungsmethode im Detail anhand der Übungen *Über Kreuz* und *Wirbelsäulendrehung* (eigener Entwurf)

Übung	**Merkmale**	**Raten in Prozent**		
		Treffer	Ausfall	Korrektklassifikation
Über Kreuz	Alle	94,29	31,52	75,59
	Singulärwerte	97,14	91,3	33,07
Wirbelsäulendrehung	Alle	96,97	3,77	96,51
	Singulärwerte	93,94	15,09	88,37
Zirkuspferd	Alle	100	22,5	88,16
	Singulärwerte	91,67	30	80,26
Rollender Ball	Alle	100	31,03	85
	Singulärwerte	54,84	24,14	65
Luftmatratze	Alle	83,02	18,89	81,82
	Singulärwerte	75,47	25,56	74,83
Raupe	Alle	40	6,6	80,14
	Singulärwerte	91,43	5,66	93,62
Gesamt	Alle	85,2	17,8	83,25
	Singulärwerte	83,86	34,15	72,2

Tabelle 4.7: Vergleich von verschiedenen Merkmalen zur Klassifizierung mithilfe von Support Vector Machines (eigener Entwurf)

4.2 Diskussion

Unterschiedliche Eignung der Methoden

Die in dieser Arbeit untersuchten Methoden eignen sich unterschiedlich zur Erkennung von Pilatesübungen bei Kindern anhand von Sensorinformationen des Apple iPhones. Template Matching kann bei genauer Ausführung und somit höherer Korrelation zum Template bessere und genauere Ergebnisse liefern als die implementierten Mustererkennungsmethoden, allerdings können Kinder nur sehr schwer die Übungen exakt bewältigen. Die implementierte Mustererkennungsmethode kann auch ungenauere Versuche erkennen, mögliche Fehler klassifizieren und somit die einzelnen Wiederholungen bewerten. Umso besser diese absolviert werden umso höher fällt deren Bewertung aus. Die Kinder werden dadurch nicht aufgrund von nicht erkannten Wiederholungen oder falschen Summen irritiert beziehungsweise frustriert, erhalten Verbesserungsvorschläge und zusätzlich einen Ansporn, die Übungen genauer auszuführen.

Template Matching

Die konkrete Zuordnung der Wiederholungen zu typischen Ausführungsfehlern konnte mit einfachem Template Matching nicht umgesetzt werden. Die Ergebnisse waren zu stark abhängig von der Musterausführung und dem Probanden bzw. der Probandin der bzw. die diese erstellt hat. Durch die Einführung mehrerer Templates und mehrfachem Template Matching konnten Verbesserungen erzielt werden, die Genauigkeit lag jedoch unter der der Mustererkennung. Weiters spielte die Geschwindigkeit der Klassifikation eine entscheidende Rolle, die Berechnung der Koeffizienten nahm, im Vergleich, sehr viel Zeit in Anspruch und lieferte lange Wartezeiten auf die Rückmeldung des Systems. Auch der Parameter Ausführungszeit stellt ein Problem dar, da Kinder Übungswiederholungen nur sehr schwer in einem bestimmten Zeitfenster absolvieren können. Wiederholungen die länger oder kürzer dauern, werden so übersehen und können nicht gezählt werden.

Mustererkennung

Die Klassifizierung mithilfe der Mustererkennung ist sehr stark von den verschiedenen Variablen abhängig. So spielt die Qualität und die Anzahl der zur Verfügung stehenden Trainingsdaten eine entscheidende Rolle, als auch die Wahl der Merkmale. Der entscheidende Vorteil liegt aber daran, dass die Ausführungszeit vernachlässigt werden kann, und auch zu schnelle beziehungsweise zu langsame Ausführungen als Versuche erkannt werden. Durch den Einsatz mehrere Klassifikationsstufen können ungenaue Versuche von schönen Ausführungen und sogar typischen Fehlern unterschieden werden. Diese Unterscheidung ermöglicht es auch, den Probanden und Probandinnen Rückmeldungen zur Genauigkeit und gegebenenfalls Verbesserungsvorschläge zu liefern.

Komplexität der Übungen

Auffällig ist, dass egal welche Methode eingesetzt wird, einfache Übungen immer besser erkannt werden und robuster gegen Falscherkennungen sind als komplexere. Dies liegt daran, das die Bewegungsdaten aller Beispiele dadurch ähnlicher sind, da die genaue Ausführung leichter vollzogen werden kann. Komplexere Übungen können oft sehr unterschiedliche Bewegungsdaten liefern, da zum Beispiel Ausführungspausen, Bewegungen in andere Richtungen, Geschwindigkeit und Instabilität diese stark beeinflussen. Diese Einflussgrößen kommen zum Beispiel durch schwinden der Kräfte, Koordinationsschwierigkeiten oder auch dem Grad der Fitness der Probanden und Probandinnen zustande.

Grenzen des Systems

Natürlich kann eine komplette Kontrolle der Bewegung anhand eines Referenzpunktes nicht gewährleistet werden, daher muss für diesen eine signifikante Stelle der Übung gewählt werden. Wird die Hauptbewegung also zum Beispiel mit den Armen ausgeführt, bietet sich als Referenzpunkt der Oberarm an. Die Bewegung des zweiten Armes und der Beine lässt sich hier nicht kontrollieren. Dadurch können falsche oder unvollständige Bewegungen als richtig erkannt werden oder Daten, die durch das einfache Bewegen des Gerätes mit der Hand erzeugt werden, positive Ergebnisse liefern. Andererseits können, aus bereits besprochenen Gründen, echte Versuche nicht erkannt werden.
Ein weiteres Problem stellen Übungen ohne Bewegung dar, die dadurch zu wenig signifikante Bewegungsdaten liefern. Im Rahmen dieser Arbeit konnte zum Beispiel die Übung *Liegestütz*, die daraus bestand die Liegestützposition über einen bestimmten Zeitraum zu halten, nicht erkannt werden beziehungsweise nicht ausreichend unterschieden werden.

Reaktion der Kinder

Bei den Besuchen in den Schulen zu Testzwecken beziehungsweise der Trainingsdatensammlung konnte eine sehr positive Reaktion der Kinder, auf das neue Medium iPhone als Hilfsmittel, beobachtet werden. Das Interesse und das Verständnis für den Umgang mit der Applikation war durchwegs vorhanden und als positiv zu bewerten.

Die letztendlich gewählte Methode und Auswahl der Merkmale konnte die Bewegungen zwar nicht immer exakt zuordnen, aber liefert ein für die Praxis brauchbares Ergebnis. Die Anforderung, ein Bewegungsprotokoll zur Kontrolle der selbstständigen Tätigkeit der TeilnehmerInnen zuhause konnte somit erfüllt werden. Auch die Rückmeldungen des Systems und die Verbesserungsvorschläge spiegeln den Bewegungsablauf wider.

4.3 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein System entwickelt, das die automatische Protokollierung von Bewegungsprogrammausführungen ermöglicht. Es wurden folgende drei Komponenten entwickelt (siehe Kapitel 3.1):

- iOS-Client zur Aufnahme der Bewegungsdaten (MotionTracker-Applikation)
- Webservice zur Auswertung und Klassifizierung der gewonnen Bewegungsdaten
- Backend zur Aufbereitung der gewonnenen Informationen (MotionTracker-Webapplikation)

Dieses System konnte den Ansprüchen, ein Bewegungsprogramm zu ersetzten und der Studienbetreuerin ein Kontrollinstrument zu liefern, gerecht werden. Die MotionTracker-Applikation nimmt die notwendigen, durch die im iPhone vorhandenen Sensoren (Accelerometer und Gyroscope), Bewegungsdaten auf und benutzt ein Webservice zur Klassifizierung. Die Ergebnisse werden dann übersichtlich dargestellt. Die MotionTracker-Webapplikation bereitet Informationen zu bereits absolvierten Bewegungsprogrammen für TeilnehmerInnen und Studienbegleiterin graphisch auf.
Zur Klassifizierung der Übungen des Wiraculix Bewegungsprogrammes wurden verschiedene Methoden und Varianten getestet, die Evaluierung (siehe Kapitel 4.1) zeigt, dass diese zwar nicht immer exakt, aber für den vorhergesehenen Zweck ausreichend erkannt werden können.
Damit konnte gezeigt werden, dass die vom iPhone gelieferten Bewegungsdaten und deren Genauigkeit sich zur Mustererkennung eignen und derartige Analysen und Klassifikationen erlauben.

Die Klassifizierung mithilfe von Support Vector Machines kann sicher noch optimiert werden. Mögliche zukünftige Schritte könnten diesbezüglich die Erzeugung qualitativ und quantitativ hochwertigerer Trainingsdaten, die Verwendung verschiedener Kernelfunktionen und die weitere Optimierung der Parameter dieser sein. In damit verbundenen Arbeiten, werden auch immer wieder weitere Machine-Learning-Techniken, wie Hidden-Markov-Models etc. zur Klassifizierung von Bewegungsdaten vorgeschlagen, welche möglicherweise bessere Ergebnisse liefern.

Ein weiterer essentieller Schritt ist sicherlich die Gamification (Ebner u. Schön, 2011) der Applikation. Dies bedeutet, dass spielerische Elemente eingebaut werden, um die Motivation der Kinder zur Benutzung zu steigern. Macht die Applikation den Kindern Spaß, wird die Teilnahme erhöht und somit das Ergebnis verbessert. Zum Beispiel

könnte ein Spiel implementiert werden, bei dem die Kinder die Übungen nacheinander ausführen müssen und abhängig von der Genauigkeit und Wiederholungsanzahl mehr oder weniger Punkte bekommen. Abhängig von dieser Punkteanzahl könnten sie in höhere Ebenen oder Schwierigkeitsstufen gelangen. Dies hätte zusätzlich den Vorteil, dass die korrekte Übungsabfolge und Wiederholungsanzahl eingehalten werden würde, da sonst zum Beispiel die nächste Stufe nicht erreicht werden kann.

Literaturverzeichnis

[Almer 2011] Almer, Stefan: *Mobile Applications for Fall Detection in the Area of Ambient Assisted Living*, Graz University of Technology, Diplomarbeit, 2011

[Apple Inc. 2014] Apple Inc.: *Event Handling Guide for iOS*. 2014. – https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/EventHandlingiPhoneOS.pdf, 15.02.2014

[Arnold u. Weber 2011] *Kapitel* Die "Netzgeneration Empirische Untersuchungen zur Mediennutzung bei Jugendlichen. In: **(Ebner u. Schön, 2011)**

[Bishop 2006] Bishop, C.M.: *Pattern Recognition and Machine Learning*. Springer, 2006 (Information Science and Statistics). http://books.google.at/books?id=kTNoQgAACAAJ. – ISBN 9780387310732

[Boser u. a. 1992] Boser, E. ; Guyon, I. ; Vapnik, V.: A training algorithm for optimal margin classifiers. In: *In Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ACM Press, 1992, S. 144–152

[Cortes u. Vapnik 1995] Cortes, Corinna ; Vapnik, Vladimir: Support-Vector Networks. In: *Mach. Learn.* 20 (1995), September, Nr. 3, 273–297. http://dx.doi.org/10.1023/A:1022627411411. – DOI 10.1023/A:1022627411411. – ISSN 0885–6125

[Deutscher Bundestag 2014] Deutscher Bundestag: *Telemedizin*. 2014. – http://www.bundestag.de/dokumente/analysen/2011/Telemedizin.pdf, 13.02.2014

[Deutschmann 2014] Deutschmann, Dietlind: *Kindgerechtes Pilates als Intervention zur Haltungsstabilisierung und Verbesserung von Wirbelsäulenfehlhaltungen bei SchülerInnen im Alter von 10-12 Jahren*, Karl-Franzens Universität Graz, Diss., 2014. – in work

[Duda u. a. 2000] Duda, Richard O. ; Hart, Peter E. ; Stork, David G.: *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000. – ISBN 0471056693

[Ebner u. Schön 2011] Ebner, Martin (Hrsg.) ; Schön, Sandra (Hrsg.): *Lehrbuch*

für Lernen und Lehren mit Technologien. 2011 http://l3t.tugraz.at/index.php/LehrbuchEbner10/issue/current

[ECMA International 2013] ECMA International: *The JSON Data Interchange Standard.* 2013. – http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, 08.04.2013

[Eysenbach 2001] Eysenbach, G.: *What is e-health?* J Med Internet Res 2001;3(2):e20, 2001. – http://www.jmir.org/2001/2/e20/,13.02.2014

[Firemint Pty Ltd 2014] Firemint Pty Ltd: *Real Racing 2.* iTunes, 2014. – https://itunes.apple.com/AT/app/id386568787?mt=8&affId=2083439, 15.02.2014

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-oriented Software.* Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0–201–63361–2

[Grimus u. Ebner 2014] Grimus, M. ; Ebner, M.: *Learning with Mobile Devices Perceptions of Students and Teachers in Lower Secondary Schools in Austria.* ED-Media, 2014. – accepted, in print

[Hsu u. a. 2010] Hsu, Chih-Wei ; Chang, Chih-Chung ; Lin, Chih-Jen: *A Practical Guide to Support Vector Classification.* 2010. – http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf, 19.02.2014

[Lange 2010] Lange, Kenneth: *Singular Value Decomposition.* Springer New York, 2010 (Statistics and Computing). – 129–142 S. http://dx.doi.org/10.1007/978-1-4419-5945-4_9. http://dx.doi.org/10.1007/978-1-4419-5945-4_9. – ISBN 978–1–4419–5944–7

[Le u. a. 2011] *Kapitel* Game-Based Learning - Spielend Lernen? In: **(Ebner u. Schön, 2011)**

[Li u. a. 2007] Li, Chuanjun ; Kulkarni, PunitR. ; Prabhakaran, B.: Segmentation and recognition of motion capture data stream by classification. In: *Multimedia Tools and Applications* 35 (2007), Nr. 1, 55-70. http://dx.doi.org/10.1007/s11042-007-0119-6. – DOI 10.1007/s11042–007–0119–6. – ISSN 1380–7501

[Mellone u. a. 2012] Mellone, S. ; Tacconi, C. ; Schwickert, L. ; Klenk, J. ; Becker, C. ; Chiari, L.: Smartphone-based solutions for fall detection and prevention: the FARSEEING approach. In: *Zeitschrift für Gerontologie und Geriatrie* 45 (2012),

Nr. 8, 722-727. http://dx.doi.org/10.1007/s00391-012-0404-5. – DOI 10.1007/s00391–012–0404–5. – ISSN 0948–6704

[ROLOCULE GAMES PRIVATE LIMITED 2014] ROLOCULE GAMES PRIVATE LIMITED: *Motion Tennis*. iTunes, 2014. – https://itunes.apple.com/de/app/motion-tennis/id614112447?mt=8, 15.02.2014

[Szeliski 2010] Szeliski, Richard: *Computer Vision: Algorithms and Applications*. Springer-Verlag, 2010. – http://szeliski.org/Book/, 19.02.2014