Universität Konstanz
FB Informatik und Informationswissenschaft
Bachelor-Studiengang Information Engineering

Masterarbeit

Midas – Ein Toolkit für das Erkennen und Analysieren von Gesten

zur Erlangung des akademischen Grades eines
Master of Science (M.Sc.)

Studienfach: Information Engineering

Schwerpunkt: Angewandte Informatik

Themengebiet: Mensch Computer Interaktion

von

Jan Oke Tennié

(Matr.-Nr. 01/737256)

Erstgutachter: Prof. Dr. Harald Reiterer
Zweitgutachter: Prof. Dr. Werner König
Betreuer: Daniel Klinkhammer
Einreichung:

Danksagung

An dieser Stelle möchte ich Paula Erdös für ihre Unterstützung bei dieser Arbeit danken. Ein weiterer Dank geht an Daniel Klinkhammer für die gute Betreuung meiner Arbeit.

Zusammenfassung

In der vorliegenden Masterarbeit wird eine softwaretechnische Lösung für das Erkennen von Hand- und Armgesten präsentiert. Die eingesetzte Echtzeiterkennung basiert dabei auf Sensoren, die die Bewegung des Armes und der Hand aufnehmen. Die eingesetzte Hardware zum Aufzeichnen der Gesten ist das *Myo* Armband. Die aufgezeichneten Daten werden dann an eine eigens dafür programmierte Windowsapplikation mit dem Namen *Midas* gesandt, in der die eigentliche Gestenerkennung stattfindet. Vor der Erkennung der Gesten muss der Anwender diese jedoch für das System trainieren, denn nur so weiß der eingesetzte Algorithmus, wie die Geste aussieht. Um den richtigen Algorithmus zu finden, wurden viele Wege ausprobiert und analysiert. Um deshalb die Flexibilität der Datenpipeline zu gewährleisten, wurde *Midas* als *Node Designer* konzipiert. Dadurch können verschiedene Algorithmen mit Hilfe der Maschine-Learning-Pipeline modelliert und getestet werden. Schlussendlich kam man zu dem Ergebnis, dass sich der *Dynamic-Time-Warping Algorithmus* am besten dazu eignet, um eine Gestenerkennung durchzuführen.

Nach dieser Erkenntnis wurden zwei Szenarien umgesetzt, um den *Dynamic-Time-Warping Algorithmus* eingehend zu testen und dessen Eignung zu untermauern. Bei dem ersten Szenario handelte es sich um eine Touch-Gestenerkennung. Das implementierte System namens *Midas Touch,* macht es möglich die Touch-Interaktion auf einem touchsensitiven Display einem ausführenden Benutzer zu zuordnen. Das zweite Szenario ermöglicht es dem Anwender beliebige Gesten in gesprochene Sprache zu übersetzten. Das dafür entwickelte System *Gvoice* ermöglicht es Gesten auf einfachste Weise zu trainieren und zu testen. Bei beiden Systemen läuft der *Midas Node Designer* im Hintergrund, womit der Anwender in Echtzeit die Möglichkeit hat, die eingesetzte Pipeline umzustellen und neue Wege zu testen. Die zwei entwickelten Einsatzszenarien – *Midas Touch* und *Gvoice*, beschreiben nur im Ansatz die Möglichkeiten der Benutzung von *Midas*. Aus diesem Grund wurde auf Basis von Buxtons State Model (Buxton 1990), ein Modell für dieses System entwickelt, um dessen Potential zu veranschaulichen.

Anschließend wurden die beiden entwickelten Szenarien mit Usern in einer Studie evaluiert. Das Ergebnis der Evaluation von *Midas Touch* zeigte, dass die mittlere Erkennungsarte der Touchgeste bei 88% liegt. Bei *Gvoice* wurden beliebige Gesten mit 85%iger Wahrscheinlichkeit richtig erkannt. Im Schnitt wurden dabei 12 Gesten pro Proband getestet. Dieses Ergebnis zeigt, dass eine Gestenerkennung mittels Sensoren, unter Verwendung des *Dynamic-Time-Warping Algorithmus*, im Rahmen des Machbaren liegt.

Abstract

In this thesis, a software technical solution for the identification of hand and arm gestures is being presented. The real-time detection used is based on sensors that record the movement of the arm and the hand. The hardware used to record these gestures is the so called *Myo* bracelet. After the data is being recorded, it is sent to an implemented Windows application named *Midas*, where the actual gesture recognition is calculated. Before *Midas* can recognize any gestures, it is necessary, for the user to train some gestures for the system, in order to make sure, that the used algorithm knows how the gesture looks like. To find the right algorithm, many approaches have been tested and analyzed. As to ensure the flexibility of the data pipeline, *Midas* was designed as Node Designer. This allows the user to model and test different algorithms, using the machine-learning pipeline. Finally, one came to the conclusion that the *dynamic time warping* algorithm works best for the task of a gesture recognition.

With this knowledge, two scenarios were implemented in order to test the *dynamic time warping* algorithm in detail and to verify its suitability. The first scenario contains a touch gesture recognition. The implemented system, called *Midas Touch*, allows an assignment of a touch interaction on a touch-sensitive display to the executing user. The second scenario enables the user to translate any gesture into spoken language. The particularly therefore developed system *Gvoice*, implements the training and the testing of gestures, in the most possible user friendly way. Both systems are supported by the *Midas Node Designer*, that runs in the background. This enables the user to change the used pipeline in real time in order to test new approaches. The two developed scenarios - *Midas Touch* and *Gvoice* only touch the surface of the possibilities *Midas* can offer. For this reason, a model of this system was created to illustrate its potential. This model is based on the *Buxton State Model* (Buxton 1990).

Subsequently, both scenarios were evaluated with users in two studies. The evaluation of *Midas Touch* showed, that the average detection rate of touch gesture accuracy is 88%. The system *Gvoice* identified different gestures with an 85% probability correctly. On average, twelve gestures per subject were tested. This result indicates that a gesture recognition using sensors, in combination with the *dynamic time warping* algorithm, lies within the scope of feasibilities.

1 Inhalt

Danksagung **2**

Zusammenfassung **3**

Abstract **4**

1 Einleitung **7**

2 Aufbau der Arbeit **8**

3 Analysephase **9**

3.1 State of the Art 9

3.2 Anforderungen 13

3.3 Vision 15

4 Entwurfsphase **18**

4.1 Sketches des dritten Prototypen (Version 2) 18

4.1.1 Sketches für das Trainieren von Gesten 18

4.1.2 Überblick über die Daten 19

4.1.3 Sketch des Servers 20

4.2 Sketche des finalen Prototypen 21

4.2.1 Design der Nodes 21

4.2.2 Überblick über die Daten 22

4.2.3 Sketch des Servers 23

4.2.4 Sketch des Clients 24

5 Prototyp Entwicklungsphase **25**

5.1 Maschinelles Lernen 25

5.2 Hardware 27

5.3 Midas Prototyp drei (Version 2) 29

5.3.1 Multiclass Logistische Regression 30

5.3.2 Fazit des Prototypen 32

5.4 Midas Node Designer (Version 3) 33

5.4.1 Installation 33

5.4.2 Bedienung 33

5.4.3 Dynamik Time Warping 34

5.4.4 Fazit Midas Nodedesigner 37
5.5 Szenarien 38
5.5.1 Toucherkennung 38
5.5.2 Midas Gvoice (Gesture Voice) 41
5.5.3 Gestenanalyse 60
5.6 Gesten State Model 63
6 Evaluation 68
6.1 Studienablauf 68
6.1.1 Midas Touch 68
6.1.2 Gvoice 69
6.2 Durchführung 71
6.3 Ergebnis der Evaluation 72
6.3.1 Midas Touch Ergebnisse 72
6.3.2 Gvoice 72
7 Diskussion 79
8 Fazit 81
9 Ausblick 82
10 Referenzen 83

1 Einleitung

Viele Systeme lassen sich heute über Gesten steuern, angefangen von dem Fernseher bis hin zu Multimediasystemen in Autos. Dabei variiert die Komplexität der Gesten stark. Im 7er-BMW gibt es insgesamt fünf einfache Handgesten[1], während für das Steuern des Fernsehers mit dem *Kymera Magic Wand*[2] dreizehn Hand- und Armgesten zu Verfügung stehen. Mit der Microsoft Kinect sind sogar noch mehr Gesten möglich und hier kann der komplette Körper eingesetzt werden. Studien zeigen, dass eine Gestensteuerung genauso funktionieren soll wie eine Spracherkennung, denn es sollen natürliche Gesten erkannt werden. Diese natürlichen Gesten variieren nicht nur in ihrer Komplexität, sondern sind auch von Land zu Land verschieden (Meier et al. 2014). Um die Wünsche der Anwender umzusetzen, braucht es Systeme, die solche Hand- und Armgesten erkennen. Dies geschieht auf zwei möglichen Wegen. Die erste Variante ist, die Gesten mit Hilfe von optischen System zu erkennen. Dabei wird meist ein Tiefenbild über Infrarot LEDs und zwei Kameras ermittelt und aus diesen dann das Skelett des Menschen extrahiert. So ein Skelett wird benötigt, um die Bewegungserkennung zu verbessern, denn so werden unnatürliche Bewegungen vermieden. Dies ist der Fall bei der *Leap Motion*[3] und der *Microsoft Kinect*[4]. Alternativ werden auch markerbasierte Systeme eingesetzt. Solche Systeme werden unteranderem in der Filmindustrie als „motion capturing“ verwendet. Den Vorteil, den markerbasierte Systeme bieten ist, dass sie eine sehr hohe Genauigkeit aufweisen. Dafür sind sie allerdings auch sehr teuer, benötigen viel Platz und sehr viel Vorarbeit. Die Vorarbeit besteht darin, dass die Marker (grafische 2D Marker oder Infrarot Marker) erst platziert und auf das System abgestimmt werden müssen.

Die zweite Variante Gesten zu erkennen bzw. aufzuzeichnen, ist über Sensoren am Körper. Auch hier gibt es bereits kommerzielle Lösungen, die wieder für die Filmindustrie entwickelt wurden (wie das *Xsens MVN*[5]). Dabei wird das Skelett durch Bewegungssensoren an mehreren Punkten am Körper berechnet. Auch hier kommt das Skelett zum Einsatz, um unmenschliche Bewegungen zu vermeiden. Die Vorteile, die solche Systeme bieten sind, dass sie verhältnismäßig günstig und nahezu überall einsetzbar sind. Denn ein auf Sensoren basiertes System ist nicht auf einen speziellen Raum angewiesen. Ein weiterer Vorteil ist, dass sie auch unter unterschiedlichen Lichtverhältnissen funktionieren, im Gegensatz zu den optischen Systemen. Dennoch ist dieses sensorbasierte System nichts für den Alltag, da es zu teuer ist und immer noch einen speziellen Anzug benötigt. In den letzten Jahren scheint allerdings auch der Trend zuzunehmen, dass immer mehr Benutzer freiwillig Sensoren am Körper tragen. Dies zeigen die ganzen *Wearables*, die auf den Markt strömen und jeder für sich mehrere Sensoren besitzen um körperliche Aktivitäten zu messen. Im Jahre 2014 wurden so 3,4 Mio. *Wearables* allein auf dem deutschen Markt verkauft, die eine Umsatz von 465,7 Millionen Euro erzeugten (Ballhaus et al. 2015). Dabei liegt die Akzeptanz und die Käufe von *Wearables* eher bei Fitness Armbändern und

[1] www.bmw.com/com/de/newvehicles/7series/sedan/2015/showroom/innovative_functionality.html
[2] http://www.kymera-wand.com/kymera-wand-manual/
[3] http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/
[4] http://www.xbox.com/de-DE/xbox-one/accessories/kinect-for-xbox-one
[5] https://www.xsens.com/products/xsens-mvn/

Smartwatches (Ballhaus et al. 2015). Aus diesem Grund soll diese Arbeit zeigen, ob es möglich ist mit diesen Wearables eine Hand- und Armgestenerkennung umzusetzten und was es dazu braucht.

2 Aufbau der Arbeit

Das Projekt und diese Arbeit sind nach dem folgenden Softwareentwicklungs-Lifecycle Modell aufgebaut:

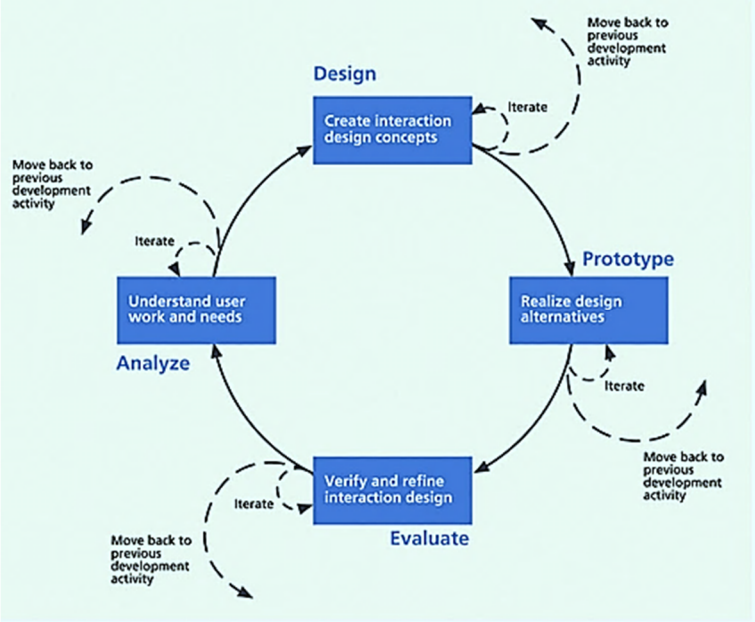


Abbildung 1: Softwareentwicklungs-Lifecycle (Hartson & Pyla 2012)

Dies hat den Vorteil, dass im *Lifecycle* die einzelnen Schritte genauestens beschrieben sind und auch dadurch die Entwicklung komplexer Software strukturiert wird. Außerdem erlaubt es dieses Modell, dass immer wieder ein Schritt zurückgegangen werden kann, wenn der Entwickler in der Phase die Notwendigkeit dazu sieht. Diese Arbeit startet also genauso wie das Projekt als Erstes mit einer Analysephase, in der die Anforderungen an das Produkt ermittelt werden.

3 Analysephase

Wie in der Einleitung bereits erwähnt, sollte ein System entwickelt werden, das in der Lage ist Hand- und Armgesten mit Hilfe von Wearables in Echtzeit zu erkennen. Da Hand- und Armgesten weit gefächerte Begriffe sind, wurde im Vorfeld die Erkennung auf eine Geste, die sich gut wiederholen und testen lässt, eingeschränkt. Diese Geste ist die *Touchgeste*, denn der Erfolg der Geste und die Durchführung kann klar mittels eines Touchscreens gemessen werden. Die Überlegung war, dass wenn die Erkennung dieser Geste erfolgreich durchgeführt werden kann, weitere Gesten folgen sollten. Damit verbunden kann ein bestehendes Problem gelöst werden, nämlich die Zuordnung der Benutzer zu der Touchinteraktion. Bevor jedoch mit der Implementierung des Projektes begonnen wurde, wurde eine Analyse der bisherigen Systeme und deren Vor- und Nachteile vorgenommen.

3.1 State of the Art

Für das Szenario der Benutzer-Touch-Zuordnung gibt es bereits mehrere Systeme, die versuchen dieses Problem zu lösen. Eines dieser Systeme ist *PhoneTouch* (Schmidt et al. 2010), dessen Aufbau in Abbildung 2 dargestellt ist. Bei diesem System toucht der Benutzer mit einer Ecke seines Smartphones auf ein spezielles Display. Dabei wird die Erschütterung des Handys, mit dem im Handy verbauten Accelerometer, gemessen. Denn auf Basis der Erschütterung wird geprüft ob eine Touch-Interaktion mit diesem Device durchgeführt wurde. Neben der Erschütterung wird auch das Audiosignal des Mikrophons des Smartphones aufgezeichnet. Diese Daten werden in Kombination mit dem Erschütterungssignal ebenso für die Erkennung verwendet. Für den Erkennungsalgorithmus kommt ein einfacher Schwellwertdetektor zum Einsatz, das heißt sobald das Audiosignal und das Accelerometersignal beide gleichzeitig einen bestimmten Wert überschritten haben, wird eine Touchinteraktion angenommen. Das System kann zusätzlich unterscheiden, ob der Benutzer mit einem Smartphone getoucht hat, oder aber mit seinen Fingern. Diese Unterscheidung geschieht über eine Kamera, die unter dem Display angebracht ist und über die Größe der Touchpunkte die Zuordnung durchführt. Dabei sei erwähnt das nur durch eine Smartphone-Interaktion der Benutzer zugeordnet werden kann. Der Ablauf dieses System ist wie folgt:

Die Infrarotkamera zeichnet das gesamte Display auf und über einem Bildalgorithmus werden die Touchpunkte extrahiert. Bei großen Touchpunkte wird *Handtouch* angenommen und es gibt keine Zuordnung. Bei kleinen Touchpunkten wird ein *Smartphone-Touch* angenommen. Jetzt werden alle Smartphones geprüft, ob deren Messungen über dem Schwellwert liegen. Ist dies bei einem Handy der Fall, kann eine Zuordnung durchgeführt werden.

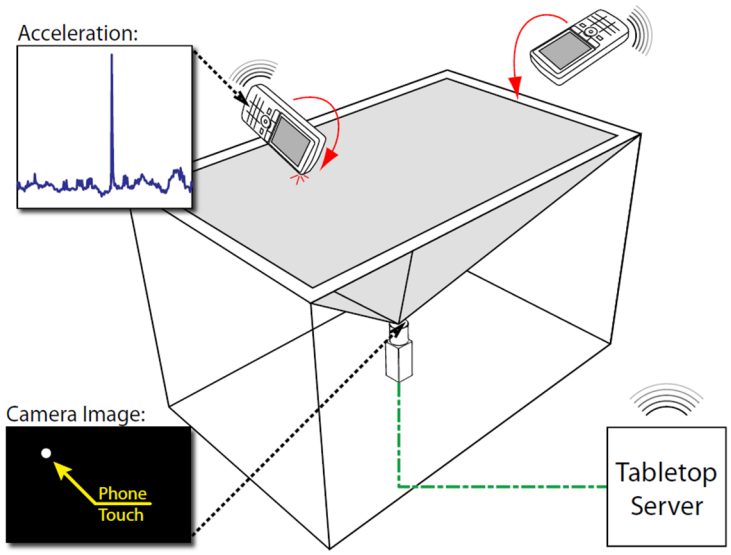


Abbildung 2: Aufbau der Technik von *PhoneTouch* (Schmidt et al. 2010)

Auf Basis dieser Technik entwickelte der Autor mehrere Einsatzszenarien, die neben Sicherheitsaspekten auch die Interaktion an Tabletops erweitern (Schmidt et al. 2012).

Eine der ersten Smartwatch Publikation nennt sich „Duet“ und wurde auf der CHI 2014 von Chen et al. veröffentlich. Die Autoren haben untersucht, inwieweit sich die Interaktionen von Smartphone und Smartwatch kombinieren lassen. Dafür wurden einfache Gesten wie Ausrichtung der touchenden Hand ermittelt. Durch diese Informationen lassen sich unterschiedliche Interaktionen triggern. So kann der Anwender mit der Seite seines Fingers den Inhalt auf dem Smartphone scrollen, während er mit den Fingerknöcheln die Inhalte markiert. Die Erkennung der einzelnen Gesten erfolgt mittels Entscheidungsbaum. Alle Gesten sind in dem folgenden Bild nochmals dargestellt.

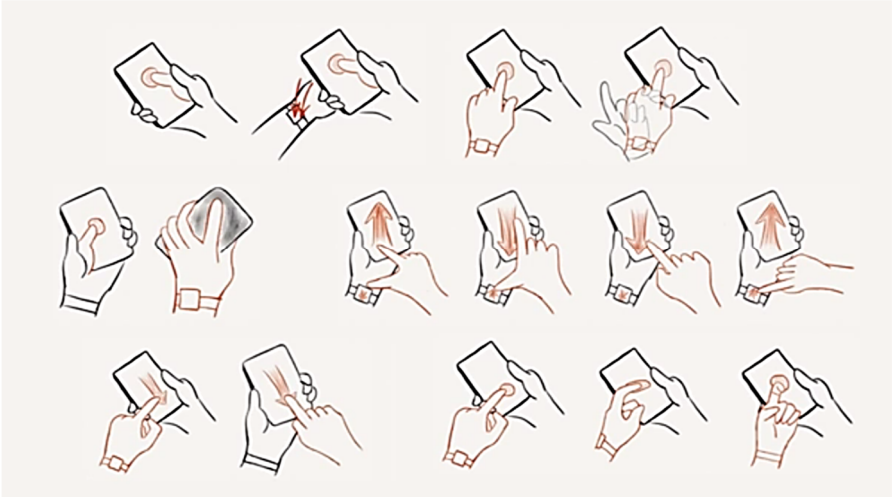


Abbildung 3: Mögliche Smartphone-Smartwatch-Interaktionen (Chen et al. 2014)

„Duet“ zeigt, dass eine Toucherkennung mittels Smartwatch durchaus im Rahmen des Machbaren liegt. Ein Schritt weiter gehen die Autoren des Papers „Finger-writing with Smartwatch: A Case

for Finger and Hand Gesture Recognition using Smartwatch“ (Xu et al. 2015). Sie versuchen Handschriftgesten zu erkennen, also von der Komplexität der Gesten im Verhältnis zu „Duet“ weitaus schwieriger. Sie verwenden allerdings keine normale Smartwatch, sondern ein medizinisches Gerät zum Tracken von Aktivitäten. Solche Geräte werden gerne von Entwickler eingesetzt, da sie gute und schnelle Schnittstellen bieten. Denn die Smartwatches stehen erst am Anfang der Entwicklung und um bestimmte Dinge auszuprobieren, ist es manchmal schneller und geschickter solche medizinischen Geräte zu verwenden, oder aber die Hardware selbst zu bauen. Dies war der Fall bei dem System „WatchConnect“ (Houben et al. 2015), bei dem die Autoren eine Smartwatch aus *Arduino* Bauteilen einfach selbst zusammengebaut haben, um so auf Sensoren einfacher zuzugreifen und auch Sensoren zu verwenden, die in keiner kommerziellen Smartwatch verbaut sind.

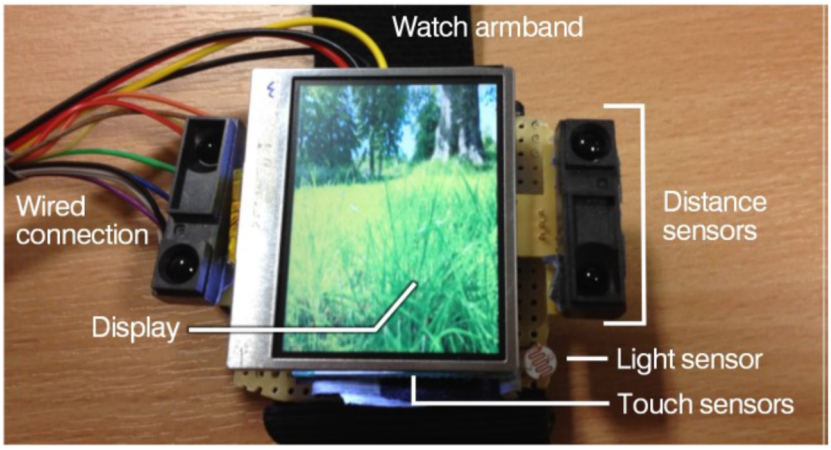


Abbildung 4: *WatchConnect Smartwatch* (Houben et al. 2015)

Auch das System *WatchConnect* besitzt eine Toucherkennung durch Sensoren und zeigt, dass die Sensoren am Arm ausreichen, um so eine Erkennung durchzuführen.

Es gibt auch auf optische Systeme, die die User Toucherkennung implementieren. Eines dieser Systeme ist *Carpus* (Ramakers et al. 2012). Das System erkennt den Handrücken der einzelnen User. Die Autoren sind der Meinung, dass diese Handrücken bei jeder Person unterschiedlich sind und daher eine Identifizierung möglich machen. Die Erkennung erfolgt über eine Kamera (siehe Abbildung 5), die über dem touchsensitiven Display angebracht ist. Der Vorteil dieser Erkennung ist im Gegensatz zu dem Sensor basierenden Systemen, dass der Benutzer nichts tragen muss. Laut den Autoren erreicht *Carpus* eine Erkennungsrate von respektablen 97,3%. Was die Autoren allerdings verschweigen ist, dass der Inhalt der Anwendung auf dem Display einen großen Kontrast zu der Farbe der Hände haben muss. Denn wenn das Display beispielsweise gelbe Kreise zeigt, dann funktioniert die Erkennung nicht mehr. Denn die Kreise werden dann als Handrücken erkannt und es kommt zu sehr vielen fehlerhaften Erkennungen. Außerdem müssen die Lichtverhältnisse immer gleich sein, da auch sonst keine Erkennung erfolgt. Ebenso die Größe des Displays, die eine bestimmte Größe nicht überschreiten darf, denn ansonsten reicht die Auflösung der Kamera nicht mehr aus, um die Hände zuverlässig zu erkennen. Dieses Beispiel zeigt nochmals, dass es durch aus große Vorteile hat ein System basierend auf *Wearables* zu verwenden. Dennoch ist es interessant

sich auch optische Systeme anzuschauen, denn die dort gezeigten Szenarios können dann vielleicht auch auf ein sensorbasiertes System angewandt werden. Die Anwendung hat nämlich ein sehr einfaches Trainingsverhalten, dass dem Anwender das Trainieren erleichtert.

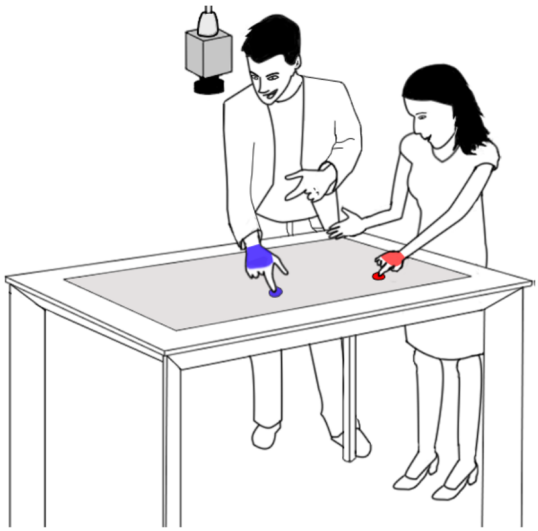


Abbildung 5: Aufbau von *Carpus* (Ramakers et al. 2012)

Dadurch, dass immer neue Wearables und Techniken auf dem Markt erscheinen, gibt es auch immer mehr wissenschaftliche Veröffentlichungen zu diesem Thema. So erscheinen zeitgleich zu diesem Projekt eine Vielzahl an weiteren Papern und Systemen. Eines dieser Systeme ist *Tomo* (Zhang & Harrison 2015). *Tomo* versucht mittels elektrischer Impedanz-Tomografie einen Blick in den Arm zu werfen und damit die Ausdehnung der Muskeln im Arm zu messen (siehe Abbildung 6). Durch unterschiedliche Handgesten werden die Muskeln im Arm unterschiedlich angespannt, was dann jeweils unterschiedlich Ausgaben erzeugt. Diese Ausgaben werden anschließend mit Hilfe von *Maschine Learning* einer Geste zugeordnet.

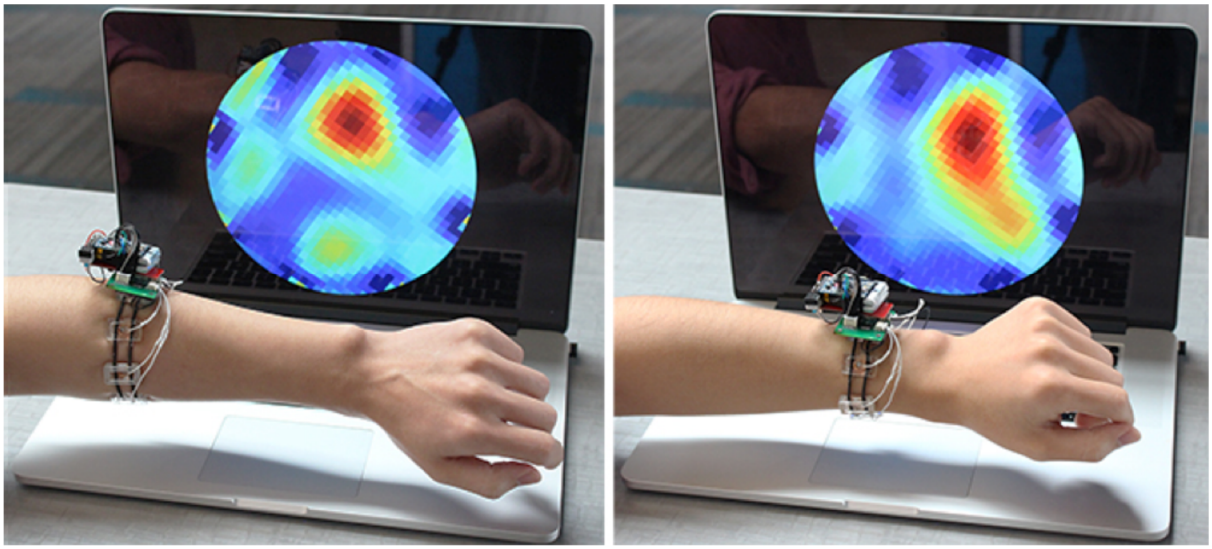


Abbildung 6: TOMO (Zhang & Harrison 2015)

Weitere Systeme für die User Touch Erkennung oder aber Gestenerkennung können in der Seminararbeit „Tabletop-Centric Multi Device Interaction“(Tennié 2015) nachgelesen werden.

3.2 Anforderungen

Die beschrieben verwandten Arbeiten zeigen alle, dass es möglich ist, mit Sensoren von Wearables Gesten zu erkennen. Doch was genau sind die Anforderungen, die ein Usertouch-Zuordnungssystem besitzt. Wie bei den meisten Echtzeitsystemen, möchte der Benutzer nicht lange auf das Ergebnis warten, sondern er will nach einer bestimmten Antwortzeit ein Resultat haben. Bei dieser Antwortzeit gibt es klare Richtlinien, denn Robert Miller untersuchte bereits 1968 wie lang solche Antwortzeiten seien dürften, bevor der Benutzer eine Reaktion erwartet (Miller 1968). Bei einer Verzögerung von 0.1s geht der Anwender von einer sofortigen Reaktion aus. Nach Miller werden Verzögerungen die im Rahmen von 0.1s -1s in den meisten Fällen akzeptiert. Auch wenn man eine App für den *Microsoft Store* entwickeln will, muss nach einer Sekunde lang dem Anwender klargemacht werden, dass das System noch Zeit braucht[6]. Ist dies nicht der Fall, wird sie nicht für den Store zugelassen. Dementsprechend nehme ich für dieses Projekt die eine Sekunde als Richtlinie. Gelichzeitig versuche ich eine Erkennungsrate von 80% zu erreichen. Ich gehe allerdings davon aus, dass diese Erkennungsrate nur erreicht werden kann, wenn der Anwender im Vorfeld die Touchgeste trainiert hat und das Device immer an der gleichen Stelle trägt. Dies ist auch in allen beschrieben verwandten Arbeiten der Fall, selbst bei *Carpus*. Aus diesem Grund soll das Trainieren von Gesten sehr einfach möglich sein. Was die Hardware anbelangt, soll nur kommerzielle Hardware zu Einsatz kommen, denn sie gewährleistet in der Regel besseren Tragekomfort als selbstgebaute Hardware. Auf der Softwareseite soll der Erkennungsalgorithmus nur die zwei folgenden States erkennen:

1. *Der Anwender führt keine Touchinteraktion mit der sensorbestückten Hand und dem trainierten Finger durch.*
2. *Der Anwender hat mit dem trainierten Finger eine Touchinteraktion durchgeführt.*

Wie bereits erwähnt, wurde auch für die Entwicklung des Projektes das Lifecycle-Modell verwendet. Insgesamt wurden 4 Prototypen entwickelt, bis eine zufriedenstellende Lösung vorhanden war (siehe Abbildung 7). Aus diesem Grund wurden auch mehrmals die Anforderungen geändert.

[6] https://msdn.microsoft.com/de-de/library/windows/apps/hh465338.aspx

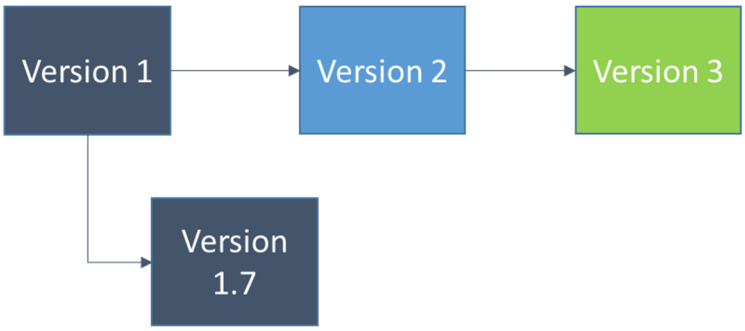


Abbildung 7: Überblick über die entwickelten Prototypen. Version 1.7 wurde komplett verworfen und bei den anderen Versionen wurden Teile mit in die neue Version übernommen. Version 3 stellt die finale Entwicklung dar.

Mit *Version 2* kamen die Änderungen der Anforderung im Bereich Datenübersicht hinzu. Denn bei der Gestenerkennung mit Sensoren gibt es jede Menge Rohdaten, die der Entwickler beobachten muss, um seine Algorithmen zu verstehen, zu testen und anzuwenden. Deswegen muss dem Entwickler die Möglichkeit gegeben werden, die Daten in einem geeigneten Format zu betrachten. Ab der Version 3 kamen noch mal neue Anforderungen hinzu. So wurde ein Konzept benötigt, mit dem der Verlauf der Erkennung und die Parameter des Erkennungsalgorithmus zur Laufzeit verändert werden konnten. Außerdem ging die Anforderung der Datenübersicht nicht weit genug, denn die Übersicht muss zu jedem Zeitpunkt und nach jeder Aktion ohne viel Aufwand möglich sein. Das heißt, wenn z.B. die Daten gefiltert wurden, braucht man eine Übersicht der Daten vor und nach der Filterung. Denn nur auf diese Weise können die erzeugten Machine Learning Pipelines richtig überprüft und angepasst werden. Vor der Entwicklung der *Version 3* war klar, dass diese Anforderungen nur ein *Node Designer* erfüllen kann. Also ein Tool, mit dem der Anwender in der Lage ist seinen Erkennungsalgorithmus und deren Parameter zur Laufzeit zu verändern. Dazu baut er seine Pipeline mit einzelnen *Nodes* auf. Jede dieser Node erfüllt dabei eine Aufgabe, wie z.B. ein Filter. Das Ergebnis solch einer Node wird dann an die nächste Node in der Pipeline weitergereicht und so weiter. Am Ende der Pipeline steht dann das Ergebnis. Was in diesem Fall einer der beiden oben genannten States ist. Auch an die Bausteine eines solchen Systems gibt es natürlich Anforderungen. So sollen diese Nodes möglichst flexibel verschiebbar sein und auch in der Größe skalierbar. Das Verbinden der einzelnen Nodes muss einfach möglich sein. Genauere Ausführungen zu dem Node Designer folgenden in den anschließenden Kapiteln.

Mit diesen Anforderungen (siehe Tabelle 1) kann das Projekt in den nächsten Schritt, dem Erstellen einer Vision, wechseln.

1. Ergebnis soll in maximal einer Sekunde vorliegen.
2. Erkennungsrate soll 80 % betragen.
3. Gesten müssen trainierbar sein.
4. Das Trainieren soll so einfach wie möglich sein.
5. Nach dem Training soll der Erkennungsalgorithmus die States *getoucht* und *nicht getoucht* erkennen.
6. Überblick über die Daten.
7. Schnelle Änderung der Pipeline.
8. Schnelles Debuggen von der Pipeline durch flexible Datenbeobachtung.
9. Nodes müssen flexibel verschiebbar sein.
10. Nodes müssen skalierbar sein.
11. Nodes müssen sich leicht verbinden lassen, um eine Pipeline aufbauen zu können.

Tabelle 1: Die weiß eingefärbten Anforderungen standen bereits am Anfang des Projekts fest. Die blauen Anforderungen sind ab Version 2 hinzugekommen und die grünen ab Version 3.

3.3 Vision

Als Nächstes ist es wichtig die Anforderungen in eine Vision umzuwandeln, um in ein greifbares Szenario zu erhalten. Diese Vision wurde bereits in der ersten Vorarbeit (Tennié 2015) zu diesem Projekt konstruiert und seitdem immer wieder verfeinert (siehe Abbildung 8). Die Vision sieht vor, dass der Anwender an der starken Hand also, wenn er Rechtshänder ist, die Smartwatch rechts trägt und wenn er Linkshänder ist, entsprechend die Smartwatch links trägt. Anschließend startet der Anwender eine App auf der Uhr, die dafür sorgt, dass alle Aktionen aufgezeichnet werden und an einen Erkennungsalgorithmus weitergereicht werden. Die zweite Komponente in der Vision bildet ein Tabletop System, auf dem ebenfalls eine App läuft, die alle Touchinteraktionen an den Erkennungsalgorithmus weitergibt. Der Algorithmus kombiniert alle Daten und gibt dann den Namen des Benutzers aus, der die Interaktion ausgeführt hat. Diese Information erhält dann die Tabletop App und kann entsprechend Aktionen ausführen. Diese Vision wurde im Laufe des Projektes ein wenig geändert. Anstatt der ursprünglich gedachten Smartwatch, kommt nun ein *Myo Armband*[7] zum Einsatz. Die Gründe hierfür können in der Projektarbeit „*Midas Touch*“ (Tennié 2016) nachgelesen werden.

Dieses Projekt erhielt den Namen Midas. „*Der Grund dafür ist, dass bei Touch Systemen oft das Problem besteht, dass alles, was berührt wird, eine Aktion ausführt, auch wenn der Anwender nur drauf zeigen wollte. Dieses Problem wird „Midas Touch Problem“ genannt. Da in der Legende von Midas nur von einer Person die Rede war, die die Gabe hatte alles zu Gold zu verwandeln und nicht jeder, wird diese Sage nur dann wahr, wenn man die Anwender unterscheiden kann. Mit so*

[7] https://www.myo.com/

einem Erkennungssystem kann nur der Anwender mit der Smartwatch alles berühren und es „verwandelt sich zu Gold““(Tennié 2016).

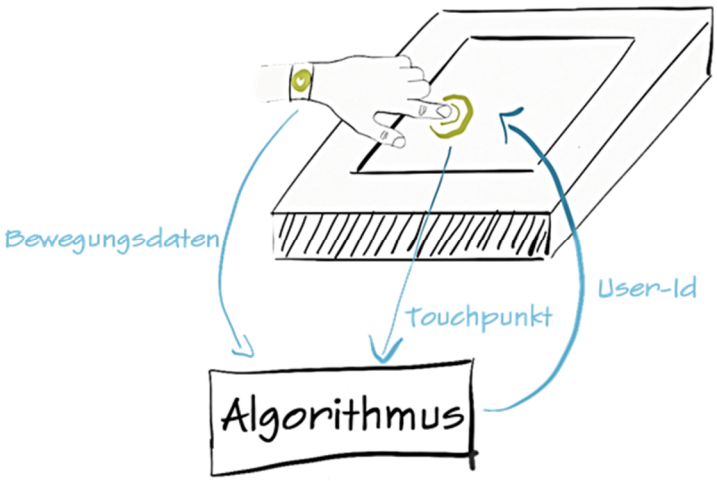


Abbildung 8: Benutzererkennung über eine Smartwatch (Tennié 2015)

Auf Basis dieser Vision wurde dann eine technische Vision (siehe Abbildung 9) entwickelt, die die Anwendungen und deren Funktionen nochmals genauer beschreibt. Nach dieser Vision sollen sämtliche Sensoren im *Myo Armband* dazu genutzt werden die Geste aufzuzeichnen. All diese Daten werden anschließend an einen Server geschickt, der diese nach Säuberung mit Hilfe eines Sliding-Window-Algorithmus analysiert. Bei einem Sliding-Window-Algorithmus betrachtet man immer nur einen Teil der Daten (siehe Abbildung 10). Die Größe dieses Teils wird durch die Länge des Fensters bestimmt. Ist dieser Teil ausgewertet, wird das Fenster verschoben und ein neuer Teil wird betrachtet. Die betrachteten Fenster können sich dabei auch überlappen. Auf diese Weise werden nach und nach alle Daten analysiert. Der Grund für das Einsetzten dieses Algorithmus ist, dass im Vorfeld nicht klar ist, wann eine Geste durchgeführt wird. Deswegen muss der Server in den Daten nach der Geste suchen. Jedes einzelne Fenster wird dann mittels Machine Learning ausgewertet und einer Geste zugeordnet. Dazu muss der Benutzer als Erstes die Touchgeste trainieren, denn nur so kann der Machine Learning Algorithmus wissen wonach er suchen soll. Wie in der ersten Vision sollen dann die Daten an einen Client weitergereicht werden, der diese dann in eine Aktion verwandelt.

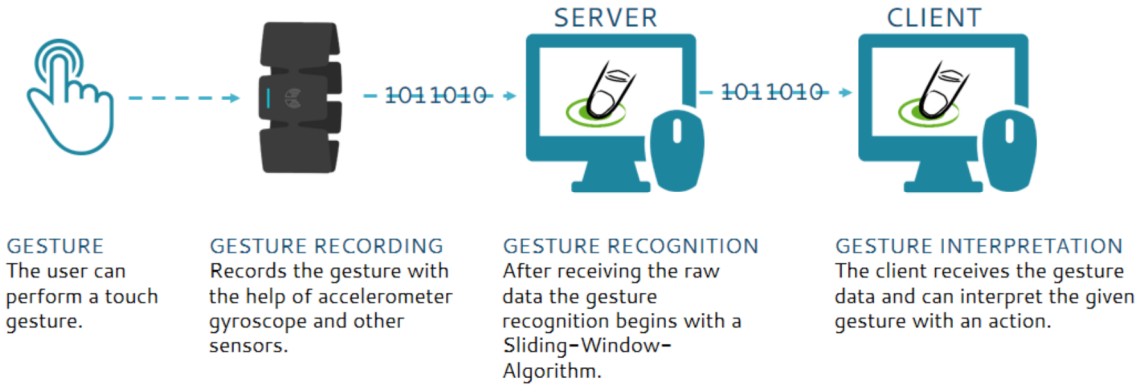


Abbildung 9: Technische Vision (Tennié 2016)

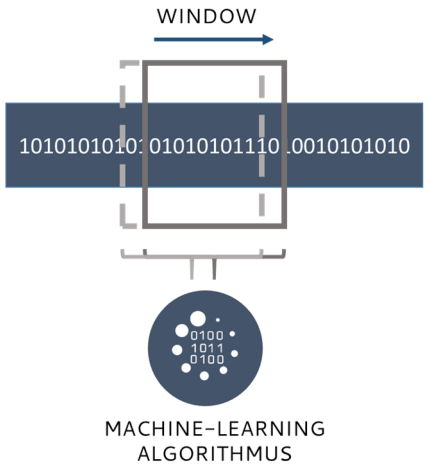


Abbildung 10: Sliding-Window-Algorithmus

4 Entwurfsphase

In Design oder Entwurfsphase (siehe Abbildung 11) werden die ersten Oberflächendesigns mittels Sketching Methoden produziert. Dazu werden die vorher aufgestellten Anforderungen als Basis für das Sketchen verwendet. Wie aus dem vorhergehenden Kapitel entnommen werden kann, werden Interfaces für zwei Anwendungen benötigt. Für dieses Projekt wurde der Lifecycle vier Mal durchlaufen, dabei wurden die ersten drei Varianten verworfen. Der Grund für das Verwerfen kann in der Projektarbeit „Midas Touch“ nachgelesen werden (Tennié 2016). Bei den ersten beiden Durchläufen wurde die Designphase übersprungen, da hier nicht der Anwender im Vordergrund stand, sondern die technische Machbarkeit und das Austesten von Algorithmen. Erst beim dritten Anlauf wurde dann der Anwender in den Mittelpunkt gestellt. Deshalb gibt es auch nur für die letzten beiden Versionen Sketches.

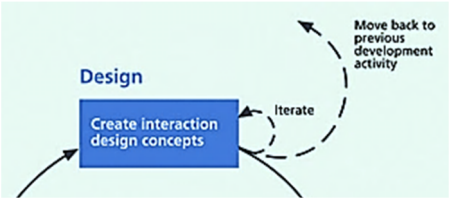


Abbildung 11: Entwurfsphase (Hartson & Pyla 2012)

4.1 Sketches des dritten Prototypen (Version 2)

Dadurch das es mehrere Prototypen gab, wurde auch unterschiedliche Sketches angefertigt. Die des Dritten werden im Folgenden dargestellt.

4.1.1 *Sketches für das Trainieren von Gesten*

Die Anforderung an das Trainieren von Gesten war, dass es möglichst einfach durchgeführt werden kann und der Anwender keine komplizierten Einstellungen vornehmen muss. Von daher war die Überlegung, dass der Anwender einfach zwischen Trainieren und Erkennen hin- und herschalten kann. Das Ergebnis ist hier zu sehen:

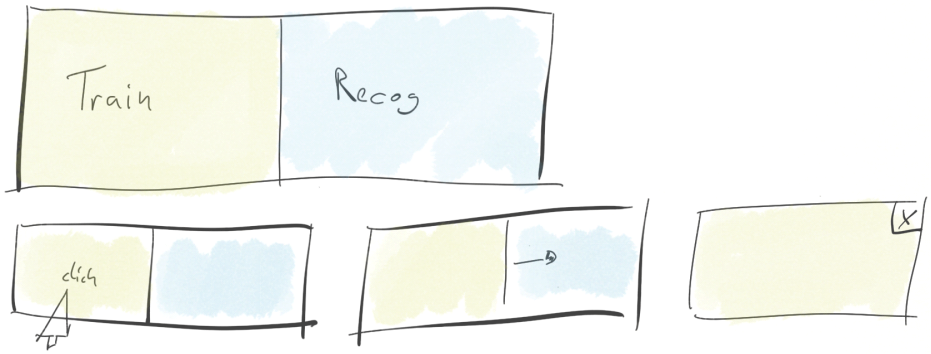


Abbildung 12: Switch Control.

Wenn der Anwender rechts geklickt hat, wurde die rechte Seite maximiert und wenn er die linke Seite ausgewählt hat, wurde entsprechend die linke Seite maximiert. Dies hatte den Vorteil, dass so erstens eindeutig sichtbar war, in welchem Modus sich die Anwendung befindet. Außerdem wurde der Platz für weitere Controls innerhalb der einzelnen Modi erweitert. Denn es baucht in jedem Schritt Einstellungen, die der Anwender wählen muss. Zum Beispiel beim Trainieren von Gesten, muss er den Zeitpunkt bestimmen ab wann aufgenommen werden soll. Anschließend muss noch der Name dieses Training oder Templates[8] festgelegt werden. Und bei der erkennenden Seite wurde Platz gebraucht, um dem Anwender zu zeigen, welches Template gerade erkannt wurde.

4.1.2 *Überblick über die Daten*

Es wurde lange überlegt, wie der Anwender einen Überblick über die Daten bekommen kann. Grundsätzlich gab es zwei Ansätze. Die Erstere war die Daten interpretiert darzustellen:

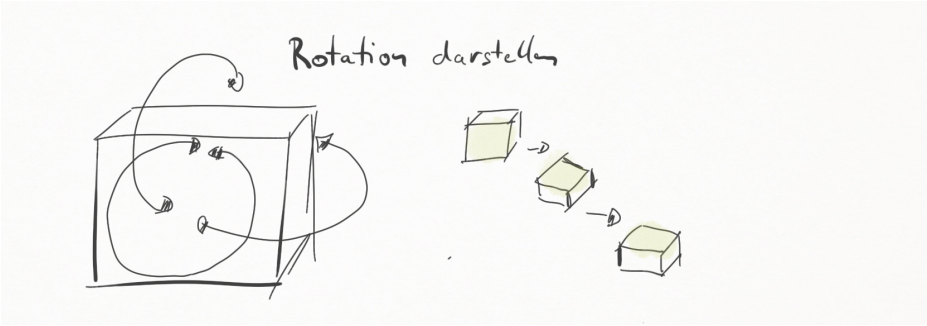


Abbildung 13: Rotation wird direkt abgebildet

Diese Darstellung hat den Vorteil, dass die Auswirkung der Daten gut zu erkennen ist, wenn das Gerät um 180° rotiert ist, wird auch die Repräsentation rotiert. Der Nachteil ist, dass schwer verglichen werden, kann ob das Gerät vorher genauso rotiert war und wie genau der Verlauf bis zu dieser Rotation war. Deswegen wurde sich schlussendlich für die im Folgenden beschriebene zweite Variante entschieden. Die zweite Variante ist ein einfaches Linien Diagramm (siehe Abbildung 14). Der entscheide Vorteil ist, dass alle Daten auf die gleiche Weise dargestellt werden können und dass auch der Verlauf sichtbar ist. Der Anwender kann außerdem auswählen, welche Daten er sehen möchte. Dazu selektiert er die Checkboxen auf der linken Seite. Jeder Graph hat auch eine unterschiedliche Farbe, diese ermöglicht eine bessere Unterscheidung bei vielen angezeigten Daten.

[8] Das Abbild einer Geste wird in dieser Arbeit als Template bezeichnet. Dabei können für das Erkennen einer Geste mehrere Templates derselben Geste verwendet werden. Ein Template sieht damit immer unterschiedlich aus, denn es ist recht unwahrscheinlich, dass ein Anwender eine Geste perfekt wiederholen kann.

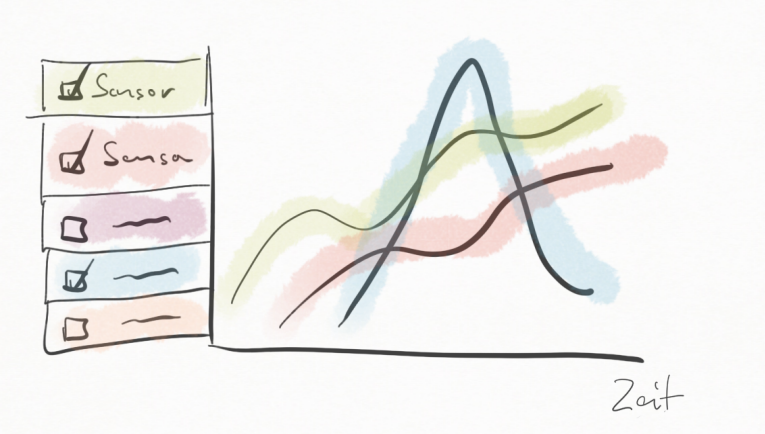


Abbildung 14: Line Graph

4.1.3 *Sketch des Servers*

Mit dem Switch Control und dem Line Graph sind alle Komponenten des Servers fertig. Die beiden Komponenten wurden dann in einem Interface vereint und das Resultat sieht dann wie folgt aus:

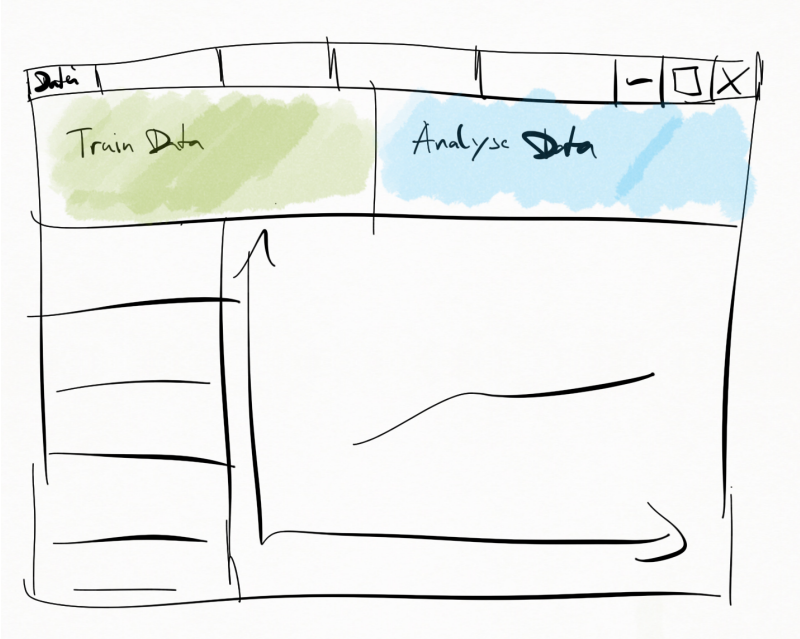


Abbildung 15: Sketch des dritten Prototypen

Dadurch, dass dieser Prototyp frühzeitig verworfen wurde, gab es auch nicht wie in der technischen Version (Abbildung 9) ein Interface Sketch für einen Client.

4.2 Sketche des finalen Prototypen

Der dritten Prototyp (Version 2) wurde verworfen, weil neue Anforderungen aufgetaucht sind, die eine Umgestaltung unmöglich machten. Eine der Anforderungen war ein bedeutender Faktor, der ein ganz neues Konzept benötigte - die Flexibilität der Pipeline. Nach langem Überlegen und Recherchieren, wurde klar, dass nur eine *Node Designer* diese Flexibilität besitz. Dieser wird auch häufig im Zusammenhang mit Gesten und Filter-Pipelines eingesetzt unter anderem in „*Squidy*“ (König et al. 2010), aber auch in „*HuddleLamp*“ (Rädle et al. 2014). Mit so einem Node Designer ist es möglich den Verlauf des Algorithmus jederzeit zu verändern, ohne dabei Code umzuschreiben. Der Grund dafür ist, dass eine Pipeline aus einzelnen Bausteinen besteht, den sogenannten Nodes. Jede dieser Node erfüllt einen eindeutigen Zweck und mit dem Zusammenschalten mehrerer Nodes kann eine Pipeline erzeugt werden, die die Daten verarbeitet. Aus diesem Grund bilden die Nodes das Herzstück des Node Designers und brauchen ein gutes Layout und Konzept.

4.2.1 *Design der Nodes*

Die Nodes sollten nicht nur genug Platz für Einstellungen bieten, sondern auch die typischen Verbindungspunkte haben. Auch das Vergrößern und Verschieben von Nodes sollte möglich sein. Das Resultat ist im olgenden zu sehen:

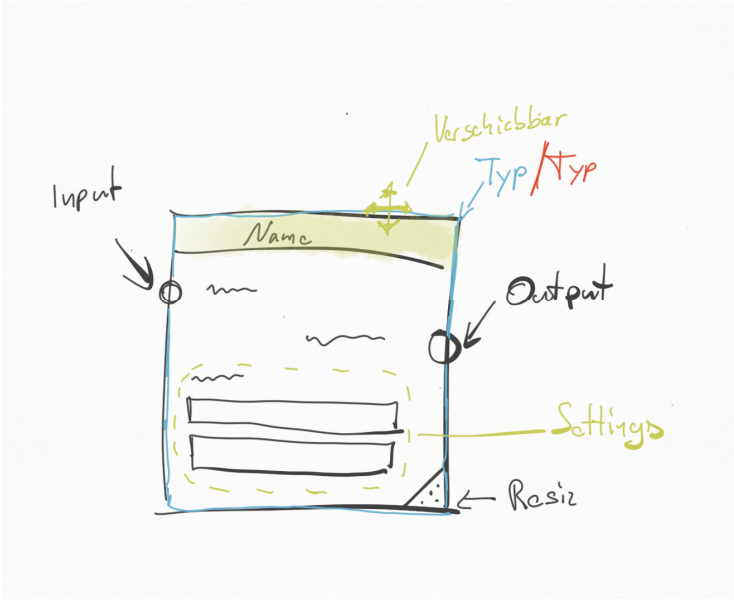


Abbildung 16: Node Sketch (Tennié 2016)

Die Node soll nur im oberen Teil des Fensters verschiebbar sein, damit ist der Bereich klar abgegrenzt. In der unteren rechten Ecke besteht dann die Möglichkeit die Größe der Node zu verändern. In beiden Fällen soll der Mauszeiger unterstützend über dessen Form anzeigen welche Funktion gerade aktiv ist. Am Rand gibt es die Runden Anknüpfungspunkte mit denen eine Verbindung zwischen den Nodes klar ersichtlich ist. Neben jedem Punkt steht eine Beschreibung, die erklärt was der Punkt zu bedeuten hat und welche Aus- bzw. Eingabe er liefert oder benötigt. Außerdem gibt die Farbe der Node an, um welchen Typ von Node es sich handelt. Also was genau dieser Node macht, ob sie filtert oder nur Eingaben liefert.

4.2.2 *Überblick über die Daten*

Die Visualisierung aus dem dritten Prototypen, der Line Graph, wurde einfach in einen Node integriert. Der Vorteil den es mit sich bringt ist, dass die Daten in jeder Stufe je nach Bedarf betrachtet werden können. Dies erhöht die Flexibilität ungemein, aber auch das Verständnis der Funktionen und deren Auswirkungen können direkt verglichen werden. Wie dieser Node aussieht kann in dem nachfolgenden Sketch betrachtet werden.

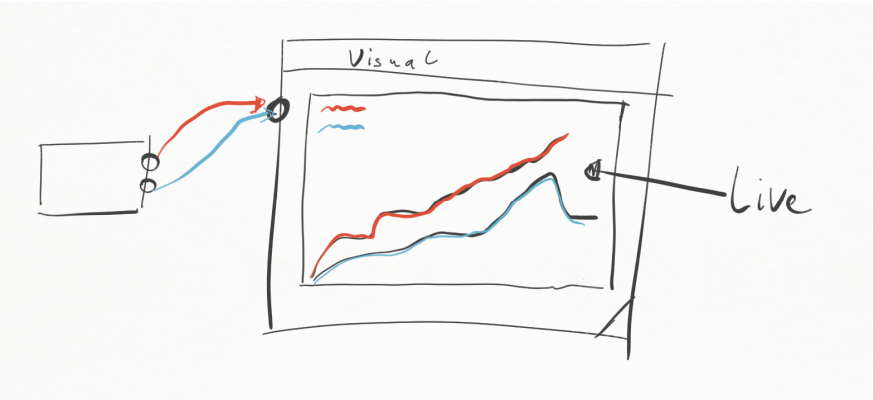


Abbildung 17: Line Graph in einer Node

Es gibt allerdings auch Änderungen was die Visualisierung anbelangt die Menüleiste an der linken Seite ist verschwunden, denn das kann mit den Verbindungen einfach gesteuert werden. Fügt der Anwender eine neue Verbindung zu dieser Node hinzu werden diese Daten angezeigt. Der umgekehrte Fall tritt ein, wenn er die Verbindung auflöst. Damit der Anwender dennoch nicht den Überblick verliert hat jede Line seine eigene Farbe und der Name der Line wird in einer Legende angezeigt.

4.2.3 *Sketch des Servers*

Das fertige Layout des Servers sieht dann wie folgt aus:

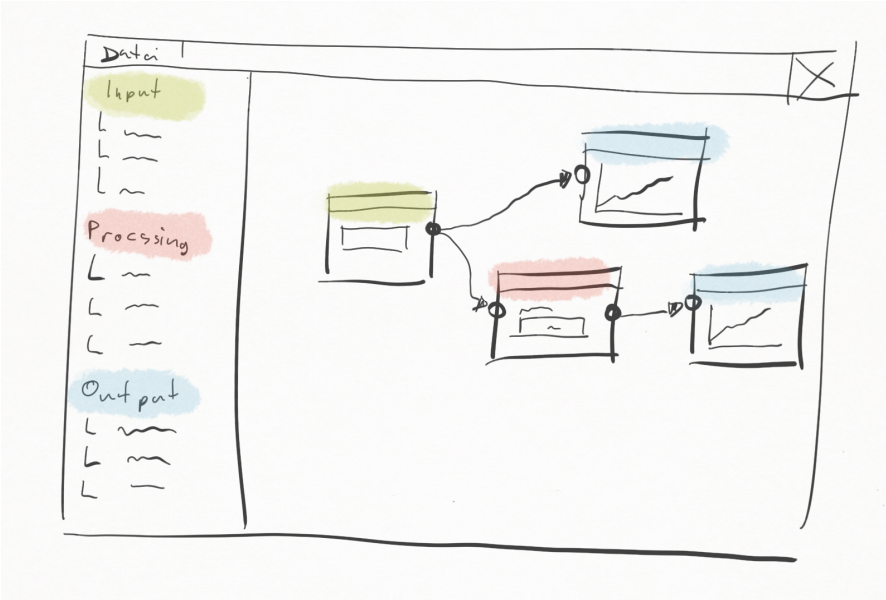


Abbildung 18: Midas Server Layout Sketch

Auf der linken Seite gibt es eine Liste mit den einzelnen Nodes die der Anwender verwenden kann. Diese ist in Input, Process und Output gegliedert. Da diese Liste eine Vielzahl von Einträgen besitzen kann, wäre es von Vorteil, wenn die einzelnen Oberkategorien aus- und einklappbar sind. Nodes des entsprechenden Typen erhalten die Farbe der Typen. Die Nodes werden dann auf der rechten Seite platziert und können dort nach Belieben verschoben und vergrößert werden. Da so eine Pipeline recht schnell groß werden kann, soll die Landschaft auf der die Nodes platziert werden zoombar und scrollbar sein.

4.2.4 *Sketch des Clients*

Da dieser Prototyp es endlich geschafft hat die Anforderungen zu erfüllen, wurde auch eine Oberfläche für den Client entwickelt, die wie folgt aussieht:

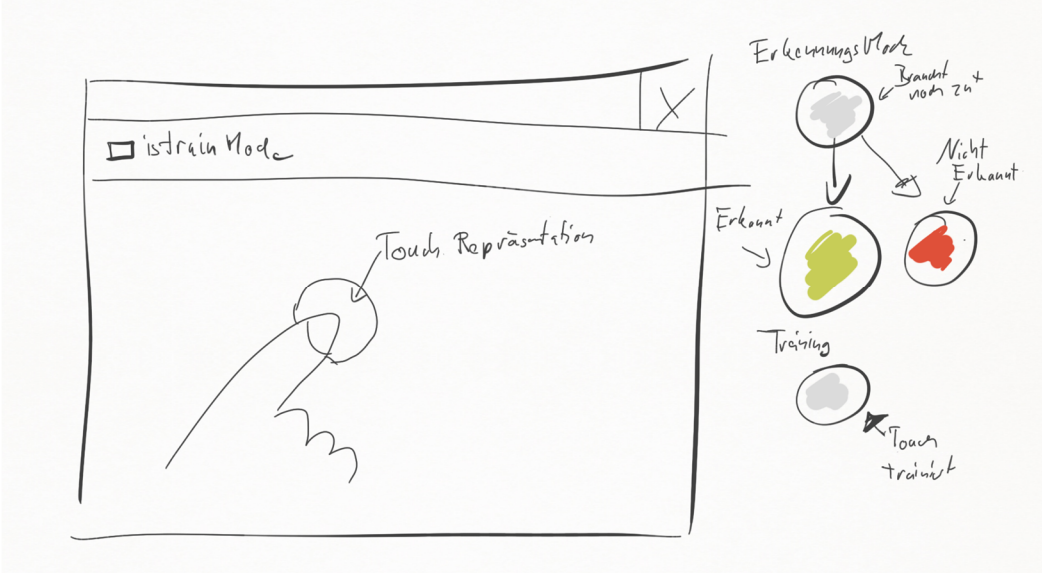


Abbildung 19: Sketch Oberfläche des Midas Client

Das Training der Gesten ist in dieser Anwendung integriert. Der Anwender ist dann sobald er die Anwendung startet im Trainingsmodus. In diesem Modus trainiert der Anwender die benutzerzugeordnete Erkennung, indem er das Display an verschiedenen Bereichen berührt. Durch große Kreise unter dem Finger, wird dem Anwender klar gemacht, dass die Geste erfolgreich trainiert wird. Im Hintergrund laufen dann alle Verknüpfungen mit dem Server, wovon der Anwender aber nichts mitbekommen soll. Über eine Checkbox kann der Anwender den Modus umschalten. Ab da an werden alle Toucheingaben analysiert. Der Kreis des Touchpunktes hat dabei drei Farben. Grau signalisiert, dass eine Anfrage gestellt wurde, grün, dass sie ihm zugeordnet werden konnte und rot dass sie ihm nicht zugeordnet werden konnte.

5 Prototyp Entwicklungsphase

Nach der Designphase konnte mit der Entwicklung der Prototypen begonnen werden. Die Prototyp Entwicklungsphase bildet damit die dritte Stufe des Entwicklungs Lifecycles. Bei der Entwicklung werden die vorhergehenden Resultate der Design- und der Anforderungsphase mitverwendet.

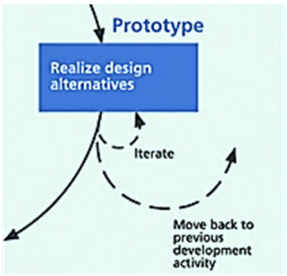


Abbildung 20: Prototyp Entwicklungsphase (Hartson & Pyla 2012)

In der technischen Vision wurden bereits einige Aspekte für den Aufbau der Prototypen vorgeschlagen. So wurde für den Server das Verwenden eines Sliding-Windows-Algorithmus genannt. In Zusammenhang mit diesem Algorithmus bestand schon die Überlegung Machine Learning für die Gestenerkennung einzusetzen. Diese Überlegung ergibt sich aus den vorgestellten Systemen in Abschnitt 3.1, denn von diesen Systemen setzten viele Machine Learning ein, um Gesten zu erkennen. Deshalb lohnt es sich diesen Prozess einmal näher zu betrachten, bevor die Entwicklung startet.

5.1 Maschinelles Lernen

Beim maschinellen Lernen, oder englisch Machine Learning, geht es darum einem Computer etwas beizubringen, so dass er später selbständig Antworten auf Fragen geben kann. Ein Beispiel für Machine Learning ist bei einer Postkarte die handschriftlichen Postleitzahlen automatisiert zu lesen (Bishop 2006). Dabei trainiert man den Algorithmus zuallererst mit Daten, bei denen man das Resultat bereits kennt. Der Algorithmus entscheidet dann auf Basis der gegebenen Trainingsdaten. In diesem Beispiel, wären die Trainingsdaten auf unterschiedliche Weise von Hand geschriebene Zahlen. Das beutet, wenn eine neue handgeschriebene Zahl eingegeben wird, wird sie mit den Trainingsdaten abgeglichen und wenn es zu einer Übereinstimmung kommt, wird das Ergebnis ausgegeben. Um die Übereinstimmungen zu finden braucht es jedoch genaue Methoden, denn ein Computer weiß in der Regel nicht was der Unterschied zwischen zwei Bildern mit Zahlen ist. Aus diesem Grund werden aus den Bildern sogenannte Feature Vektoren generiert, die numerische Werte beinhalten. So ein Feature Vektor kann zum Beispiel die Anzahl der farbigen Pixel abbilden. Mit diesen Features kann dann die Differenz zwischen zwei Feature Vektoren genau berechnet werden. Diese Funktion ist elementar für den eigentlichen Machine Learning Algorithmus. Dieser Algorithmus bestimmt nämlich aus Basis der einzelnen Differenzen der Feature Vektoren, welche der Trainingsdaten am ähnlichsten sind. Die aufgeführten Schritte zeigen, dass einiges unternommen werden muss, damit der Computer „selbst entscheiden“ kann. Aus diesem Grund gibt es auch ein Model, das den Ablauf von Machine Learning abbildet. Das folgende Modell ist eine Pipeline und stammt aus der Projektarbeit „Midas Touch“ (Tennié 2016) und aus „Machine

Learning: An Algorithmic Perspective, Second Edition“ (Marsland 2014). Diese Pipeline umfasst sechs Schritte, die im Folgenden näher erläutert werden:

1. *Daten sammeln:*
   Die Basis dieser Pipeline sind Daten, diese müssen natürlich erst gesammelt werden. Bei diesem Projekt sind das die Sensor Daten vom Myo Armband.

2. *Vorbereiten:*
   Es kann durchaus vorkommen, dass die Daten Fehler beinhalten. Diese müssen in dieser Phase bereinigt werden, denn sonst besteht die Gefahr der Fehlinterpretation. Daneben kann es sein, dass die Daten noch gefiltert werden müssen, weil sie z.B. in ihrer Rohform nicht so aussagekräftig sind.

3. *Feature Selektion:*
   In diesem Schritt werden die Features ausgewählt, aus denen der Vektor gebildet wird. Auf diesem Vektor wird dann der Vergleich durchgeführt. Im Falle dieses Projekts muss in dieser Phase überlegt werden welche Teile der Daten und in welcher Form sie sich am besten für einen Vergleich eignen.

4. *Algorithmus*:
   Für die Analyse der Feature Vektoren braucht es einen Algorithmus. Dieser Machine Learning Algorithmus bildet das Herzstück der Pipeline. Er entscheidet welche Resultate schlussendlich berechnet werden. Dabei sein erwähnt, dass die Wahl des Algorithmus stark mit der Aufgabenstellung zusammenhängt. Für dieses Projekt wurden zwei Algorithmen ausgewählt und getestet. Zum einen die Logistik Regression und zum anderen der Dynamik Time Warping. Beide Algorithmen werden im Verlauf dieser Arbeit nochmal genauer beschrieben.

5. *Training*:
   Damit der Algorithmus eine Antwort auf die Frage, die gestellt wurde, berechnen kann muss er zunächst alle Antwortmöglichkeiten und deren Eigenschaften wissen. Dazu gibt man dem Algorithmus Trainingsdaten, mit welchen die Parameter so eingestellt werden, damit das gewünschte Ergebnis erzielt wird. Im Falle von Midas, bestehen die Trainingsdaten aus Feature Vektoren der einzelnen Gesten.

6. *Evaluation*:
   Ist der Algorithmus trainiert, muss natürlich überprüft werden wie gut. Deshalb werden Daten an die Pipeline übergeben, von denen nur der User das Ergebnis kennt. Anschließend wird überprüft, ob der Algorithmus auch das erwartete Ergebnis liefert. Ist dies nicht der Fall, muss die Pipeline oder die Tranigsten überarbeitet werden. Im Falle von diesem Projekt wird die trainierte Pipeline mit neuen Aufnahmen von bereits trainierten Gesten gefüllt.

Die folgende Grafik fasst die wichtigsten Aspekte nochmals zusammen.

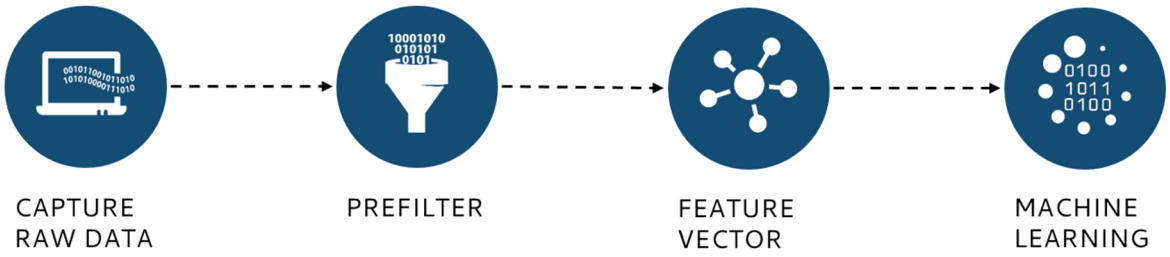


Abbildung 21: Machine Learning Pipeline (Tennié 2016)

Wie gezeigt wurde, bestehet eine Machine Learning Pipeline aus mehreren Grundbausteinen, die sich mehr oder minder von Szenario zu Szenario unterscheiden. Daher müssen diese Bausteine immer auf das jeweilige Einsatzgebiet angepasst werden. Für dieses Projekt wurden mehre Ansätze aus bereits veröffentlichten Papers betrachtet. Das Problem, das bei den meisten Veröffentlichungen besteht ist, das dort immer nur ein Teil der Pipeline beschrieben ist und meistens nicht so genau wie erhofft. Deshalb wurde zunächst das Paper „Gesture Recognition with a Wii Controller“ (Schlömer et al. 2008) als Grundlage für die Implementierung genommen. Der Vorteil bei diesem Paper war, dass die Autoren den Code veröffentlicht haben. Was die Autoren in diesem Paper untersucht haben ist, wie gut eine Gestenerkennung mit der Wii Remote[9] durchgeführt werden kann. In einer Wii Remote befindet sich ein Accelerometer und später dann auch ein Gyroskop. Es stellte sich jedoch heraus, dass die Gesten die mit diesem System erkannt werden, nur große Formgesten sind und nicht fein genug, um diese für das Szenario der Toucherkennung zu verwenden. Deshalb wurde die Pipeline aus dem Paper „Finger-writing with Smartwatch: A Case for Finger and Hand Gesture Recognition using Smartwatch“(Xu et al. 2015) verwendet. Die Autoren haben zwar nicht den Code zur Verfügung gestellt, referenzieren bei ihrem Prefiltering aber auf eine Arbeit („Using machine learning for real-time activity recognition and estimation of energy expenditure“ (Tapia 2008)), die die genutzten Formeln genauestens beschreibt und auch deren Wirkung. Deshalb diente dieses Paper als Grundlage für den dritten Prototypen.

5.2 Hardware

Wie bereits erwähnt, kommt bei diesem Projekt das *Myo Armband* von *Thalmic Labs Inc.* zum Einsatz. Dieses Armband besteht aus acht Sensorblöcken, die um den Arm des Trägers angebracht sind. Die Sensoren in den Blöcken, sind vor allem EMG Sensoren, die die Aktivität von Muskeln messen. In einem Block ist noch eine IMU-Einheit verbaut. Diese IMU-Einheit besteht aus einem Accelerometer und einem Gyroskop und erlaubt die Berechnung der Orientierung des Armbandes. Der Aufbau des Armbandes ist in dem folgenden Bild dargestellt:

[9] https://de.wikipedia.org/wiki/Wii-Fernbedienung

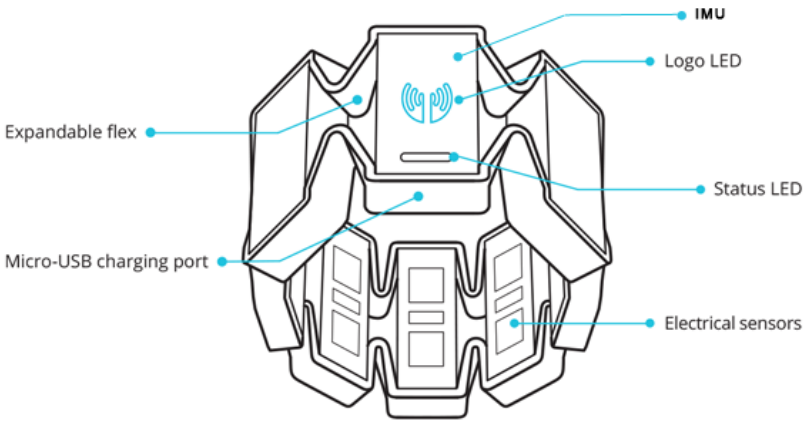


Abbildung 22: Aufbau des Myo Armbandes (Quelle: Thalmic)

Der eigentliche Verwendungszweck dieses Armbandes ist es, die folgenden rudimentären Gesten zu erkennen:

Abbildung 23: Gesten des Myo Armbandes, die über den Treiber erkannt werden (Quelle: Myo Keyboard Mapper)

Diese Gesten werden bereits von dem Treiber erkannt und können unter anderem zum Steuern von Präsentationen verwendet werden. Für die Erkennung der Gesten kommen nur die EMG Daten zum Einsatz. Ein EMG Sensor misst die Kontraktionsspannung des darunterliegenden Muskels. Das heißt, wenn der Muskel stark kontrahiert, dann sind die Spannungswerte höher, entsprechend sind die Spannungswerte bei kleineren Kontraktionen geringer.

5.3 Midas Prototyp drei (Version 2)

Die Entwicklung der vorhergehenden Versionen können im Projektbericht “Midas Touch“ (Tennié 2016) nachgelesen werden. Als Grundlage dienten die Anforderungen aus Abschnitt 3.2 und die Sketches aus Abschnitt 4.1. Das Resultat der Anwendung ist in den folgenden Bildern zu sehen:

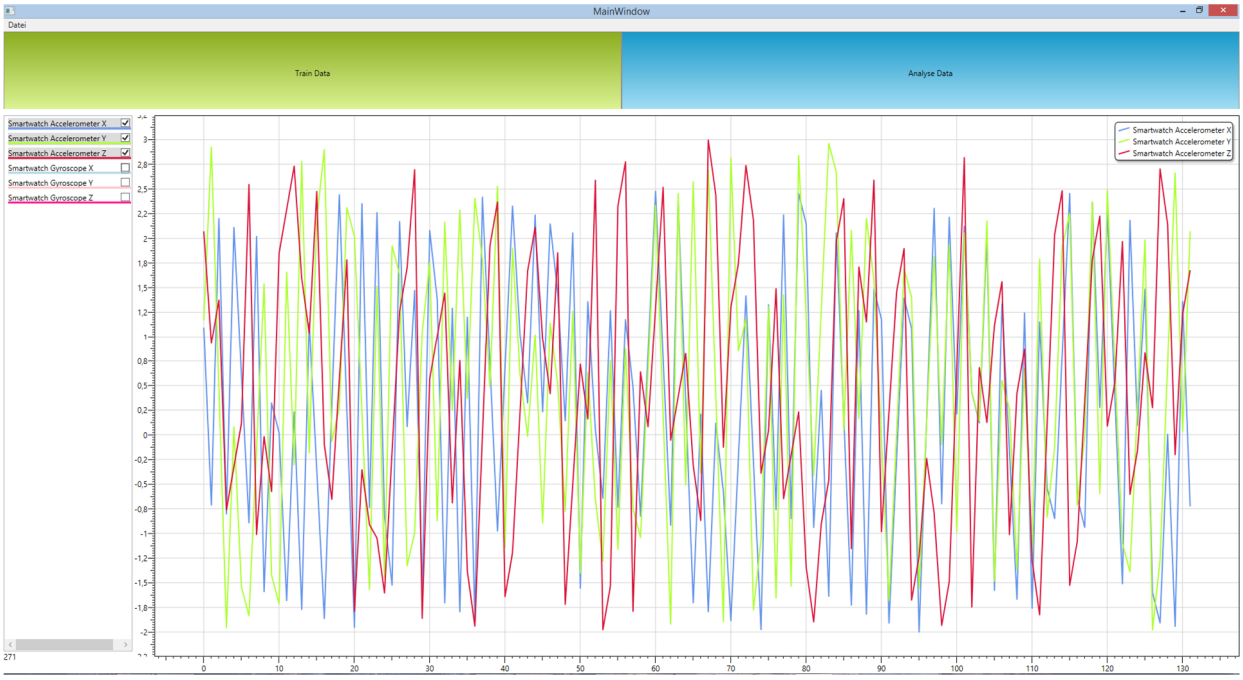


Abbildung 24: Screenshot des Midas Prototypen drei (Hauptmenü) (Tennié 2016)

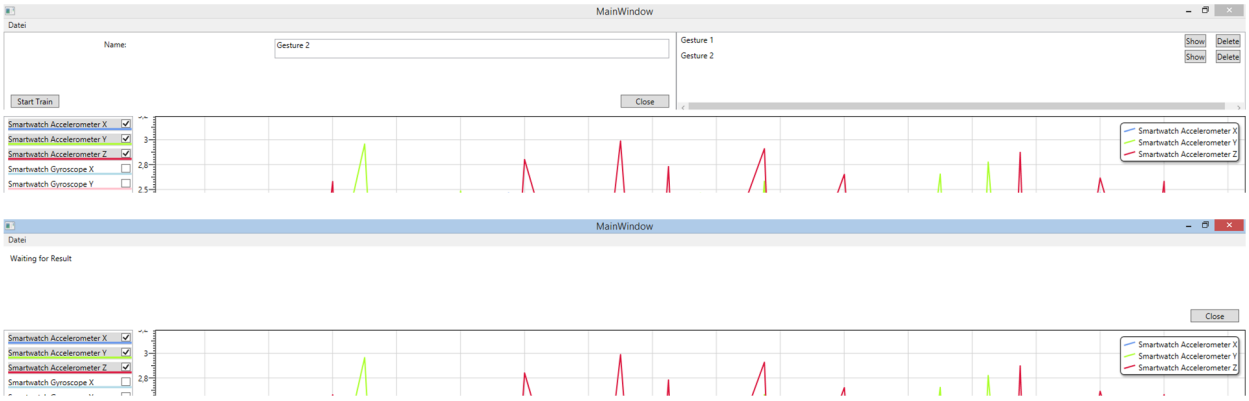


Abbildung 25: Screenshot des dritten Midas Prototypen. Oben das Trainieren, unten das Erkennen. (Tennié 2016)

Auf dem Hauptbildschirm, (siehe Abbildung 24) werden wie in den Sketches die Echtzeitdaten angezeigt. Dabei kann der Anwender über das Selektieren und Desektieren auswählen, welche Daten angezeigt werden. Neben der Visualisierung wurde auch das Menü aus den Sketches übernommen. Das Anlegen der Trainingdaten geschieht auf folgende Weise. Der Anwender trägt zunächst den Namen der Geste ein und startet das Trainieren durch das Drücken auf den „Start Train“ Button. Drückt der Anwender nochmal auf den Button, wird die Aufnahme gestoppt und ein Template der Geste wird angelegt. Ein Template beschreibt in dieser Arbeit ein Abbild einer

Geste, im speziellen Falle dieser Anwendung wäre das ein Feature Vektor mit dem dazugehörigen Resultat. Der Anwender kann nach dem Aufnehmen die aufgenommenen Daten bei Bedarf anzeigen lassen und auch wieder löschen. Wenn der Anwender anschließend seine Gesten testen möchte, kann dieser in den Test Modus wechseln. Dort werden dann automatisch die Gesten analysiert und das Ergebnis ausgegeben. Die gesamte Anwendung wurde für Windows entwickelt und ist mit C# und WPF programmiert worden. Weitere Informationen zu diesem Prototypen kann in der Projektarbeit „Midas Touch“ (Tennié 2016) nachgelesen werden.

Wie bereits erwähnt, wurde die Pipeline von Xu et al. verwendet. Was diese Pipeline auszeichnet ist, dass sie jede Menge Feature Vektoren aufweist, die dann als Grundlage für den Logistischen Regressionsklassifizierer dienen. Die Bestandteile der Pipeline sind in der folgenden Grafik dargestellt.

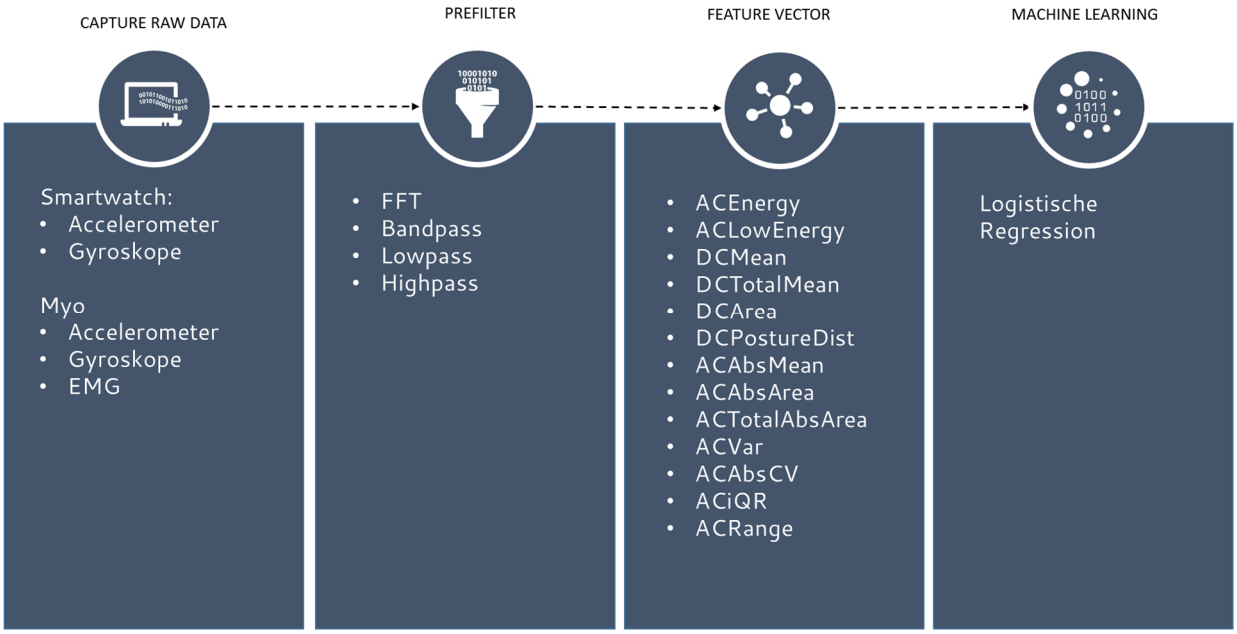


Abbildung 26: Pipeline des dritten Prototyps

5.3.1 *Multiclass Logistische Regression*

Um die Logistische Regression zu verstehen, braucht es zu nächst ein Verständnis für eine Lineare Regression. Bei der Linearen Regression geht es darum eine gerade Linie zwischen den Daten zu ziehen, sodass der Abstand der Punkte zu der Linie minimal ist. Nehmen wir ein konkretes Beispiel bei dem die Daten sich in zwei Klassen, so wie in dem folgenden Bild, separieren lassen.

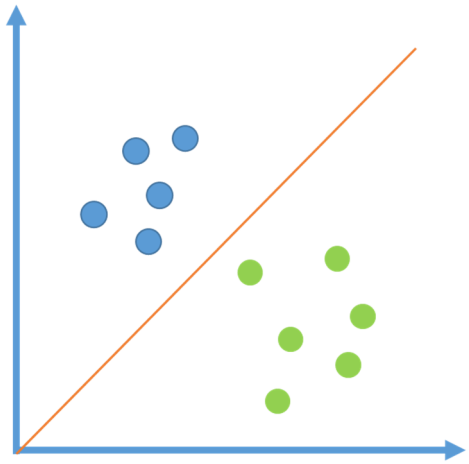


Abbildung 27: Lineare Regression

Wenn nun ein neuer Punkt hinzukommt, kann genau gesagt werden zu welcher Klasse dieser gehört. In dem Fall, dass der Punkt unterhalb der orangen Linie liegt, wird er der grünen Klasse zugeordnet. Wenn er oberhalb liegt der blauen. Man kann nun die Wahrscheinlichkeiten bestimmen, ob ein neuer Punkt zu einer bestimmten Klasse gehört. Bei dieser Methode kann es jedoch vorkommen, dass Wahrscheinlichkeiten über eins oder unter null berechnet werden. Dies ist meist der Fall, wenn die Punkte außerhalb der trainierten Werte liegen. Deshalb wird in der Logistischen Regression keine Grade verwendetet, sondern eine Kurve.

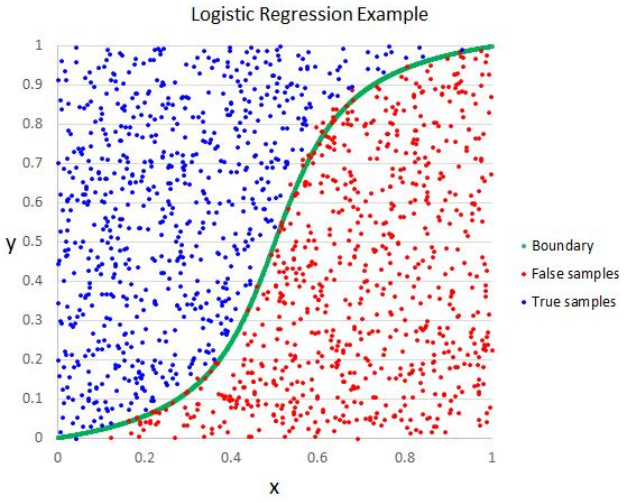


Abbildung 28: Beispiel für eine Logistische Regression (Quelle: https://www.mssqltips.com/sqlservertip/3471/introduction-to-the-sql-server-analysis-services-logistic-regression-data-mining-algorithm/)

Diese Kurve gewährleistet jetzt, dass die berechnete Wahrscheinlichkeit immer im Bereich von 0 bis 1 liegt. Diese Kurve ist natürlich nicht bei allen Daten gleich, sondern wird ebenfalls den Trainingsdaten angepasst. Wenn man mehrere Features hat, die eine Klasse beschreiben gibt es auch mehrere Kurven und dann werden die berechneten Wahrscheinlichkeiten der einzelnen Features kombiniert. Aus dieser Gesamtwahrscheinlichkeit wird dann die Klasse bestimmt.

Mit dieser Methode können Daten nur in zwei Klassen klassifiziert werden. In diesem Projekt hat man allerdings mehr als zwei Klassen, denn man möchte mitunter mehr als zwei Gesten erkennen. Um dieses Problem zu lösen, wird ein Trick verwendet, der einfach so tut als würde es nur zwei Klassen geben. Zum Beispiel haben wir drei Gesten: Klatschen, Winken und Werfen. Dann wir zunächst die Kurve für Klatschen und nicht Klatschen bestimmt, dann für Winken und nicht Winken und für Werfen und nicht Werfen. Auf diese Weise werden die Wahrscheinlichkeiten für die einzelnen Klassen bestimmt. Der Nachteil bei diesem Algorithmus ist, dass er immer auf Feature Vektoren angewiesen ist und daher auch die Performanz davon abhängt.

5.3.2 *Fazit des Prototypen*

Was dieser Prototyp gezeigt hat ist, dass ein funktionierendes Machine Learning, recht schwer zu entwickeln ist, auch wenn die Pipeline bereits gegeben ist. Der Grund dafür ist, dass maschinelles Lernen, durch die hohe Komplexität, nicht immer verständlich ist. Und es braucht viele Versuche und Anpassung, um das Szenario robust zu verwirklichen. Im Falle dieser Anwendung gab es bereits erste Gesten, die erkannt worden sind, allerdings beschränkte sich dies auf sehr eindeutige und nur wenige Gesten. Die Ursachenforschung stellte sich als nahezu unmöglich heraus, da die Ausgabe von Daten zu jedem Zeitpunkt ohne Visualisierung keinen Aufschluss über die Fehler ergab. Diese Visualisierung hätte nur durch immer wiederkehrendes Umschreiben des Codes realisiert werden können. Auch war unklar, ob die Filterung der Daten überhaupt erfolgreich durchgeführt wurde. Denn es gibt keine Referenzen, an die man sie halten kann und die voraussagen, welcher Output bei welchem Input und welchem Filter zu erwarten ist. Deshalb ist es nötig in diesem Fall Bibliotheken zu verwenden, die dies übernehmen, in der Hoffnung, dass die Berechnungen stimmen. Der Nachteil ist allerdings, dass der Entwickler dann auf die Performanz des Codes der Bibliothek angewiesen ist.

Die Performanz ist das nächste Problem, denn die Daten erreichen die Anwendung im Bereich von 50-60Hz. Das heißt, es müssen innerhalb von 1,7ms alle Ergebnisse berechnet sein, denn andernfalls stauen sich die Daten und die Software ist somit keine Echtzeitanwendung mehr. Diese 1.7ms konnte man mit dieser Anwendung jedoch bei mehreren Gesten oder Templates nicht einhalten. Der Algorithmus weiß nicht wann eine Geste auftritt und aus diesem Grund müssen alle Daten durchsucht werden. Außerdem wurden 13 Features, für jeweils 8 EMGs, 3 Gyroskope und 3 Accelerometer Signale, berechnet. Das ergibt insgesamt eine Berechnung von 182 Features, die alle 1.7ms neu errechnet werden müssen. Dies ist in Realtime nicht möglich, deswegen wurde versucht die Anzahl der Feature zu reduzieren, was allerdings mit Codeänderungen verbunden war. Auch war nicht klar welche der Features sich rausstreichen lassen, denn das hängt stark von den Daten ab.

All dies führte dazu, dass dieser Prototyp verworfen wurden und mit einem neuen Konzept weitergemacht wurde.

5.4 Midas Node Designer (Version 3)

Das neue Konzept sollte auf jeden Fall deutlich einfacher zu warten und gut erweiterbar sein. Wie bereits in den Anforderungen aus Abschnitt 3.2 und bei dem Sketch aus Abschnitt 4.2.1 beschrieben, sind diese Bedingungen nur mit Hilfe eines Node Designers machbar. Denn nur auf diese Weise können ohne große Änderungen im Code, neue Wege modelliert und getestet werden. Dazu baut der Anwender aus kleinen Bausteinen, aus sogenannten *Nodes*, eine Pipeline zusammen. Diese Pipeline sollte nicht nur die Flexibilität erhöhen, sondern auch dazu beitragen, dass die Berechnung der Zuordnung der Gesten effektiver und effizienter als im letzten Prototypen durchgeführt werden kann. Der neue Prototyp wurde ebenso für Windows konzipiert und ist in C# und WPF programmiert. Die finale Anwendung sieht wie das folgenden Bild aus:

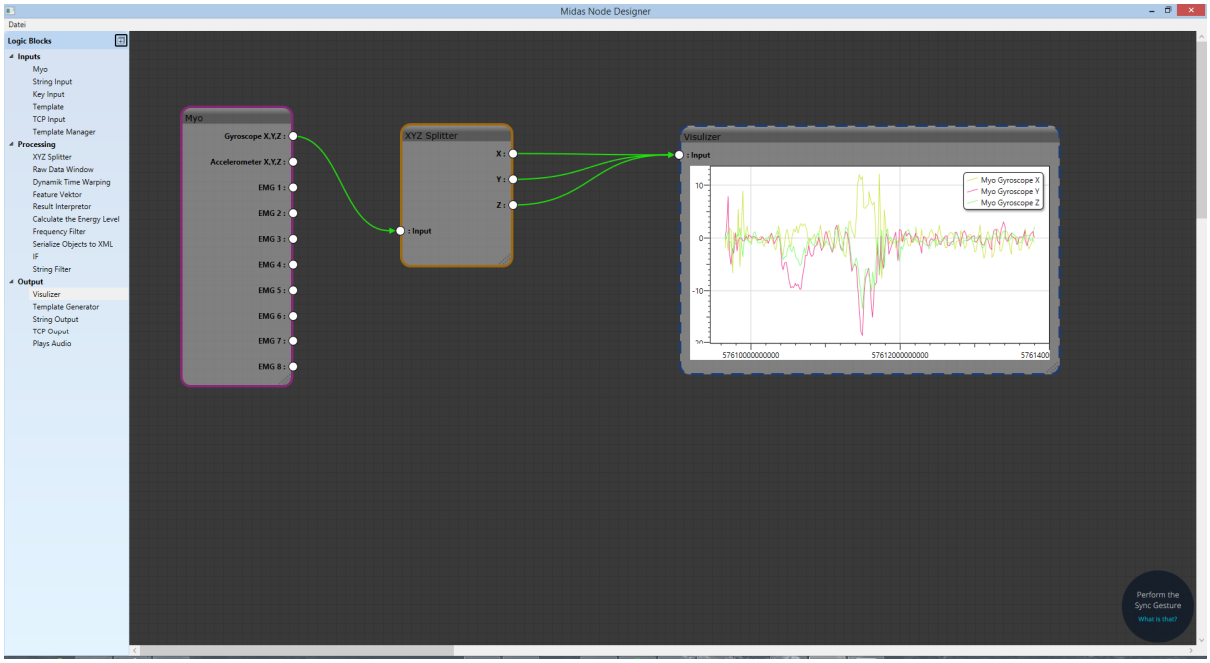


Abbildung 29: Midas Node Designer. Der Visualizer Node ist ausgewählt, dies sieht man an der gestrichelten Linie.

5.4.1 *Installation*

Damit die Anwendung funktioniert, braucht es ein Windows Rechner (Windows 7 und aufwärts) mit einer Installation vom *Myo Connect Treiber* (https://www.myo.com/start). Daneben sollte der Rechner eine möglichst hohe CPU Power besitzen und viele Kerne haben, denn die Berechnungen werden parallelisiert durchgeführt. Je mehr Kerne und Gigahertz zur Verfügung stehen, desto mehr Gesten können unterschieden werden.

5.4.2 *Bedienung*

Auf der linken Seite befinden sich, wie in den Sketchen bereits zu sehen war, das Menü mit allen einzelnen Bausteinen, die der Anwender verwenden kann. Diese zieht der User dann auf die Landschaft links. Dort kann er sie nach seinen Bedürfnissen verschieben und skalieren (siehe Abbildung 30). Außerdem kann er per Drag and Drop die Nodes miteinander verbinden und so

eine Pipeline erstellen. Der Anwender kann über das Drücken der Entf-Taste die Node bzw. die Verbindung wieder entfernen. Dazu sei erwähnt, dass die Komponente, die entfernt werden soll zunächst ausgewählt werden muss. Dazu klickt der User mit der linken Maustaste einfach auf die Komponente. Ob eine Komponente ausgewählt ist, wird dem Anwender über einen gestrichelte Line um die Komponente herum angezeigt (siehe Abbildung 29).

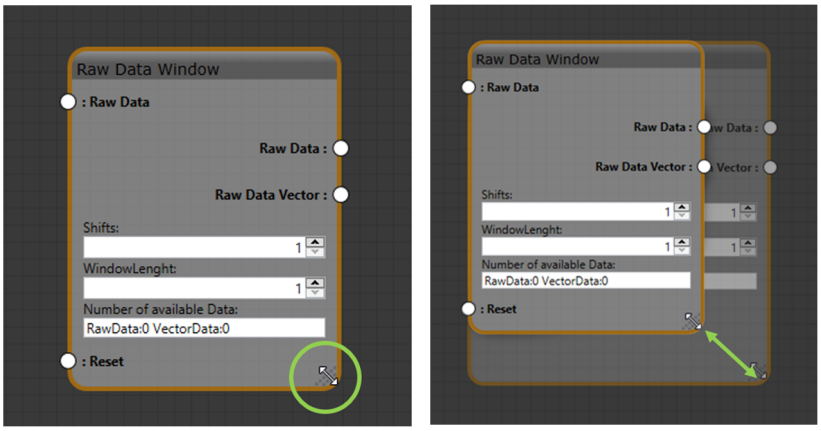


Abbildung 30: Skalieren eines Blockes (Tennié 2016)

Die Landschaft auf dem die Nodes platziert werden, ist mit gedrückter Alt-Taste und dem Scrollrad zoombar. Außerdem kann die Landschaft über die Scrollbalken verschoben werden. Ausführliche Bedienungsanleitungen dieses Prototypen können in der Projektarbeit Midas Touch (Tennié 2016) nachgelesen werden.

5.4.3 *Dynamik Time Warping*

Da aus dem vorhergehenden Prototypen klar war, dass die Verwendung von Logistischer Regression zum einem nicht zufriedenstellenden Ergebnis führt, wurde nach einer Alternative gesucht. Dabei sollte diese Alternative einfacher zu verstehen sein und auch mit weniger Filter auskommen. Aus diesem Grund wurde für diesen Prototyp eine Pipeline aus dem Paper „WATCHCONNECT: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications Steven" (Houben et al. 2015) ausprobiert. In dieser Arbeit verwenden die Autoren eine Pipeline, die interessanterweise nur mit den Rohdaten arbeitet. Denn sie verwenden einen Algorithmus mit dem Namen *Dynamic-Time-Warping*. Ursprünglich wurde dieser Algorithmus für die Spracherkennung verwendet (Vintsyuk 1972). Dabei war die Herausforderung, die Ähnlichkeit zwischen zwei Audiosignalen zu bestimmen. Der Algorithmus berechnet also die minimalen Kosten, um ein Signal in ein anderes zu überführen. Je geringer die Kosten sind, desto ähnlicher sind sich die beiden Signale. Diese Aufgabe lässt sich wunderbar auf eine Gestenerkennung erweitern, denn die Daten sind einem Audiosignal ähnlich. Damit ist das Grundverständnis dieses Algorithmus schon mal viel einfacher zu erlangen, denn er funktioniert auch, wenn die Daten unverändert bleiben. Gleichzeitig können aber immer noch Filter vorgeschaltet werden, was ein klarer Vorteil gegenüber den bisher eingesetzten Algorithmen ist. Ein weiterer Vorteil dieser Pipeline ist, dass der verwendete Klassifizierungsalgorithmus in der Länge skaliert. Das bedeutet, es können unterschiedlich lange Signale miteinander verglichen werden, denn durch den Algorithmus werden die Signale in der Länge gestaucht bzw. gedehnt. Dies spielt eine wichtige Rolle, wenn der Proband eine Geste mal

schneller durchführt, als er sie trainiert hat, denn der Algorithmus erkennt trotzdem die Ähnlichkeit und errechnet entsprechend eine geringen Kostenfaktor. Dieser Dynamic-Time-Warping Algorithmus funktioniert wie folgt (siehe auch Abbildung 31):

1. Man nimmt eine Tabelle die so viele Spalten hat, wie das erste Signal Werte und so viele Zeilen, wie das zweite Signal Werte hat.
2. Die einzelnen Werte des ersten Signals, werden über die erste Spalte geschrieben und die des zweiten Signals links neben die erste Zeile.
3. Jetzt wird langsam die Tabelle gefüllt. Dazu wird nach dem Muster vorgegangen, indem zunächst die absolute Differenz der zugeordneten Daten gebildet wird (Zelle 1;1: erster Wert des Signals eins – erster Wert des Signals zwei). Anschließend wird der errechnete Wert mit dem Minimum des vorangegangenen Wertes addiert. Vorangegangene Werte sind immer die Werte, die an die aktuellen Zelle grenzen. Dieser neu bestimmte Wert wird in die Zelle eingetragen. Zusätzlich zu dem Wert, wird auch die Zelle, aus der der minimale Wert bestimmt wird, gespeichert. Anschließend wird mit der nächsten Zelle weitergemacht.
4. Wenn alle Werte ausgefüllt wurden, wird ausgehend von der letzten Zelle der minimale Pfad bestimmt. Dazu wird immer wieder auf die gespeicherte minimale Zelle zurückgegriffen. Dieser Warping Pfad (siehe Abbildung 32) beschreibt dann die einzelnen Verschiebungen, die unternommen werden müssen, um die Signale zu transformieren. Wenn nur die Kosten berechnet werden sollen, reicht es den Wert der letzten Zelle auszugeben.

Dieser oben beschriebene Algorithmus kann durch unterschiedliche Distanz- und Kostenfunktionen entsprechend an die Signale angepasst werden. Leider ist die gesamte Berechnung der Tabelle nur in $O(n)$ möglich. Aus diesem Grund haben Salvador und Chan 2007 einen sogenannten Fast Dynamic-Time-Warping Algorithmus vorgestellt. Sie gehen davon aus, dass in dieser Tabelle viele Werte gar nicht berechnet werden müssen. Denn wenn zwei Signale ähnlich sind, dann wird meist ein Pfad gefunden, der in der Mitte der Tabelle verläuft (Abbildung 32). Denn Werte am Rand ergeben meist recht hohe Werte. Aus diesem Grund wird nur der mittlere Bereich der Tabelle berechnet und dadurch wird viel wertvolle Rechenkapazität eingespart (Salvador & Chan 2007).

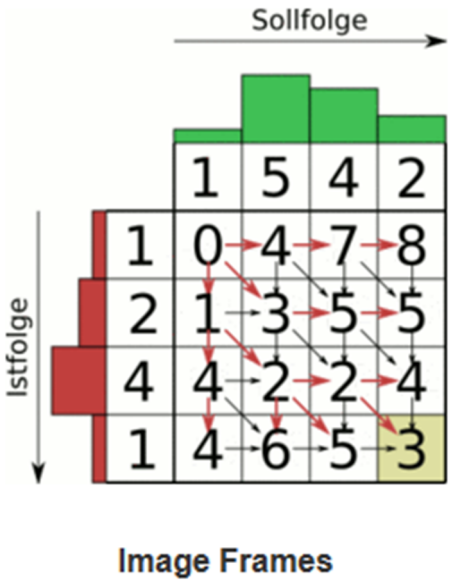


Abbildung 31: Berechnung von Dynamik Time Warping (Quelle: https://de.wikipedia.org/wiki/Dynamic-Time-Warping)

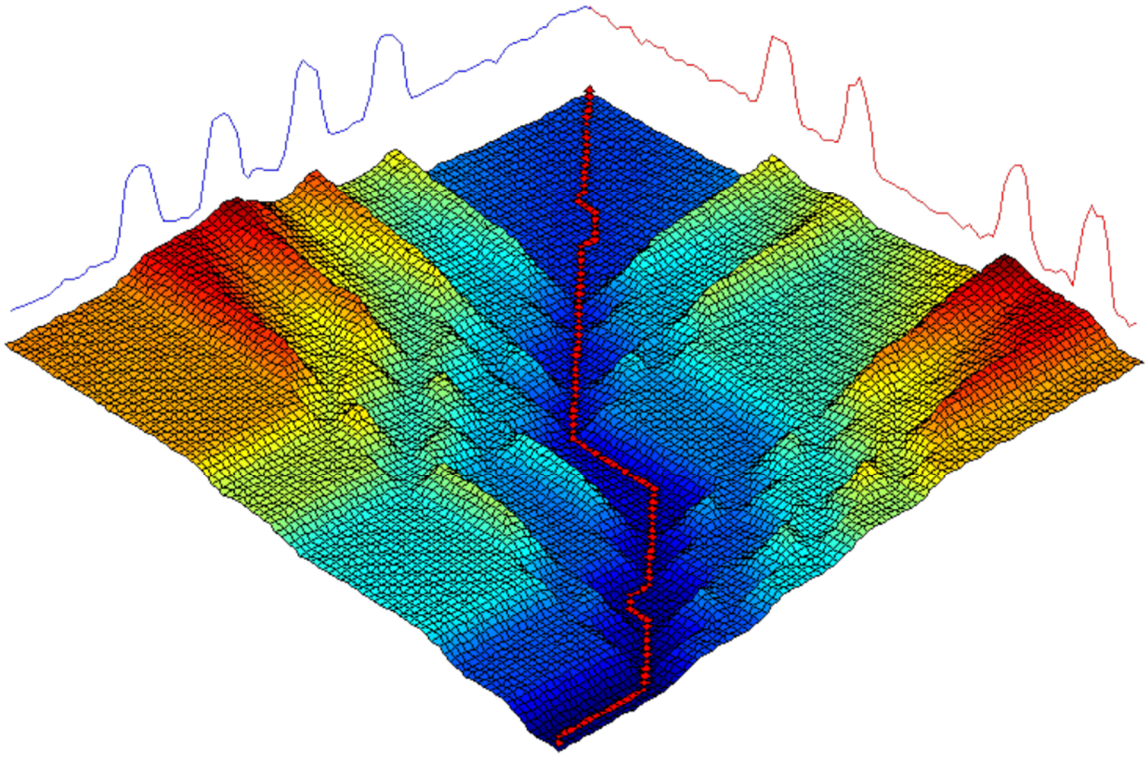


Abbildung 32: Dynamic Time Warping (Quelle: http://www.cs.ucr.edu/~eamonn/teaching/cs179materials/time_series.ppt von Prof. Eamonn Keogh)

5.4.4 *Fazit Midas Nodedesigner*

Das Umstellen auf einen Node Designer hat sich auf jeden Fall gelohnt, denn so hat der Entwickler endlich die Möglichkeit die Pipeline auf eine angemessene Art und Weise zu Debuggen. Außerdem kann die Pipeline immer wieder angepasst und verändert werden. Und wenn neuer Code für neue Funktionen geschrieben wird, dann ist dieser in der Regel wartbar und für sich auch testbar, denn der Code ist in einer Node gekapselt. Einen Nachteil hat jedoch auch diese Software, denn bis jetzt unterstützt dieser Node Designer keine Parallelisierung über mehrere Nodes hinweg. Deswegen ist dieser Designer manchmal nicht ganz so schnell, als wenn der Code direkt implementiert wäre. Dennoch bleibt dieser Nachteil gegenüber dem Vorteil nahezu unbedeutend. Außerdem ermöglicht dieses Tool nicht nur die Erkennung von Touchgesten, sondern auch andere Gesten, durch ein einfaches Ändern der Pipeline. Auch zeigt sich, dass sich die Pipeline von Houben et al. besser zur Echtzeit Gestenerkennung eignet und deutlich performanter ist als die Pipeline von Xu et al. Damit erfüllt der Node Designer die aufgestellten Anforderungen und es kann mit der Implementierung der Szenarien weitergehen.

5.5 Szenarien

Mit dem Midas Node Designer wurde endlich ein Tool entwickelt, das in der Lage ist mehrere Gesten zu erkennen. Deshalb wurde nicht nur ein Szenario wie ursprünglich versprochen umgesetzt, sondern gleich mehrere, um zu überprüfen in wie weit sich der Node Designer anpassen und ausreizen lässt.

5.5.1 *Toucherkennung*

Das Szenario der User Touch Erkennung war das ursprüngliche Szenario, das durch den Node Designer endlich umgesetzt werden konnte. Zu allererst wurde jedoch eine Anwendung aus den Sketchen aus Abschnitt 0 entwickelt. Das Resultat kann in dem nachfolgenden Bild betrachtet werden:

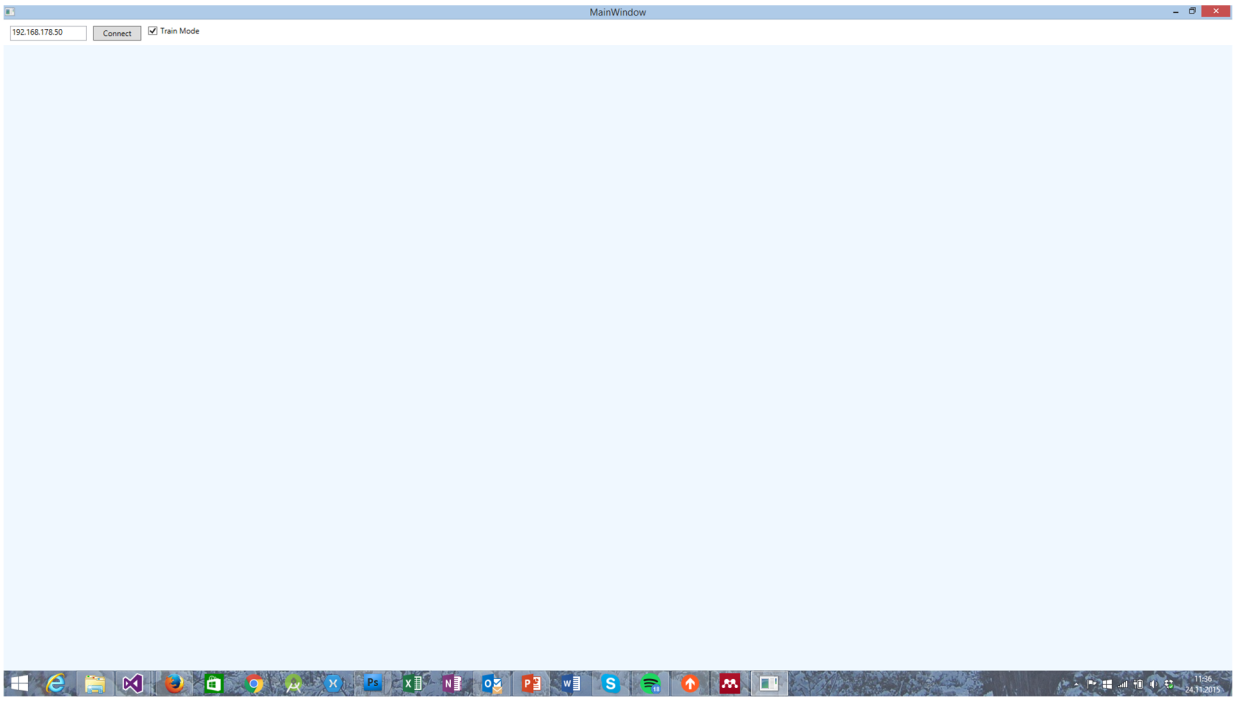


Abbildung 33: Midas Touch Client (Tennié 2016)

Auf der blaugrauen Fläche kann der Anwender touchen, wobei die Touches dann automatisch an den Server weitergereicht werden. Ähnlich wie in den Sketches, wird dem Anwender auch signalisiert, dass dieser einen Interaktion durchgeführt hat (siehe Abbildung 34). Das Wechseln zwischen den Modi wird über eine einfache Checkbox realisiert, die bei Start der Anwendung ausgewählt ist. Dies signalisiert dem Anwender, dass er sich im Trainingsmodus befindet. Neben der Checkbox muss der Anwender aber zu nächst die IP Adresse des Servers eintragen und dann bestätigen, denn nur so wird eine Verbdingung zu dem Server aufgebaut. Fortan werden die Interaktionen dann im Midas Node Designer entgegengenommen.

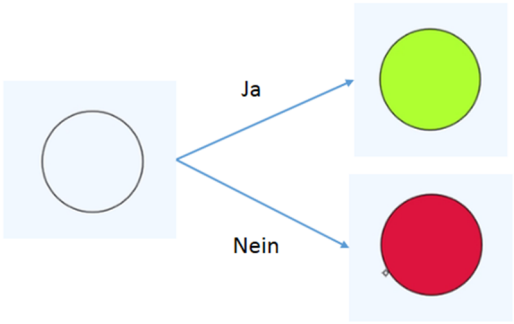


Abbildung 34: Touchrepräsentation bei dem Midas Client (Tennié 2016)

Im Midas Node Designer ist die Pipeline wie folgt aufgebaut:

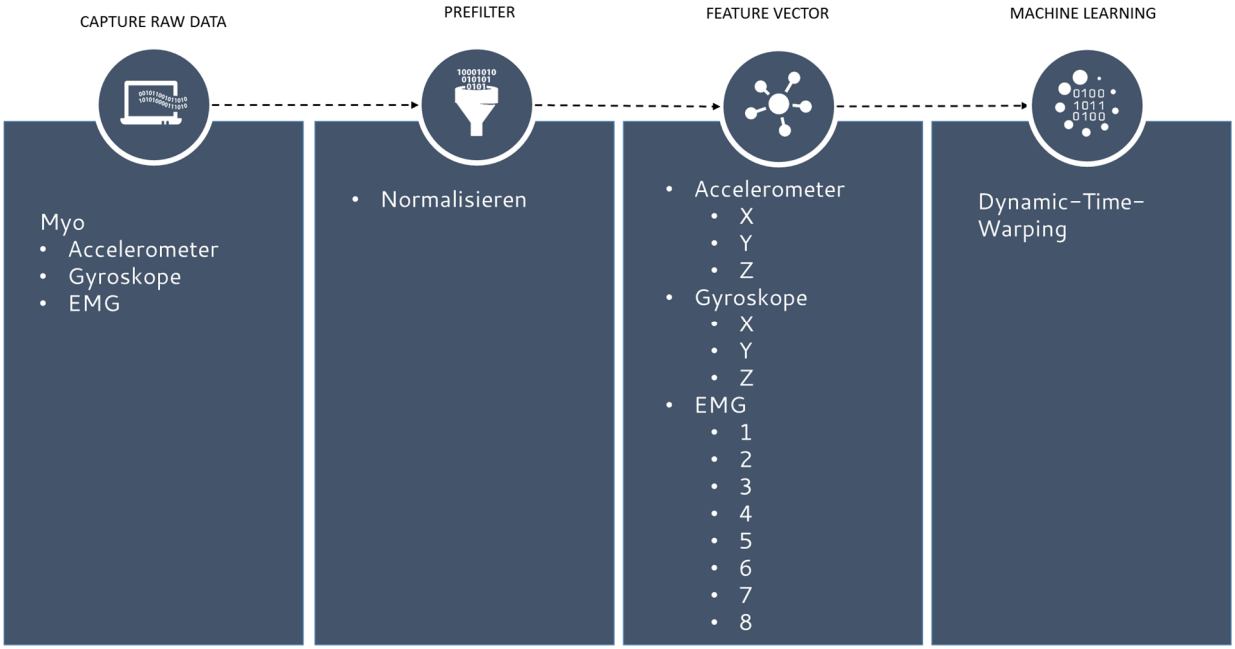


Abbildung 35: Pipeline zur Toucherkennung

Alle Daten des Myo Armband werden als Eingabe für die Pipeline verwendet. Diese Daten werden dann jeder für sich normalisiert, damit alle Kurven die gleiche Höhe haben. Dazu wurden die folgenden Werte für die Min-Max-Normalisierung verwendet:

	Min	Max
Gyroskop	-10	10
Accelerometer	-530	530
EMG	-130	130

Tabelle 2: Normalisierungswerte

Nach der Normalisierung werden alle Daten in einem Feature Vektor zusammengefasst und danach über einen Dynamic Time Warping Algorithmus mit den Trainingssets verglichen.

Der Ablauf für diese Anwendung ist Folgender:

1. Starten von Midas und Laden der vorher erzeugten Pipeline.
2. Nachdem der Anwender das Myo Armband angezogen hat und den Midas Touch Client auf einem multitouchfähigen System gestartet hat, wird die IP des Midas Server eingestellt und eine Verbindung hergestellt.
3. Danach kann der Anwender das System trainieren in dem er auf dem Display toucht. Wichtig zu erwähnen ist, dass er auf unterschiedliche Art und Weise toucht und auch an unterschiedlichen Positionen.
4. Anschließend wird in den Testmodus gewechselt. Der Anwender toucht nun erneut auf dem Display. Über die grafische Darstellung wird dem Anwender vermittelt, ob der Touchpunkt seiner Hand zugewiesen werden kann.
5. Gegebenenfalls müssen in Midas die Schwellwerte für die einzelnen Touchgesten noch eingestellt werden, wenn der Defaultschwellwert nicht ausreicht.

Die Schwellwerte sind die Kosten, die benötigt werden um das derzeitige aufgenommen Signal in einen der trainierten Signale zu verwandeln. Das bedeutet sie geben an, wie stark das derzeitige Signal dem trainierten ähneln muss, bevor es als eine Touchgeste gilt. Aus diesem Grund muss der Schwellwert korrekt eingestellt werden, denn wenn dieser Wert zu hoch ist kann es zu Fehlinterpretationen kommen. Wenn der Wert dagegen zu niedrig eingestellt wird, kann es vorkommen, dass sie gar nicht erkannt werden. Diese Werte sind von Person zu Person unterschiedlich, denn sie hängen davon ab wie gut die Gesten widerholt werden können und wie unterschiedlich die Gesten trainiert wurden. Tests zeigen, dass um die zwanzig aufgenommene Templates eine hohe Zuverlässigkeit bieten. Außerdem ist es wichtig, dass das Training immer mal wieder wiederholt wird, um möglichst viele unterschiedliche Arten zu erhalten. Denn wenn alles auf einmal trainiert wird, kann es vorkommen, dass die trainierten Templates sich sehr stark ähneln. Wird dagegen nach fünf Templates eine kleine Pause von einigen Minuten gemacht, in der der Anwender eine andere Geste ausführt und danach nochmals trainiert, unterscheiden sich die neu trainierten Templates signifikant von den vorher aufgenommenen. Dies erzeugt dann ein heterogeneres Abbild der Gesten und somit eine zuverlässigere Erkennung. Alternativ können aber auch die Schwellwerte angepasst werden.

5.5.1.1 Fazit der Toucherkennung

Mit dem Midas Touch Client können Touchgesten zuverlässig erkannt werden. Die verwendete Pipeline ist durch ihre Einfachheit besonders gut geeignet, denn an jedem Punkt kann genau nachvollzogen werden, warum das Ergebnis erfolgt ist. Allerdings hat auch diese Toucherkennung ihren Nachteil, denn sie ist in dieser Form nicht Multiuser fähig. Denn es kann bei gleicher Touchinteraktion nicht entschieden werden, welcher Anwender wo getoucht hat. Dazu würden Positionssensoren gebraucht werden und diese sind in der benötigten Genauigkeit in keinem Wearable verbaut. Dieses Problem haben alle sensorbasierten Touchgestenerkennungs Systeme, denn für eine zuverlässige Positionserkennung, wird in den meisten Fällen ein optisches System benötigt. Dennoch lassen sich auch mit diesem System einige Szenarios verwirklichen.

5.5.2 *Midas Gvoice (Gesture Voice)*

Nachdem die Toucherkennung abgeschlossen war, wurde der nächste Schritt durchgeführt und geschaut in wie weit sich die Pipeline auch für andere Gesten erweitern lässt. Zunächst wurde jedoch überlegt, welche Gesten man als Grundlage für Test verwenden könnte. Aus diesem Grund wurde wieder eine neuer Entwicklungs-Lifecycle (Hartson & Pyla 2012) gestartet und mit dem Sammeln von Anforderungen begonnen.

5.5.2.1 Analysephase

Um herauszufinden was genau die Anforderungen an ein solches System sind, wurde ein *Inspiration Card Workshop* abgehalten (Feyer 2015). Die zentrale Frage, die die Probanden beantworten sollten war „Welche Art von Gesten würdest du mit Midas verwenden?“. Dazu wurde den Probanden zunächst das System beschrieben und sie damit vertraut gemacht. In dem anschließenden Workshop, sollten die Probanden dann sogenannte *Inspiration Cards* erstellen. Das sind Karten, die etwas zeigen, das dazu dient eine Idee oder ein Konzept zu visualisieren. Diese *Inspiration Cards* wurden dann an einem Whiteboard zu einem Schaubild zusammengefügt. Das Ergebnis zeigt das nachfolgende Bild:

Abbildung 36: Ergebnis des Inspiration Card Workshops

Es gab viele Überlegungen was für Gesten die Anwender mit Midas erkennen lassen könnten. In diesem Zug wurden unterschiedliche Arten von Bewegungen vorgestellt.

1. Gesten, wie typische Handbewegungen zum Beispiel die „Ok“ Geste, oder mit den Fingern zählen.
2. Daneben gab es aber auch Gesten, die eine realweltliche Interaktion nachahmten wie z.B. das Drücken eines Buttons oder das Öffnen einer Tür.
3. Gesten die durch Sportübungen entstehen, wie z.B. Liegestütz oder Strecken.
4. Im Laufe des Workshops offenbarte sich dann allerdings auch, dass es bereits eine Datenbank mit etlichen Gesten gibt, die gut voneinander unterscheiden werden können. Die Rede war von der Gebärdensprache.

Aus dieser Idee der Gebärden wurde dann ein ganzes System, dass Gesten in gesproche Sprache überführt. Es war jedoch auch klar, dass Midas zu komplizieret für normale User ist, deswegen sollte eine neue Anwendung entwickelt werden, die die Basisfunktionen von Midas besitzt. Das Trainieren und Testen von Gesten, sowie eine Möglichkeit für das einfache Einstellen von Schwellwerten. Mit dieser Funktion sinkt die Einstiegshürde. Die Anforderungen sind nochmals in der folgenden Tabelle zusammen gefast.

1.	Erkennung von Teilen der Gebärdensprache oder anderen Gesten
2.	Ausgabe von Sound nach der Erkennung der Geste
3.	Gesten müssen einfach trainierbar sein
4.	Gesten müssen einfach testbar sein
5.	Automatische Schwellwerteinstellung
6.	Gesten müssen löschbar sein
7.	Qualität der Gesten im Vergleich zu den bereits trainierten muss erkennbar sein

Abbildung 37: Anforderungen an Gvoice

Aus diesen Anforderungen wurde dann eine grafische Vision erstellt, die einen Menschen zeigt, der an beiden Armen ein Myo Armband trägt und diese Aktionen dann zu Midas gesendet werden. Neben den Myos, trägt der User noch einen Lautsprecher für die Ausgabe.

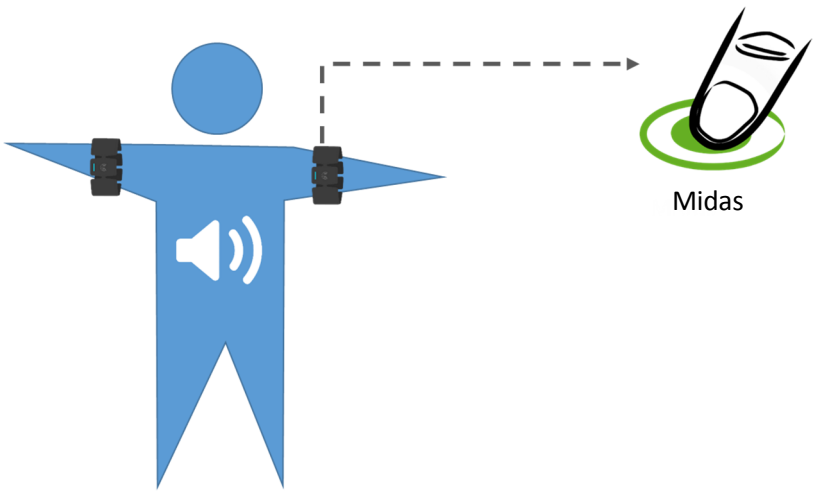


Abbildung 38: Gvoice Vision

Neben der einfachen Vision braucht es auch eine technische Vision. Der Anwender trägt dabei einen Lautsprecher, der mit Bluetooth mit einem Rechner verbunden ist, der sich im Rucksack des Anwenders befindet. Auf dem Rechner läuft zum einen Midas, welcher die Gestenerkennung durchführt, zum anderen auch die neue Software, die bei erkannter Geste eine Audiodatei abspielt. Gleichzeitig soll die neue Software das Trainieren und Testen von Gesten übernehmen, sodass der Anwender nichts im Midas Node Designer einstellen muss. Der Anwender trägt neben einem Rucksack natürlich auch das Myo Armband und zwar an dem Arm mit der starken Hand. Dies ist grafisch nochmal in der folgenden Bild zusammengefasst:

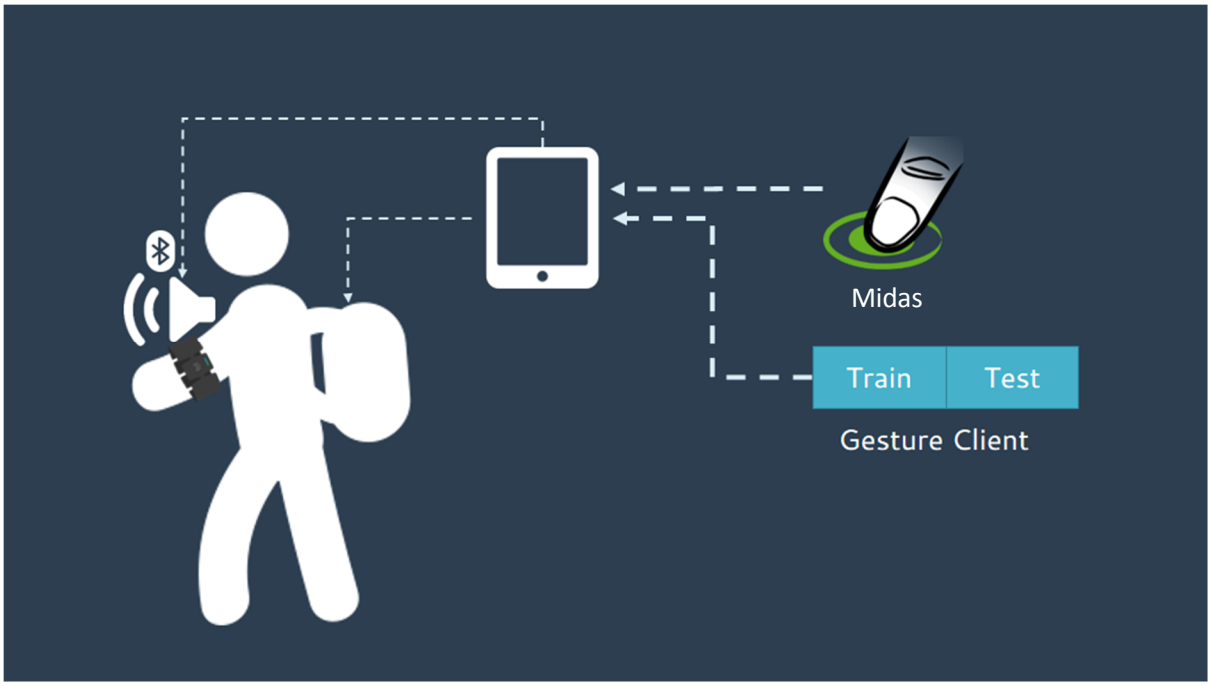


Abbildung 39: Technische Vision von Gvoice

Die oben dargestellte Vision zeigt die Verwendung von Gvoice mit Midas, allerdings muss sowohl Midas, als auch Gvoice vor der eigentlichen Verwendung so eingestellt werden, dass der Anwender mit den Systemen arbeiten kann. Dabei wird von einem Experten einen Pipeline in Midas angelegt und diese kann dann von dem eigentlichen Anwender einfach nur geladen werden und benutzt werden. Denn der Gesture Client kommuniziert mit Midas über eine TCP Verbindung und sorgt dafür, dass der Anwender nichts mit dem Node Designer zu tun hat, außer dass er diesen starten muss. Dies ist in dem folgenden Schaubild nochmals dargestellt.

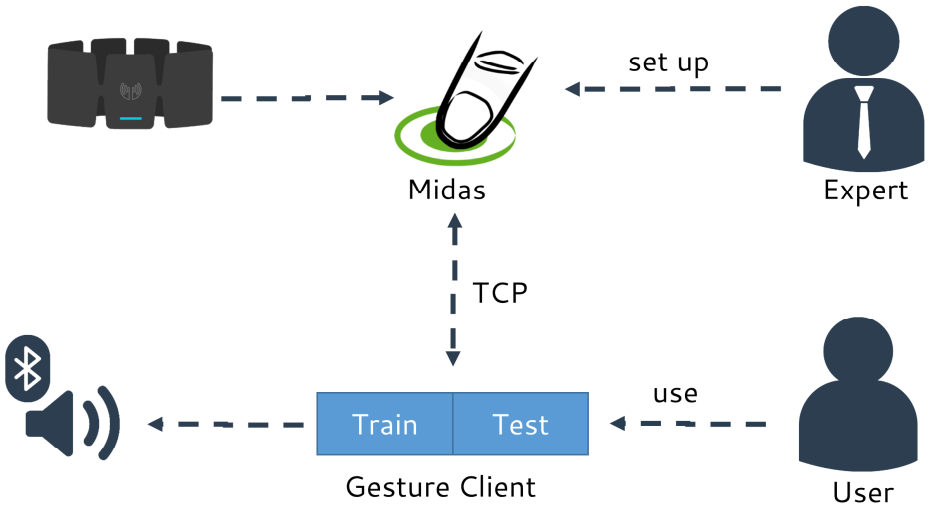


Abbildung 40: Einrichten von Gvoice

Mit diesen Ergebnissen ging es dann in die nächste Stufe des Lifecycles - die Designphase.

5.5.2.2 Entwurfsphase

Auf Basis der Anforderungen und der Visionen, wurden dann Sketches für die Anwendung entwickelt. Dazu wurde zunächst überlegt, aus wie vielen Interfaces die Anwendung gemäß den Anforderungen bestehen muss. Man ist dabei zu dem Ergebnis gekommen, dass vier Interfaces alle Anforderungen abdecken (siehe Abbildung 41). Der erste Screen bildet dabei den Hauptbildschirm, auf dem der Anwender einer der drei Seiten auswählen kann. Die Unterseiten sind in diesem Fall eine Seite mit der die *Gesten trainiert* werden und eine Seite mit der die Gesten dann in eine *Aktion* überführt werden. Außerdem braucht es noch eine Seiten mit der die *Einstellungen* für die Verbindung zwischen der Anwendung und dem Midas Node Designer regelt wird.

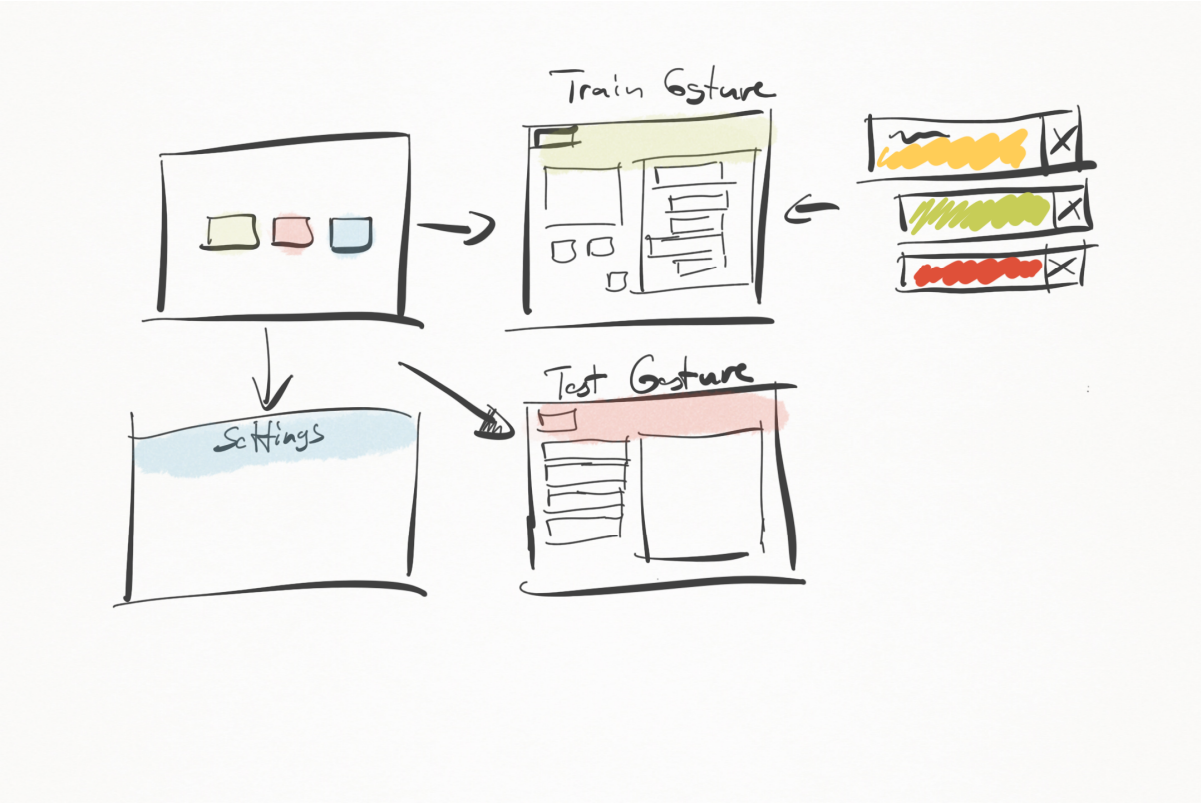


Abbildung 41: Die einzelnen Screens in Gvoice

In der Anforderung wurde deutlich gemacht, dass das Trainieren von Gesten so einfach wie möglich sein sollte. Dazu wurde die folgende Userstory erdacht:

Der Anwender möchte eine neue Geste trainieren, dazu wechselt er vom Hauptbildschirm in den Trainingsunterbildschirm. Dort trägt er zunächst oben rechts den Namen der Geste ein, die er trainieren möchte. Dann wählt er die geschätzte Dauer der Geste aus und drückt auf den Starten Knopf. Jetzt hat er fünf Sekunden Zeit sich vorzubereiten, die ablaufende Zeit wird sichtbar angezeigt. Nach dem der Timer abgelaufen ist, signalisiert die Anwendung ihm, dass er mit der Geste starten kann. Er führt seine Geste durch und sieht danach in der Anwendung einen Hinweis das die Aufnahme gestoppt hat und die Geste erfolgreich trainiert wurde. Das Trainieren der Geste wird anschließend mehrmals wiederholt, um mehre Variationen zu trainieren.

Da viele Gesten freihändig durchgeführt werden, wurde sich dafür entscheiden, einen Timer zu verwenden, der dem Anwender genügend Zeit verschafft seine Geste vorzubereiten. Für diesen Zeitraum wurden fünf Sekunden genommen. Das Starten des Timers kann über mehrere Eingaben erfolgen. So hat der Anwender die Möglichkeit den Timer über einen Button, eine Keyboard Taste oder aber über einen Presenter zu starten und zu beenden. Der Anwender muss außerdem nach seiner Geste nicht zwingend auf Stop drücken, denn nach dem Vorbereitungstimer startet ein zweiter Timer, der automatisch stoppt sobald dieser abgelaufen ist. Die Länge des Timers kann der Anwender selbst einstellen. Eine weitere Anforderung besagt, dass der Anwender die Möglichkeit haben soll zu sehen, wie gut eine Geste trainiert ist und wie gut sie sich erkennen lässt. Um dies dem Anwender deutlich zu machen, wird mit Farben gearbeitet. Hierbei kommt eine Farbskala zum Einsatz, die der Anwender von Ampeln kennt: *grün* heißt dabei „ok“, *gelb* „mittel

gut“ und *rot* „schlecht“. Dem Anwender wird auf diese Weise gezeigt wie groß die Variationen innerhalb einer Geste und deren Trainingsets sind. Es wird aber auch dargestellt, wie groß der Unterschied zu anderen Gesten ist, das heißt ob es zu Fehlerkennungen kommen kann. Die folgenden Sketches zeigen das Resultat aus allen aufgestellten Ideen und Entwürfen.

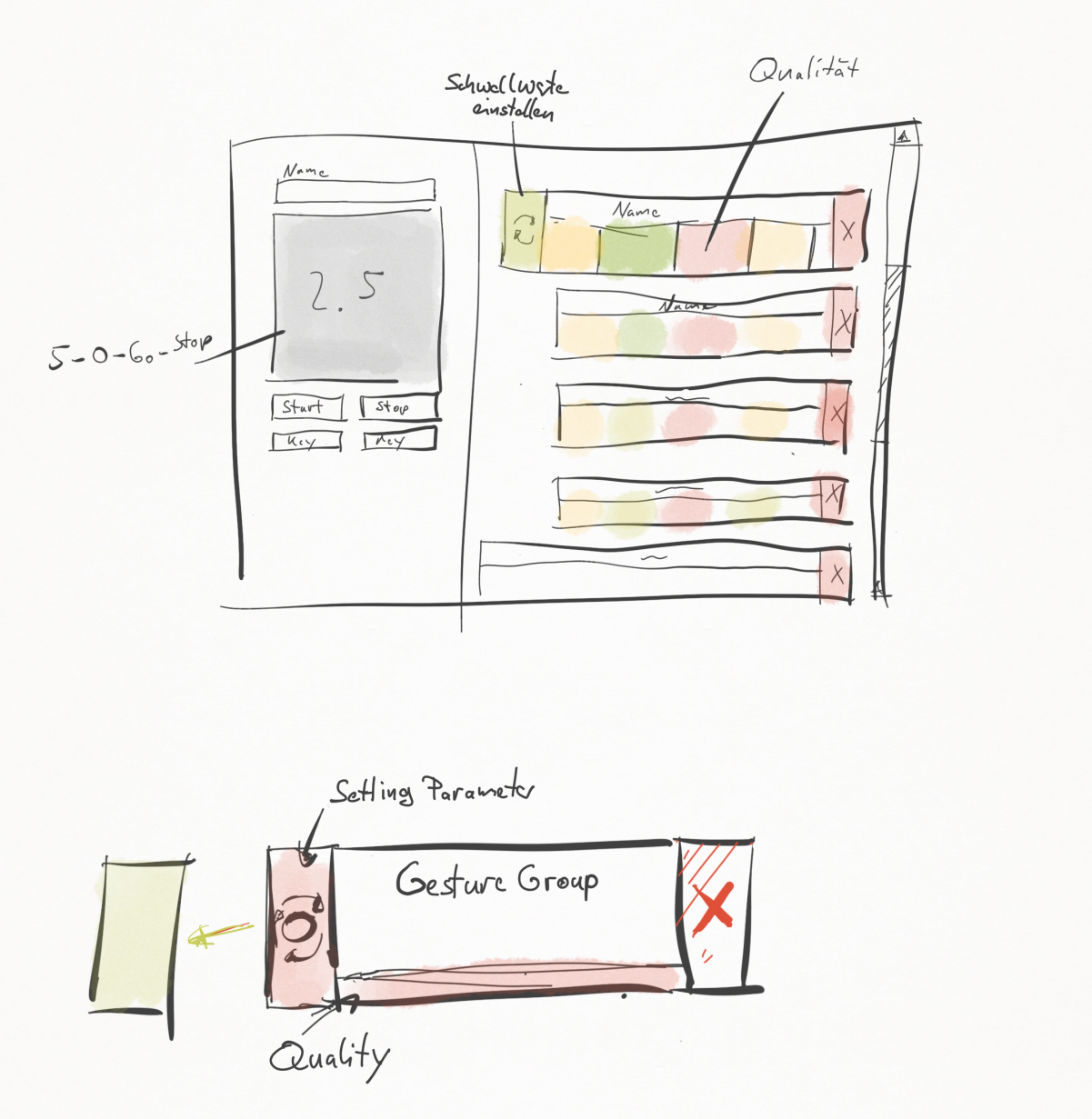


Abbildung 42: Trainingsseite von Gvoice. Die untere Abbildung zeigt die Repräsentation einer Geste.

Damit der Anwender den Überblick über die trainierten Gesten behält, sind diese folgendermaßen gegliedert. Wenn eine Geste trainiert wird, erscheinen zwei Einträge. Der obere Eintrag ist immer der Headereintrag, dieser fasst alle Informationen zu den untergeordneten Templates zusammen. Diese Templates werden dann unter dem Header eingerückt angezeigt. Auf diese Weise ist klar ersichtlich, welches Template zu welcher Geste gehört.

Als nächstes braucht es noch ein einfaches Interface für das Festlegen der einzelnen Aktionen für eine Geste. Dazu wurde auch hierfür wieder einen User Story entworfen:

Der Anwender hat mehre Gesten trainiert und auch so eingestellt, dass er sie wiederholen kann und die Gesten erkannt werden. Danach wechselt er zurück ins Hauptmenü und wählt dort „Testen“ aus. Auf der Seite fügt er zunächst eine Aktion über einen Button unten auf der Seite hinzu. Danach wählt er die entsprechende Geste aus, bei der er eine Aktion hinterlegen möchte. Nun wählt er einen Audio Datei aus, die nach dem Ausführen dieser Geste abgespielt werden soll. Anschließend testet er, ob alles funktioniert hat, indem er einfach die Geste durchführt. Ist er zufrieden, stellt er eine Aktion für die nächste trainierte Geste ein.

Beim Einstellen der Sounds soll der Anwender, wie in der User Story, einfach ein Audiofile eingeben können. Allerdings wäre es auch von Vorteil diese direkt über einen Mikro aufzunehmen. Und später dann beim Ausführen der Geste, diese dann wiederzugeben. Selbstverständlich soll der Anwender jederzeit alle Einträge löschen können. Das finale Ergebnis für die Testseite ist in dem folgenden Sketch dargestellt.

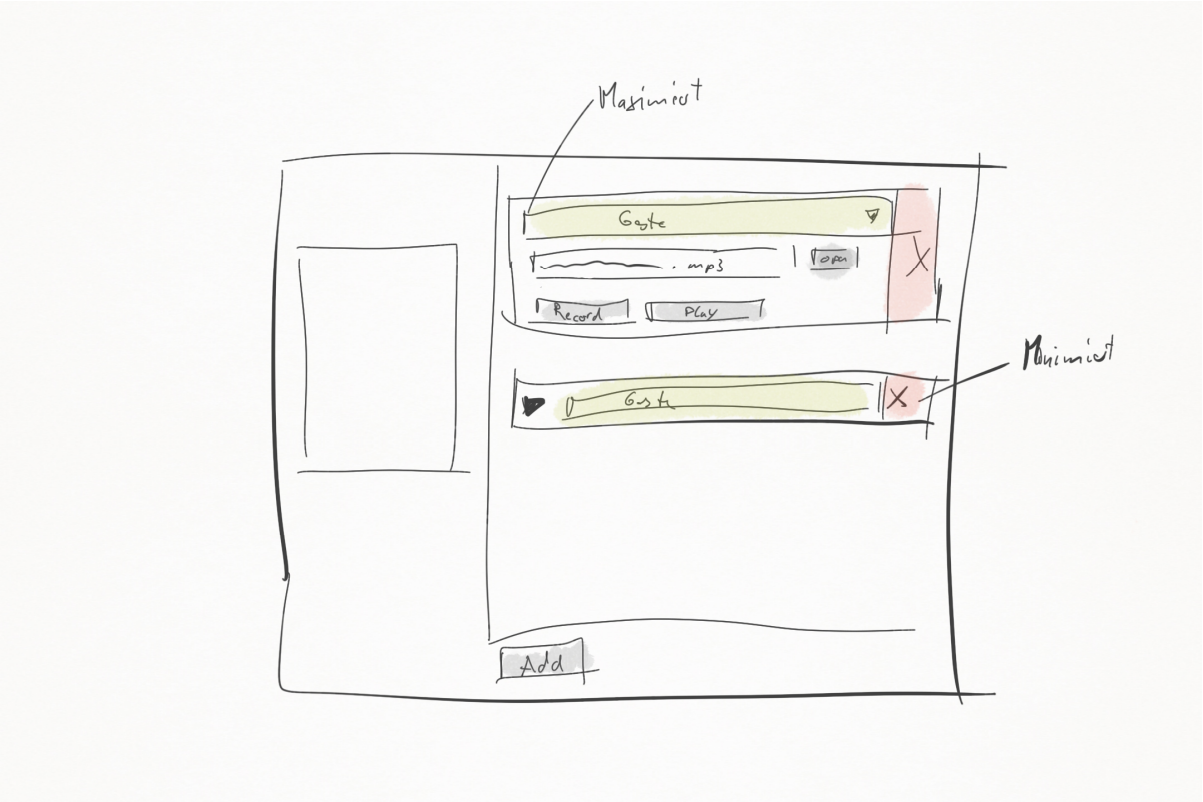


Abbildung 43: Gvoice Testseite - auf der rechten Seite sind die derzeit eingestellten Aktionen sichtbar

Ähnlich wie bei den Gestentemplates gibt es auch unter der Testenseite Einträge auf der rechten Hälfte. In diesen Einträgen definiert der Anwender die Audiodatei für eine bestimmte Geste. Der Übersicht halber soll jeder Eintrag minimierbar sein, das bedeutet es ist dann nur noch zu sehen welche Gesten eingestellt ist und der Löschen Button.

Damit liegt die wichtigsten Teile der Anwendung als Sketch vor und es kann mit der nächste Stufe des Lifecycles, der Prototypphase weitergehen.

5.5.2.3 Prototyp Entwicklungsphase

Mit all den vorher aufgestellten Anforderungen und Sketches kann nun mit der Entwicklung fortgefahren werden. Genauso wie Midas, wurde auch Gvoice in C# und WPF umgesetzt. Dies hat den Vorteil, dass jede Menge Code aus Midas wiederverwendet werden konnte. Dennoch bestand die Herausforderung herauszufinden, wie der Midas Node Designer und Gvoice untereinander kommunizieren, ohne dass der Anwender etwas davon mitbekommt.

Kommunikation:

Bei dem Midas Touch Client wurden bereits Informationen an den Node Designer über TCP weitergegeben, allerdings nur in sehr geringer Komplexität. Dennoch wurde sich dafür entschieden die bereits vorhandene TCP-Kommunikation über weitere Nodes auszubauen. Dazu wurde überlegt, wie am besten Ergebnisse und Objekte übertragen werden können. Die Lösung ist die Verwendung von einem Interface mit dem Namen „XMLSerilizer“. Diese Interfaces werden standardmäßig vom .Net-Framework bereitgestellt und sorgen dafür, dass einfache Objekte fast automatisch in XML überführt werden. Die so erzeugten XML Daten können dann einfach über das Netzwerk übertragen werden und auf der anderen Seite wieder deserialisiert werden. Deswegen wurde in Midas eine Node integriert, die automatisch die Objekte mit Hilfe des „XMLSerilizer“ Interface umwandelt. Allerdings können nur vorher vorbereitete Objekte auf diese Weise serialisiert werden, denn die meisten verwendeten Objekte waren zu komplex für eine automatische Konvertierung. Deshalb wurden Objekte wie die Templates so umgebaut, dass eine Konvertierung erfolgen kann. In Gvoice wurde dann für diese Objekte das entsprechenden Gegenstück implementiert, damit die Objekte wiederverwendet werden können. In dem Zusammenhang wurde auch die TCP Input und Output Node im Midas Node Designer verbessert und erweitert. Damit konnten die ganzen Templates und die Start-Stopp-Signale für das Aufnehmen von Gesten gesendet werden. All die gesendeten Befehle und Daten werden über ein Protokoll geregelt. Dieses Protokoll besteht aus einem Befehl (siehe Tabelle 3), gefolgt von einem Doppelpunkt. Nach diesem Doppelpunkt können dann noch Daten folgen.

BEFEHLE	DATEN	BESCHREIBUNG
Connect:	keine	Baut eine Verbindung zum Node Designer auf. (Handshake)
AskForTemplates:	keine	Mit diesem Befehl kann nach Templates gefragt werden.
AskForThreshold:	keine	Mit diesem Befehl werden alle Schwellwerte der Templates erfragt.
TemplateName:	Name der nächsten Templates	Mit diesem Befehl wird der Name für die nächsten aufgenommenen Templates festgelegt.
StartRecord:	keine	Die Aufnahme für ein Template wird gestartet.
StopRecord:	keine	Die Aufnahme für ein Template wird gestoppt.

TemplateTrain:	Guid des Templates	Für ein gewisses Set an Templates wird versucht Schwellwerte automatisch zu berechnen.
TemplateDelete:ALL:	Guid des Templates	Mit diesem Befehl wird ein Templateset komplett gelöscht.
TemplateDelete:	Guid des Templates	Mit diesem Befehl wird nur ein Template gelöscht.
Thresholds:	XML von dem Schwellwert Objekt	Dieser Befehl legt einen Schwellwert für ein Template fest.

Tabelle 3: Protokoll Anfragebefehle

Wurde vom Client einer der oben aufgeführten Befehle an Midas gesandt, wird entsprechend dem Befehl eine Antwort gesendet. Die Antworten sehen wie folgt aus:

BEFEHLE	DATEN	BESCHREIBUNG
Template:	Templates als XML Objekt	Mit dieser Antwort erhält man alle derzeitigen Templates.
Result:	Erkannte Gesten als XML Objekt	Mit dieser Antwort werden alle gerade erkannten Gesten übertragen.
Thresholds:	Schwellwerte als XML Objekt	Diese Antwort enthält alle Schwellwerte der Templates

Tabelle 4: Protokoll für die Antworten

In den Protokollen stehen jede Menge Befehle und Antworten, die die Schwellwerte beinhalten. Der Grund dafür ist, dass sie die zweit wichtigste Komponente von Gvoice sind. Denn die Schwellwerte entscheiden darüber, ob eine Geste richtig erkannt wurde. Ist der Schwellwert einer Geste zu hoch, kann es passieren, dass die Geste häufig fehlerhaft erkannt wird. Wenn der Schwellwert zu niedrig ist, dann wird die Geste mitunter gar nicht mehr erkannt. Deshalb ist es wichtig den richtigen Schwellwert einzutragen und zu verwenden.

Schwellwerte:

Damit der Anwender aber nicht selbst die Schwellwerte einstellen muss, wurde ein Verfahren entwickelt mit dem sich diese automatisch berechnen lassen. Dazu wurde im Midas Node Designer die Node namens „Calculate Thresholds“ hinzugefügt. Diese berechnet dabei die einzelnen Schwellwerte nach dem folgenden Algorithmus:

```
Function ClaculateThreshold(SetofTemplates)
     For all Template in SetofTemplates
          Max = claculateMaxDistanze(Template, SetofTemplates);
          Avg = claculateAvgDistanze(Template, SetofTemplates);
          Template.Thresholde=(Max-Avg)/2.0 + Avg;
     End for
End function
```

Anfangs bin ich davon ausgegangen, dass man die Schwellwerte einfach automatisch berechnen kann, ohne diese später noch nachjustieren zu müssen. Mit dieser Annahme lag ich falsch, denn unteranderem hängt die Berechnung der Schwellwerte davon ab, wie gut die trainierten Templates sind. Denn wie bereits bei dem Touch Client festgestellt wurde, neigen die Anwender dazu, beim Trainieren der Gesten dazu diese perfekt zu wiederholen. Das bedeutet, es werden Templates angelegt, die sich nur minimal unterscheiden. Dies führt bei dem oben aufgeführten Algorithmus dazu, dass ganz kleine Schwellwerte berechnet werden. Meistens kann der Proband die Geste allerdings nach einiger Zeit nicht mehr so perfekt reproduzieren. Der Grund dafür ist das beim Trainieren der Geste das Muskel Memory zum Tragen kommt und so dafür sorgt, dass die Gesten identisch sind. Aus diesem Grund müssen in den meisten Fällen die Parameter im Nachhinein noch ein wenig angepasst werden. Um dies zu ermöglichen, wurde die „Calculate Thresholds“ Node um die Funktion für manuelle Schwellwerteingaben erweitert.

Manchmal reicht allerdings das Einstellen von Schwellwerten nicht aus, um eine Geste zuverlässig zu erkennen. Der Grund dafür ist, dass die Geste möglicherweise zu ähnlich zu einer anderen ist und deswegen nicht richtig erkannt wird. Dehalb ist es wichtig dem Anwender zu verdeutlichen, dass die Geste zu ähnlich sind.

Qualität der Gesten:

Die Ähnlichkeitvisualisierung von Gesten soll nach den Sketchen, über eine Farbskala realisiert werden. Dazu braucht es eine Funktion, die den Unterschied zwischen den einzelnen Templates ermittelt. Dieser Unterschied ist natürlich abhängig von dem verwendeten Klassifizierungsalgorithmus, deshalb wurde die „Dynamic-Time-Warping“ Node um die Funktion der Berechnung der Qualität erweitert. Diese Funktion berechnet den Abstand von jedem Template zu jedem anderen und ermittelt so den minimalen, maximalen, median und durchschnittlichen Abstand zu den einzelnen Template Sets. Dies ist in dem folgenden Algorithmus dargestellt.

```
Function ClaculateGestureQuality(AllTemplates)
     SetsofTemplates = ClaculateSets(AllTemplates);
     For all Template in AllTemplates
          For all SetofTemplate in SetsofTemplates
               For all TemplateofaSet in SetofTemplate
                    Costs.add(CalculateCost(Template,
                    TemplateofaSet));
               End for
               Max = MaxDistance(Costs);
               Min = MinDistance(Costs);
               Avg = AvgDistance(Costs);
               Median = MedianDistance(Costs);
               Template.stats.add(Max,Min,Avg,Median,
               SetofTemplate.Name);
          End for
     End for
End function
```

Auf diese Weise kann genau ermittelt werden, wie gut ein Template im Verhältnis zu den anderen Templates ist. Damit im Midas Node Designer die Qualität auch überprüft werden kann, wurde wieder eine weitere Node dem Designer hinzugefügt, der „Template Quality Manager“. Die errechneten Qualitäten sind in den Template Objekten gespeichert, das heißt sie werden automatisch an Gvoice übertragen. Dort wird dann die errechnetet Qualität in die Farbvisualisierung überführt, mit Hilfe des Medians für Templates des gleichen Sets. Und bei Templates eines anderen Sets werden dann die maximale Distanzen verwendet.

```
Function ClaculateGestureQualityColor(AllTemplates)
     For all Template in AllTemplates
          For all statistic in template.stats
               If(statistic.name is template.name)
                    Statistic.Quality = statistic.Median –
                    Median(template.stats.median);
               Else
                    Statistic.Quality = statistic.Max;
          End for
     End for
End function
```

Der Median wird verwendet um zu verhindern, dass wenn der Abstand zu einem anderen Template in der gleichen Gruppe zu groß ist, die Qualität der Training Sets trotzdem in einem farblich grünen Bereich liegt. Angenommen es werden für eine Geste vier Templates aufgenommen, die Geste wurde dabei mit unterschiedlichem Grad an Anstrengung aufgenommen. Das würde bedeuten, dass das Template, welches mit wenig Kraft aufgenommen wurde, einen großen Abstand zu einem Template aufweist, welches mit viel Kraft aufgenommen wurde. Die Darstellung des großen Abstands würde drauf hindeuten, dass die Geste in sich schlecht trainiert ist, denn es besteht große Varianz. Dies ist aber eigentlich nicht der Fall, denn es gibt ja noch zwei Templates dazwischen. Mit dem Median werden diese Abstände ebenfalls berücksichtigt, und erzeugen so ein homogeneres Bild der Templates. Gerade bei komplexen Gesten, die sehr viel Kraft benötigen (z.B. Liegestütz) kann es dennoch vorkommen das die Abstände der einzelne Templates recht hoch sind. Dennoch kann es sein, das die Geste gut erkannt wird. Das liegt meist daran, dass die eingesetzte Kraft bei keiner anderen Geste vorhanden ist. Um diese auch zu verdeutlichen wird dem Medianabstand innerhalb Geste noch der Median des Median aller anderen Abstände zu anderen Gesten abgezogen. Das bedeutet, eine Geste ist nicht nur dann gut trainiert, wenn die Sets kleine bis mittlere Abstände haben, sondern auch, wenn keine anderen Gesten ähnlich sind. Damit der Anwender dennoch genau sehen kann zu welcher Geste die Geste ähnlich ist werden die Qualitäten im Verhältnis zu den anderen Gesten nochmals aufgeschlüsselt dargestellt. Die Grundlage für die Ermittlung ist dabei der maximale Abstand zu einer anderen Geste. Das bedeutet, wenn der maximale Abstand einen kleiner Wert hat, ist es sehr wahrscheinlich, dass es zu Fehlinterpretationen kommt.

Nach dem all diese fundamentalen Entwicklungen erfüllt wurden, konnte das Interface entsprechend gebaut werden.

Interface:

Wenn der Benutzer die Anwendung startet, erscheint zu allererst das folgende Hauptmenü in dem er dann in die einzelnen Unterseiten wechseln kann. Das Design ist an das *Materialize Design* [10] angepasst, welches unteranderem bereits von Apple, Google und Microsoft für Anwendungen eingesetzt wird. Die Besonderheit bei diesem Design ist, dass mit keinen Farbverläufen gearbeitet wird und deswegen nur wenige Farben verwendet werden. Aus diesem Grund sind alle Icons auch so dargestellt, dass sie auf das Wesentliche reduziert sind. Der Hauptbildschirm sieht wie folgt aus:

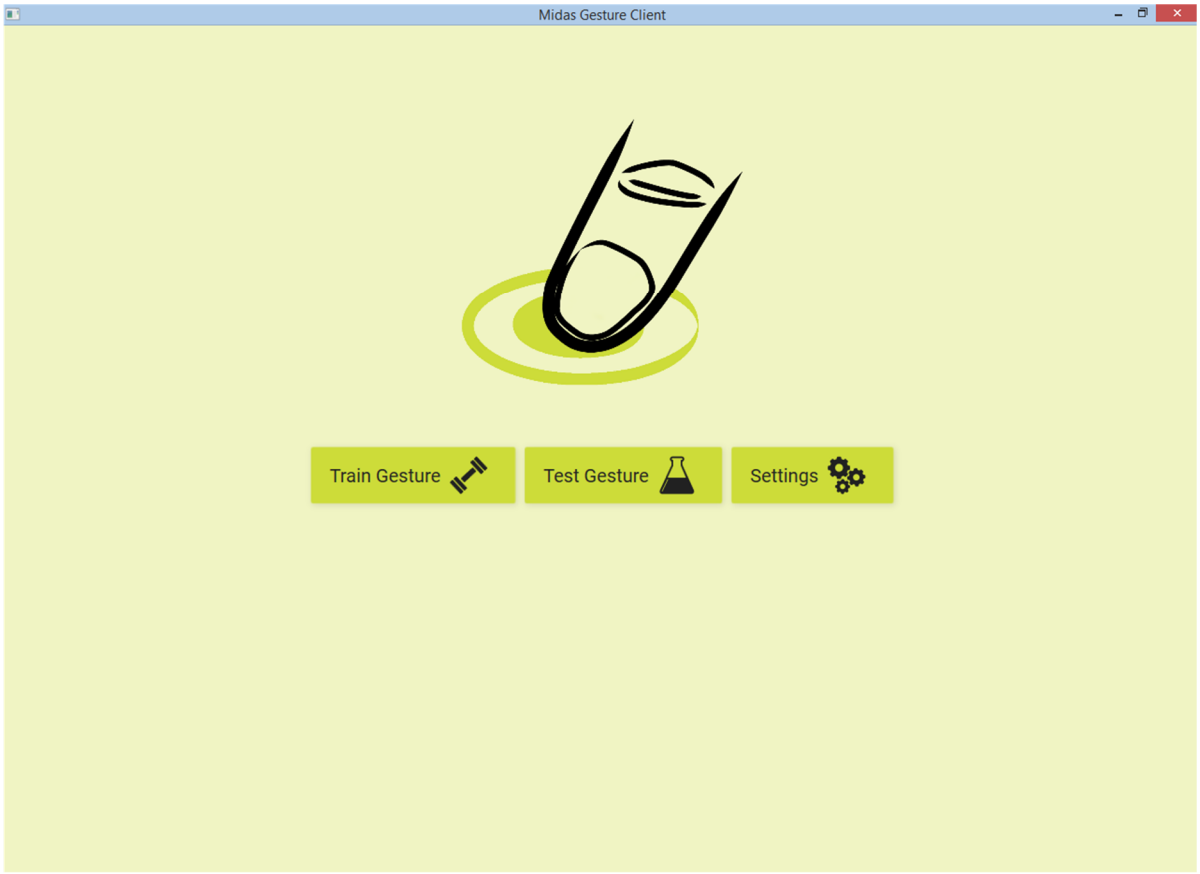


Abbildung 44: Hauptmenü von Gvoice

Die Buttons sind so angeordnet, dass der Anwender von links nach rechts die Schritte durchgehen kann. Die erste Auswahlmöglichkeit bildet Train Gesture. Damit kann der Anwender Gesten trainieren (siehe Abbildung 45). Wenn der Anwender einen Geste trainieren möchte, dann gibt er zunächst oben links den Namen ein. Danach kann er die Dauer der Aufnahme bestimmen, indem er auf den Button unterhalb des Countdowns drückt. Bei wiederholtem Drücken des Buttons wechselt die Aufnahmelänge der Geste, angefangen von 1s über 2s, 3s, 5s, 10s, 20s bis hin zu 30s. Wenn er dann auf Start drückt, läuft im Countdown Fenster die Zeit von 5 Sekunden ab, wenn die 5 Sekunden abgelaufen sind, kann mit dem Aufnehmen begonnen werden. Dann wird entsprechend der voreingestellten Länge „Go“ angezeigt und danach „Stop“. Dies signalisiert dem Anwender, dass er Aufhören kann seine Geste auszuführen, denn die Aufnahme hat gestoppt. Er kann die

[10] https://www.google.com/design/spec/material-design/introduction.html#

Aufnahme auch vorzeitig über das Drücken des Stop Buttons beenden. Wie bereits in den Sketchen angedacht, kann der Anwender Hotkeys für die Buttons festlegen. Dafür stehen unter dem Button zwei Comboboxes zur Verfügung. Manchmal ist allerdings nicht klar, wie ein Taste auf der Tastatur heißt, dies ist besonders der Fall bei Sonderzeichen. Deswegen wird die letzte gedrückte Taste unter den Comboboxen angezeigt. Das heißt, wenn der Anwender nicht weiß wie die Taste heißt drückt er sie einfach und kann dann sie entsprechend in der Combobox auswählen. Auf diese Weise lässt sich auch ein Presenter recht einfach einbauen.

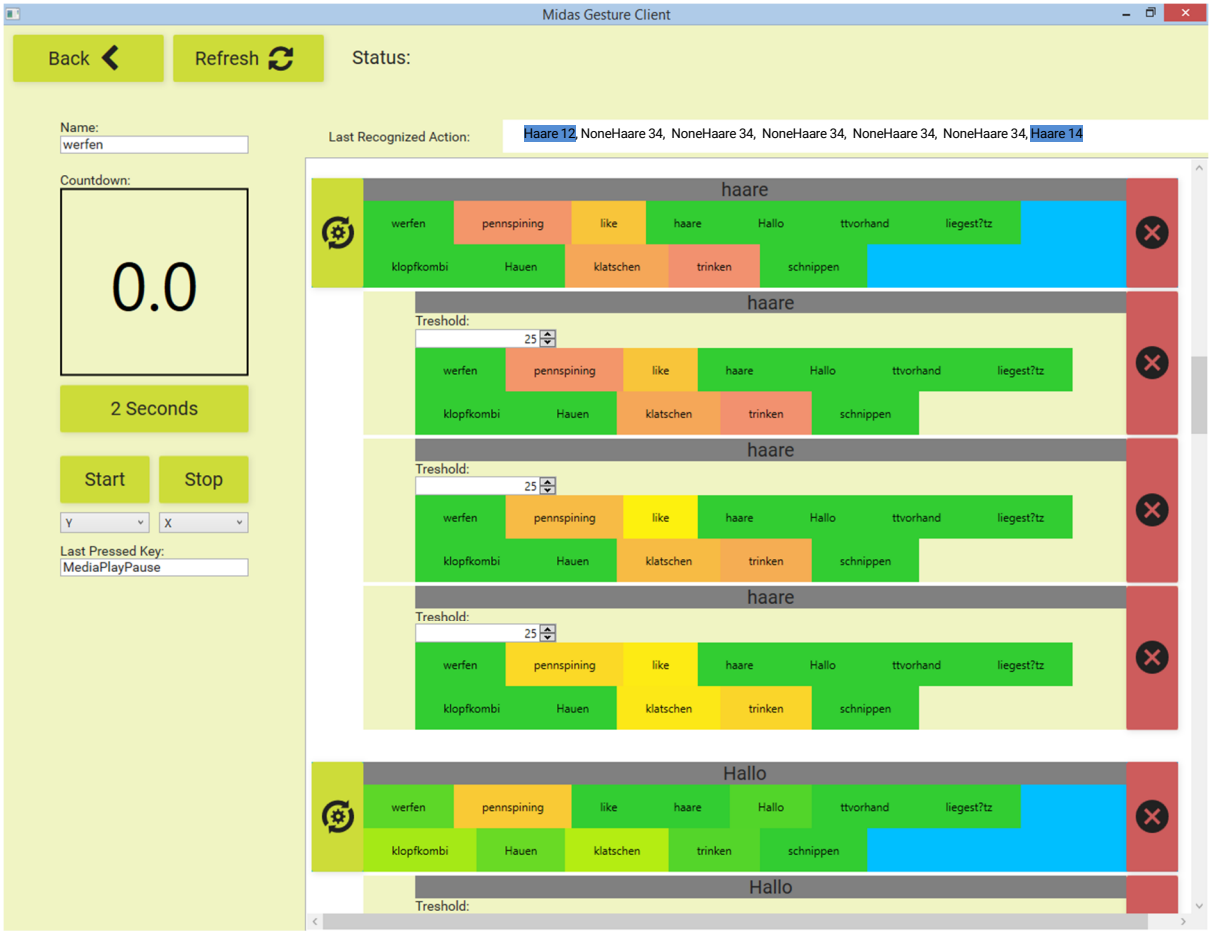


Abbildung 45: Gvoice Trainigsseite

Nach dem der Anwender eine Geste erfolgreich aufgenommen hat, erscheinen zwei Elemente in der rechten Liste. Das obere Element ist die Geste an sich, in welchem alle Informationen über die Geste enthalten sind. Unter diesem oberen Headerelement erscheinen dann die einzelnen Templates der Geste. Der Aufbau der beiden Elemente ist sehr ähnlich. Bei beiden wird der Name der Geste zu der sie gehören angezeigt, daneben besitzen beide einen Löschen Button. Der Löschen Button bei dem Header löscht nach einer Bestätigung die gesamte Geste, während der andere nur das Template löscht. Das Header Element hat einen grünen Button, der die Schwellwerte kalkuliert bzw. eine Nachricht an den Node Designer sendet, wo dann die Berechnung durchgeführt wird. Das Ergebnis dieser Berechnung wird in den Nummernfeldern in den Templates angezeigt. Dort kann der Schwellwert dann auch angepasst werden. Über die bunten Kästen, kann der Anwender nachlesen, wie gut eine Geste ist. In der gezeigten Abbildung 45 ist beispielsweise die Geste mit dem Namen *Haare* zu den meisten anderen Gesten recht verschieden. Allerdings gibt es auch ein

paar Gesten wie z.B. *penspining*, die ähnlich sind. Dagegen ist die Geste in sich im grünen Bereich was bedeutet, dass sie gut erkannt wird. Nachdem der Anwender die Gesten erfolgreich trainiert hat, kann er die Geste mit einer Aktion verknüpfen. Zu diesem Zweck wechselt er auf die Testseite (siehe Abbildung 46). Um eine Aktion anzulegen, geht der Anwender auf den Add Button unten. Damit wird eine Aktion der Liste hinzugefügt. Dort wählt der Anwender in der Combobox die Geste, die er verknüpfen möchte aus. Auf diese Weise wird die definierte Aktion ausgeführt, sobald die Geste erkannt wird.

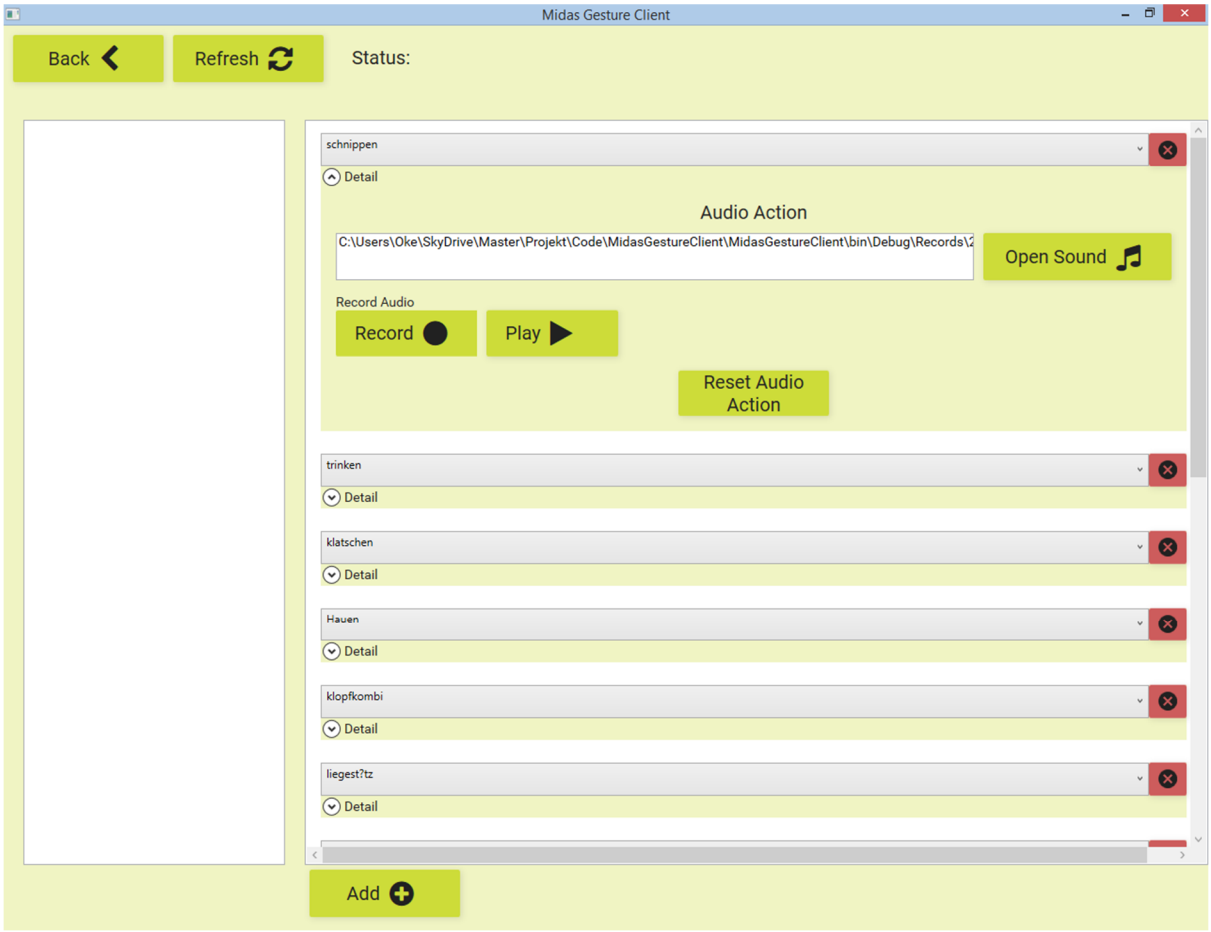


Abbildung 46: Testseite

Um die Aktion jetzt im Detail einzustellen, geht der Anwender auf Detail und sieht jetzt die einzelnen Aktionen. Im Moment gibt es nur eine Audio Action, die Liste kann aber beliebig erweitert werden, entsprechende Interfaces sind bereits vorhanden. In der Audio Action hat der Anwender die Möglichkeit entweder eine Audio Datei zu öffnen, oder aber direkt eine aufzunehmen. Letzteres ist eine praktische Lösung, die es ermöglicht schnell Audio Content zu produzieren.

Damit diese Aktionen jedoch ausgeführt werden können, muss die Anwendung wissen wann. Das heißt die Ergebnisse der Erkennung müssen an die Anwendung gelangen.

Ergebnisse:

Der Erkennungsalgorithmus berechnet jede 1,7 ms ein neues Ergebnis, das an den Client übertragen werden müsste. Dies würde dazu führen, dass extrem viele Daten über das Netzt

geschickt werden. Außerdem passiert es häufig, dass wenn eine Geste erkannt wird, diese oft mehrmals direkt hintereinander erkannt wird. Der Grund dafür ist, das beim Verschieben des Fensters, die Ähnlichkeit an einer bestimmten Geste immer noch hoch genug ist, dass sie unter dem Schwellwert liegt und deswegen erkannt wird. Um all dies ein wenig aufzufangen, wurde eine Node mit dem Namen „Result Filter“ entwickelt. Diese stellt sicher, dass wenn eine Geste erkannt wurde erst nach einer vom User bestimmten Zeit eine weitere Geste erkannt wird. Nachdem diese Resultate gefiltert wurden, werden sie an Gvoice über das Protokoll gesendet. Dort werden sie nicht nur für das Auslösen von Aktionen verwendet, sondern sie werden dem Anwender auch präsentiert. So sieht der Anwender auf der Trainingsseite (siehe Abbildung 45) diese Resultate oberhalb der trainierten Gesten. Auf diese Weise werden ihm die letzten Resultate zusammen mit den errechnetet Schwellwert präsentiert. Die Schwellwerte helfen dann auch beim Einstellen der Werte für die Geste. Dem Anwender werden neben den positiven Ergebnissen, auch die negativen, also nicht erkannten Ergebnisse präsentiert. Der Unterscheid zwischen den beiden Ergebnissen wird zum einen über den Namen ersichtlich, denn alle nicht erkannten Gesten starten mit „None“ und zusätzlich werden die positiven Resultate blau eingefärbt. Dies hebt sie deutlich von den anderen ab. Ähnlich wie ein Newsticker, laufen sie von rechts nach links, wobei immer jede Sekunde ein neues Ergebnis rechts hinzugefügt wird. Da auf diese Weise nur festgestellt werden kann, welche Geste erkannt wurde und nicht welches Template dafür gesorgt hat, wurde eine zweite Technik verbaut. Wenn also ein bestimmtes Template für die Auslösung einer Geste gesorgt hat, blinkt der Hintergrund des Namens dieses Template auf und wird von anfangs grau zu weiß. So kann genau nachvollzogen werden welches Template erkannt wurde, auch wenn das Blinken nicht wahrgenommen wurde.

Speichern:

Damit der Anwender nicht immer alles beim Neustart des Programms von nochmals einstellen muss, werden die Einstellungen, die in Gvoice gemacht werden, automatisch beim Schließen des Programms gespeichert. Startet der Anwender Gvoice erneut, werden die Settings automatisch geladen. Dennoch darf nicht vergessen werden, die Daten in Midas ebenfalls zu speichern, denn hier wird manuell gespeichert und auch geladen.

5.5.2.4 Tests

Nachdem die Entwicklung abgeschlossen wurde, ging es darum die Pipeline aufzubauen und zu testen. Bei dem ersten Ansatz wurde einfach die Pipeline für die Touchgestenerkennung verwendet. Es wurde recht schnell klar, dass mehrere Gesten nicht so einfach funktionierten. Der Grund dafür schienen die EMG Daten zu sein. Die EMG Daten repräsentieren die Aktivierungsspannung der Muskeln. Da es sich um ein oszillierendes Signal handelt gibt es sowohl positive, als auch negative Werte. Das liegt daran, dass bei der Muskelaktivierung immer wieder ein Potenzial auf- und danach abgebaut wird. Die Spannung schwankt dadurch immer zwischen positiven und negativen Werten (-70mv bis +40mv) (Gottlieb & Agarwal 1971). Ursprünglich bin ich davon ausgegangen, dass das EMG Signal reproduziert werden kann, wenn die gleiche Geste mit der gleichen Kraft durchgeführt wird, dies ist allerdings nicht der Fall. Der Test wurde mit einem Schaumstoffball durchgeführt, der immer auf die gleiche Weise und mit der gleichen Kraft zusammengedrückt wird. Das Ergebnis war, dass die gemessene Kurve immer unterschiedlich aussieht. Um dieses Problem zu umgehen,

wurde versucht das EMG Signal umzuwandeln in ein Signal das die Aktivität des Muskels wiedergibt. Dazu wurden die folgenden Schritte angewendet.

1. Von dem Signal wird das Absolut gebildet. Damit fallen schon alle negativen Werte weg.
2. Das Signal wird normalisiert, dies vereinfacht später den Vergleich mit dem Gyroskop und dem Accelerometer Signal.
3. In einem dritten Schritt wird das Signal dann geglättet. Dazu wurden zwei Algorithmen ausprobiert. Zum einen ein Kalman Filter (Kalman 1960), zum anderen ein Durchschnittsfilter. Letzterer bildet aus den letzten Werten, zusammen mit dem neuen Werten, den Durchschnitt und gibt diesen aus. Der Durchschnittsfilter stellte sich als beste Wahl heraus, da er am schnellsten berechnet werden kann und das gewünschte Ergebnis erzielt.

Auf diese Weise lässt sich das EMG Signal besser zwischen den einzelnen Gesten unterscheiden. Denn das Signal zeigt nun im übertragenen Sinne die derzeit eingesetzte Kraft.

Die finale Pipeline sah dann wie folgt aus:

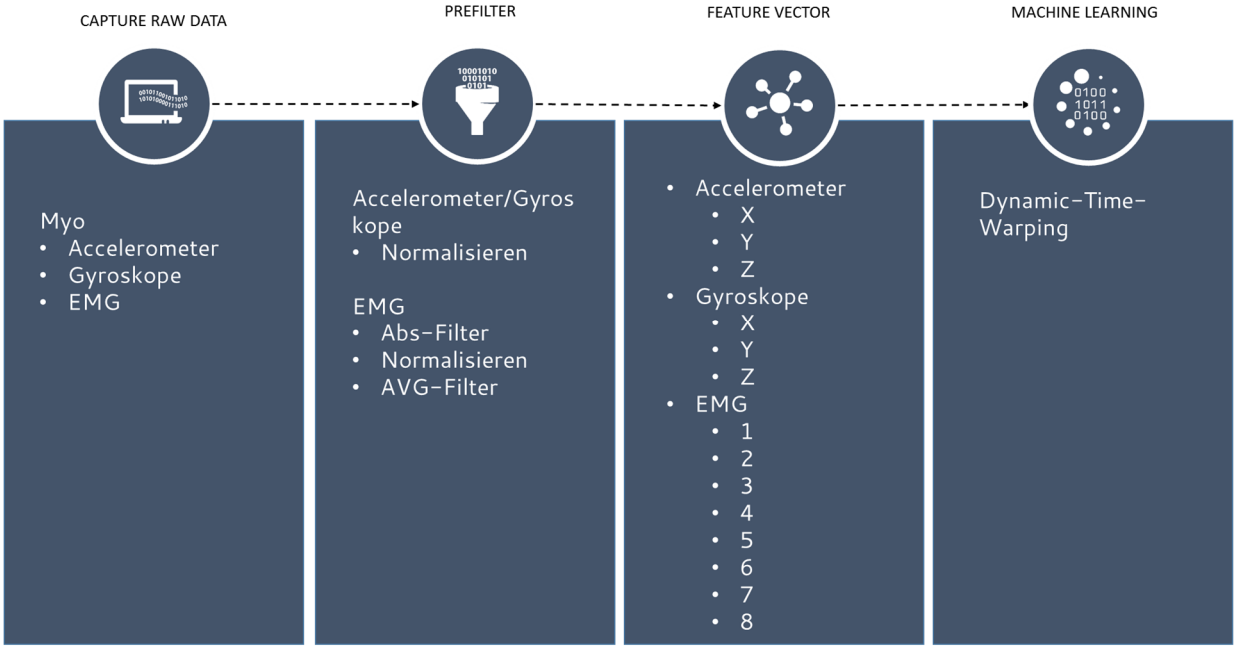


Abbildung 47: Finale Pipeline

Mit dieser Pipeline konnten 5 bis 6 unterschiedliche Gesten erkannt werden. An diesem Punkt erschien ein Paper mit dem Namen „Scepeter“ (Paudyal et al. 2016). Das genau das Gleiche machte, das ich entwickelt hatte.

5.5.2.5 Scepeter

Die Autoren von „Scepeter“ haben ein System entwickelt, das mit Hilfe von zwei Myo Armbändern in der Lage war Zeichensprache zu erkennen. Das Interessante dabei ist, dass sie bis zu 60 Gesten unterscheiden können. Sie verwenden dazu genauso wie Midas Dynamic-Time-Warping als Klassifizierungsalgorithmus und kommen auch zu dem Schluss, dass dieser Algorithmus sich am besten eignet. Was sie auch erkannt haben ist, dass die EMG Daten als Rohdaten nicht zu verwenden sind und deswegen kumulieren sie sie über die Laufzeit einer Geste hinweg. Dies führt dazu, dass sie Rechenzeit einsparen und damit mehr Gesten erkennen können. Neben alldem, verwenden sie bei jeder Geste nur die Veränderungen, das heißt sie ziehen bei jedem Fenster den Anfangswert von allen ab. Dies ermöglicht es die Geste auch unteranderen Bedingungen durchzuführen, z.B. bei gleichmäßigem Vorwärtslaufen zu erkennen, oder aber, wenn der Anwender die Geste mit unterschiedlichen Startpositionen durchführt.

5.5.2.6 Verbesserungen

Teile von *Scepeter* wurden dann auch in Midas übernommen, zum Beipiel das Verwenden von nur einem EMG Wert pro Sensor. Aber auch das Abziehen des ersten Wertes von dem Rest der Werte des Fensters. Für beide Filter wurde einfach eine neue Node in Midas angelegt. Diese beiden Änderungen führten dazu, dass deutlich mehr Gesten erkannt wurden, da über die Hälfte der Rechenzeit eingespart wurde. Zusätzlich wurden Gewichtungen der Feature Vektoren eingebaut. Diese stellten sicher, dass die EMG Daten (Faktor 0,375) bei der Kostenberechnung keine so große Rolle mehr spielten. Dieser Faktor von 0,375 ergibt sich dadurch dass die anderen Sensoren jeweils drei Einheiten besitzen, die jeweils mit Faktor 1 gewichtet sind, da der EMG Sensor aus acht Einheiten besteht werden sie durch den Faktor soweit reduziert, dass sie der Gewichtung eines Sensors mit drei Einheiten entspricht. In dem Zuge wurde der Dynamic-Time-Warping Algorithmus durch Multi-Threading weiter verbessert, so dass mit dieser Version ca. 20 Gesten unterscheiden werden können, ohne dass es zu lange dauert eine Berechnung durchzuführen.

Die Pipeline sieht jetzt wie folgt aus:

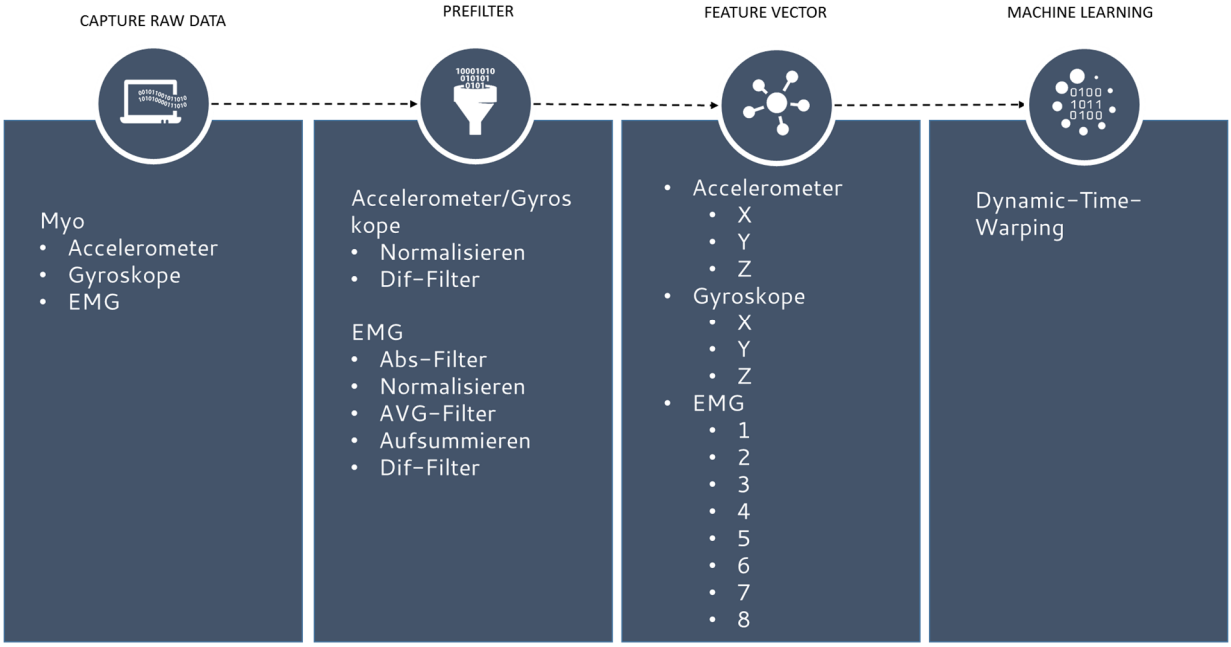
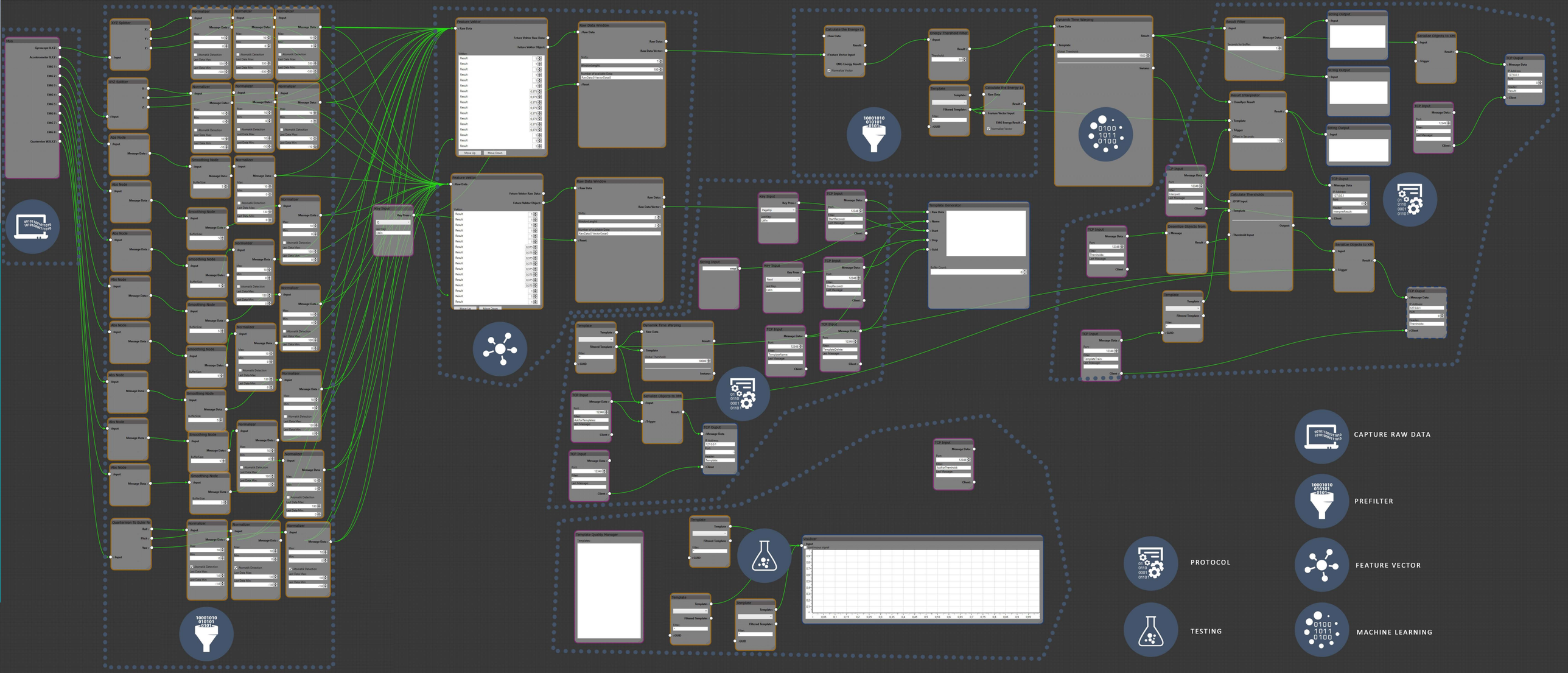


Abbildung 48: Pipeline nach der Verbesserung

Anschließend wurden weitere Verbesserungen vorgenommen, wie ein Value Filter. Dieser Filter stellt sicher, dass nur Signale an den Klassifizierungsalgorithmus weitergeben werden, in denen auch etwas passiert. Das heißt, bewegt der Proband den Arm nicht, muss auch keine Gesten erkannt werden. Dieser Filter wurde ebenfalls als Node in Midas hinzugefügt und spart ebenso Rechnenzeit. Gleichzeitig wurde die Myo Node verändert, denn das Myo SDK stellt neben Accelerometer, Gyroskop und EMG auch die Orientierung zur Verfügung. Deshalb wurden nun auch die Orientierungsdaten als Eingabedaten für die Pipeline verwendet. Vorteil dieser zusätzlichen Daten ist, dass die Erkennung besser geworden ist. Allerdings sorgen diese Daten für mehr Rechenzeit. Es muss also abgewogen werden was besser ist. Parallel zu diesem Projekt wurden weitere Szenarien entwickelt und getestet.


CAPTURE RAW DATA
PREFILTER
PROTOCOL
FEATURE VECTOR
TESTING
MACHINE LEARNING

5.5.3 *Gestenanalyse*

Das folgende Szenario entspringt der Idee, Analysen von Studien möglichst automatisiert durchzuführen. Dies würde dazu führen, dass Studien schneller und umfangreicher ausgewertet werden können. Derzeit wird das Verhalten von Probanden entweder über Beobachtungen, oder über Fragebögen erfasst (Erdös 2014a). Im Falle von Beobachtungen, wird häufig Videocoding eingesetzt. Dies ist ein Verfahren bei dem die aufgenommenen Videos der Studie, im Nachhinein auf bestimmte Verhalten der Probanden analysiert werden. Deshalb ist Videocoding ein sehr zeitaufwendiges Verfahren, dass leider auch keine Erfolgsgarantie bietet (Erdös 2014a). Aus diesem Grund wurde im Rahmen dieses Projekts überlegt, wie ein solches Verfahren mit Midas möglichst automatisch durchgeführt werden kann.

5.5.3.1 Touchzuordnung

Bei vielen kollaborativen Studien an Multitouchsystemen, wird im Nachhinein über Videocoding bestimmt wie viele Touchaktionen die Probanden jeweils durchgeführt haben. Ich habe hierfür im Rahmen von einem anderen Projekt eine Software geschrieben, die in der Lage war das Videocoding dieser Touchinteraktion deutlich zu vereinfachen (siehe Abbildung 49). In diesem Tool wurden die Touch Logging Daten hineingeladen und das dazugehörige Video. Dadurch, dass die Logging Daten mit einem Zeitstempel versehen sind, kann der Anwender auf das Event klicken und das Video wird an die entsprechende Stelle gespult. So braucht der Anwender nur noch anhand des Videos entscheiden, wer die Aktion ausgelöst hat. Mit diesem Tool braucht man etwa 15 Minuten für eine Stunde Video Material, im Vergleich zu Stunden mit herkömmlichen Videocoding Tools wie *Noldus Observer* (Erdös 2014b).

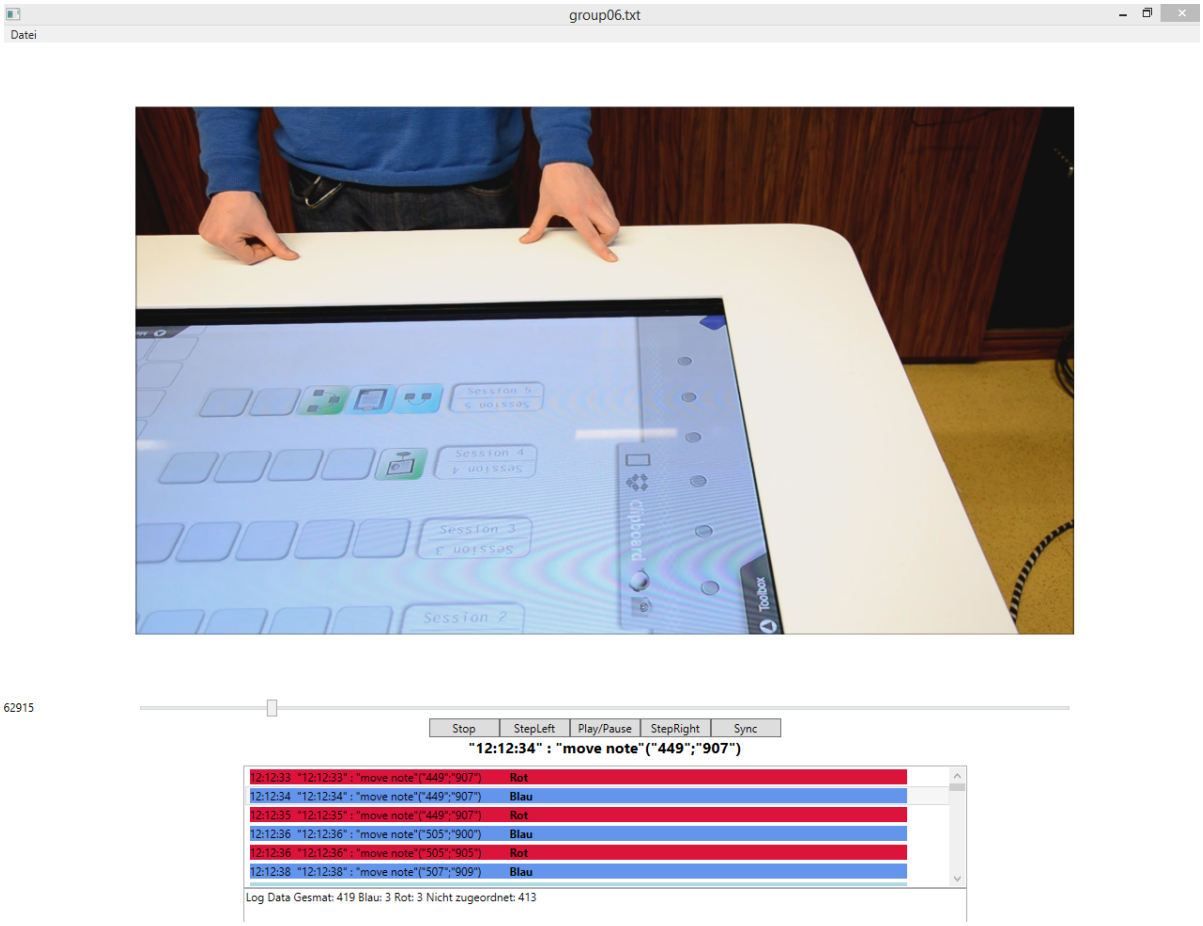


Abbildung 49: Videocoding Tool. Oben ist das zu loggende Video zu sehen und unten kann der Anwender die vom System geloggten Daten sehen und sie mit der linken Taste dem einen Probanden zuordnen und mit der rechten dem anderen.

Trotz dieses Tools, dauert das Auswerten immer noch eine Weile. Mit Midas könnte man das nun weiter verbessern, denn Midas ist in der Lage Touch Interaktion zu identifizieren. Damit würde man schon viele Interaktionen automatisch erfassen können, nur bei den gleichzeitigen Interaktionen müsste noch von Hand nachcodiert werden.

5.5.3.2 Energie

Ein weiteres System wird parallel zu dieser Arbeit getestet. Auch hier ist einer der Ansätze eine möglichst automatisierte Auswertung zu erzeugen. In dieser Studie wird untersucht ob das Eingabegerät Einfluss auf das *spatial memory* hat. Jeder der Probanden trug ein Myo Armband, wobei sämtliche Daten während dem Verlauf der Studie aufgezeichnet und anschließend als CSV gespeichert wurden. Die Aufgabe die die Probanden während der Studie lösen mussten, war ein Memory Task. Bei dieser Aufgabe wird ein vorgegebenes Bild auf einer Landschaft die größer ist als das Display durch horizontales und vertikales pannen gesucht (siehe Abbildung 50). Dieser Schritt wiederholt sich dann immer und immer wieder. Die Bilder auf der Landschaft sind immer an der gleichen Position und auf diese Weise wird das spatial memory trainiert. Anschließend mussten die Probanden auf einer leeren Landschaft die Positionen der Bilder zuordnen. Über den Abstand von der ursprünglichen Position zu der tatsächlichen, kann die Qualität des spatial

memory abgelesen werden. Für das Bewegen der Landschaft wurden drei unterschiedliche Geräte verwendet:

1. Playstation Move Controller
2. TouchPad (Logitech T650)
3. Touch in Form von einem touchsensitiven Display

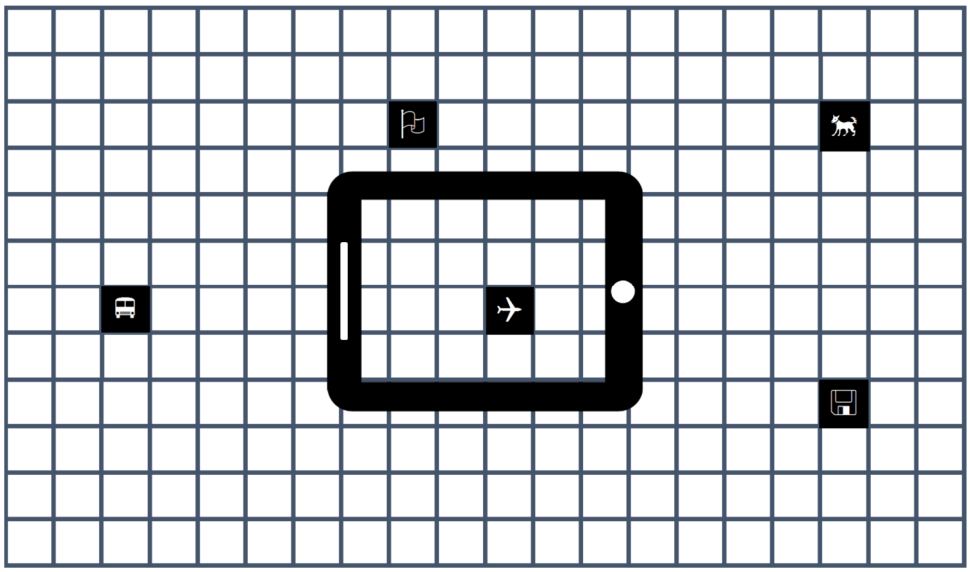


Abbildung 50: Memory Studie

Um die Hypothese, „ob die der Bewegung Einfluss auf das Ergebnis der Studie hat“ zu überprüfen wurde unter andrem versucht ein Maß der Bewegung zu ermitteln. Dieses Maß ist das *Energy Level*. Es wird wie folgt berechnet:

$$energy = \sum_{i=1}^{window_{lenght}/2} magnitude^2$$

Dazu müssen die Daten erst im Bereich von 0.1-20Hz gefiltert werden. Anschließend wird aus dem Daten die FFT Koeffizienten bestimmt, die in der Formel die Magnitude bilden (Tapia 2008).

Mit Midas lassen sich die einzelnen Bestandteile schnell in einer Pipeline abbilden. Was dafür gefehlt hat, war nur noch eine Funktion die die CSV Daten in die Pipeline übergibt und auch wieder Daten speichern kann. Deshalb wurden die zwei Funktionen in jeweils eine Node überführt. Damit sieht die Pipeline so aus wie in Abbildung 51 dargestellt. Nachdem die Rohdaten eingelesen worden sind, müssen sie normalisiert werden, damit sie besser vergleichbar sind. Anschließend werden immer 30 Einheiten jeweils einer Achse eines Sensors gepuffert. Das bedeutet immer etwa eine halbe Sekunde Daten werden zusammengefasst. Anschließend werden sie Bereich von 0.1-20hz mit einem *Bandpass* gefiltert. Dieser Filter funktioniert auch nur, wenn ein entsprechender Bereich gefiltert wird. Deswegen werden die Daten im Vorfeld gepuffert. Gleiches gilt für die Anschließende FFT Transformation. Aus den FFT Koeffizienten wird dann gemäß der oben aufgeführten Formel die Energie berechnet und mit der CSV Output Node als CSV gespeichert. Mit Hilfe dieser Daten

kann dann festgestellt werden, wann sich eine Person wie stark bewegt hat. Ob der Energie Level nun korreliert mit dem Spatial Memory kann leider noch nicht gesagt werden, denn das Ergebnis dieser Studie steht noch aus.

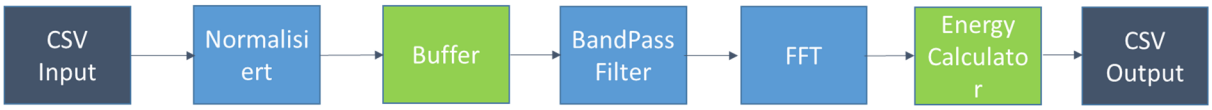


Abbildung 51: Pipeline zum Berechnen des Energie Levels

Wie gezeigt wurde, kann Midas recht flexibel eingesetzt werden. Diesen Vorteil bietet der Node Designer, und zeigt abermals, dass die aufgestellten Anforderungen mit diesem Konzept mehr als erfüllt sind. Um zu zeigen, dass auch weitere Szenarien mit Midas durchaus denkbar sind, gilt es eine Möglichkeit zu finden, mit der das gesamte Potential in einem Modell zusammengefasst wird. Dies wird in dem folgenden Abschnitt versucht.

5.6 Gesten State Model

Bill Buxton hat 1990 ein Paper vorgestellt, in dem er untersucht hat, welche States eine Maus- und eine Toucheingabe haben. Er war der Meinung, dass dies eine wichtige Komponente beim Entwerfen von Systemen ist, denn nur auf diese Weise kann der Entwickler überprüfen, ob das Eingabegerät zu seiner Anwendung passt. Das vorgestellte *State Model* für die Maus sah wie folgt aus:

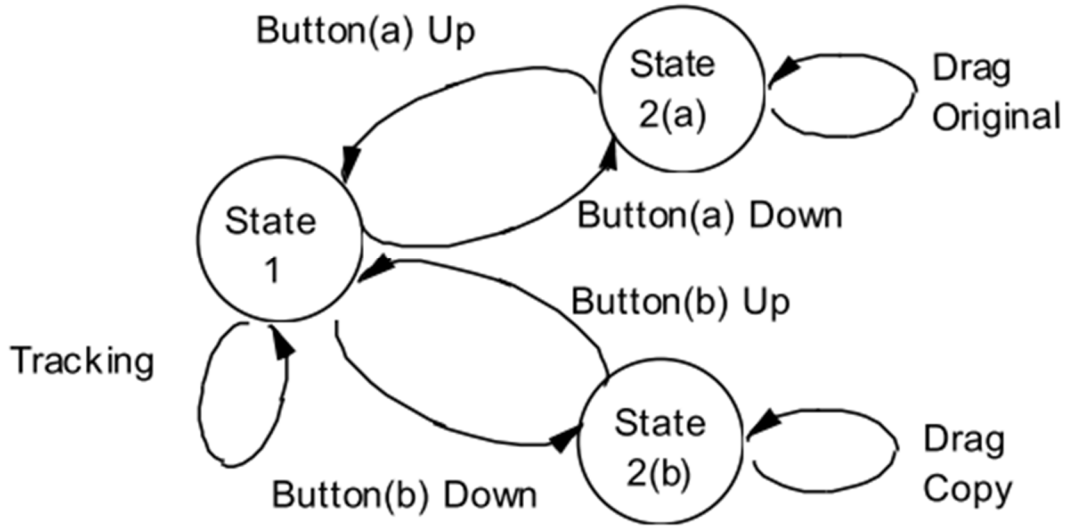


Abbildung 52: State Model der Multibutton Maus (Buxton 1990)

Bei diesem Modell befindet sich die Maus in State 1, das heißt, die Anwendung weiß ganz genau wo die Maus ist und kann jede Bewegung tracken. Drückt der User nun einen Button, wechselt er den State der Maus in den zweiten State. Dabei ist dieser State abhängig davon, welcher Button

gedrückt wurde. So führt das Drücken von Button a zu einem Verschieben der Originaldatei, während das Drücken des Buttons b nur eine Kopie der Datei verschiebt. Lässt der Anwender die Taste los, wechselt die Maus wieder in den State 1. Ein bisschen anders sieht das State Model für eine Toucheingabe aus:

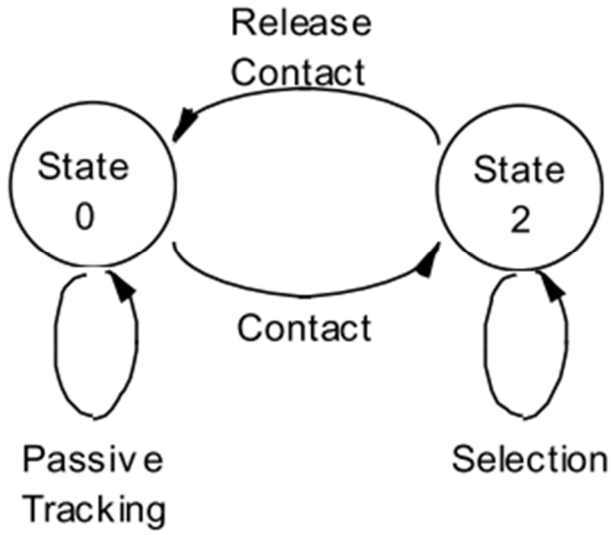


Abbildung 53 State Model für Touchinput (Buxton 1990)

Wenn der Finger den Bildschirm nicht berührt, dann kann die Anwendung auch nicht entscheiden, wo genau sich der Finger befindet. Buxton bezeichnetet dies als State 0 und die Aktion, die in diesem State durchgeführt wird ist *passives tracking*. Das heißt, die Anwendung versucht zu messen wo sich der Finger befindet. Da aber in diesem State kein wirkliches tracking vorliegt, bezeichnete Buxton diesen als State 0 und nicht wie bei der Maus als State 1. Berührt der Anwender mit der Hand den Bildschirm, dann wird in State 2 gewechselt. Dieser State ist genauso wie bei der Maus der State, bei dem eine Aktion durchgeführt wird. In diesem Fall kann es die Selektion, aber auch das Dragging sein. Was Buxton damals zwar schon geahnt hat ist, dass Gesten eine Rolle spielen. Er hat dafür aber noch keine State Model erzeugt, deswegen wird jetzt in dieser Arbeit versucht ein Modell für dieses Szenario zu entwickeln.

Gesten verhalten sich ähnlich zu dem Touch Modell, denn wenn der Anwender keine Geste durchgeführt hat, dann kann sie auch nicht erkannt werden. Das bedeutet es gibt einen State 0 in der das Modell am Anfang ist. Führt der Anwender dann eine Geste aus, kann diese Geste eine Aktion auslösen, die vergleichbar ist mit dem Drücken einer Tastenkombination. Dies ist z.B. bei Gvoice der Fall, denn wenn eine Geste durchgeführt wird, dann wird nur eine Aktion ausgeführt. Das State Model zu diesem Fall sieht wie folgt aus.

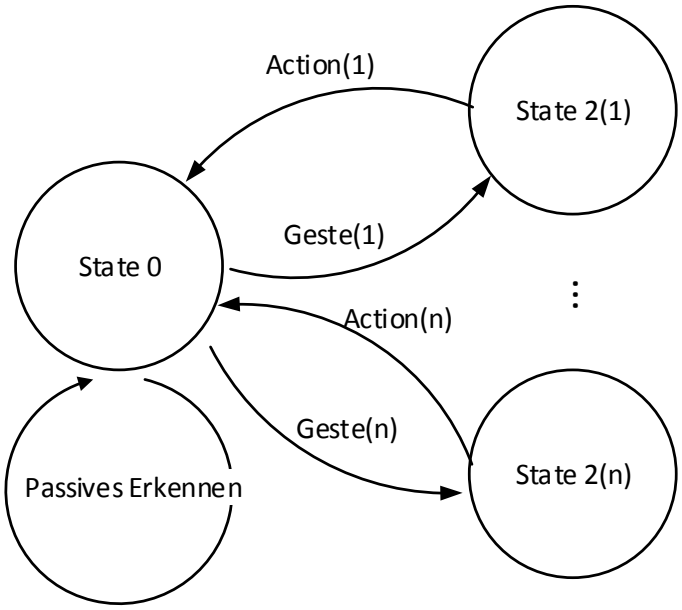


Abbildung 54: State Model für eine einfache Gestenerkennung

Da es eine unbestimmte Anzahl an Gesten gibt die durchgeführt werden können, gibt es n States 2. Sobald der State 2 erreicht wird, wechselt er wieder in den State 0. Auf diese Weise können auch Verkettungen von Gesten durchgeführt werden, denn sie enden immer in einer Aktion. Ein Beispiel für eine Verkettung von Gesten sind z.B. die Standard Gesten des Myo Armbands, die bereits mit dem Treiber erkannt werden. Denn wenn man eine Geste durchführen will, muss zunächst die Initialgeste (Mittelfinger und Daumen werden zweimal hintereinander schnell berührt) durchgeführt werden. Erst nach dieser Geste werden andere Gesten erkannt, dieses Verfahren stellt sicher, dass Gesten nicht fehlerhaft durchgeführt werden. Um dies mit Hilfe eines Modells abzubilden wurde Folgendes entwickelt:

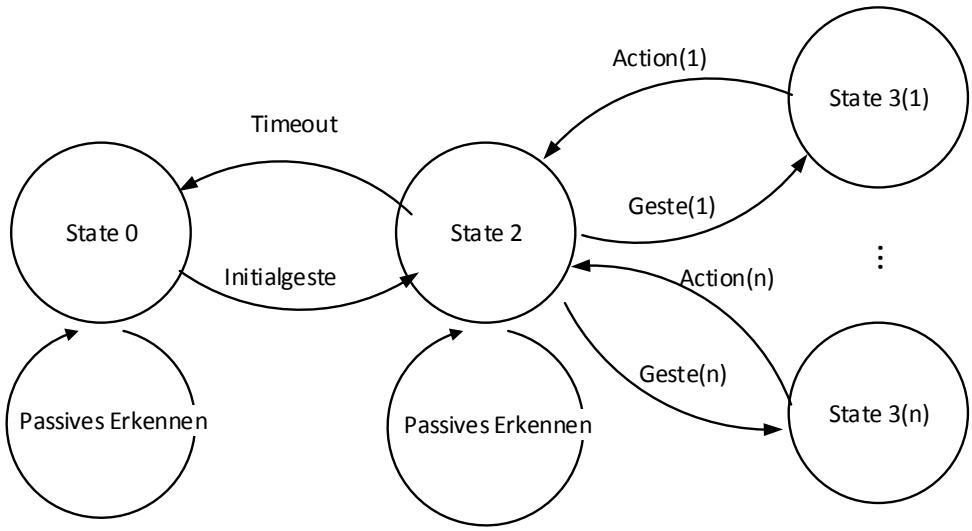


Abbildung 55: Verkettete Gesten

Mit der Initialgeste wird in den nächsten State gewechselt. Von dort aus kann dann eine weitere Geste durchgeführt werden, oder aber z.B. nach einer Zeit wird dann wieder in den State 0

gewechselt. Beim Myo Treiber ist es so, dass nachdem eine Geste durchgeführt wurde, wieder in den State 2 gewechselt wird und weitere Gesten durchgeführt wurden, denkbar wäre aber auch das nach der Aktion der State 0 wieder erreicht wird.

Ein weiteres Szenario das mit dem Myo Armband möglich ist, wäre *Force Touch*[11] nachzubauen. Bei *Force Touch* entscheidet der Druck mit dem der Touch ausgeführt wurde darüber, was für eine Aktion durchgeführt wird. Mit dieser Funktion erweitert sich auch das Touch State Model wie folgt:

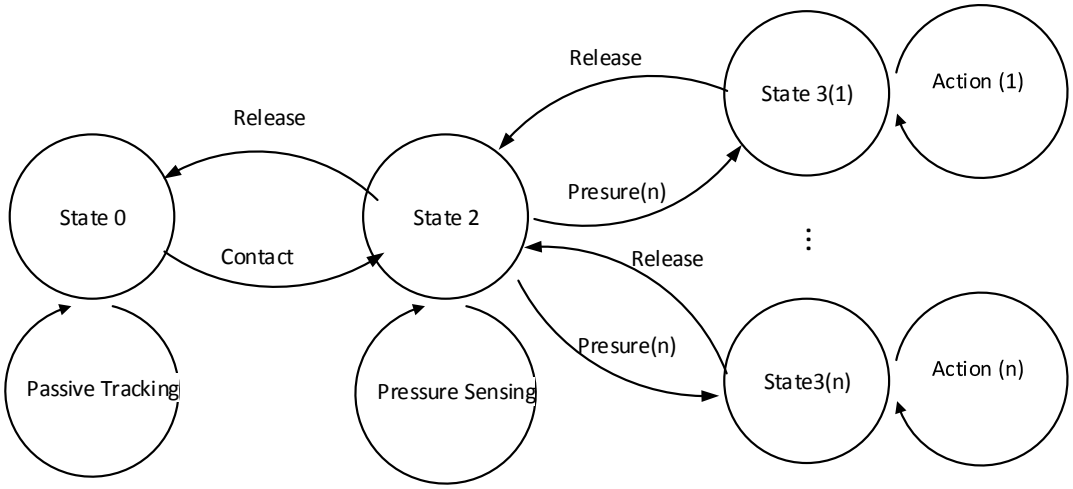


Abbildung 56: Force Touch State Model mit Myo

Im State 2 wird der Druck gemessen, der für das Touchen benötigt wird. Über diesen Druck entscheidet sich, in welchen State 3 gewechselt wird. In diesem State kann dann der Anwender seine Aktion durchführen, solange er den Finger auf dem Display hat. Sobald er dieses nicht mehr berührt, wechselt das Modell von State 3 zu State 2 und von dort dann direkt zu State 0. Man könnte also eigentlich auch von State 3 direkt in den State 0 wechseln.

Natürlich spielt Druck nicht nur bei Touch eine Rolle, sondern kann auch bei Gesten angewendet werden. Das bedeutet, wenn der Anwender eine Faust macht, dann kann er dies mit unterschiedlicher Stärke tun. Dieses Verhalten ist bereits in Abbildung 54 modelliert, denn man kann auch jede dieser Druckstufen als eigene Gesten behandeln.

Bis jetzt sind wir davon ausgegangen, dass Gesten immer nur eine Aktion auslösen und nicht gehalten werden. Diesen Fall des Haltens einer Geste gibt es aber auch, denn wenn der Anwender eine Faust macht dann kann er diese bewegen und halten solange er will. Dies geht aber nicht bei jeder Geste, denn ein Schnipsen kann nicht gehalten werden, sondern nur beliebig oft wiederholt werden. Mit haltenden Gesten sind dann wieder ganz andere Gesten möglich, wie das folgende State Model zeigt:

[11] https://en.wikipedia.org/wiki/Force_Touch

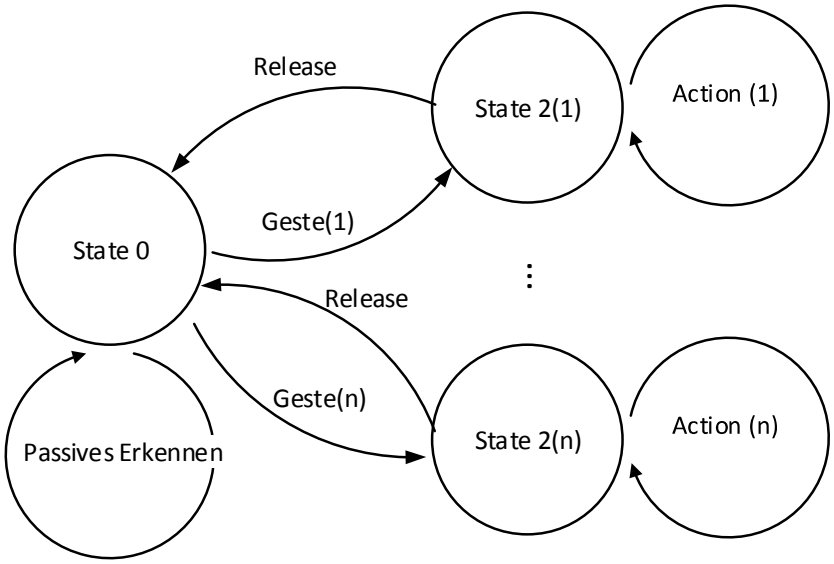


Abbildung 57: State Model für haltende Gesten

Durch das Halten kann, wie bei Touch, eine Aktion über das gesamte Halten der Geste durchgeführt werden. Wie wir gesehen haben, gibt es nicht nur ein Modell, das für die Verwendung bei Gesten geeignet ist, deswegen muss man genau sehen, welche Gesten erkannt werden können und welche Systeme verwendet werden. Was diese Systeme zeigen ist aber auch, dass sie bestehende Eingaben erweitern können. Somit besteht die Möglichkeit Toucheingaben dahingehend zu erweitern, dass Gesten der nicht-touch Hand den Input beeinflussen. So könnte z.B. die rechte Hand für Toucheingaben verwendet werden, während die linke Hand Zahlen zeigt und so Textfelder mit der Zahl ausgefüllt werden. Auf diese Weise lassen sich unzählige neue Szenarien ergründen.

6 Evaluation

Nachdem die Entwicklung abgeschlossen ist, geht es nun in die letzte Phase des Lifecycles, bevor er wieder von vorne beginnt. In dieser Evaluationsphase wird die geschriebene Anwendung eingehend drauf geprüft, ob sie die angestrebten Ziele auch erreichen kann.

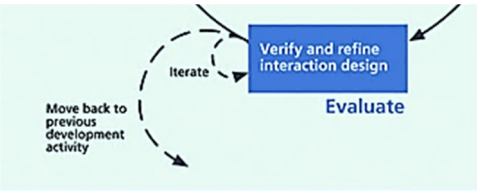


Abbildung 58: Evaluationsphase (Hartson & Pyla 2012)

In diesem Abschnitt werden zu Evaluationszwecken die zwei implementierten Systeme mit Benutzern getestet, zum einen *Midas Touch* und zum anderen *Govice*.

6.1 Studienablauf

In beiden Studien soll hauptsächlich die Qualität der zugrundeliegenden Pipeline getestet werden. Aus diesem Grund wurde eine technische Evaluation gewählt. Im Falle von *Midas Touch* gibt es entsprechende Richtlinien, die eingehalten werden sollten.

6.1.1 *Midas Touch*

Zu diesen Richtlinien zählen unteranderem die Präzision und die Performanz. Um beide Parameter zu bestimmen wurde der folgende Testlauf vorbereitet.

1. Die Probanden wurden gefragt, ob sie Rechts- oder Linkshänder sind. Entsprechend der Aussage wurde dann das Myo Armband angebracht.
2. Es wurde geprüft, ob das Armband richtig am Arm positioniert war und die richtigen Werte erfassen kann. Dazu sollten die Probanden eine Box-Geste durchführen.
3. Midas Touch wurde den Anwendern vorgestellt, sodass sie ein mentales Modell der Software aufbauen konnten.
4. In diesem Schritt wurden die Probanden angewiesen das System mit einem Finger zu trainieren. Dazu touchten sie an verschiedenen Stellen auf dem Display.
5. Nach etwa 10 Trainingsinstanzen, wurden die Probanden angewiesen eine andere Geste für etwa eine Minute lang durchzuführen. Zum Beispiel in die Hände klatschen, oder malen. Dieser Schritt dient als Ablenkung für das muscle memory, um verschiedene Gesten zu erhalten.
6. Nach der Ablenkung sollten die User weiter trainieren, bis etwa 20-30 Trainingsset angelegt waren.
7. Anschließend wurden die Schwellwerte für die Gesten vom Studienleiter festgelegt, bis die Erkennung zufriedenstellend war.

8. Mit dem letzten Schritt war die Studie soweit vorbereitet, dass die Parameter abgeprüft werden konnten. Dazu sollten die Probanden 20-mal mit dem trainierten Finger touchen. Die Ergebnisse wurden dabei mitgeloggt.
9. Danach sollten sie 20-mal mit der anderen Hand touchen.
10. Wurde der letzte Schritt beendet, sollten sie wieder 20-mal mit der Hand touchen, mit der sie das System trainiert haben. Jetzt sollten sie jedoch mit einem anderen Finger der Hand touchen.
11. Im letzten Teil wurden die Probanden angewiesen nochmals 20-mal mit dem Finger zu touchen, mit dem sie das System trainiert haben, allerdings sollten sie 5 bis 10 Sekunden warten, bis sie mit der nächsten Touchinteraktion weitermachten. Auch hier ging es wieder darum, das muscle memory zu vergessen.
12. Abschließend wurde der Proband verabschiedet.

Die Dauer der oben bestehenden Studien wurde auf etwa 10 Minuten geschätzt. Aus diesem Grund wurde auch keine Vergütung ausgeschüttet. Die Anzahl der Teilnehmer sollte um die 6 Probanden liegen. Die geforderte Antwortzeit von Midas nach einer Interaktion sollte unter einer Sekunde liegen.

Ganz anders sieht der Studienaufbau bei der Evaluation von Gvoice aus.

6.1.2 *Gvoice*

Auch bei Gvoice sollte die Performanz und Präzision überprüft werden, allerdings nicht nach einem vorgegebenen Gestenpool. Das Ziel dieser Evaluation war außerdem herauszufinden, welche Gesten gut funktionieren und welche eher schlecht. Dazu sollten die Probanden möglichst viele und unterschiedliche Gesten trainieren. Hierfür wurde der folgende Ablauf befolgt:

1. Der Proband wurde begrüßt und es wurde ihm der Inhalt der Studie erläutert.
2. Anschließend wurde er aufgefordert eine Einverständniserklärung zu unterzeichnen mit dem der Proband der Auswertung und der Aufnahme der Studie zustimmte.
3. Der Proband musste einen Vorfragebogen beantworten, in dem nicht nur das Alter und das Studienfach abgefragten wurden, sondern auch die Kompetenz im Umgang mit PCs, Wearables und Gestenerkennung. Daneben wurde auch gefragt, ob der Proband regelmäßig Sport ausübt. Es wurde auch sichergestellt, dass der Proband keine körperlichen Einschränkungen besitzt.
4. Anschließend wurde das Myo am starken Arm des Probanden angelegt und die korrekte Position mittels einer Box-Geste verifiziert.
5. In diesem Schritt wurde der Proband mit den Funktionen des Myo Armbandes vertraut gemacht, um ein mentales Modell vom Bandes aufzubauen. Die einzelnen Signale die das Myo aufzeichnet wurden erläutert.
6. Es wurde besprochen, wie die Gestenerkennung grob arbeitet und dass ein Computer immer ein Abbild der Gesten braucht um sie zu erkennen.
7. Anschließend wurde die erste Geste trainiert. Als Beispielgeste wurde schnipsen gewählt, konnte der Proband nicht schnipsen, wurde klatschen verwendet. So wurden drei Trainigsets angelegt.

8. Anschließend wurden die Parameter vom Studienleiter eingestellt und eine Audiodatei geladen, die den Namen der Geste abspielte. So konnte immer überprüft werden wann die Geste ausgeführt wurde. Nach dieser Einstellung sollte der Proband nochmal die Geste wiederholen, um zu zeigen, dass alles funktioniert.
9. Im Anschluss durfte der Proband eine halbe Stunde lang sich selbst Gesten überlegen und trainieren. Ihm wurde allerdings vorher noch kleinere Hinweise zur Qualität der Erkennung gegeben. Auch wurden sie darauf hingewiesen, dass sie alle Gegenstände im Raum benutzen können und sich frei bewegen dürfen. Nach jedem Trainieren einer Geste, wurde der Name der Geste als Audio Datei aufgenommen und dann abgespielt, wenn die Geste erkannt wurde.
10. Nach der halben Stunde werden die trainierten Gesten nochmals wiederholt und gegebenenfalls erneut trainiert.
11. Abschließend wurde geprüft, welche der Gesten die Probanden wiederholen konnten. Dabei galt die Geste als nicht wiederholbar, wenn sie nach dreimaligen Versuch nicht erkannt wurde.
12. In diesem Stadium durften die Probanden das Armband ablegen und es wurde eine Frage zu dem Tragekomfort gestellt.
13. Anschließend sollten der Anwender noch einen AttrakDiff (Hassenzahl et al. 2008) Fragebogen beantworten.
14. Nach dem Fragebogen musste der Proband den Erhalt von 8€ quittieren und wurde anschließend verabschiedet.

Die Bedienung der Software wurde vom Studienleiter durchgeführt, um den Probanden zu ermöglichen, dass dieser sich voll und ganz auf das Ausführen und Finden von Gesten konzertieren konnte. Außerdem wurde die Studie von zwei Kameras und zwei Mikrophonen aufgezeichnet. Für diese Studie wurde eine Teilnehmerzahl von 8 Probanden angestrebt.

Der Aufbau der Studie sah wie folgt aus:

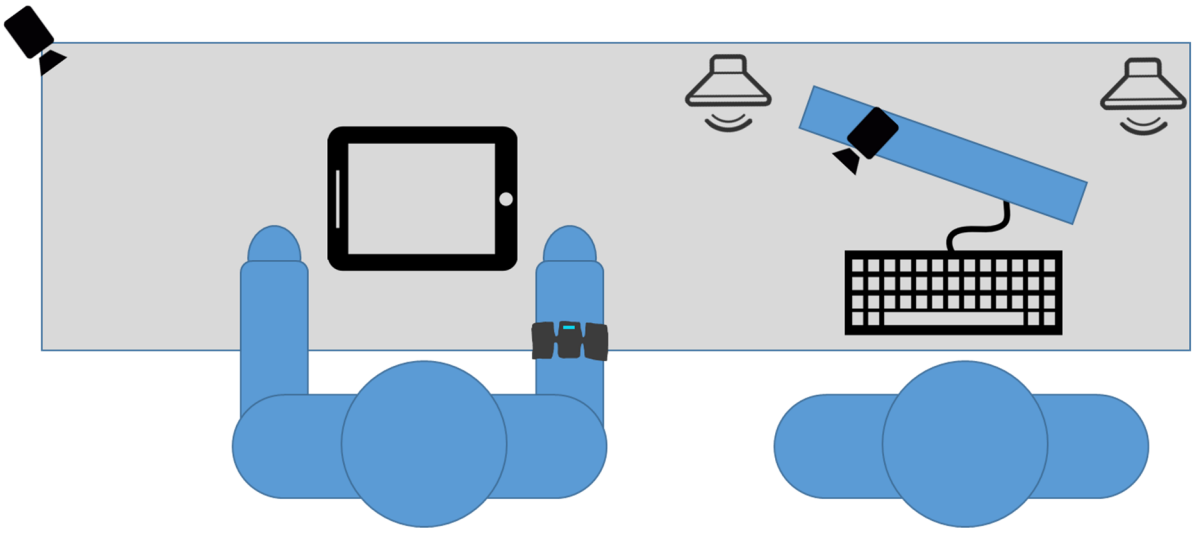


Abbildung 59: Studienaufbau Gvoice

6.2 Durchführung

Um den geplanten Studienablauf zu testen, wurde mit zwei Probanden eine Vorstudie durchgeführt. Diese Vorstudie bestätigte das Studiendesign dahingehend, dass die Teilnehmer kreativ genug sind sich Gesten auszudenken und dass es besser ist, wenn die Probanden sich auf die Gesten konzentrieren und nicht mit der Bedingung der Anwendung beschäftigt ist. Anschließend wurden die Studien wie im Ablauf beschrieben durchgeführt. Bei der ersten Midas Touch Studie wurden 6 Teilnehmer und Teilnehmerinnen getestet. Die zweite Studie zu Gvoice hatte 9 Teilnehmer und Teilnehmerinnen. Wie bereits geschätzt, dauerte die Midas Touch Studie etwa 10 Minuten, während die Gvoice Studie je nach Teilnehmer zwischen einer Stunde und 50 Minuten dauerte. Die ersten Fragebögen von Gvoice, die Fragen zur Person beinhalteten, wurden vom Studienleiter in einem Interview ausgefüllt. Auf diese Weise konnte der Studienleiter mehrere Dinge besser erklären, vor allem was *Wearables* sind. All diese Informationen wurden mit einem Google Forms Fragebogen erhoben. Dies hat den Vorteil, dass die Ergebnisse schon digital vorliegen. Der AttrakDiff Fragebogen wurde über das Portal eSURVEY[12] durchgeführt und ausgewertet. Auch dieses Portal ermöglicht eine digitale Auswertung der Daten. Dieser Fragebogen wurde in der Regel von dem Probanden selbst ausgefüllt.

[12] https://esurvey.uid.com

6.3 Ergebnis der Evaluation

Nachdem die Studie durchgeführt wurde, sind die einzelnen Studien ausgewertet worden. Die Resultate sind in den folgenden Abschnitten beschrieben.

6.3.1 *Midas Touch Ergebnisse*

Die angestrebte Antwortzeit von unter einer Sekunde war bei allen Probanden geben. Damit ist schon eine der Anforderungen erfüllt. Die Probanden erreichten zudem die folgenden Erkennungsraten:

Getoucht mit	Erkannt	Nicht erkannt	Rate in %
Trainiertem Finger	114	6	95
Nicht trainierter Hand	0	120	100
Trainierte Hand aber anderem Finger	103	17	85,83
Trainierte Finger zwischendrin Pause von 5-10 Sekunden	101	19	84,17

Tabelle 5: Ergebnis – die grün markierten Felder sind die richtig erkannten Touches, die roten die fälschlicherweise erkannten Touches (false positives).

Die Erkennung mit dem gleichen Finger liegt mit 95% höher als die geforderten 80%, selbst wenn ein anderer Finger verwendet wird (86%), oder eine Pause zwischen den Touches eingebaut wird (84%).

6.3.2 *Gvoice*

6.3.2.1 Auswertung Vorbefragung

Von den 9 Probanden waren alle Studenten, davon 5 weiblich und 4 männlich. Das Alter lag zwischen 18 und 31. Um herauszufinden, ob die Probanden bestimmte Bewegungsabläufe routiniert trainierten, wurde nach ihren sportlichen Aktivitäten befragt. Dabei übte nur ein Teilnehmer keinen Sport regelmäßig aus. Die Sportarten reichten von Ballsportarten wie Fußball, Volleyball, Handball und Tennis, bis hin zu Fitness, joggen und Radfahren, einer der Teilnehmer übte Yoga aus. Bei der Vertrautheit im Umgang mit dem PC, konnten die Probanden auf einer Likert Skala (Likert 1932) (1 = sehr vertraut, 5 = noch nie genutzt) auswählen, wie vertraut sie damit sind. Dabei waren die meisten sehr vertraut mit dem Umgang von PCs (Durchschnitt 1,78, Standartabweichung 0,79). Anders sah das aus bei der Vertrautheit mit Gestensteuerung, dort waren viele dabei, die noch nie oder selten eine Gestensteuerung verwendet haben (Durchschnitt 2,78, Standartabweichung 1,47). Die am meisten verwendete Gestensteuerung war die Nintendo Wii und die Microsoft Kinect, gefolgt von der Playstation Move[13]. Mit der Verwendung von Wearables waren sie dagegen noch unvertrauter. Nur lediglich zwei der Probanden hatten schon mal Wearables getestet (Durchschnitt 4,33 Standartabweichung 1,33). Einer der Probanden hatte

[13] https://en.wikipedia.org/wiki/PlayStation_Move

bereits das Myo Armband im Vorfeld getestet. Des Weiteren hatten die Probanden alle keine Einschränkung am Arm oder der Hand, somit erfüllten alle Probanden die Voraussetzung um an dieser Studie teilzunehmen. Alle der teilnehmenden Personen waren zudem Rechtshänder.

6.3.2.2 Ergebnis der Gestenerkennung

In der halben Stunde, die die Probanden Zeit hatten Gesten mit Midas zu trainieren, trainierten sie zwischen 10 und 16 unterschiedliche Gesten (Durchschnitt 12,22; Standardabweichung 2,25). Da jeder Proband frei war in der Auswahl der Gesten, wurde auch sehr viele unterschiedliche Gesten trainiert. Es gab allerdings auch einige Gesten die mehrfach von unterschiedlichen Personen trainiert wurden:

Name der Geste	**Häufigkeit**	**Erkennungsrate**
Klatschen	7	85,71%
Winken	5	80%
Werfen	5	60%
Boxen	5	60%
Daumen hoch	3	100%
Drehen/Schrauben	2	50%
Schwimmen	2	50%
Tür aufmachen	2	100%
Recken	2	100%
Schreiben	2	50%
Brille auf-/absetzten	2	100%
Durch die Haare streichen	2	100%
Liegestütz	2	100%
Prellen	2	50%
Tennisaufschlag	2	100%
Blättern	2	100%
Rühren	2	100%
Zeigen	2	100%

Tabelle 6: Häufig trainierte Gesten

Vor allem Winken und Klatschen kamen bei der Hälfte der Probanden vor. Interessanterweise hat jeder die Geste anders durchgeführt. Manche haben gleich mehrmals geklatscht, andere mit der hohlen Hand und andere mit der flachen Hand. Genauso unterschiedlich fällt auch die Erkennung der Geste aus. Bei manchen wird die Klatschgeste am Ende nicht mehr erkannt bei anderen sofort. Die Erkennungsrate bei allen Gesten liegt im Schnitt bei 85,78% (Standardabweichung 10,76%; Max 100% Min 63,93%) mit einer durchschnittlichen Qualität von 1,67 benötigten Wiederholungen (Standardabweichung 0,32). Das bedeutet, dass eine Geste ungefähr 2-mal durchgeführt werden muss bis sie erkannt wird.

Um genauer aufzuschlüsseln welche Gesten gut funktionieren, wurden die trainierten Gesten entsprechenden Klassen zugeordnet:

1. **Hand- oder Fingergesten:** In dieser Klasse sind alle Gesten, die hauptsächlich mit der Hand oder den Fingern durchgeführt werden. Eine Beispielgeste dafür wäre die „Ok“ Geste, oder die „Faust“ Geste.
2. **Armgesten:** Alle Gesten, die mit dem Arm und der Hand durchgeführt werden, fallen unter diese Kategorie. (z.B. winken)
3. **Ganzkörpergeste:** Gesten oder Bewegungen, die mit dem ganzen Körper durchgeführt werden fallen unter diese Kategorie, so z.B. das Springen.

Diese drei Hauptkategorie teilen alle Gesten eindeutig zu, das heißt es kann ganz genau betrachtet werden, welche dieser Kategorien bei den Probanden wie abgeschnitten hat. Um weitere Eigenschaften zu finden, wird zusätzlich in die folgenden Kategorien gruppiert.

1. **Kraftintensive Geste:** Bei Bewegungen oder Gesten die mit sehr viel Kraft durchgeführt werden, wie z.B. Liegestütz, werden dieser Kategorie zugeordnet.
2. **Wenig Kraft Geste:** Bei Bewegungen oder Gesten die mit wenig Kraft durchgeführt werden, wie z.B. Winken, werden dieser Kategorie zugeordnet.

Anschließend wird nochmals unterteilt zwischen:

1. **Emblem**s: Unter dieser Kategorie fallen Gesten die in Gesprochene Sprache übersetzt werden können wie die „Ok“ Geste (Kipp 2005).
2. **Beats:** Zu dieser Kategorie gehören Gesten die immer wieder widerholt werden, z.B. das „Schnipsen“ (Kipp 2005).
3. **Sportbewegung/-geste:** Da viele Probanden Bewegungen gemacht haben, die zu einer Sportübung gehören, werden mit dieser Kategorie diese Bewegungen einzeln betrachtet.
4. **Interaktion mit Gegenständen:** Es kam auch häufiger vor, dass Gesten mit Gegenständen durchgeführt wurden. So benutzen viele beim Werfen einen Gegenstand, um das Gefühl und die Bewegung zu verbessern.
5. **Sonstiges:** Hierunter fallen alle Gesten, die sich nicht in die 4 Unterkategorien zuordnen lassen.

Klasse	Häufigkeit bei den Probanden	Gesamte Häufigkeit	Erkennungsrate in %	Durchschnittliche Wiederholung bis zur Erkennung
Hand- oder Fingergeste	9	30	86,67	1,68 (SD 0,59)
Armgesten	9	67	83,58	1,68 (SD 0,43)
Ganzkörper	7	14	92,86	1,38 (SD 0,45)
Viel Kraft	9	44	81,82	1,71 (SD 0,63)
Wenig Kraft	9	67	88,06	1,64 (SD 0,44)
Emblems	6	15	93,33	1,95 (SD 0,64)
Beats	9	29	82,76	1,84 (SD 0,78)
Sportgeste	7	24	70,83	1,24 (SD 0,39)
Geste mit Gegenständen	9	37	89,19	1,63 (SD 0,58)
Sonstiges	8	23	91,30	2,01 (SD 0,55)

Tabelle 7: Ergebnis der Gvoice Studie

Betrachtet man die Aufteilung zwischen den einzelnen Geschlechtern, waren die Frauen ein bisschen besser als die Männer. Die Erkennungsrate für die weiblichen Teilnehmer liegt bei 85,5% (SD 7,810), während die bei den Männern bei 81,13% (SD 10,97) liegt. Auch was die Qualität anbelangt waren die Frauen besser, sie mussten die Geste im Schnitt 1,75 (SD 0,39) mal wiederholen, bis sie erkannt wurde, während die Männer 1,77 (SD 0,39) Mal brauchten.

Nachdem die Teilnehmer den aktiven Part beendet hatten, sollten sie einen AttrakDiff Fragebogen beantworten. Das Ergebnis dieses Fragebogens wird im folgenden Abschnitt beschrieben.

6.3.2.3 *Ergebnis des AttrakDiff Fragebogens*

Bei dem AttrakDiff Fragebogen wurden die Probanden angewiesen nur das Trainieren mit dem Myo, die Erkennung und die Erfahrung zu berücksichtigen. Die Software hingegen sollte von ihnen nicht bewertet werden, da sie diese nur indirekt bedient haben.

Der AttrakDiff Fragebogen lieferte die folgenden Ergebnisse:

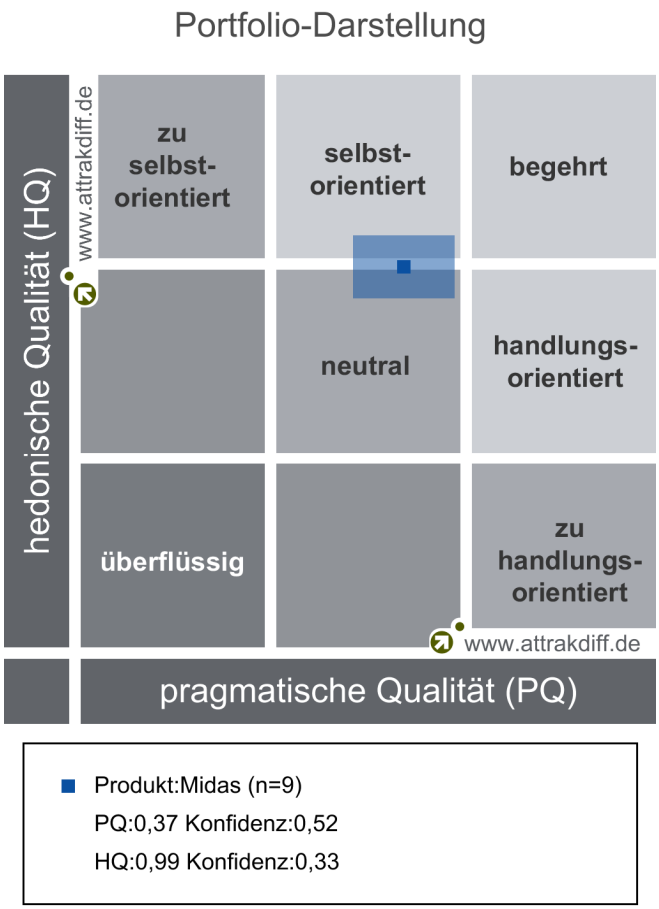


Abbildung 60: Ergebnisüberblick -Portfolio

Die Teilnehmer sahen die Nützlichkeit dieses System eher als neutral an. Dieses Ergebnis deckt sich auch mit den Beobachtungen, denn es wurde häufig am Ende der Studie gefragt warum und wofür man ein solches System benötigen könnte. Außerdem ist die Software noch nicht hundertprozentig zuverlässig und es kam zu Fehlinterpretationen. Während die pragmatische Qualität also eher neutral ausfiel, ergab die hedonistische Qualität ein positiveres Bild. Denn sie liegt über dem neutralen Bereich. Damit hat die Anwendung den Probanden Freunde und Spaß bereitet. Auch dieses Ergebnis spiegelt sich in den Beobachtungen wieder, denn viele Probanden sagten sowas wie „das Benutzen des Systems hat richtig Spaß gemacht“ oder aber „es war witzig so etwas mal ausprobieren zu dürfen“.

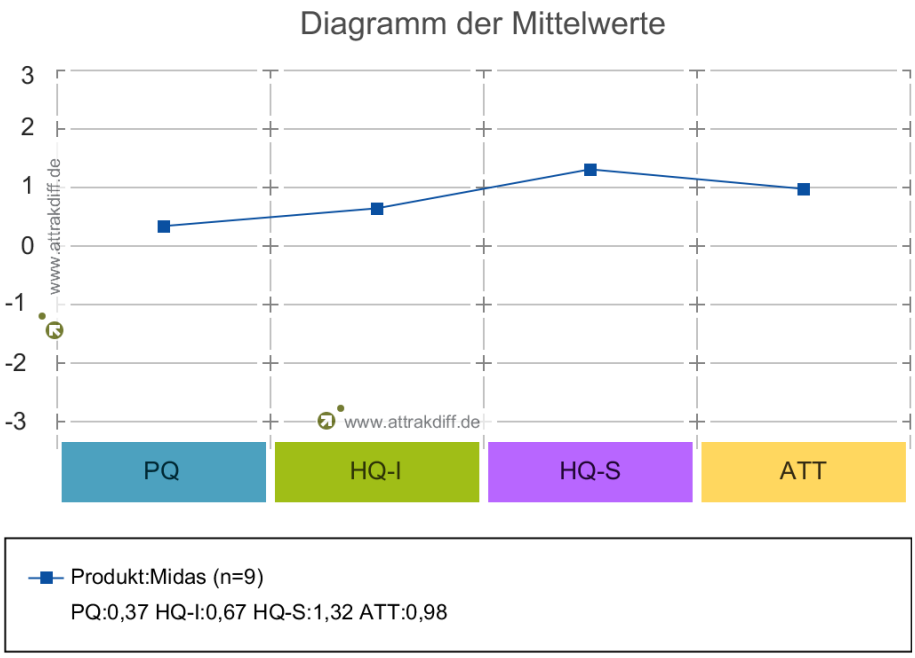


Abbildung 61: Diagramm der Mittelwerte. PQ steht für pragmatische Qualität also die Nützlichkeit, HQ-I steht für hedonistische Qualität Identität also wie gut konnten sich die Probanden mit dem System Identifizieren, HQ-S steht für hedonistische Qualität Stimulation, ATT steht für die Attraktivität.

Betrachtet man die hedonistische Qualität nochmal aufgeschlüsselt, stellt man fest, dass die Probanden durch Midas überdurchschnittlich gefesselt bzw. stimuliert wurden (signalisiert durch HQ-S siehe Abbildung 61), aber auch die gesamte Attraktivität ist recht hoch. Lediglich der Wert der Identifikation mit dem Produkt scheint nicht ganz so hoch zu sein (signalisiert durch HQ-I siehe Abbildung 61). Dies hängt wahrscheinlich damit zusammen, dass der Nutzen nicht eindeutig war. In der folgenden Grafik sind nochmal alle Werte beschrieben, die in dem Fragebogen abgeprüft worden sind.

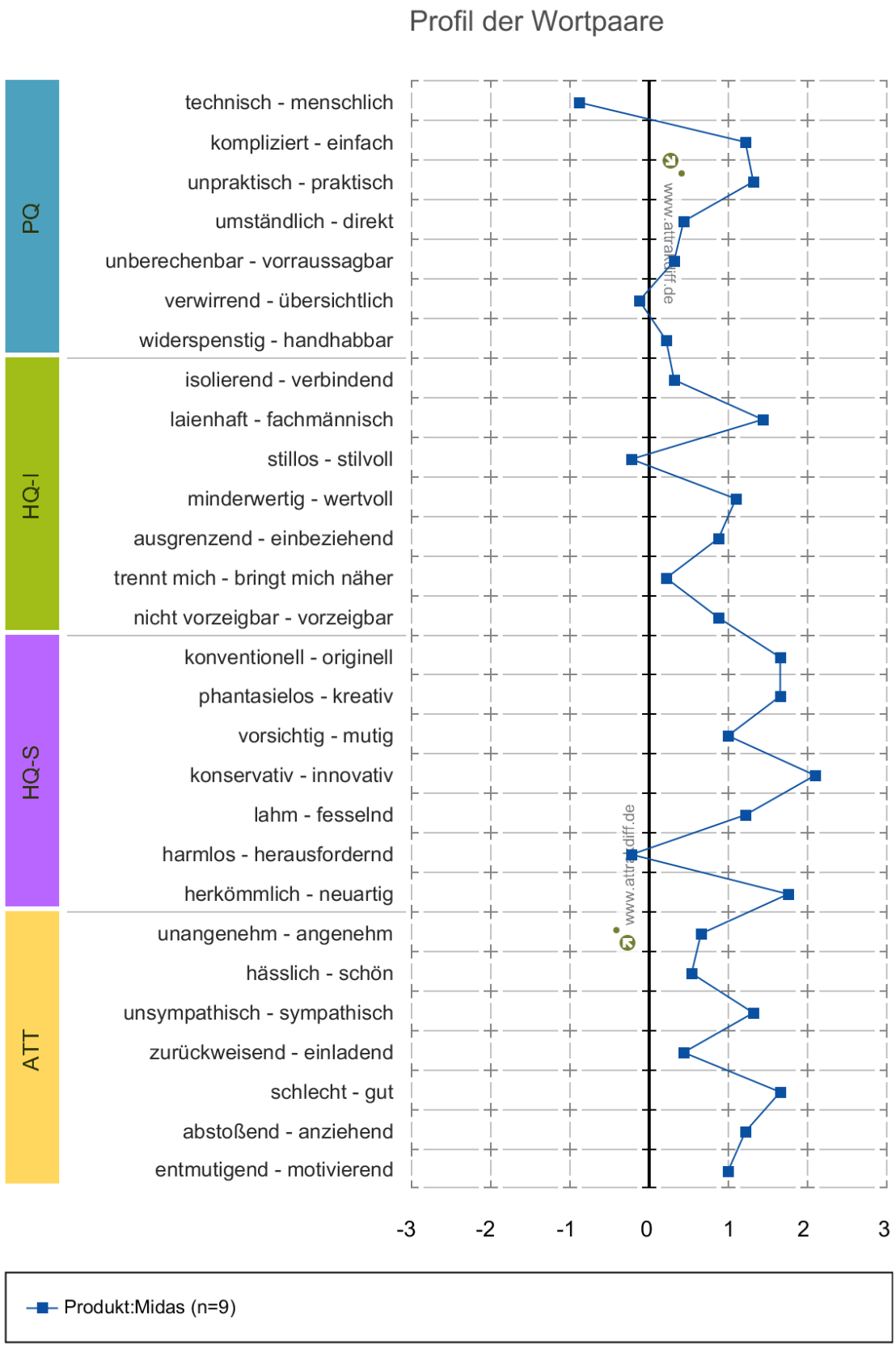


Abbildung 62: Profil der Wortpaare

7 Diskussion

Für das Szenario der Toucherkennung wurden klare Anforderungen an die Software gestellt (siehe Abschnitt 3.2). Diese ergaben sich zum einen aus der Literatur, und zum anderen aus der Praxis. Bei der Toucherkennung *Midas Touch*, sollte es eine Echtzeitanwendung sein, die in unter einer Sekunde eine Zuordnung berechnen kann. Gleichzeitig sollten die Ergebnisse mit einer Rate von mindestens 80% richtig zugeordnet werden. Die Untersuchung hat gezeigt, dass diese Werte eingehalten wurden und dass die Erkennungsrate sogar höher liegt als gefordert. Der beste Wert liegt im Schnitt bei 95% und zeigt, dass die Erkennung bei schnellen Touchbewegungen fast immer richtig funktioniert. Selbst wenn der Anwender einen anderen Finger nimmt, als den mit den er das System trainiert hat, liegt die Erkennungsrate immer noch bei respektablen 85,85%. Interessanterweise ist die Erkennung der Finger, mit entsprechender Pause, kleiner als die durchschnittlichen 95 %. Der Grund dafür ist, dass die Probanden diese Pause nutzten, um die Hand zurück an den Körper zu führen. Anschließend bewegten sie die Hand wieder auf das Display zu. Diese Geste wurde nicht trainiert, weswegen die Erkennung nicht ganz so gut funktioniert. Um die Erkennung in dem Bereich zu verbessern, sollte man diese Gesten mitaufnehmen. Vielleicht müsste man auch die Anwender über den Tag verteilt immer wieder trainieren lassen, um ein möglichst vielseitiges Set an Daten zu bekommen. Dies zeigt, dass Geste nicht gleich Geste ist, und dass zwischen den Gesten eine hohe Varianz besteht, die durch die Software gewährt werden muss. In dieser *Midas Touch* Studie haben die Teilnehmer um die 20 bis 30 Trainingstemplates unterschiedlicher Art angelegt.

Mit der *Gvoice* Studie sollte geprüft werden, ob es möglich ist nicht nur eine Geste zu trainieren und zu erkennen, sondern auch verschiedene Gesten zu unterscheiden. Dazu sollten auch deutlich weniger Trainingstemplates angelegt werden, nämlich im Schnitt nur 3 bis 5 pro Geste. Daneben sollte herausgefunden werden, welche Art von Gesten sich am Besten mit Midas und der Pipeline erkennen lassen. Die Studie zeigte, dass die Probanden die unterschiedlichsten Hand- und Armgesten mit Midas trainieren konnten und diese auch erkannt wurden. Allerdings hing diese Erkennung damit zusammen, wie genau sich die Probanden an die Ausführung der Geste erinnerten. So wurden Gesten, die im Alltag häufiger vorkommen und der Anwender gelernt hat sie immer gleich durchzuführen, besser erkannt als jene die weniger häufig benutzt werden. Auch wurden ausgeführte Gesten durch Gegenstände deutlich besser erkannt. Dennoch hat die Studie ergeben, dass sich beliebige Gesten mit einer Erkennungsrate von 85,78% richtig erkennen lassen und dass im Schnitt die Geste nur zweimal wiederholt werden musste bis sie erkannt wurde. Dieses Ergebnis liegt noch hinter dem der *Midas Touch* Studie. Der Grund dafür ist, dass nicht soviel Templates angelegt worden ist wie bei der Touchgeste (3-5 zu 20-30 Templates). Die Touchgeste ist allerdings auch keine Freihandgeste an sich, sondern ist eine Interaktion mit einem Gegenstand. Betrachtet man diese Erkennungsrate bei *Gvoice*, ist die Erkennung deutlich besser, mit fast 90 Prozent. Die Ursache für diesen Anstieg liegt darin, dass die Geste in gewisser Weise geführt wird und so die Möglichkeiten der Ausübung nicht so stark variieren. Denn bei den Freihandgesten führt der Probanden die Gesten immer auf unterschiedliche Weise aus, wenn zwischen dem Ausüben der Geste und dem Trainieren der Geste eine gewisse Zeit vergangen ist. Das bedeutet konkret, wenn ein Proband zum Beispiel schnipst, dann tut er das während dem Trainieren mit einer gewissen Stärke, in einer gewissen Höhe und Bewegung. Nach zwei bis zehn Minuten kann es dazu kommen,

dass er die Geste komplett anders durchführt. Am stärksten ist dieser Effekt beim der Geste „winken“ zu beobachten, denn hier variiert nicht nur die Höhe und die Stärke, sondern auch die Geschwindigkeit und die Anspannung. Für den Probanden ist das natürlich die gleiche Geste, wenn man diese dann jedoch erneut trainiert, stellt man einen sehr großen Unterschied fest. Dieser Effekt ist dem Muskelgedächtnis oder Motor Learning geschuldet, denn eine Geste wird bei kurzen zeitlichen Abständen immer auf die gleiche Art und Weise wiederholt. Nach einer Zeit ist diese Erinnerung dann nicht mehr so präzise vorhanden und es muss erst mehrmals getestet werden, bis diese Erinnerung wieder erzeugt ist (Schmidt & Lee 2011). Bei Interaktionen mit Gegenständen wird diese Variation minimiert, denn sie geben z.B. die Form der Geste vor. Am stärksten ist dies bei der Geste „Tür aufmachen“ zu beobachten, denn die Probanden müssen um die reale Tür zu öffnen, immer die gleiche Kraft aufwenden um den Türgriff herunterzudrücken und das Öffnen der Tür passiert auch immer im gleichen Winkel. Deswegen wird diese Geste in der Studie immer erkannt und auch beim ersten Mal detektiert. Durch die geringe Varianz dieser Geste, musste sie auch nur einmal trainiert werden. Ähnlich sieht das bei Gesten aus, die geübt sind, wie z.B. bei Zeichen, die als Sprachersatz fungieren, wie die „O.K.“ Geste. Hier liegt die Erkennungsrate im Bereich von 93,33%. Diese Gesten sind im Langzeitgedächtnis gespeichert und werden daher immer wieder ähnlich durchgeführt (Schmidt & Lee 2011). Ich habe allerdings auch erwartet, dass ein ähnlich hoher Wert bei Sportgesten erzieht wird, denn die Probanden haben in der Regel diese Bewegungen ebenso gelernt. Interessanterweise ist dies nicht der Fall, denn Gesten aus dieser Kategorie werden nur zu 70% erkannt. Betrachtet man die Gesten aus dieser Gruppe genau, stellt man fest, dass viele Probanden die Geste „Werfen“ verwendet haben. Diese Geste wird von den meisten Probanden ohne Gegenstand durchgeführt, das heißt sie führen diese Geste in der Luft aus. Jedes Mal, wenn die Geste so trainiert worden ist, kann sie nicht wiederholt werden. Wird dagegen ein Gegenstand verwendet steigt die Erkennungsrate. Bei den Probanden, die Handball spielen, wird die Geste sofort erkannt. Eine weitere Problemgeste scheint „Boxen“ zu sein, denn die Kraft- und Geschwindigkeitsvariation ist anscheinend zu hoch, zumal keiner der Probanden Boxsport praktiziert. Nimmt man bei den Sportgesten nur Gesten, die der Proband beherrschen sollte, liegt die Erkennungsrate nahezu bei 94%. Dementsprechend spielt das Lernen von Gesten eine entscheidende Rolle bei der Möglichkeit die Geste zu erkennen. Diese Annahme bestätigt sich auch, als ein Proband Musikgesten wie Klavier und Gitarre spielen trainiert. Denn diese werden auch ohne Problem detektiert. Es zeigt sich allerdings auch, dass wenn der gesamte Körper eingesetzt wird, die Erkennung besser ist. Der Grund dafür ist, dass die Gesten meist natürlicher sind und somit besser wiederholt werden können, denn sie sind natürlicherer Art. Ein Beispiel sind hier die Sportgesten wie „Liegestütz“ oder das „Ausstrecken“, die in jedem Fall immer erkannt wurden. Interessant ist auch, dass die Gestenerkennung bei Frauen besser funktioniert. Dies ist eventuell so zu erklären, dass Frauen dünnere Arme haben und somit die Muskelkontraktion besser gemessen werden kann. Ein weiterer Grund könnte sein das die Arme von Männern meist behaart sind und somit der Kontakt der Sensoren zur Haut nicht ideal ist. Doch kamen alle Probanden mit dem System zurecht und hatten sogar Spaß am Benutzen des Systems. Leider war den Probanden der Nutzen eines solchen System nicht ganz klar. Dies mag daran liegen, dass die Teilnehmer darüber im Vorfeld nicht aufgeklärt wurden und daran, dass sie selbst kein Nutzen davon hatten, wenn die Geste richtig erkannt wurde, denn es wurde mit keiner richtigen Aktion verknüpft.

Schlussendlich kann festgehalten werden, dass Gesten mit *Midas* immer gut erkannt werden, wenn der Proband diese exakt wiederholen kann. Die Wiederholung wird durch mehrfaches Training der Geste verbessert, oder durch vorgegebene Bewegungen. Das bedeutet bei zukünftigen Systemen muss darauf geachtet werden, dass die Probanden die Ausführung der Gesten verändern und dadurch eine große Varianz der Trainingsets sichergestellt wird. Bei Gesten, die im Langzeitgedächtnis der Anwender verankert sind, bleibt die Erkennung einfacher.

8 Fazit

Wie die Studien von *Midas Touch* und *Gvoice* gezeigt haben, ermöglichten diese Systeme im Vorfeld trainierte Gesten zuverlässig zu erkennen. Allerdings funktioniert nicht jede Geste, denn die Gesten müssen unterscheidbar und gut reproduzierbar sein. Die Reproduzierbarkeit wird erhöht durch Gesten, die im Muskelgedächtnis verankert sind, oder mit Gegenständen ausgeführt werden. Dies trägt nicht nur dazu bei, dass die Geste natürlicher wirkt, sondern auch, dass weniger Trainingsets benötigt werden. Im Moment funktioniert *Midas* mit 10 bis 20 Gesten, was für die meisten Szenarien und Tests ausreicht. Es wurde auch gezeigt, dass die verwendete Pipeline mit dem Machine-Learning Algorithmus *Dynamic Time Warping* geeignet ist, um Gesten zu erkennen. Ein weiterer Vorteil der Pipeline und des *Node Designers* ist, dass sie die Abläufe visuell darstellen und damit das Verständnis für die Vorgänge gewährleisten. Damit ist es zudem möglich Wartungen durchzuführen und Fehler effizient zu finden. Denn es stellte sich heraus, dass für die Erkennung von Gesten eine Vielzahl von Vorgehensweisen getestet werden müssen, bis das gewünschte Ergebnis erzielt wird. Hier ist es entscheidend, dass allzeit das Gleichgewicht zwischen Performanz und Genauigkeit gehalten wird, damit die Anwendung in Echtzeit läuft. All dies ist nur möglich, da Midas ein *Node Designer* ist, mit dem mehrere Vorgehensweisen ohne großen Aufwand gut ausprobiert werden können. Auch die Verwendung des *Myo* Armbands war die richtige Entscheidung. Zum einen ist das Interface für das Erlangen der Daten effektiv und effizient, und zum anderen liefert es mehr Daten als herkömmliche Wearables, auch wenn die EMG Daten nicht ganz so vielversprechend waren wie angenommen. Mit Hilfe der Filter können diese Daten jedoch dazu verwendet werden, um die Erkennung zu erweitern. All dies zeigt, das es durchaus möglich ist mit Hilfe von Sensoren eine Gestenerkennung zu entwickeln. Die entwickelten Szenarien beweisen, dass *Midas* nicht nur flexibel genug ist um neue Ideen und Konzepte zu verwirklichen, sondern auch die Auswertung von Studien zu vereinfachen kann. Die aufgestellten Modelle zeigen zusätzlich, dass noch viel Potential in *Midas* steck, das durch weitere Szenarien zum Vorschein gebracht werden kann.

9 Ausblick

Möchte man die Anzahl der Gesten die mit Midas erkannt werden erweitern, gilt es zu allererst die Berechnung effizienter zu gestalten, denn das ist das größte Problem. Hierfür gibt es die Möglichkeit die Pipeline parallesierbar zu machen und so die Arbeit auf mehre CPUs oder gar GPUs zu verteilen. Beim Trainieren der Gesten werden viele Trainigsets aufgenommen, die vielleicht gar nicht gebraucht werden, denn sie lassen sich einfach über die Schwellwerte und einem bereits existierenden Template beschreiben. Auf diese Weise würden mehr qualitativ hochwertige Templates entstehen, wobei die Anzahl der Templates reduziert werden könnte und damit Rechenzeit gespart werden würde. Ein weiterer großer Faktor ist, dass sehr viele Berechnungen durchgeführt werden, die zu keinem Ergebnis führen können, denn sie haben zu große Abweichungen in Bezug auf die trainierten Gesten. Werden solche Datenfenster im Vorhinein ausgefiltert, bleibt mehr Zeit für weitere Berechnungen. Gleichzeitig gibt es die Möglichkeit den Dynamic-Time-Warping Algorithmus zu verbessern, dahingegen, dass die Daten schneller verarbeitet werden, dies kann z.B. durch das Einschränken der Genauigkeit erreicht werden. Neben softwaretechnischen Lösungen können natürlich auch hardwaretechnische Lösungen erfolgen. So kann durch eine Erhöhung der Sensorenanzahl die Genauigkeit weiter verbessert werden, denn dadurch könnten nicht nur mehr Werte erfasst werden, sondern es könnte unter Umständen auch dazu beitragen, ein besseres Modell des Körpers abzubilden. Auch könnte mit dieser Technologie die exakte Position im dreidimensionalen Raum berechnet werden. Mit dieser Information kann *Midas Touch* vollständig multiuserfähig gemacht werden, da so die genaue Position der Hand über dem Display ermittelt werden kann und auf diese Weise entschieden werden kann, welcher User wo getoucht hat und nicht nur überprüft werden, ob getoucht wurde. Neben mehr Sensoren, kann auch *Midas* durch weitere Nodes erweitert werden. Da der Dynamic-Time-Warping Algorithmus auch dafür verwendet wird Spracherkennung durchzuführen, wäre es ein Versuch wert *Midas* dahingehend zu erweitern. Damit würden auch die Möglichkeiten der Anwendung und deren Szenarien deutlich steigen. In den letzten Jahren wird das Machine-Learning auch immer mehr von den globalen Firmen, wie IBM, Microsoft und Google vorangetrieben. Diese veröffentlichen nicht nur den Code von ihren Anwendungen, sondern stellen auch Dienste bereit, mit denen die Berechnung von solchen Machine-Learning Pipelines in der Cloud durchgeführt werden können. Vielleicht ist es möglich die Implementierung in die Cloud zu verlagern, um noch mehr Rechenpower nutzen zu können und damit die Gestenerkennung weiter auszubauen.

10 Referenzen

Ballhaus, W. et al., 2015. *Media Trend Outlook Wearables: Die tragbare Zukunft kommt näher*,

Bishop, C.M., 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Available at: http://dl.acm.org/citation.cfm?id=1162264 [Accessed April 8, 2016].

Buxton, W.A.S., 1990. A THREE-STATE MODEL OF GRAPHICAL INPUT. In *INTERACT '90 Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*. North-Holland Publishing Co. Amsterdam, The Netherlands, pp. 449–456. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.139.1700 [Accessed April 19, 2016].

Chen, X. "Anthony" et al., 2014. Duet: Exploring Joint Interactions on a Smart Phone and a Smart Watch. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, pp. 159–168. Available at: http://dl.acm.org/citation.cfm?id=2556288.2556955 [Accessed September 8, 2014].

Erdös, P., 2014a. *Evaluierung des Einflusses von Interaktionstechniken auf kollaborative Designprozesse bei Tabletop-Anwendungen*. Universität Konstanz.

Erdös, P., 2014b. *Video Coding with Noldus Observer XT 11.5*. Uni Konstanz.

Feyer, S., 2015. *Rack of Inspiration – Design und Evaluation einer Toolbox zur Unterstützung des Inspiration Card Workshops*. Universität Konstanz.

Gottlieb, G.L. & Agarwal, G.C., 1971. Dynamic relationship between isometric muscle tension and the electromyogram in man. *J Appl Physiol*, 30(3), pp.345–351. Available at: http://jap.physiology.org/content/30/3/345 [Accessed April 16, 2016].

Hartson, R. & Pyla, P.S., 2012. *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*, Elsevier. Available at: https://books.google.com/books?id=w4I3Y64SWLoC&pgis=1 [Accessed April 1, 2016].

Hassenzahl, M., Burmester, M. & Koller, F., 2008. Der User Experience auf der Spur. *Usability Professionals 2008*, pp.78–82. Available at: http://attrakdiff.de/sience.html.

Houben, S. et al., 2015. WATCH CONNECT : A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. In *CHI '15*.

Kalman, R.E., 1960. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1), p.35. Available at: http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402 [Accessed July 10, 2014].

Kipp, M., 2005. *Gesture Generation by Imitation: From Human Behavior to Computer Character Animation*, Universal-Publishers. Available at: https://books.google.com/books?id=OIyArJx7mEQC&pgis=1 [Accessed May 16, 2016].

König, W.A., Rädle, R. & Reiterer, H., 2010. Interactive design of multimodal user interfaces. *Journal on Multimodal User Interfaces*, 3(3), pp.197–213. Available at: http://link.springer.com/10.1007/s12193-010-0044-2 [Accessed April 6, 2016].

Likert, R., 1932. A technique for the measurement of attitudes. *Archives of Psychology*, 22 140, p.55.

Marsland, S., 2014. *Machine Learning: An Algorithmic Perspective, Second Edition* 2nd Editio. B. Raton, ed., CRC Press. Available at: https://www.crcpress.com/Machine-Learning-An-Algorithmic-Perspective-Second-Edition/Marsland/9781466583283 [Accessed January 5, 2016].

Meier, A., Goto, K. & Wörmann, M., 2014. Thumbs up to gesture controls? A cross-cultural study on spontaneous gestures. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 211–217.

Miller, R.B., 1968. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*. New York, New York, USA: ACM Press, p. 267. Available at: http://dl.acm.org/citation.cfm?id=1476589.1476628 [Accessed April 3, 2016].

Paudyal, P., Banerjee, A. & Gupta, S.K.S., 2016. SCEPTRE. In *Proceedings of the 21st International Conference on Intelligent User Interfaces - IUI '16*. New York, New York, USA: ACM Press, pp. 282–293. Available at: http://dl.acm.org/citation.cfm?id=2856767.2856794 [Accessed March 4, 2016].

Rädle, R. et al., 2014. HuddleLamp. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces - ITS '14*. New York, New York, USA: ACM Press, pp. 45–54. Available at: http://dl.acm.org/citation.cfm?id=2669485.2669500 [Accessed January 22, 2016].

Ramakers, R. et al., 2012. Carpus: a non-intrusive user identification technique for interactive surfaces. In *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, p. 35. Available at: http://dl.acm.org/citation.cfm?id=2380116.2380123 [Accessed October 28, 2014].

Salvador, S. & Chan, P., 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5), pp.561–580.

Schlömer, T. et al., 2008. Gesture recognition with a Wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction - TEI '08*. New York, New York, USA: ACM Press, p. 11. Available at: http://dl.acm.org/citation.cfm?id=1347390.1347395 [Accessed February 26, 2016].

Schmidt, D. et al., 2012. A cross-device interaction style for mobiles and surfaces. In *Proceedings of the Designing Interactive Systems Conference on - DIS '12*. New York, New York, USA:

ACM Press, p. 318. Available at: http://dl.acm.org/citation.cfm?id=2317956.2318005 [Accessed November 8, 2014].

Schmidt, D. et al., 2010. PhoneTouch: A Technique for Direct Phone Interaction on Surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*. New York, New York, USA: ACM Press, p. 13. Available at: http://dl.acm.org/citation.cfm?id=1866029.1866034 [Accessed November 12, 2014].

Schmidt, R.A. & Lee, T.D., 2011. *Motor Control and Learning* 5th ed., Human Kinetics. Available at: https://books.google.com/books?hl=de&lr=&id=vBP091HCz38C&pgis=1 [Accessed June 2, 2016].

Tapia, E.M., 2008. Using machine learning for real-time activity recognition and estimation of energy expenditure. *Architecture*, p.493. Available at: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=15716969842745266469related:JdFbxszkHdoJ\nhttp://dspace.mit.edu/handle/1721.1/44913.

Tennié, J.O., 2016. *Midas touch*. Uni Konstanz.

Tennié, J.O., 2015. Seminararbeit : Tabletop - Centric Multi Device Interaction.

Vintsyuk, T.K., 1972. Speech discrimination by dynamic programming. *Cybernetics*, 4(1), pp.52–57. Available at: http://link.springer.com/10.1007/BF01074755 [Accessed April 11, 2016].

Xu, C., Pathak, P.H. & Mohapatra, P., 2015. Finger-writing with Smartwatch: A Case for Finger and Hand. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications - HotMobile '15*. New York, New York, USA: ACM Press, pp. 9–14. Available at: http://dl.acm.org/citation.cfm?id=2699343.2699350 [Accessed March 7, 2015].

Zhang, Y. & Harrison, C., 2015. Tomo. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*. New York, New York, USA: ACM Press, pp. 167–173. Available at: http://dl.acm.org/citation.cfm?id=2807442.2807480 [Accessed November 12, 2015].