Faculty of Science
Department of Computer Science
and Applied Computer Science

A 3D Gesture Recognition Extension for iGesture

Graduation thesis submitted in partial fulfillment of the
requirements for the degree of Master in Applied Informatics

Johan Bas

Promoter: Prof. Dr. Beat Signer

SEPTEMBER 2011

Abstract

When Nintendo released its Wii gaming platform, they introduced motion sensing to the general public. Motion sensing is a technique which can help users to control other electronic devices by performing hand or full body movement. Motion sensing is a more natural way to communicate with computers compared to keyboards, mice, buttons or other input devices. Motion sensors are capable of measuring specific physical quantities including acceleration, rotation, magnetic fields. Most consumer electronics are equipped with these motion sensors to provide a better user experience. For example, photo cameras may have a tilt sensor, which is used to sense whether a picture has been taken holding the camera in portrait or landscape mode. The motion sensors embedded in electronic devices can not only provide data to the device itself, but also provide input for other applications users want to interact with. This enables developers to create applications that can be controlled trough user motion without having to develop and sell specific motion sensing hardware.

The iGesture framework has been developed to help application developers to register and recognise gestures. Originally, iGesture supported 2D gestures recorded with a digital pen or a mouse. Later, 3D gesture recognition has been added to the framework. iGesture distincts itself from other frameworks trough its support of multiple input devices. Developers can add new devices to the framework in such a way that existing gesture recognition algorithms can be reused.

As part of this thesis, we first did a thorough investigation of common sensors and how they can be used in gesture recognition. In addition, we analysed existing devices for motion sensing and inspected their capabilities. Beside these two studies, the iGesture framework has been analysed to get familiar with its current support for 3D gesture recognition.

In a second phase, support for new IP-based devices has been integrated into iGesture. IP-based devices are input devices with embedded motion sensors capable of communicating using the IP protocol. An Android application has been developed to support iGesture's IP-based com-

munication on motion sensing devices running the Android operating system. The addition of new devices introduces more flexibility and provides a larger collection of devices for future iGesture developers to choose from. Besides the integration of these new devices, a 3D gesture recognition algorithm based on the Dynamic Time Warping algorithm has been developed and integrated.

Finally, an extensive set of tests to evaluate the recognition rates of our new algorithm has been performed. Different input devices were used by different users to perform various gestures. The results show that the current algorithm implementation in combination with the existing input devices and new Android input devices can be used for 3D gesture recognition.

Samenvatting

Wanneer Nintendo zijn Wii spel platform uitbracht, introduceerden ze bewegingssensoren als invoerapparaat aan het algemene publiek. Motion sensing, het gebruik van bewegingssensoren, is een techniek die gebruikers helpt elektronische apparaten te controleren door hand of lichaamsbewegingen. Motion sensing is een meer natuurlijke manier om te communiceren met computers in tegenstelling tot toetsenborden, muizen, knoppen of andere invoerapparaten. Bewegingssensoren zijn in staat een natuurlijke grootheid te meten zoals acceleratie, rotatie en magnetische velden. De meeste elektronische apparaten bevatten deze bewegingssensoren om een betere gebruikservaring aan te bieden. Bijvoorbeeld, foto camera's kunnen uitgerust zijn met een tilt sensor, deze tilt sensor wordt gebruikt om te meten of een foto is getrokken in landschap of portret modus. De bewegingssensoren ingebouwd in deze elektronische apparaten kunnen niet alleen data verzamelen voor het toestel zelf, maar ook voor andere applicaties waarmee de gebruiker wil communiceren. Dit staat ontwikkelaars toe om applicaties te bouwen die door middel van beweging gestuurd kunnen worden zonder bijpassende hardware te ontwikkelen en te verkopen.

Het iGesture framewerk is ontstaan om applicatie ontwikkelaars te helpen om bewegingen te registreren en te herkennen. Oorspronkelijk ondersteunde iGesture enkel 2D bewegingen geregistreerd met een elektronische pen of een muis. Later werd 3D bewegingsherkenning toegevoegd aan het framewerk. iGesture onderscheidt zichzelf van de andere framewerken door zijn ondersteuning van meerdere invoerapparaten. Ontwikkelaars kunnen nieuwe apparaten toevoegen aan het framewerk zodat de bestaande bewegingsherkenning algoritmes hergebruikt kunnen worden.

Als onderdeel van deze thesis, hebben we een grondig onderzoek gedaan naar de meest voorkomende sensoren en hoe deze sensoren gebruikt kunnen worden voor bewegingsherkenning. Daarnaast hebben we een analyse gemaakt van de bestaande apparaten voor bewegingsherkenning en hebben we de mogelijkheden van deze apparaten geïnspecteerd. Bovenop deze twee studies werd het iGesture framewerk geanalyseerd om bekend te raken met zijn huidige status

van 3D bewegingsherkenning.

In een tweede fase werd onder-steuning voor IP gebaseerde apparaten toegevoegd aan iGesture. IP gebaseerde apparaten zijn invoerapparaten met bewegingssensoren welke kunnen communiceren door middel van het IP protocol. Een Android applicatie is ontwikkeld om iGesture's IP communicatie op Android toestellen met bewegingssensoren te ondersteunen. Het toevoegen van deze nieuwe apparaten introduceert meer flexibiliteit en biedt een grotere keuze aan invoerapparaten voor toekomstige iGesture gebruikers. Naast het integreren van deze Android apparaten, werd er een 3D bewegingsherkenning algoritme gebaseerd op Dynamic Time Warping ontwikkeld.

Als laatste werden er uitgebreide testen uitgevoerd om de herkenningsgraad van dit nieuw algoritme te bepalen. Verschillende invoerapparaten werden gebruikt door verschillende gebruikers om verzamelingen van bewegingen te classificeren. De resultaten tonen aan dat de huidige implementatie van het bewegingsherkenning algoritme in combinatie met zowel, de bestaande als de nieuwe invoerapparaten, kan gebruikt worden voor 3D bewegingsherkenning.

Acknowledgments

Four years ago, I decided to pursue a master's degree. Combining a full-time job with university studies and still maintaining my private life has proven to be a challenge. I could not have obtained this diploma without the support of my loving girlfriend, my family and friends, my co-workers and all the other persons who supported me. The hours spent behind my books and computer, the stress of failing or passing has weighted on them too. Their understanding, their motivational speeches and encouraging words have driven me to reach this point.

I would like to thank my promoter Prof. Dr. Beat Signer for the opportunity to work on the iGesture framework, for the counselling and challenges as well as for his personal assistance in the realisation of this master's thesis.

I also want to thank Dr. Bart Jansen for the enlightening conversation we had when I found myself at a dead end. It changed my way of thinking and made me explore new directions.

Contents

Abstract i

Samenvatting iii

Acknowledgments v

1 Introduction 1

1.1 Context . . . 1
1.2 Problem Description . . . 2
1.3 Approach . . . 3
1.4 Contributions . . . 3

2 Background and Related Work 5

2.1 3D Motion Capture Devices . . . 5
2.1.1 Wii Remote . . . 5
2.1.2 Wii Motion Plus . . . 6
2.1.3 Kinect for XBox 360 . . . 6
2.1.4 Playstation Move . . . 7
2.1.5 Android Devices . . . 8
2.2 3D Recognition Frameworks . . . 8
2.2.1 WiiGee . . . 9
2.2.2 Invensense . . . 9
2.2.3 LiveMovePro . . . 10
2.3 Sensors . . . 10
2.3.1 Accelerometer . . . 10
2.3.2 Gyroscope . . . 11
2.3.3 Magnetic Sensor . . . 13
2.4 Sensor Related Issues . . . 13
2.4.1 Tilt . . . 13

2.4.2 Sensor Fusion . . . 14
2.4.3 Acceleration, Velocity and Position . . . 16

3 3D Gesture Recognition in iGesture 17
3.1 Existing iGesture framework . . . 18
3.1.1 iGesture . . . 18
3.1.2 iGesture Tools . . . 21
3.2 A New Gesture Recognition Algorithm . . . 24
3.2.1 k-Nearest Neighbour . . . 24
3.2.2 Dynamic Time Warping . . . 24

4 Implementation Details 29
4.1 Dynamic Time Warping Acceleration Metrics . . . 29
4.1.1 Euclidean Norm Metric . . . 29
4.1.2 Plain x,y,z Metric . . . 32
4.1.3 DTW for the x,y,z Axes Separately . . . 32
4.2 IP Communication . . . 33
4.2.1 Architecture . . . 33
4.2.2 Message protocol . . . 34
4.3 Proof of Concept Application . . . 40

5 Evaluation 42
5.1 WiiGee Gestures . . . 42
5.2 Boxing gestures . . . 45
5.3 Cellphone Gestures . . . 50

6 Future Work 53
6.1 Sensor Fusion Using the Wii Motion Plus . . . 53
6.2 Extending and Adding Recognition Algorithms . . . 53
6.3 Processing of Data Streams . . . 54

7 Conclusion 55

A Wii Remote 56
A.1 BlueCove . . . 56
A.2 WiiGee . . . 57

B Source Code 58

Bibliography 96

Chapter 1

Introduction

This first chapter describes the context of this thesis, provides a description of the problems that we addressed together with the approach that was used to solve these problems

1.1 Context

Motion sensing devices have become a part of our daily life. Most people come in contact with cellphones, handhelds, tablets, game controllers and other motion sensing devices on a day-to-day basis. These devices are all equipped with motion sensors to give their users a better experience when using them. Unfortunately, these devices are dedicated systems specially developed for one purpose only. Due to this fact, motion sensing on the PC platform has not had its breakthrough yet, as there is no motion sensing hardware available for the PC platform. The iGesture framework could fill in this gap through the integration of existing motion sensing hardware such as 3D gesture input devices.

iGesture has been developed to help developers to implement customised gesture recognition in their applications. iGesture eliminates the complicated task of researching and implementing gesture recognition algorithms. iGesture currently contains multiple recognisers for 2D gestures, however 3D gesture recognition is lacking. With the introduction of motion sensing devices to iGesture, good 3D gesture recognition would add a valuable feature to the iGesture framework.

1.2 Problem Description

When Nintendo released its Wii gaming platform, motion sensing was introduced to the general public. Since then, other manufactures have tried to build new applications and platforms featuring motion sensing. Unfortunately, the PC platform has been left out of this new technology. No dedicated input devices have been sold to a widespread audience, no application software has been developed.

One of the biggest questions of motion sensing application development is *how to process sensor data coming from these input devices to something you can use?* The average programmer is not familiar with these devices, let alone there capabilities. Programmers are not trained in developing 3D device drivers and recognition algorithms. They know how to develop mainstream software and they are capable of integrating other components with this software.

This is where iGesture comes into play. iGesture is an open source Java framework built to help programmers to implement gesture recognition as part of their application. Until now, iGesture has an outstanding reputation concerning 2D gestures. Unfortunately, decent 3D gesture support is lacking. Currently, the only supported 3D device is the Nintendo Wii Remote and the implementation and integration of the Wii Remote has still some open issues. The drawing presented to the user when preforming a gesture with the Wii Remote does not resemble the gesture that the user has executed. Beside this aesthetic issue there are some doubts on the accuracy of the recognition algorithms. The already present knowledge of 2D gestures has been reused for 3D gesture recognition. This might not be the most efficient way for 3D gesture recognition. A new algorithm, capable of working with acceleration values for its gesture recognition needs to be developed.

Up until now the Wii Remote is the only 3D motion sensing input device integrated into iGesture. In the meanwhile other 3D motion sensing capable devices have been adapted by potential iGesture's users. It would give the iGesture framework an extra edge if new, commonly available, motion sensing devices could be integrated into the framework. This would give developers the possibility to choose between different input devices when searching for the best match for their applications.

1.3 Approach

Until now 3D gesture recognition has been performed by double integration acceleration values to obtain `x,y,z` values. When using these integrated values, datasets for the three different planes `xy,xz,yz` can be computed. Once these three datasets are computed, 2D gesture recognition algorithms can be applied on each plane. The actual 3D gesture recognition is realised through a combination of gesture recognition results from these three planes. In our new approach, we will develop an algorithm that works directly with acceleration values, eliminating the double integration and dataset computations.

Firstly, we have performed a detailed study of the sensors that were to be used in our experiments. This gave us a good idea of the possibilities and error margins of the acquired sensor data. This analysis helped us in developing a good strategy to process this sensor data to a format usable for gesture recognition. After studying the sensors, we looked into different techniques on how to combine different sensors to maximise sensor data accuracy.

Secondly, when working with acceleration values, the previously mentioned 2D recognition algorithms are no longer applicable. A new 3D gesture recognition algorithm had to be developed from scratch. This algorithm should only use acceleration values without the double integration in its recognition process.

1.4 Contributions

In this section, we will give an overview of the different contribution this master thesis has made.

- An in depth analysis of the different sensors and sensor combinations, devices and frameworks (including iGesture), for 3D gesture recognition has been made. This analysis helped us to evaluate input device candidates, gesture recognition algorithms and application possibilities/limitations.
- A new IP based communication protocol for iGesture input devices has been developed. This communication protocol will extend iGesture capabilities to different mobile devices like cellphones, handhelds and smartphones running different operating systems.
- An Android application implementing iGesture's IP communication has been developed. This enables iGesture users to, use Android devices for gesture recognition in combination with iGesture.

- Support for the Wii Motion Plus extension has been added to iGesture. A description on how to use the Wii Motion Plus extension in combination with the Wii Sensor Bar and the Wii Remote is provided.
- A new 3D gesture recognition algorithm based upon the k-Nearest Neighbour and Dynamic Time Warping algorithms has been implemented. This algorithm is configurable to be executed with three different metrics. The new algorithm is integrated with iGesture's workbench enabling iGesture users to simulate different algorithm configurations. This will help them to optimise their custom 3D gesture recognition results.
- Extensive testing using different versions of the algorithm, different gestures and different devices has been performed to analyse the performance of our new algorithm.

Chapter 2

Background and Related Work

2.1 3D Motion Capture Devices

A multitude of motion capture devices are commercially available. All these devices use different sensors in order to provide some kind of motion sensing experience. It is important to know the capabilities and limitations of these devices before using them. We will give a brief overview of some of the most commonly known devices.

2.1.1 Wii Remote

In December 2006, Nintendo released its Wii gaming platform in Europe. Nintendo tried to distinguish the Wii from other consoles (e.g. Xbox and Playstation) by introducing the Wii Remote (commonly known as the Wii Mote). The wireless Wii Remote incorporates an accelerometer and an infrared sensor. The Nintendo Wii is a closed source, proprietary platform. After some reverse engineering effort, the following sensor details became available[1]:

- The accelerometer is a ADXL 330 with a ± 3g sensing range[2].
- Inside the Wii Remote, a PixArt imaging camera is incorporated. This camera is capable of tracking up to 4 infrared lights. It has a 1024 * 768 (4:3) resolution at a 100Hz refresh rate. It uses 4 bit to express the size and 8 bit for the intensity. The camera is supposed to track the infrared bar that needs to be placed on top or beneath the television. The sensor bar consists of 3 LEDs on each side of the bar. One rather strange fact about the camera

[1] http://www.parleys.com/#st=5&id=1637

[2] http://www.analog.com/en/mems-sensors/inertial-sensors/adxl330/products/product.html

is its positioning inside the remote. Vertically it has a 30 degree view port 15 degrees up and 15 degrees down. Horizontally the sensor is capable of viewing 18 degrees to the left and 22 degrees to the right. Presumably Nintendo introduced a small bias in favour of right handed people.

The Wii Remote is connected to the console via a Bluetooth connection. Using the Bluetooth protocol, it is possible to connect a Wii Remote to a computer.

2.1.2 Wii Motion Plus

In June 2009 Nintendo released the Wii Motion Plus[3], a Wii Remote extension that adds a gyroscope to the Wii Remote. Nintendo aims to improve the Wii Remote accuracy so that game developers can push the limits of motion-based game development even further. The Wii Motion Plus contains a 3-axis gyroscope enabling Wii developers to compute true linear acceleration[4].

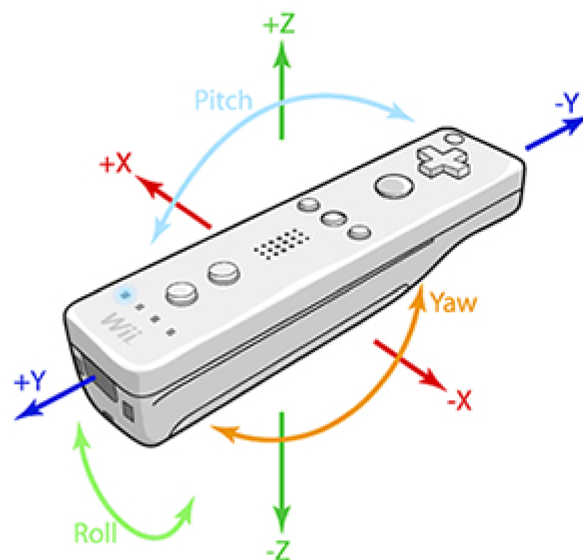

Figure 2.1: Wii Remote with the acceleration and rotation axis indicated

2.1.3 Kinect for XBox 360

The Kinect is an addition for Microsoft's Xbox 360 gaming platform. The Kinect is designed to bring motion sensing to the Xbox. The Kinect is a camera add-on which enables users to

[3]http://en.wikipedia.org/wiki/Wii_MotionPlus/
[4]http://invensense.com/mems/gaming.html

control the Xbox without any handheld device. The camera is capable of full-body 3D motion capturing with facial and voice recognition. Trough reverse engineering[5] it has been revealed that the Kinect has an eight bit VGA resolution (640*480) with depth sensing.

Figure 2.2: Microsoft Kinect

2.1.4 Playstation Move

The Playstation Move[6] is an input device for Sony's Playstation 3 gaming platform first revealed in June 2009. It is capable of motion sensing trough a set of different sensors. The Playstation Move uses an Eye camera to track the controller position. The controller is equipped with an glowing orb at the top of the controller. The orb can glow in a full range of RGB colours. This enables the eye camera to minimise external colour interference while tracking the controllers. Furthermore, the Playstation Move is equipped with a 3-axis accelerometer and a 3-axis rate sensor. A rate sensor[7] is measures angular rate like a gyroscope, but is also used for devices with a low cut off frequency that is other than zero. The Playstation Move uses the Bluetooth 2.0 protocol to communicate with the Playstation console.

Figure 2.3: Playstation Eye

Figure 2.4: Playstation Move

[5] http://openkinect.org/wiki/Main_Page/
[6] http://en.wikipedia.org/wiki/PlayStation_Move/
[7] http://en.wikipedia.org/wiki/Rate_sensor/

2.1.5 Android Devices

Android was released in November 2007 by Google. Android is an open source platform built on top of the Linux kernel. The majority of Android applications are written in a stripped down version of the Java programming language. Android was primarily intended to be used as a cellphone platform. Cellphone manufacturers quickly adapted Android due to its open source licensing model. Google does not limit manufacturers to run android on devices with certain hardware specifications. This leads to a large variety of existing android devices.

Most android phones have integrated Wifi, accelerometers, GPS and magnetic sensors. Some newer models, provide an integrated gyroscope[8][9][10]

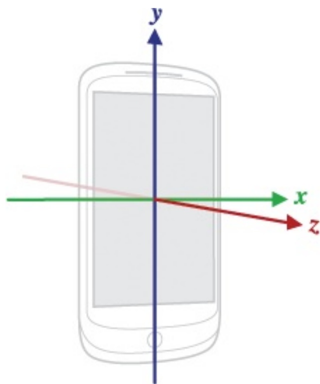

Figure 2.5: Android acceleration axis

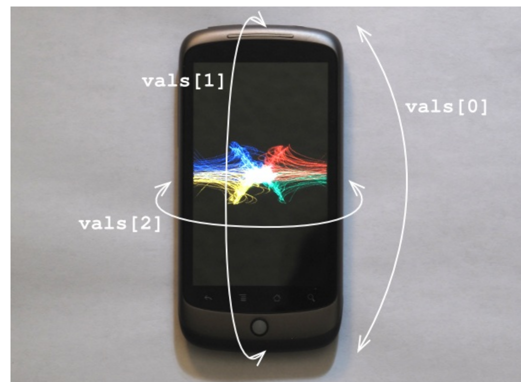

Figure 2.6: Android rotation axis

2.2 3D Recognition Frameworks

A number of applications for 3D gesture recognition for the PC platform have been developed in the past [24, 12, 9, 3]. Unfortunately, the majority of these applications are just a proof of concept. Due to the absence of stable software for 3D gesture recognition, there is an opportunity for 3D gesture recognition on the PC. This application shortage is the result of an absence of, by default, integrated 3D devices on PC platforms. This forces gesture recognition developers to first integrate a 3D device with the PC platform before they can develop any algorithms. This is where one of the advantages of iGesture comes into play: the combination of device integration with gesture recognition.

3D gesture recognition software is mainly executed on dedicated systems like the Nintendo

[8]http://www.samsungnexuss.com/nexus-s-specs/
[9]http://www.htc.com/europe/product/sensation/specification.html
[10]http://www.samsunggalaxys2.net/nieuws/touchwiz-4-0-gyroscope/

Wii, Microsoft's XBox and the Playstation (Move). Besides these gaming platforms, custom interactive motion sensing applications have been developed [8, 2]. These custom installations focus on a predefined, limited set of gestures that cannot easily be changed and extended.

2.2.1 WiiGee

WiiGee [19] is an open source gesture recognition library for accelerometer-based gestures. WiiGee only supports a single input device, the Wii Remote. While WiiGee is capable of detecting the Wii Motion Plus and a Sensor Bar, only the accelerometer is used in the gesture recognition process. WiiGee uses Hidden Markov Models (HMM) as its gesture recognition algorithm. The WiiGee library can be trained to recognise arbitrary gesture sets.

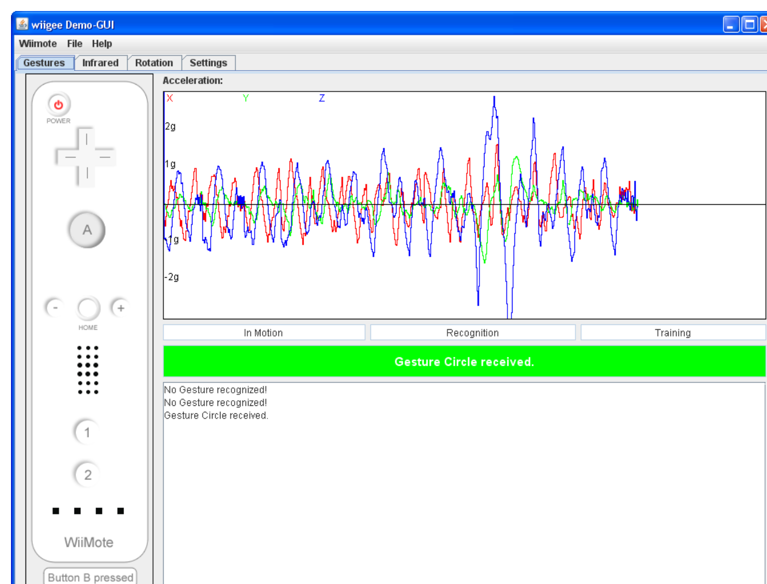

Figure 2.7: WiiGee Demo GUI

2.2.2 Invensense

Invensense[11] call themselves the leader in motion processing solutions. They have developed a closed source motion processing platform to help devices in detecting, measuring, synthesising, analysing and digitising an object's motion in three-dimensional space [14]. The Invensense motion processing platform is capable of using accelerometers, gyroscopes and other sensors. It uses sensor fusion to improve sensor reading. Invensense developed its own gyroscope and accelerometer chips integrating their own sensor fusion technology. This hardware is primarily

[11]http://invensense.com/mems/platform.html

sold for gaming, handheld and tablet purposes. Invensense also developed their own remote controller and image stabilisation software.

2.2.3 LiveMovePro

LiveMovePro[12] is a proprietary framework for Motion recognition and tracking. AiLive collaborated with Nintendo to design the Motion Plus hardware and is offering LiveMove 2 to help game developers to take full advantage of its capabilities. LiveMovePro features

- Full support for the Wii MotionPlus
- Simultaneous classification to provide semantic information about moves
- Classification of two-handed, coordinated motions
- Classification can use additional MotionPlus data to improve performance

2.3 Sensors

Sensors are electronic or mechanical components designed to measure a physical quantity. A sensor can sense one physical quantity and translate it into electric signals so that the particular quantity can be interpreted by computers.

More and more commercially available electronics come with built in sensors. For instance, digital cameras incorporate gyroscopes for image stabilisation and GPS sensors for Geo tagging. Smart phones have accelerometers, compasses, a GPS and in some cases even gyroscopes. Game controllers are equipped with accelerometers, gyroscopes and image sensors. It is important to fully understand the inner workings, capabilities and limitations of these different sensors before one can interpret the generated data.

2.3.1 Accelerometer

An accelerometer is a small mass inside a reference frame connected through beams. These beams are flexed under the forces of acceleration (movement) or gravity. The flexing of these beams can be measured resulting in acceleration values. This design of an accelerometer has two surprising effects:

[12] http://www.ailive.net/

- When an accelerometer lays still on a surface, it measures the down force of gravity. The reference frame is kept still due to the surface. The forces of gravity pulls the mass downwards flexing the top and bottom beams. Therefore the accelerometer will measure the force of gravity.
- When a device is in free fall, there is no movement between the mass and the reference frame. The accelerometer will measure zero on all axes.

Accelerometers cannot distinguish between acceleration caused by gravity or acceleration caused by movement. Gravity could be filtered out by applying a high-pass filter. Conversely a low-pass filter can be used to isolate the force of gravity. Using high- or low pass filters will decrease the sensor response time and its accuracy.

In order to get better response times and higher accuracy, gyroscopes can be used in combination with accelerometers.

2.3.2 Gyroscope

The inner workings of a gyroscope are based on the Coriolis effect[13] . Using the Coriolis effect the angular velocity of a device can be measured. In order to get an orientation one has to integrate the angular velocity. This integration step is shown in Equation 2.1.

$$\int \cos(2\pi ft) = \frac{1}{2\pi ft} \sin(2\pi ft) \tag{2.1}$$

The integration step has important side effects: high frequency jittering (noise) is reduced, which is a good thing. Unfortunately, low frequency jittering is turned into drift which is a bad thing. Figure 2.8 shows gyroscope data samples gathered by holding the device still during one second. Figure 2.9 shows the drift occurring when integrating these gyroscope values over time. Finally Figure 2.10 shows the drift of a Wii Motion Plus lying still on a table.

[13] http://en.wikipedia.org/wiki/Coriolis_effect/

When integrating the angular velocity to calculate the next angle based upon a previous angle as seen in Equation 2.2, the time measurement accuracy is as important as the gyroscope accuracy. The consequences of the time measurement being off by 2% are as severe as if the angular velocity accuracy would be off by 2%. This error margin can only be reduced through a fast sensing rate and accurate time stamping.

$$\Theta_{n+1} = \Theta_n + \omega \triangle t \tag{2.2}$$

Additional sensors like a magnetic sensor, an optical sensor or even an accelerometer can be used to compensate for these errors and the drifting effect.

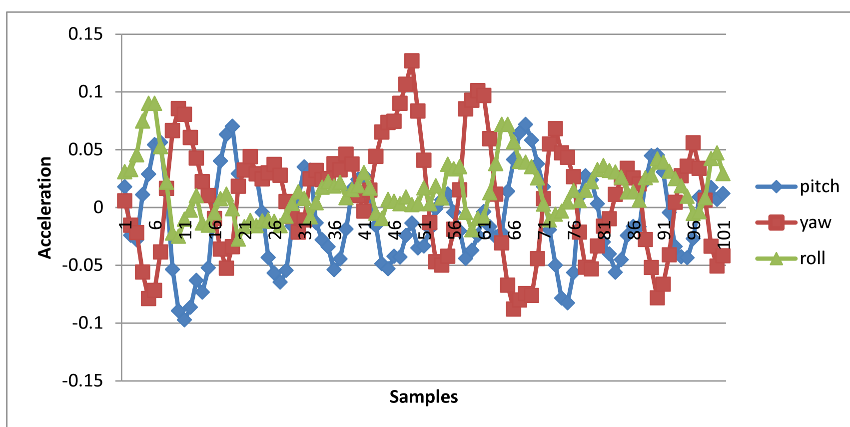

Figure 2.8: 100 gyroscope samples over a time span of 1.011243 seconds taken from a Samsung Galaxy SII while holding the device as still as possible

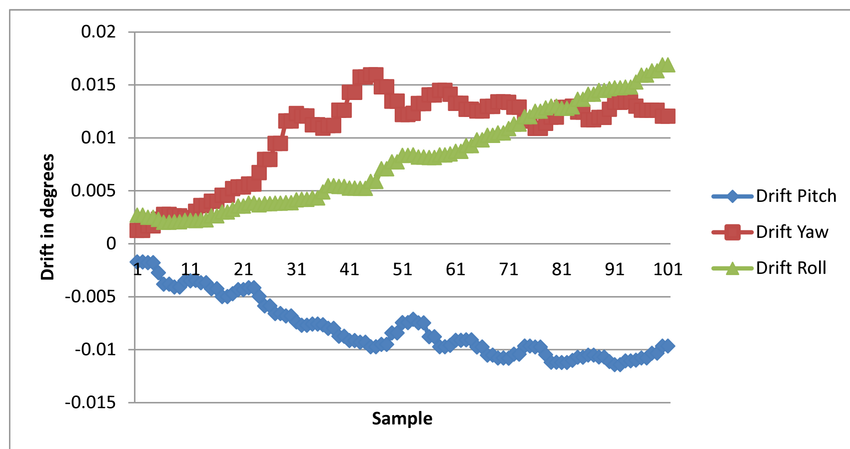

Figure 2.9: 100 gyroscope samples integrated showing drift taken from a Samsung Galaxy SII

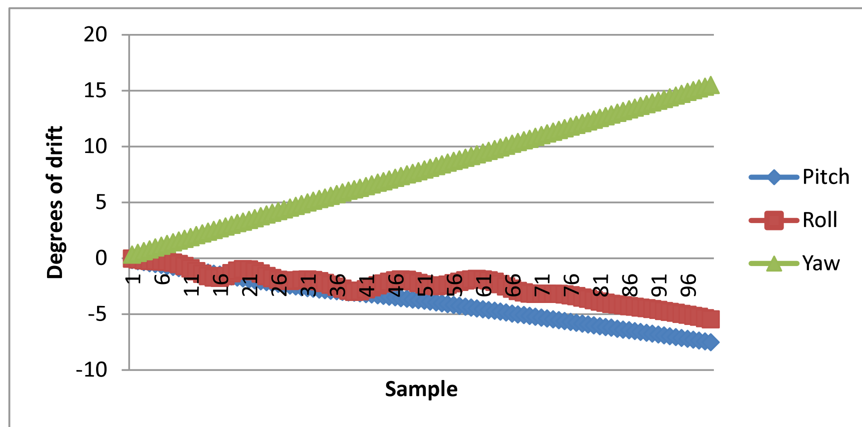

Figure 2.10: 200 gyroscope samples taken from a Wii Motion Plus, lying still on a table, integrated (by the WiiGee), to illustrate drift

2.3.3 Magnetic Sensor

A magnetic sensor, commonly known as a compass, senses magnetic fields. Due to the principles of the Hall effect[14] the magnetic sensor will point to the North Magnetic Pole[15]. A 2-axes magnetic sensor is sufficient to find magnetic north if the axes are perpendicular to the earth's surface. If the user is allowed to rotate the device, a 3-axes magnetic sensor and an accelerometer to calculate the tilt are needed to find Magnetic North. In order to find true North, a GPS sensor is needed to provide a reference point for the device relative to the Magnetic North Pole. Note that magnet sensors tend to fail when placed near strong magnetic fields or when the sensor is close the north or south pole.

2.4 Sensor Related Issues

Besides the previously described sensor design limitations, we have investigated the most common techniques and problems using one or combining multiple sensors for 3D gesture recognition.

2.4.1 Tilt

When gesture recognition is performed by using an accelerometer only, gravity will not be accurately compensated for. If the user would hold the accelerometer perfectly levelled and perform the same gesture over and over again, more or less the same acceleration values would

[14]http://en.wikipedia.org/wiki/Hall_effect
[15]http://en.wikipedia.org/wiki/North_Magnetic_Pole

be output. There would be a strong resemblance between two accelerometer sequences. However, while performing a gesture it is very difficult to hold the device perfectly levelled. Without any intention, the user often holds the device with different orientations while performing gestures. This difference in orientation might only be a few degrees but this can be enough to have a major impact on the acceleration values. The readings of the accelerometer will not directly reflect the external motions performed with the sensor. In real world situations, there will be small orientations around the 3-axis (pitch, roll and yaw) influencing accelerometer reading in x,y,z directions. Detecting tilt by using only an accelerometer is very challenging. Some algorithm implementations try to compensate for tilt with limited results [12].

An orientation-independent system might be the solution for the tilt problem. However, when developing such a system one needs to be aware of the consequences. It would no longer be possible to distinguish a gesture sliding from the left to the right from another gesture sliding from the right to the left. These two gestures only differ in the device orientation. When rotating the device by 180 degrees around the y-axis (roll) the gestures are the same.

The best solution for the tilting problem is to use other sensors like gyroscopes and compasses in order to compensate for tilt.

2.4.2 Sensor Fusion

Sensor fusion is a technique in which different sensors are combined to improve the data coming from a single sensor. The following are examples of sensor fusion:

- An accelerometer senses linear motion and gravity. However the accelerometer is not capable of distinguishing gravity from motion. Trough the help of other sensors, like gyroscopes, magnetic sensors and/or optical sensors, it is possible to remove gravity from the accelerometer values and to isolate the linear acceleration.
- A gyroscope senses angular velocity. The data from a gyroscope needs to be integrated to get angular velocity which introduces drift. Over time this drift would render the gyroscope data useless. Accelerometers, magnetic sensor, GPS and optical sensors can be used to calculate the tilt and provide an absolute reference point. This data can then be used to compensate for drifting.

Trough sensor fusion it is possible to obtain more accurate and more usable data.

Sensor Fusion using Wii Motion plus

When using a Wii Motion Plus, gyroscope pitch and roll can be compensated using accelerometer data in order to compensate for drifting. During the usage of the Wii Motion Plus, the user will hold the Wii Remote still. In these moments, when the accelerometer and gyroscope are not moving, it is advised to reset the current rotation, pitch and roll. Resetting pitch and roll will remove drift for these two orientations. The accelerometer is gravity sensitive, providing an absolute reference. This absolute reference can be used to reset the pitch and roll. Unfortunately, yaw cannot benefit from the accelerometer estimate. After a major rotation, it is advised to reset the yaw value. This is possible when the Wii Remote is pointed to the middle of the sensor bar (horizontally).

By using sensor fusion from the accelerometer, gyroscope and sensor bar better sensor data can be acquired. Once a good orientation is computed, one can use this data to calculate linear acceleration.

Sensor Fusion on Android

In Android sensor fusion is performed automatically when calling the correct API functions[16]. When listening to the `Sensor.TYPE_ACCELEROMETER`, `Sensor.TYPE_GYROSCOPE` or `Sensor.TYPE_MAGNETIC_FIELD`, raw sensor values are retrieved. On the other hand, when listening to `Sensor.TYPE_GRAVITY`, `Sensor.TYPE_LINEAR_ACCELERATION` or `Sensor.TYPE_ORIENTATION` android tries to use all available device sensors to improve the captured data. Best results will be obtained when using an Android device with an accelerometer, magnetic sensor and a gyroscope. Compensating the gyroscope with a magnetic sensor in combination with an accelerometer will provide an absolute reference point and rotation to calculate the device's orientation. The orientation can then be used to improve the acceleration values by subtracting gravity resulting in linear acceleration.

[16]http://developer.android.com/reference/android/hardware/SensorEvent.html

2.4.3 Acceleration, Velocity and Position

In order to calculate a device position based on acceleration values, one needs to perform a double integration over the acceleration values. The first integration will transform the acceleration values into velocity (2.3). A second integration will convert velocity into position (2.4).

$$v = \int at \tag{2.3}$$

$$p = \int vt \tag{2.4}$$

Linear acceleration is defined as acceleration minus gravity. The process to obtain linear acceleration is also called gravity compensation. The first thing one needs to do before integrating is to subtract gravity from the acceleration values to obtain linear acceleration. The isolation of gravity can only be performed after the device's orientation is calculated. This will give us the knowledge of the gravitational spread on the different axes so one can subtract it. A gyroscope is an excellent sensor for calculating orientation. Once gravity is compensated, the acceleration values can be double integrated. As explained earlier in section 2.3.2 when integrating noisy data, low frequency jittering will be turned into drift. When double integrating noisy data, this drift will ever increase. Therefore, if you double integrate acceleration values, significant drifting is to be expected.

Drifting due to noise is not the biggest concern when double integrating acceleration values. In order to compensate for gravity, gyroscopes are used to calculate the devices orientation. This gyroscope data is not perfect. Suppose that the gyroscope data is off by one degree, this bias adds a constant to the integration equation. Double integrating a constant will result in a parabola (2.5). This parabola shows the exponential bias that is added to the devices estimated position. Therefore, when working with accelerometers, it is important to work directly with acceleration values and to avoid integration as much as possible.

$$x = \frac{1}{2}at^2 \tag{2.5}$$

Chapter 3

3D Gesture Recognition in iGesture

The iGesture[20] development started in 2006. iGesture was developed due to the emerging need of a general and extensible framework that provides an integrated platform for the design and evaluation of gesture recognition algorithms, as well as for their deployment to a wide audience. The iGesture framework is capable of defining, evaluating, optimising and recognising gesture sets using different gesture recognition algorithms. iGesture also provides easy mechanisms for integrating new input devices and new recognition algorithms. In the beginning of iGesture only 2D devices and 2D gestures were supported. iGesture was released to the public as an open source, Java framework.

In 2009 the iGesture framework has been extended to support 3D gesture recognition [23]. In this specific approach, the gesture representation in three-dimensional space is projected onto three two-dimensional planes (`xy, xz, yz`). This technique enables iGesture to reuse the already present 2D algorithms.

In 2010 multimodal gesture recognition was added to the iGesture framework [17]. Multimodal gestures are gestures performed with multiple input devices by one or more users. Besides combined gestures, multi-modal gesture support for TUIO devices[1] was added to iGesture.

In order to have a good knowledge of the iGesture framework and to better understand the next chapters, we provide a short introduction to the design of the framework, its usage and functionality.

[1] http://www.tuio.org/

3.1 Existing iGesture framework

iGesture can be broken down into three major blocks. There is the recogniser, a management console and tools for testing gesture sets and evaluating the algorithms used. These three components all use the same common data structure.

3.1.1 iGesture

The most important data structure to understand when using iGesture is the gesture representation shown in Figure 3.1. The `GestureClass` represents one gesture a circle, a rectangle, a punch when developing a boxing game. Multiple gesture classes are organised in a `GestureSet`. A `GestureSet` represent the set of possible gestures when performing gesture recognition. The `GestureClass` uses the Visitor pattern[2] [6] to separate the object structure from the recognition algorithms.

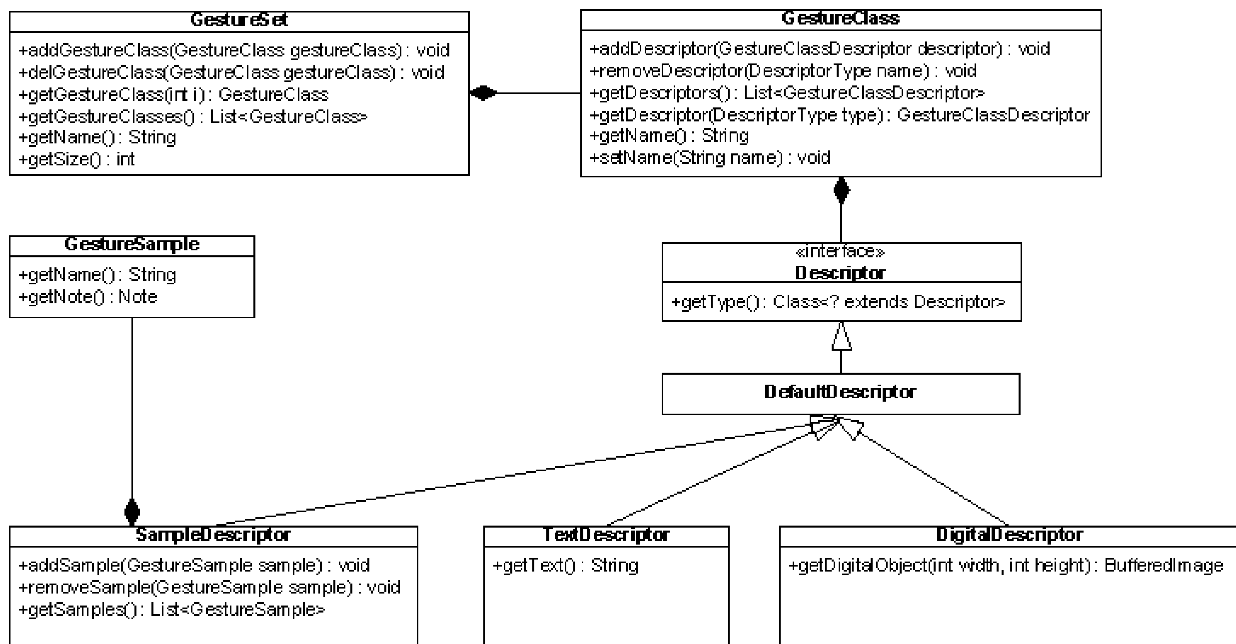

Figure 3.1: iGesture Gesture datastructure

Different algorithms may require different gesture representations. Therefore, a `Descriptor` interface has been introduced, each `GestureClass` must have at least one `Descriptor`. Different gesture descriptors have been implemented:

- The `SampleDescriptor` describes a gesture based on a set of training samples. This

[2] http://en.wikipedia.org/wiki/Visitor_pattern

Descriptor is the most widely used descriptor for gesture recognition

- The TextDescriptor describes a gesture on a textual basis, for example a character String representing directions between characteristic points of the gesture.
- The DigitalDescriptor represents a gesture trough a digital image. This Descriptor is less suitable for the gesture recognition. It is meant to provide a digital image of a gesture to be used in the graphical user interface. This allows developers to show an image of a recognised gesture to the user.

The Recogniser component shown in Figure 3.2 is responsible for the actual gesture recognition process. It acts as a facade[3] [6] meant to hide away the complexity of gesture recognition algorithms. A recogniser is initialised with a configuration object containing information about the used gesture set (1..*), the algorithm(s) to be used (1..*), the parameters for the algorithms, the minimal accuracy as well as the size of the result list in the result set. A recogniser contains multiple methods for gesture recognition which behave differently. The recognise(Note note) will evaluate the input gesture trough sequentially evaluating the chosen algorithms. This process will stop as soon as one algorithm returns a valid match. The recognise(Gesture<?> gesture, boolean recogniseAll) method on the other hand will continue to execute all the selected algorithms and will return a combined result of all algorithms in form of a ResultSet.

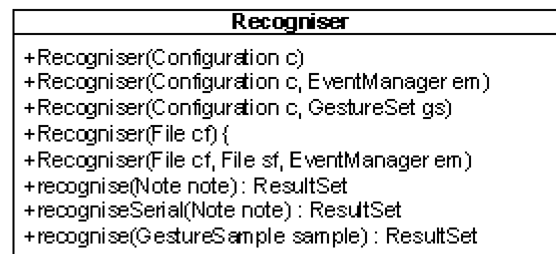

Figure 3.2: Recogniser API

A Note is a data structure representing gestures captured from 2D input devices using Traces defined by timestamped Points. The Note3D data structure represents gestures captured from a 3D input device in the form of Point3D and AccelerationSamples.

The core of the gesture recognition functionality lies in the Algorithm. iGesture strives to provide algorithm developers some flexibility in the design and usage of their algorithms. A minimal interface is provided as shown Figure 3.3. There are three important factors when

[3] http://en.wikipedia.org/wiki/Facade_pattern/

working with algorithms: the initialisation, the recognition and the registration of an event manager.

When an algorithm is initialised, an instance of the `Configuration` class needs to be provided. The `Configuration` contains gesture sets, event manager and algorithm-specific parameters. The parameters are `key/value` pairs organised in a `Collection`. The configuration object can be created in Java code or stored in an XML document. Examples are provided in Appendix B. New algorithms need to implement the `Algorithm` interface. All `Algorithm`s are responsible for validating the configuration objects provided during initialisation. Algorithm implementations need to make use of the minimal accuracy and the maximal result set size stored in the configuration object. Algorithms need to notify the event manager in case of positive recognitions. The `AlgorithmFactory` class provides static methods to create algorithms with a configuration instance and uses dynamic class loading to instantiate the algorithms.

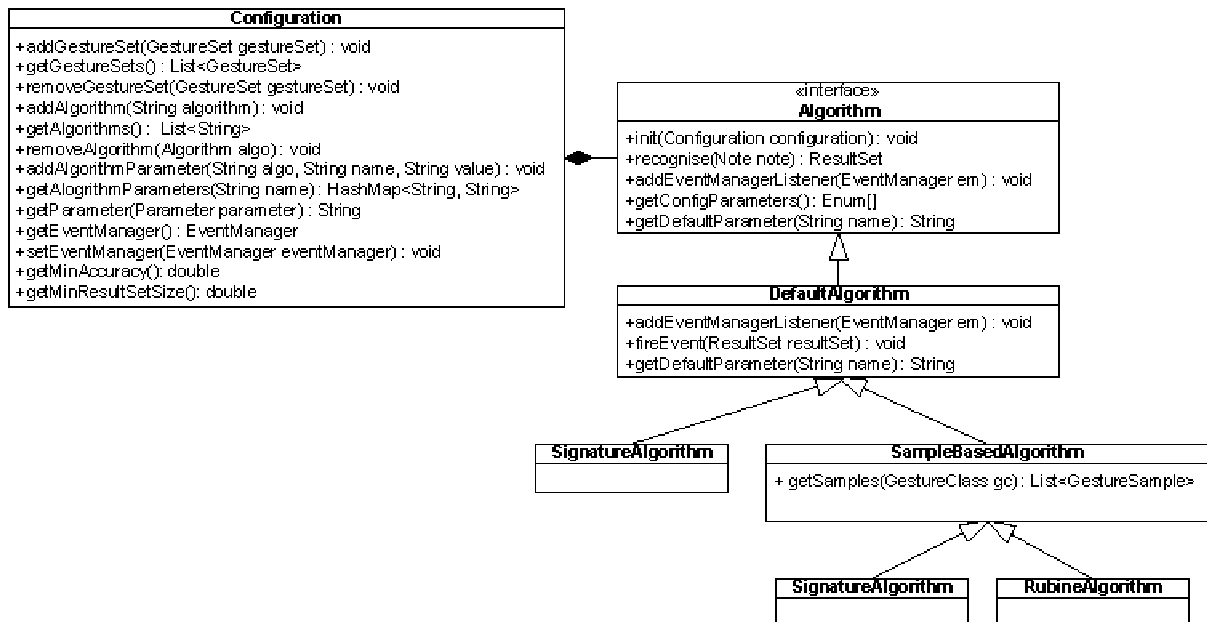

Figure 3.3: Algortihm API

iGesture supports different input devices. To support these different devices without relying on their specific hardware specifications, the `Device` and `GestureDevice` interfaces have been created. All new devices need to implement these interfaces before they can be used in iGesture as shown in Figure 3.4. When an application wants to use a certain device, it must register itself as a `GestureEventListener` with this device. If a gesture is performed with this device, the listener will inform all registered applications, passing the captured gesture sample. The application can then evaluate the gesture sample based on application-specific logic before passing it to a `Recogniser`. When the `Recogniser` has evaluated a gesture sample, it notifies all registered

`EventManagers`. An `EventManager` implements the `GestureHandler` interface to specify what should happen when a certain gesture is recognised.

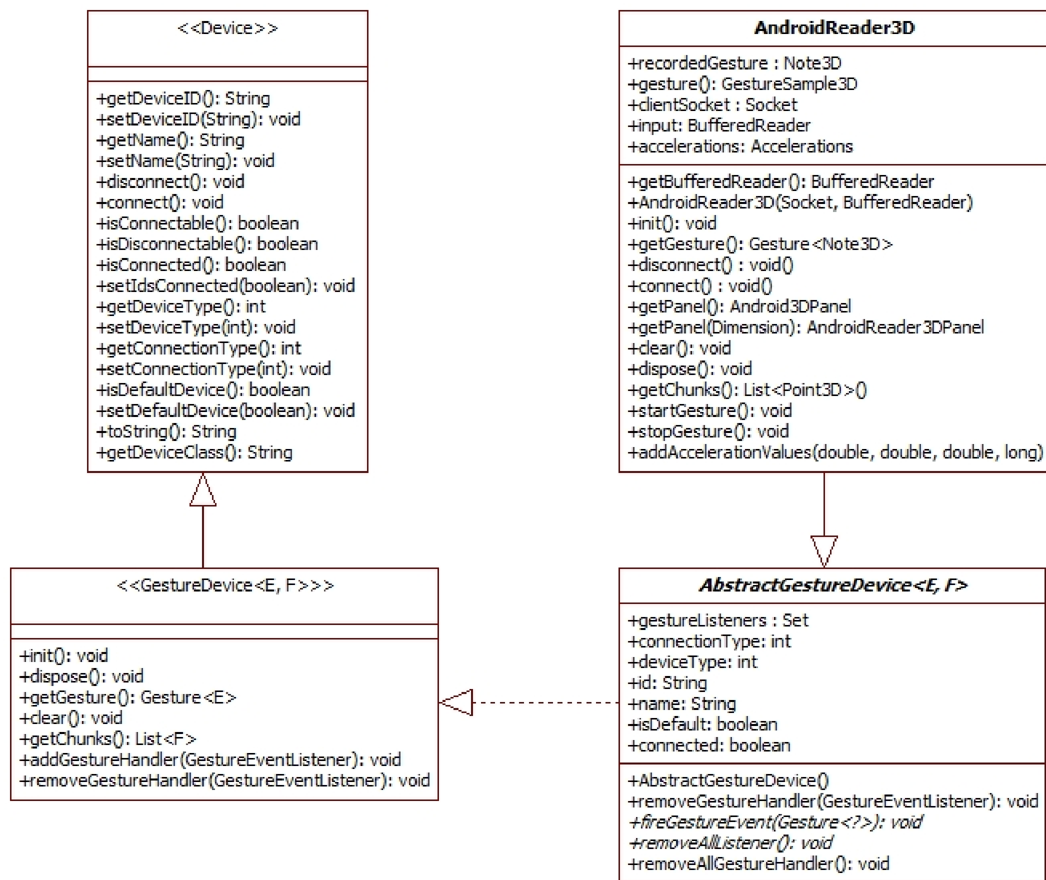

Figure 3.4: Device implementation

3.1.2 iGesture Tools

The iGesture framework contains a tool to help users define and test gesture sets. These tools are grouped in the iGesture Workbench. In the file menu, workbench projects can be created, opened and saved. Once a project is opened or created, 5 tabs are added to the main window: Gesture Set, Test Bench, Batch Processing, Test Set and Composite Test Bench. In the toolbar, the Device Manager can be found under Devices. The Device Manager is used to connect and disconnect new devices to the workbench. Currently, TUIO 3D, Bluetooth and Tuio 2D are supported. The mouse is automatically added as a 2D input device.

As just explained earlier, there are different gesture descriptors in iGesture. Therefore, different representations of these descriptors are available. In the left frame of Figure 3.8, it is shown how to create a gesture set with different descriptors. In the right bottom screen, the user has

an overview of all devices that can be used in combination with the selected descriptor. When a device is selected, a gesture can be performed. Once the gesture is finished, it is visualised in the bottom right corner. The top right frame shows all the gesture samples for the selected descriptor.

The test bench shown in Figure 3.6 is used to test a gesture recognition with an arbitrary sample. On the left-hand side, an overview of the available recognition algorithms is given. For each algorithm, multiple configurations can be created. This allows developers to quickly test configuration parameters when evaluating the training gesture set. A training gesture set can be created in the bottom right frame

The test set tab shown in Figure 3.8 can be used to create different sets of gestures. These sets of gestures are meant to be used during batch processing. An overview of the batch processing settings can be seen in Figure 3.7. Batch processing is a function in iGesture used to optimise different algorithm parameters to achieve optimum recognition rates. Batch processing permutes all the algorithm's parameters within predefined boundaries. These boundaries must be provided in a batch processing configuration file. A batch processing configuration file must be created manualy, an example can be found in Appendix B.12. The gesture set used during batch processing, is a set of gestures created in the gesture set tab. This set of gestures is used to recognise the test set that was created in the test set tab. The output will generate an HTML result file containing the recognition rates for all the configuration permutations.

Finally, the composite test bench seen in Figure 3.9 can be used to test composite gestures. Composite gestures are gestures performed using multiple devices.

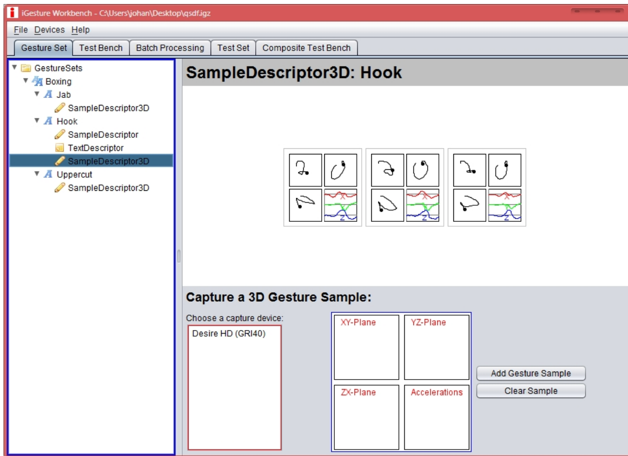

Figure 3.5: Workbench Gesture Set

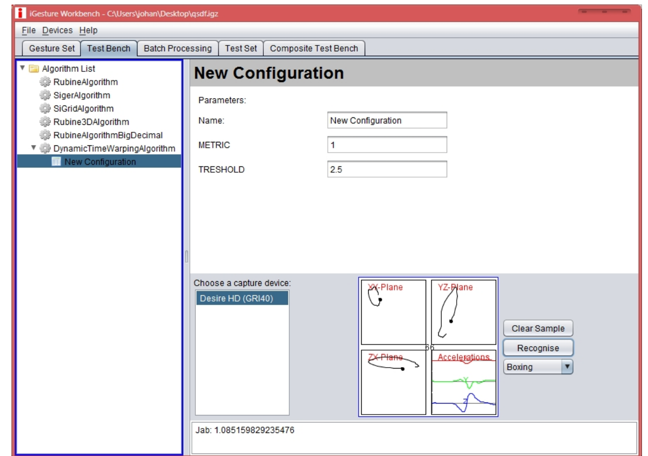

Figure 3.6: Workbench Test Bench

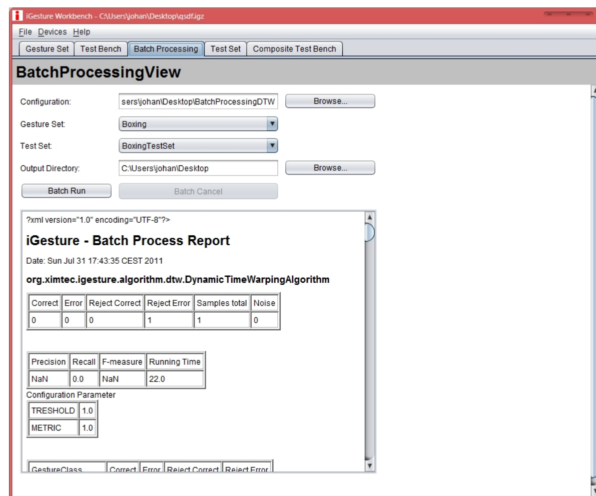

Figure 3.7: Workbench Batch Processing

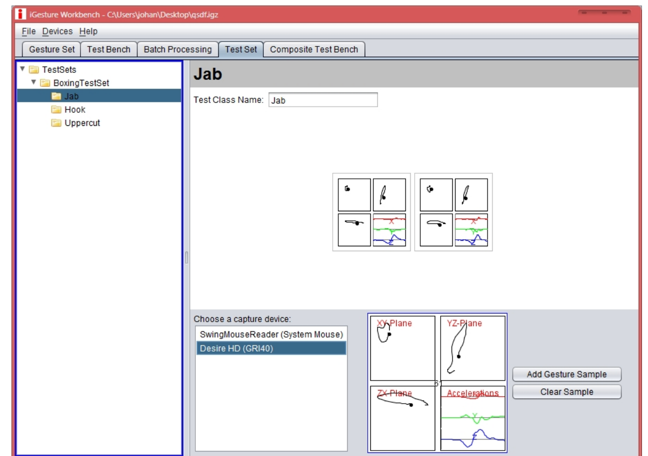

Figure 3.8: Workbench Test Set

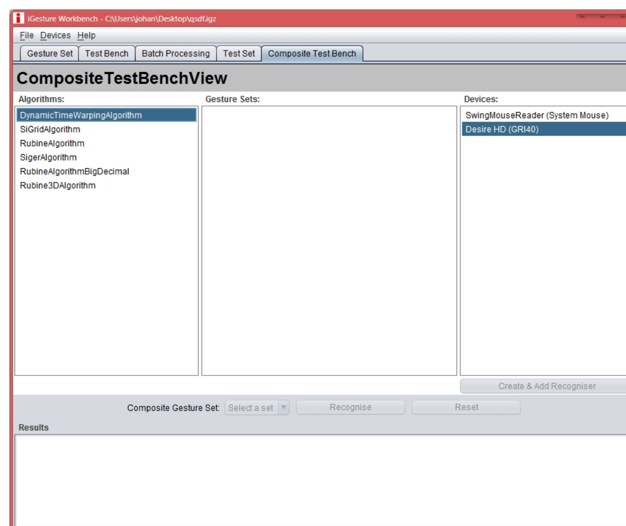

Figure 3.9: Workbench Composite Test Bench

3.2 A New Gesture Recognition Algorithm

As part of this thesis, a new gesture recognition algorithm has been developed. This algorithm is a combination of the k-Nearest Neighbour algorithm and the Dynamic Time Warping algorithm.

3.2.1 *k*-Nearest Neighbour

A naive version of the k-nearest neighbour algorithm has been implemented to classify a sample gesture. A sample gesture will be matched against all training samples to define its best match. The best match will determine the sample's classification ($k=1$). The metric used to define the nearest neighbour is Dynamic Time Warping.

3.2.2 Dynamic Time Warping

Dynamic time warping (DTW) is a computer algorithm designed to measure the difference/similarity between two sequences which may vary in time. DTW allows a computer to find a match between two time series. DTW is a commonly used algorithm in speech [10], image [4, 16], handwriting [13] and accelerometer gesture recognition. The sequences are 'warped' non-linearly in the time dimension. This results in a measurement of the non-linear variations in the time dimension. Intuitively, this means that it should not matter whether a person draws a square fast or slow, the performed gesture should still be recognised as a square.

The DTW algorithm generates an $M \times N$ matrix. N is the amount of values defining the test sample. M is the amount of values defining the training sequence. Each data entry the matrix shows the difference between two acceleration events. The entire matrix represents the difference of all acceleration samples between the test sequence and the training sequence.

Table 3.1: DTW example matrix

	1	8	19	32	41	48	57	65	77	88
0	1	8	19	32	41	48	57	65	77	88
10	9	2	9	22	31	38	47	55	67	78
20	19	12	1	12	21	28	37	45	57	68
30	29	22	11	2	11	18	27	35	47	58
40	39	32	21	8	1	8	17	25	37	48
50	49	42	31	18	9	2	7	15	27	38
60	59	52	41	28	19	12	3	5	17	28
70	69	62	51	38	29	22	13	5	7	18
80	79	72	61	48	39	32	23	15	3	8
90	89	82	71	58	49	42	33	25	13	2

Once the matrix is generated, the shortest path in the matrix is calculated with the following constraints:

- The endpoints of the two time series must match. This means that the path goes from $[x_0, y_0]$ to $[x_n, y_m]$.
- The use of the Manhattan distance metric[4] is a valid option. However, the best path probably lies near the diagonal. This can be reflected in the algorithm by applying a cost of 0.5 for a diagonal move and a cost of 1 for a horizontal or vertical move.
- Time only moves forward. This implies that from the start point in the upper left corner, one can only go right, down or down-and-right.

To compute the shortest path, a dynamic programming[5] algorithm is used. The start to end path finding problem is subdivided into smaller problems, the shortest path from any point in the matrix to the last point in the matrix. If one would start at the beginning, $[x_0, y_0]$ some sort of brute force backtracking algorithm is needed. Instead, when starting from $[x_n, y_m]$ an optimum distance measurement can be calculated for each element in the matrix. When looping trough the matrix bottom up, all one needs to do is to evaluate the values one position down,

[4]http://en.wikipedia.org/wiki/Taxicab_geometry/
[5]http://en.wikipedia.org/wiki/Dynamic_programming/

one position tot the right and the value diagonal down/right. In this way, the computational complexity of the shortest path is reduced to $O(n^2)$.

Listing 3.1: Dynamic Time Warping shortest path algorithm

```java
/**
 * Calculate shortest path in the matrix from 0;0 to i;j
 * (top left -> down right)
 *
 * The algorithm starts at i;j and works its way up to 0:0 with the
 * following constraints:
 *
 * 1. diagonal move counts as 0.5 whereas down or left move counts as 1
 * 2. only go forward in time: down and right move is not allowed
 * (0:0 is up/left and i:j down/right)
 * 3. the path must go from 0:0 to i:j
 *
 * @param matrix
 * @return
 */
private float calculateShortestPath(Float[][] matrix) {
    int rowCount = matrix.length - 1;
    int columnCount = matrix[rowCount].length - 1;

    // traverse the matrix from right->left
    for (int column = columnCount; column >= 0; column--) {
        // down->up
        for (int row = rowCount; row >= 0; row--) {

            if (column == columnCount && row == rowCount) {
                // last point in path, the lowest remaining cost till the end is
                // this cost

            }
            else if (column == columnCount) {
                // last column, the lowest cost to the end is on down 1 position
                matrix[row][column] += matrix[row + 1][column];

            }
            else if (row == rowCount) {
```

```
36                  // last rows, the lowest cost to the end is on it's right side
37                  matrix[row][column] += matrix[row][column + 1];
38
39              }
40              else {
41                  // pick the lowest values down/right/(diagonal*0.5)
42                  float down = matrix[row + 1][column];
43                  float right = matrix[row][column + 1];
44                  float diagonal = (float)(matrix[row + 1][column + 1] * 0.5);
45
46                  matrix[row][column] += min(down, right, diagonal);
47              }
48          }
49      }
50
51      return matrix[0][0];
52  }
```

If we calculate the shortest path for the given matrix shown in Figure 3.10 with the described constraints, we get a shortest path measurement of: 2.7265625.

Figure 3.10 shows the recognition rates at different threshold values for a set of 225 gestures performed by one user. The threshold value is the maximum path value for positive gesture recognition. As one can see, the square and z gestures have lower threshold values for positive classification then the roll and circle gestures. All gestures were classified at a threshold value of 1.6. Figure 3.11 shows the same threshold calculation as in Figure 3.10, except that it is performed with a different gesture set. This gesture set contains the same gesture types as the previous set, however all gestures where performed at different speeds, gesture sizes and orientations. The purpose of this gesture set is to train the algorithm to match gestures performed by users with different interpretations of the gesture set. With this training set it does not matter how fast or how large the user is performing the gestures or even at what angle he is holding the device. The algorithm will still recognise the performed gestures. This explains why gesture recognition is only complete at a threshold of 5.3 instead of a threshold around 1.6. This gesture set is probably not useful in a real situation since application developers generally only use an ideal gesture set for which the user goes through a learning curve to try to execute it as close as possible to the original training set.

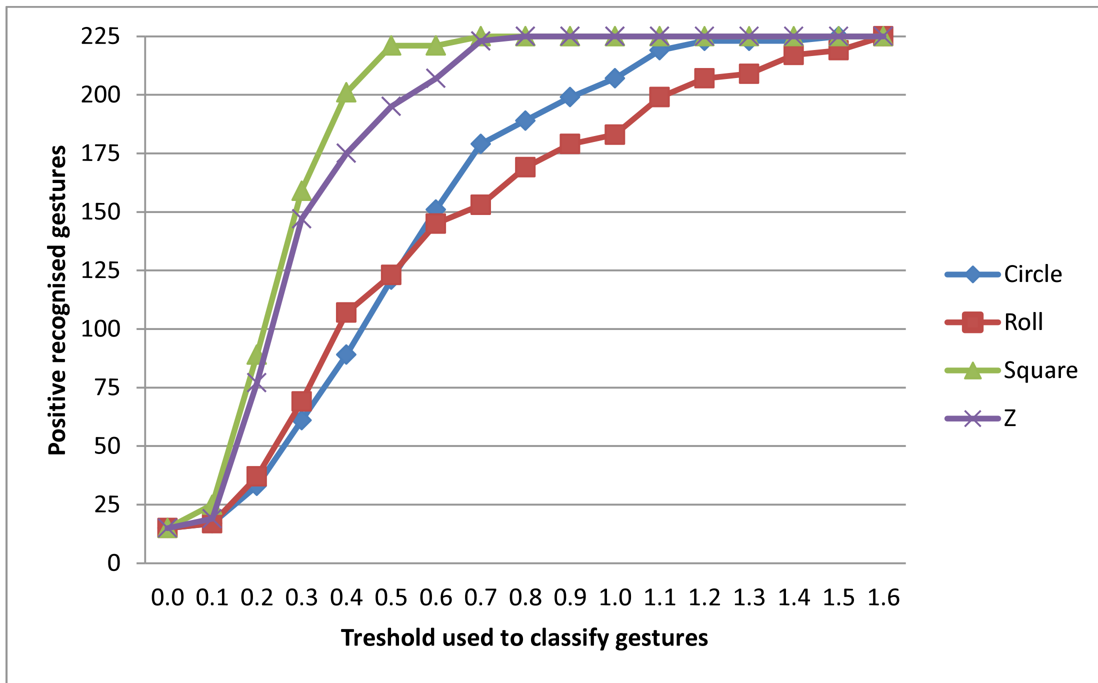

Figure 3.10: User gesture shortest path lengths, 225 calculations for each gesture

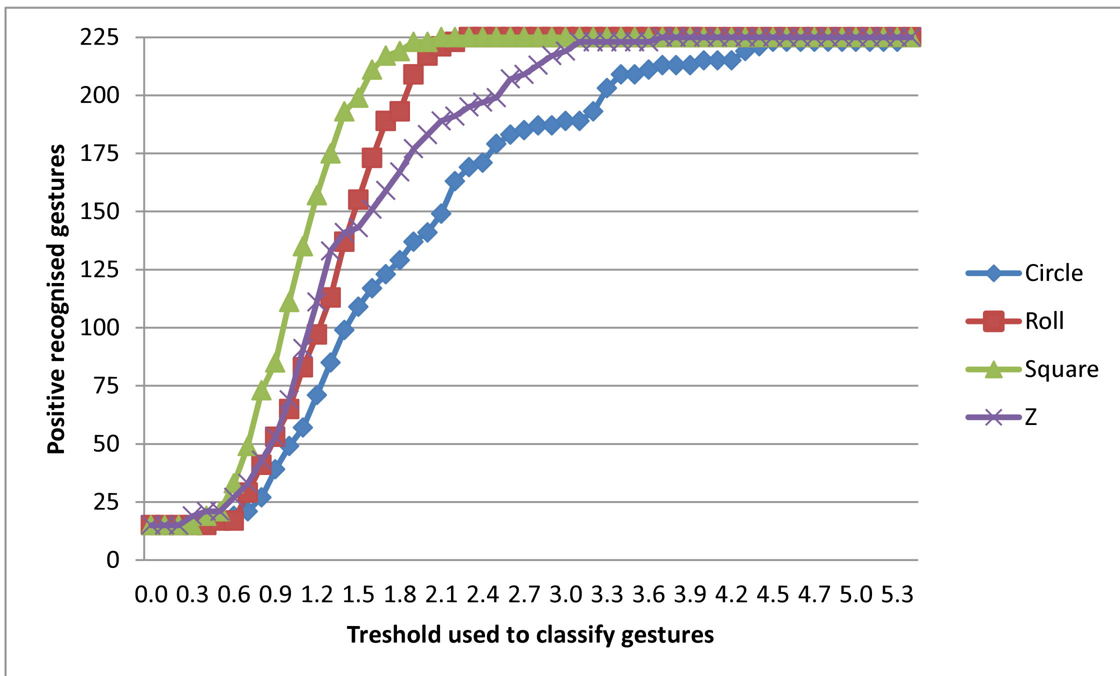

Figure 3.11: Develop gesture shortest path lengths, 225 calculations for each gesture

Chapter 4

Implementation Details

In this chapter, we will give an overview of the changes that have been applied to the iGesture framework.

4.1 Dynamic Time Warping Acceleration Metrics

The DTW algorithm relies on subtracting two acceleration events (`x,y,z`) in order to get a number representing the difference between these two acceleration events. Different metrics[1] can be defined to calculate this acceleration event subtraction. The following subsections describe the different metrics used in combination with Dynamic Time Warping.

4.1.1 Euclidean Norm Metric

The Euclidean norm is defined by Equation 4.1. If the vector v is a 3-tuple, the Euclidean Norm is defined as in Equation 4.2

$$||v|| = \sqrt{< v, v >} \tag{4.1}$$

$$||v|| = \sqrt{v_1 \cdot v_1 + v_2 \cdot v_2 + v_3 \cdot v_3} \tag{4.2}$$

[1] http://en.wikipedia.org/wiki/Metric_(mathematics)/

Listing 4.1: Code for the Euclidean Norm metric

```
/**
 * Calculates AccelerationSample EuclideanNorm:
 * sqrt(x^2+y^2+z^2)
 *
 * This technique eliminates device rotation and gravity
 *
 * @param challenge
 * @return
 */
private double euclideanNorm(AccelerationSample a, AccelerationSample b) {
  return Math.sqrt((a.getXAcceleration() * a.getXAcceleration())
              + (a.getYAcceleration() * a.getYAcceleration())
              + (a.getZAcceleration() * a.getZAcceleration()))
        - Math.sqrt((b.getXAcceleration() * b.getXAcceleration())
              + (b.getYAcceleration() * b.getYAcceleration())
              + (b.getZAcceleration() * b.getZAcceleration()));
}
```

The Euclidean Norm measurement has been designed to be orientation independent. This algorithm should be a valid solution for the earlier described tilting problem. When the Euclidean Norm is calculated, a graph can be drawn showing the gesture's characteristic function. These graphs in Figures 4.1, 4.2, 4.3, 4.4 represent the gesture for the number 1 performed with a HTC Desire HD using only raw acceleration values. The fourth graph is drawn from values captured from the device while holding it with approximately 45 degrees roll.

One can visually confirm that the functions in Figures 4.1, 4.2, 4.3 and 4.4 are good candidates for Dynamic Time Warping.

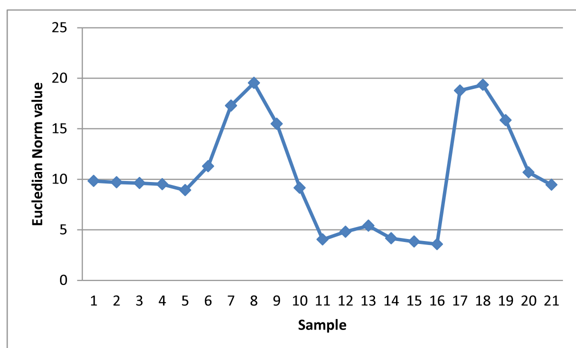

Figure 4.1: DTW of the number 1 - Euclidean Norm

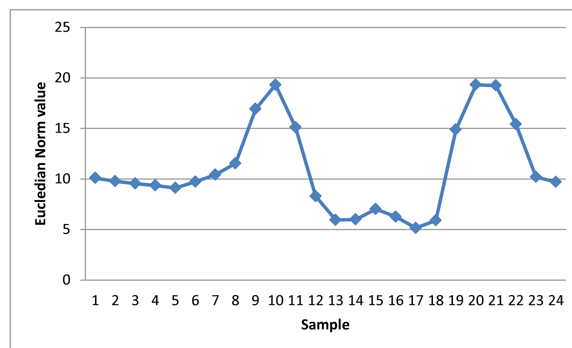

Figure 4.2: DTW of the number 1 - Euclidean Norm

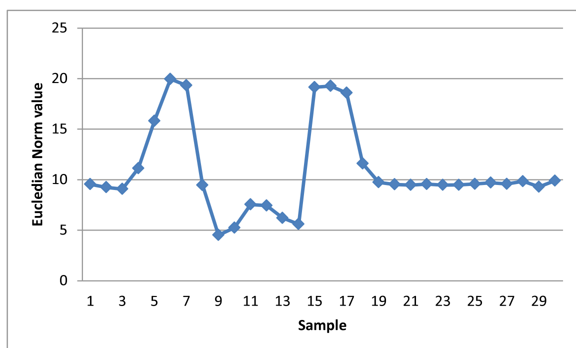

Figure 4.3: DTW of the number 1 - Euclidean Norm

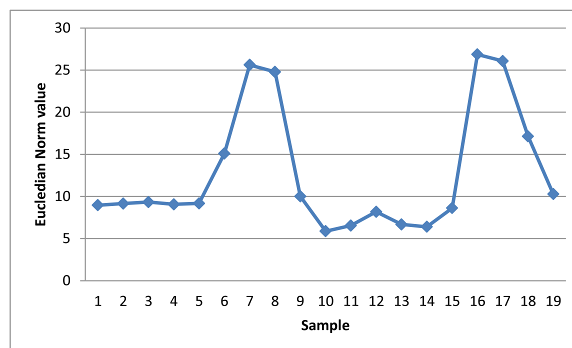

Figure 4.4: DTW of the number 1 - Euclidean Norm

4.1.2 Plain x,y,z Metric

The plain x,y,z difference algorithm is the most simple algorithm. This algorithm's formula is defined as shown in Equation 4.3. Note that this metric is sensitive for the tilting problem explained earlier in Section 2.4.1.

$$|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2| \tag{4.3}$$

Listing 4.2: Code for plain x y z metric

```
1  /**
2   * Calculate the most naive difference between two AccelerationSamples
3   *
4   * |x1-x2| + |y1-y2| + |z1-z1|
5   *
6   * @param base
7   * @param challenge
8   * @return double plainDiff
9   */
10 private double plainDiff(AccelerationSample base, AccelerationSample challenge) {
11     return Math.abs(base.getXAcceleration() - challenge.getXAcceleration())
12             + Math.abs(base.getYAcceleration() - challenge.getYAcceleration())
13             + Math.abs(base.getZAcceleration() - challenge.getZAcceleration());
14 }
```

4.1.3 DTW for the x,y,z Axes Separately

The third version of the difference function is a full DTW calculation for each acceleration plane. This algorithm is more CPU demanding due to the triple execution of the path finding algorithm. The sum of these three warping distances is the DTW performance index. This algorithm is also sensitive to the tilting problem, but not with the same magnitude as the plain x,y,z metric.

4.2 IP Communication

In this section, the specification and implementation of our IP based communication protocol is explained.

4.2.1 Architecture

IP communication is possible on all devices supporting the standard IP stack. On most mobile devices like smartphones, handhelds and tablets, a Wi-Fi connection is available. These Wi-Fi capabilities can be used to set up a connection to a computer running iGesture as shown in Figure 4.5. The only requirement is that a connection to the PC running iGesture can be opened on port 80. IP support is implemented into iGesture and an Android application is developed to make a connection to iGesture. The software running on the device decides whether it sends its sensor data to the computer or not.

This Android application can be used to train and analyse iGesture recognisers using the iGesture tools on a PC while using sensor data coming from an Android cellphone. The communication with the Android application can also be integrated into a custom PC application. This PC application can then be controlled using the Android device. If this is the case, it is advised to redesign the Android application interface to match the PC application interface.

Figure 4.5: iGesture IP communication architecture

4.2.2 Message protocol

One of the goals of this master's thesis was to implement additional input devices. Due to Android's high market penetration and its low development requirements, we have chosen to add Android support to iGesture. Android support extends the existing supported input devices with all Android based devices. In order to use an Android device as input device, a messaging system between an Android cellphone and a computer running iGesture had to be implemented. Almost all computers and Android phones have IP support (Wi-Fi/cable network) making the IP protocol an ideal candidate for sending and receiving messages. The default IP stack is used to implement a socket connection listening on port 80. The first message sent to the host computer is the devices unique identifier, `android.os.Build.MODEL + ':' + android.os.Build.ID`. All the following commands must follow strict formatting guidelines:

Code	Value	Meaning	Example
S	Start of Gesture!	indicating that the device will record and send a new gesture	`S - Start of Gesture!`
Q	End of Gesture!	indicating that the recorded gesture is finished	`Q - End of Gesture!`
A	x:y:z:timestamp	xyz values for the measured acceleration values (gravity is filtered)	`A - 1.1727:-2.0887: 10.392:3193478543000`

This messaging system can be used with any device supporting motion sensing and IP based communication. An Android application has been developed to support this messaging system on all Android-based devices. This Android application is responsible for transforming the captured sensor data to comply with the massage format. If other users would like to support other devices like iPhone, Windows phones, or other custom IP based hardware, it is sufficient to implement the previously explained message commands.

Listing 4.3: Process data coming from an IP device according to the message protocol specifications

```
1
2
3 package org.ximtec.igesture.io.android;
4
5 import java.io.IOException;
6
7
```

```
8 /**
9  *
10 * @author Johan Bas
11 *
12 */
13 public class AndroidStreamer extends Thread {
14
15     private static final String PARSE_ACCELERATION = "A";
16     private static final String PARSE_IDLE = "I";
17     private static final String PARSE_QUIT = "Q";
18     private static final String PARSE_STOP = "S";
19     private static final String PARSE_BYE = "Bye.";
20     private boolean recording = false;
21     private AndroidReader3D device;
22
23
24     public AndroidStreamer(AndroidReader3D device) {
25         this.device = device;
26     }
27
28
29     @Override
30     public void run() {
31         String inputLine;
32
33         try {
34             while ((inputLine = device.getBufferedReader().readLine()) != null) {
35
36                 if (inputLine.equals(PARSE_BYE)) {
37                     System.out.println(PARSE_BYE);
38                     break;
39                 }
40                 else if (inputLine.startsWith(PARSE_STOP)) {
41                     this.device.startGesture();
42                     recording = true;
43                 }
44                 else if (inputLine.startsWith(PARSE_QUIT)) {
45                     // end gesture
46                     this.device.stopGesture();
```

```
47                    recording = false;
48                }
49                else {
50                    if (recording) {
51                        if (inputLine.startsWith(PARSE_IDLE)) {
52                            // idle state
53                        }
54                        else if (inputLine.startsWith(PARSE_ACCELERATION)) {
55                            inputLine = inputLine.substring(4);
56                            String[] coordinated = inputLine.split(":");
57
58                            double x = Double.parseDouble(coordinated[0]);
59                            double y = Double.parseDouble(coordinated[1]);
60                            double z = Double.parseDouble(coordinated[2]);
61                            long time = Long.parseLong(coordinated[3]);
62
63                            this.device.addAccelerationValues(x, y, z, time);
64
65                        }
66                    }
67                }
68            }
69        }
70        catch (IOException e) {
71            e.printStackTrace();
72        }
73    }
74 }
```

Listing 4.4: Android application responsible for processing and sending acceleration sensor data

```
1   /**
2    * Try to connect to given server IP
3    */
4   private void connectToServer() {
5       try {
6           InetAddress serverAddr = InetAddress.getByName(serverIP);
7           Log.d("IGesture", "Connecting... " + serverAddr + ":" + serverPort);
8           socket = new Socket(serverAddr, serverPort);
9
```

```
10            try {
11                Log.d("IGesture", "Sending command.");
12                socketWriter = new PrintWriter(new BufferedWriter(
13                        new OutputStreamWriter(socket.getOutputStream())), true);
14
15                connected = true;
16                socketWriter.println(android.os.Build.MODEL + ":"
17                        + android.os.Build.ID);
18                TextView msg = (TextView)findViewById(R.id.Msg);
19                msg.setText("CONNECTED");
20            }
21            catch (Exception e) {
22                connected = false;
23                Log.e("IGesture", "Error: ", e);
24            }
25        }
26        catch (Exception e) {
27            connected = false;
28            Log.e("IGesture", "Error: ", e);
29        }
30    }
31
32    @Override
33    public void onSensorChanged(SensorEvent event) {
34        switch (event.sensor.getType()) {
35
36            case Sensor.TYPE_LINEAR_ACCELERATION: // sensor fusion
37                if (record)
38                    socketWriter.println("A - " + event.values[0] + ":"
39                            + event.values[1] + ":" + event.values[2] + ":"
40                            + event.timestamp);
41
42                break;
43
44            default:
45                break;
46
47        }
48    }
```

Figure 4.6 shows the iGesture device manager with IP support. The code used to discover IP based devices can be found in Appendix B.4. Figure 4.8 shows the initial screen of the Android application. The IP address of the computer running iGesture needs to be entered. When pressing connect, iGesture's device manager will automatically proceed to the device discovered screen as shown in Figure 4.7. The Android application will go to the connected screen as seen in Figure 4.9. When tapping the capture button, the Android application will begin streaming sensor data to iGesture starting with `S - Start of Gesture!`. When tapping this button again, the Android application will stop sending data and the last command will be `Q - End of Gesture!`. This will enable iGesture to automatically draw the received gesture. In order to test the Android application, a small standalone Java application, the socket server shown in Figure 4.10 has been created. This socket sever enables developers to debug the communication protocol when implementing IP support on other devices.

The Android application uses sensor fusion (see Section 2.4.2) to optimise acceleration data. The source code for the complete application can be found in Appendix B.5. The `onSensorChanged` method only uses the `Sensor.TYPE_LINEAR_ACCELERATION` sensor. When using only this sensor, sensor fusion is automatically performed by Android[2]. Android will perform sensor fusion on a low, systems level. Processing sensor data at the kernel level will result in more accurate sensor data.

[2] http://developer.android.com/reference/android/hardware/SensorEvent.html#values

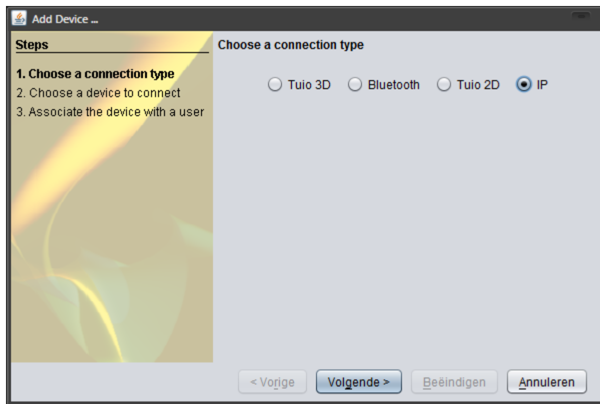

Figure 4.6: Device manager IP protocol

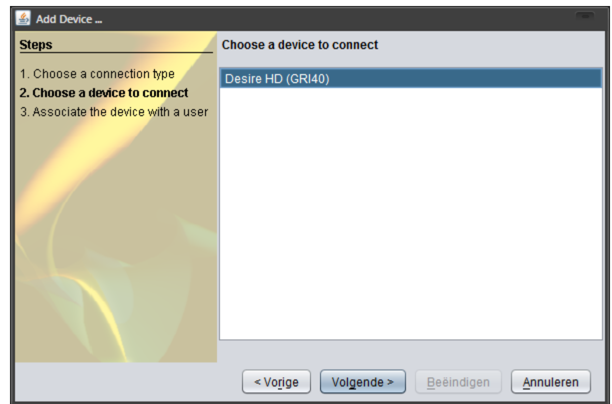

Figure 4.7: Device Manager IP device discovery

Figure 4.8: AndGesture connection screen

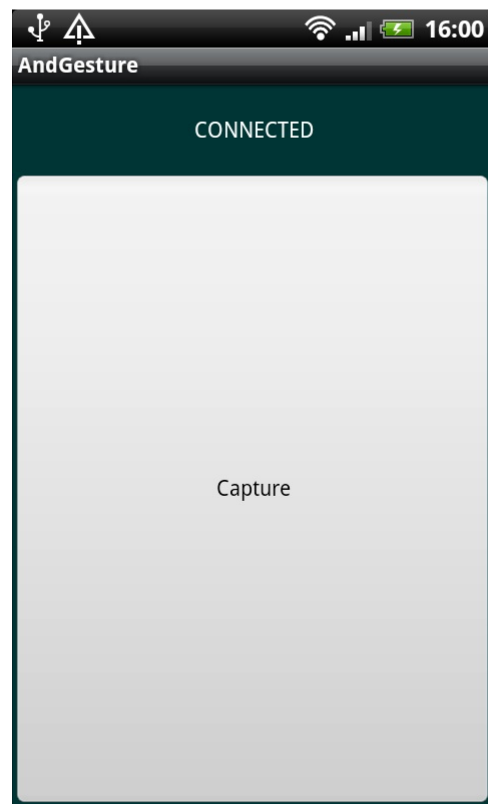

Figure 4.9: AndGesture connected screen

Figure 4.10: Socket server developed to test the Android application showing gesture data from one gesture

4.3 Proof of Concept Application

A proof of concept application has been developed (see Figure 4.11). This application is able to recognise two distinct gestures. It can detect if the cellphone is moved sideways, or if the cellphone is flipped over. When a gesture is recognised, it is printed at the bottom of the screen together with the shortest path value. These gestures are recognised using Dynamic Time Warping and the plain difference metric as explained in Section 4.1.2. The DTW algorithm is trained using ten gestures, six flip gestures and four sideways gestures. The flip gesture is recognised when a threshold of four is reached, the sideways gesture is recognised when a threshold of five is reached. The DTW algorithm is executed on the last twenty acceleration events every four acceleration events. The sensors sampling rates are set to `SensorManager.SENSOR_DELAY_GAME`. The high sensing rate in combination with the constant drawing of the three graphs uses a lot of the cellphone CPU resources. It is due to these resource limitations that only ten training samples are used.

This applications is an example of sensor fusion on Android. When performing the same move-

ment under different device orientations, the graphs will show the same acceleration readings. The cellphone accomplishes this using the magnetic sensor and, if available, the gyroscope. When a gyroscope is not present, gravity is filtered out using high pass and low pass filters. This explains small delays in the graph representation and gesture recognition.

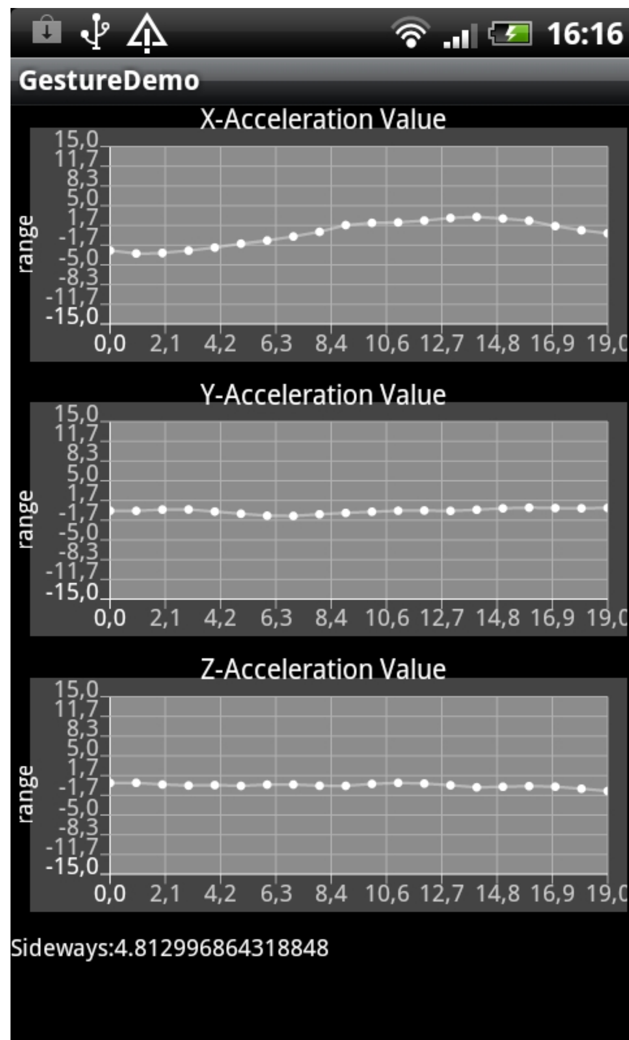

Figure 4.11: Proof of concept Android application

Chapter 5

Evaluation

Different gesture sets have been used to evaluate the implemented Dynamic Time Warping algorithm. In this section, we will discuss the gesture sets together with the test results.

5.1 WiiGee Gestures

As reference gestures, the four gestures shown in Figure 5.1 will be used. These are the same gestures that have been used in the WiiGee project, except for the *tennis gesture* due to its ambiguous interpretation. These gestures have been executed by 8 persons, female and male aged between 18 and 55. Test persons have average to no Wii gaming experience. The raw values have been stored to simulate and test different algorithm configurations.

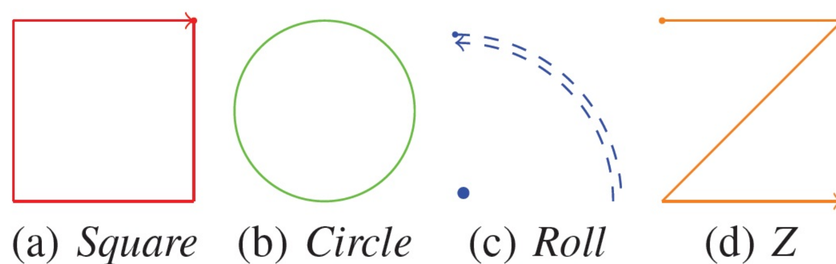

Figure 5.1: Sample Gestures

WiiGee uses a Hidden Markov Model-based algorithm [19] for its gesture recognition. We have used the same gesture set and the same amount of gesture samples as WiiGee. Table 5.1 shows the recognition results for Dynamic Time Warping using three classifications, one for each axis. Table 5.2 shows the recognition results for the plain difference version of the Dynamic Time

Warping algorithm. Finally, table 5.3 shows the recognition rates for the Euclidean Norm version of the Dynamic Time Warping algorithm.

Table 5.1: Recognition rates for the 3 axes separated DTW algorithm

3-axes DTW	Circle	Roll	Square	Z
Circle	92.5%	0%	5%	2.5%
Roll	0%	100%	0%	0%
Square	4.17%	0%	94.17%	0.83%
Z	3.33%	0%	3.33%	93.33%

Table 5.2: Recognition rates for the plain difference DTW algorithm

Plain Diff. DTW	Circle	Roll	Square	Z
Circle	90%	0%	5.83%	4.17%
Roll	0%	100%	0%	0%
Square	5.83%	0%	92.5%	0.83%
Z	3.33%	0%	3.33%	93.33%

Table 5.3: Recognition rates for the Euclidean Norm DTW algorithm

Euclidean Norm. DTW	Circle	Roll	Square	Z
Circle	77.5%	3.33%	14.17%	5%
Roll	5.83%	80.83%	2.5%	10.83%
Square	15.83%	3.33%	75%	5.83%
Z	8.33%	5%	11.67%	75%

Figure 5.2 shows the recognition rates for these three versions of Dynamic Time Warping and the test results of WiiGee. From these results, we can conclude that the Euclidean Norm is not a good recogniser for the given test set. Furthermore, it can be seen that for circle, roll and square, Dynamic Time Warping outperforms WiiGee's Hidden Markov model algorithm. The results for the z gesture are close to each other.

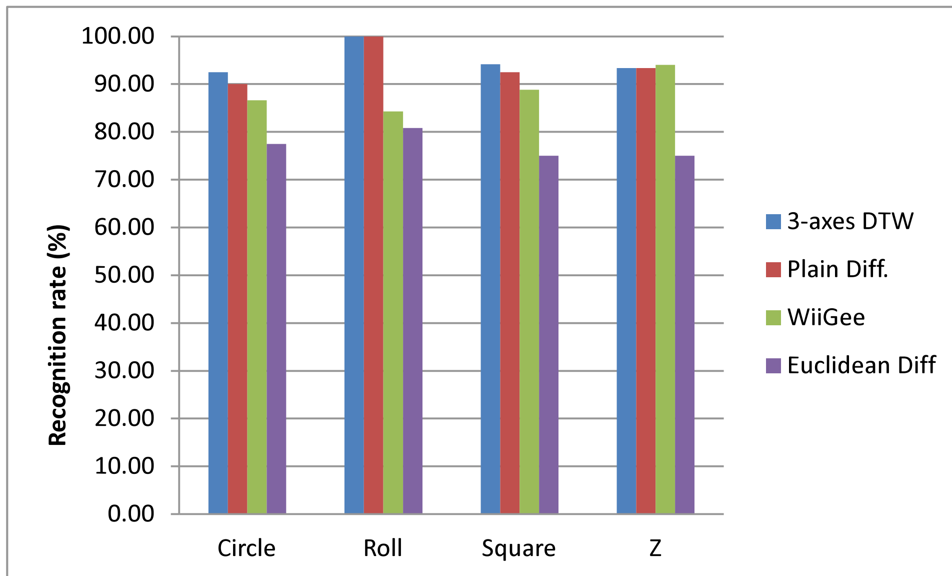

Figure 5.2: WiiGee recognition rates

Figure 5.3 shows the recognition rates per user, three of them had litle or none experience with the Wii Remote. However, five users show recognition rates of 95% and higher. Two users are having recognition rates around 90%. Notice that for these two users the Euclidean Norm seems to be a valid recognising algorithm. For one user, the recognition rates are around 85%.

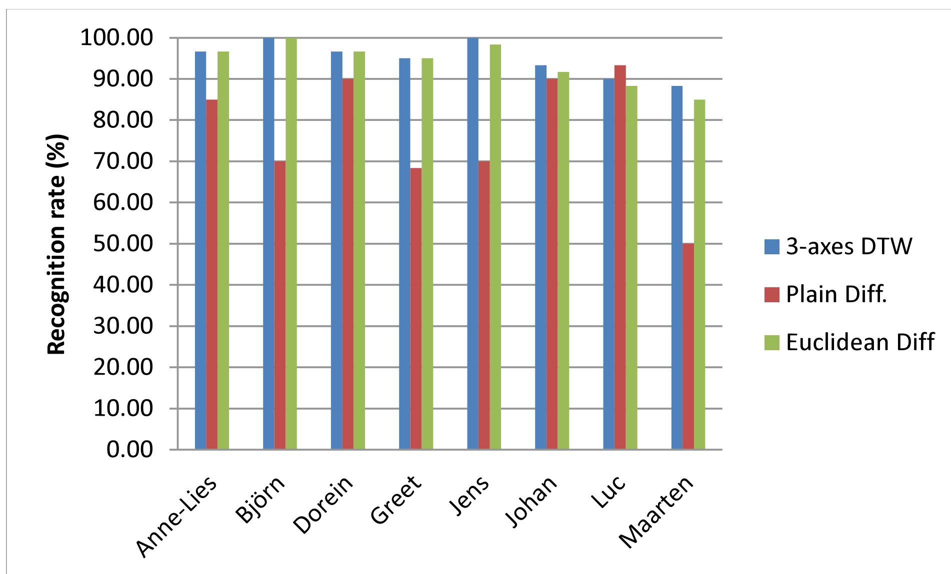

Figure 5.3: WiiGee user recognition rates

5.2 Boxing gestures

The gestures used to compare this algorithm to the WiiGee implementation are 2D gestures performed with a 3D device. We therefore used a second set of gestures to test the Wii Remote. This second set consists out of 5 different boxing gestures as used in Nintendo Wii Sport[1]:

1. A *Jab*, a straight forward punch *to the head* as shown in Figure 5.4.
2. A *Jab to the body.*
3. A *Hook*, a sideways punch *to the head* as shown in Figure 5.5.
4. A *Hook, to the body.*
5. An *Uppercut*, an upwards punch targeting the opponents chin, as shown showed in Figure 5.6

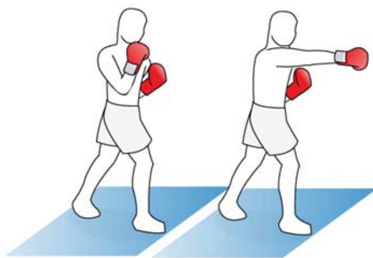

Figure 5.4: Jab

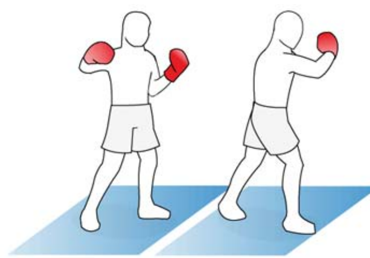

Figure 5.5: Hook

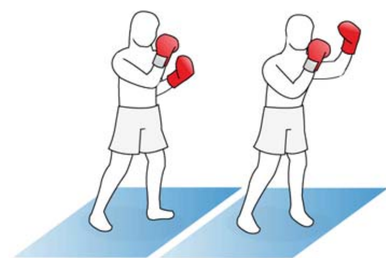

Figure 5.6: Uppercut

Three different people, one right handed female, age 24, one right handed male, age 29 and one left-handed male, age 28 have performed these gestures 15 times for each hand. This gives a total of 90 training samples for each gesture. The interesting question concerning this gesture is, *When do we classify a gesture as a false positive?* If a Jab to the head is classified as a Jab to the body, the algorithm is only partly wrong. Therefore, two sets of results are presented and compared. One set where the five gestures are evaluated differently and one where the Jabs and Hooks count as one gesture.

[1]images courtesy of http://www.talkboxing.co.uk/guides/boxing_moves.html

Tables 5.4, 5.5 and 5.6 show the recognition results for the three versions of the Dynamic Time Warping algorithm using five gesture classifications.

Table 5.4: Boxing, 3-axes DTW for 5 gestures

3-axes DTW	Face Jab	Body Jab	Face Hook	Body Hook	Uppercut
Face Jab	96.67%	3.33%	0%	0%	0%
Body Jab	7.78%	86.67%	1.11%	1.11%	3.33
Face Hook	1.11%	0%	78.89%	18.89%	1.11%
Body Hook	0%	1.11%	15.56%	77.78%	5.56
Uppercut	6.67%	4.44%	2.22%	4.44%	82.22

Table 5.5: Boxing, plain difference for 5 gestures

Plain Diff DTW	Face Jab	Body Jab	Face Hook	Body Hook	Uppercut
Face Jab	88.89%	6.67%	0%	0%	4.44%
Body Jab	5.56%	86.67%	1.11%	1.11%	5.56
Face Hook	0%	0%	78.89%	17.78%	3.33%
Body Hook	0%	1.11%	13.33%	80%	5.56%
Uppercut	7.78%	6.67%	1.11%	4.44%	80%

Table 5.6: Boxing, Euclidean Norm for 5 gestures

Euclidean Norm DTW	Face Jab	Body Jab	Face Hook	Body Hook	Uppercut
Face Jab	70%	15.56%	6.67%	3.33%	4.44%
Body Jab	18.89%	51.11%	16.67%	8.89%	4.44
Face Hook	10%	14.44%	45.56%	16.67%	13.33%
Body Hook	4.44%	13.33%	14.44%	60%	7.78%
Uppercut	4.44%	6.67%	14.44%	8.89%	65.56

Tables 5.7, 5.8 and 5.9 show the recognition rates for the three versions of the Dynamic Time Warping algorithm using only three gesture classifications.

Table 5.7: Boxing, 3-axes DTW for 3 gestures

3-axes DTW	Jab	Hook	Uppercut
Jab	97.22%	1.11%	1.67%
Hook	1.11%	95.56%	3.33%
Uppercut	11.11%	6.67%	82.22%

Table 5.8: Boxing, plain difference for 3 gestures

Plain Diff DTW	Jab	Hook	Uppercut
Jab	93.89%	1.11%	5%
Hook	0.56%	95%	4.44%
Uppercut	14.44%	5.56%	80%

Table 5.9: Boxing, Euclidean Norm for 3 gestures

Euclidean Norm DTW	Jab	Hook	Uppercut
Jab	77.78%	17.78%	4.44%
Hook	21.11%	68.33%	10.56%
Uppercut	11.11%	23.33%	65.56%

Figure 5.7 shows a good overview of gesture recognition using the 5 different gesture sets. One can see that the Euclidean Norm is not a good recogniser for the boxing gesture set. The highest recognition rate is only 70% and the hook to the face even drops below 50%. The two other recognition algorithms have more or less the same recognition rates.

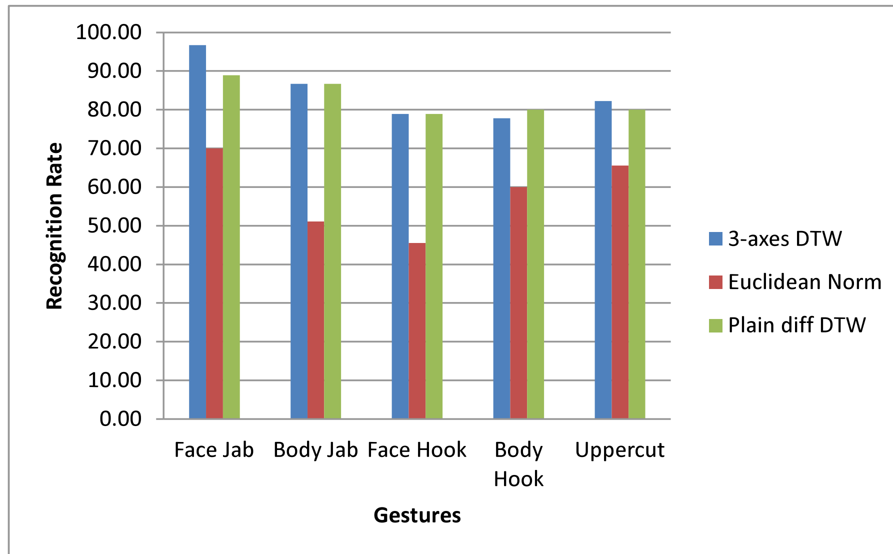

Figure 5.7: Recognition rates for Head Jab, Body Jab, Head Hook, Body Hook and Uppercut

Figure 5.8 shows the gesture recognition rates using 3 gesture sets. Here we can see that the recognition rates are much higher than the previous results. Again, the Euclidean Norm is the worst recogniser for this gesture set. The two other recognisers manage to get recognition results around 95% for the Jab and the Hook. The uppercut scores around 80%, mostly due to false classifications of Jabs.

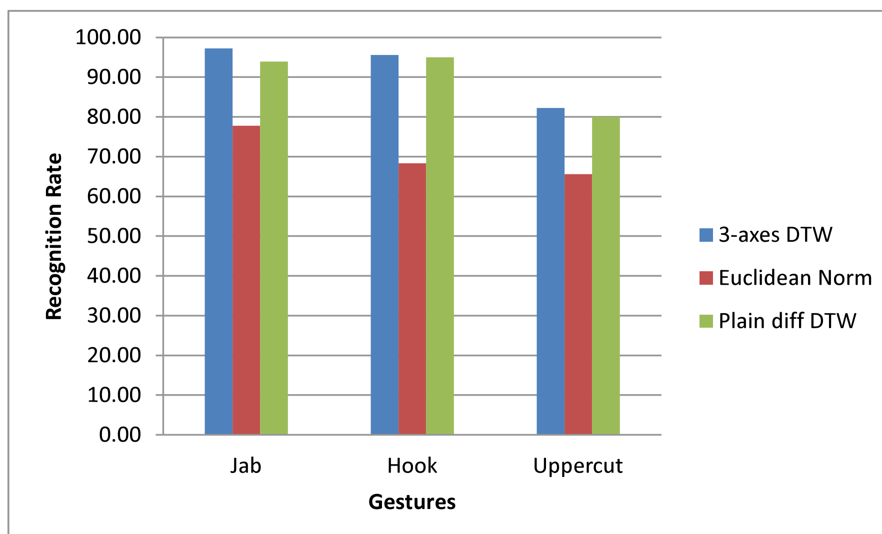

Figure 5.8: Recognition rates for Jab, Hook and Uppercut

Finally, Figure 5.9 shows the recognition rates for our three users. The user results indicated with *(2)* are the recognition results using only three gestures.

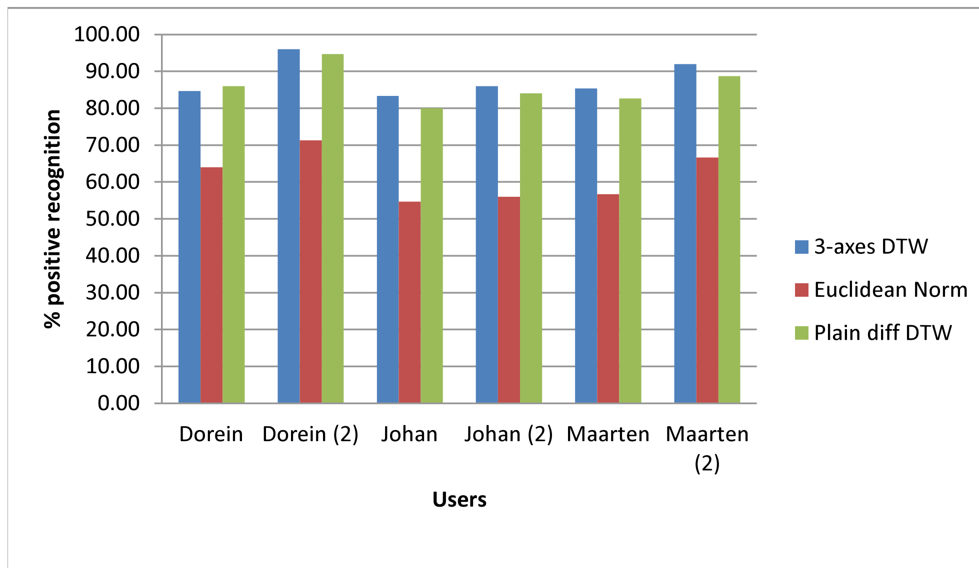

Figure 5.9: Recognition rates for each user for the boxing gesture set

5.3 Cellphone Gestures

In order to test Android gestures preformed with a cellphone, another set of gestures, shown in Figures 5.10 to 5.15, have been defined (Figures courtesy of Alp [1]).

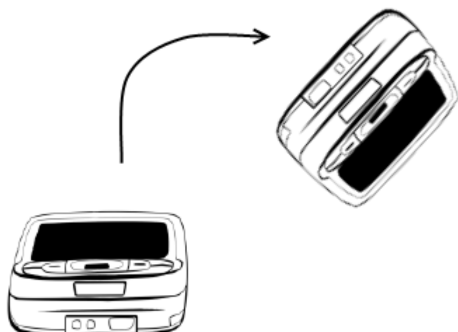

Figure 5.10: Turn the phone upside down to mute

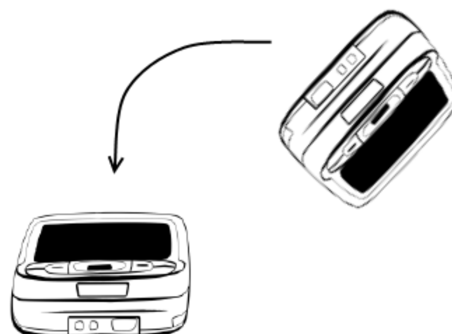

Figure 5.11: Turn the phone upside down to unmute

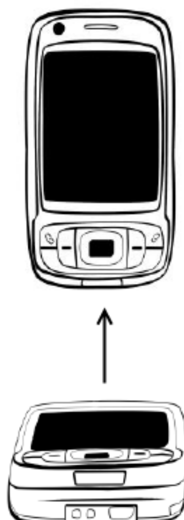

Figure 5.12: Pick the phone up from table

Figure 5.13: Jiggle the phone

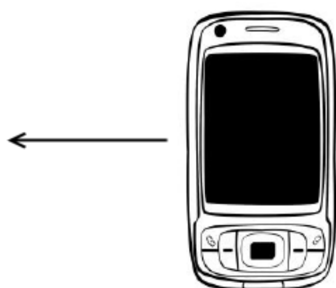

Figure 5.14: Movement to the left

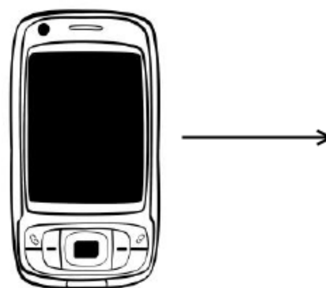

Figure 5.15: Movement to the right

When training the recogniser, it is important to clearly define the start and end of a gesture. The iGesture training application shown in Figure 4.9 requires the user to touch the screen when the gesture is started and the user needs to touch the screen again when the gesture is finished. When training the flip gesture, this can be a challenge. We therefore decided to generate the training samples only ourselves. The cellphone gestures where performed by one male, 28 years old and each gesture was executed ten times.

Tables 5.10, 5.11 and 5.12 show the results for the three versions of the Dynamic Time Warping algorithm.

Table 5.10: Cellphone results for three times DTW

3-axes DTW	Flip	Flop	Jiggle	Pickup	Left to Right	Right to Left
Flip	10	0	0	0	0	0
Flop	0	10	0	0	0	0
Jiggle	0	0	10	0	0	0
Pickup	0	0	0	10	0	0
Left to Right	0	0	0	0	9	1
Right to Left	0	0	0	0	1	9

Table 5.11: Cellphone results for Plain Difference

Plain Diff.	Flip	Flop	Jiggle	Pickup	Left to Right	Right to Left
Flip	10	0	0	0	0	0
Flop	0	10	0	0	0	0
Jiggle	0	0	10	0	0	0
Pickup	0	0	0	10	0	0
Left to Right	0	0	0	0	9	1
Right to Left	0	0	0	0	1	9

Table 5.12: Cellphone results for Euclidean Norm

Euclidean Norm	Flip	Flop	Jiggle	Pickup	Left to Right	Right to Left
Flip	10	0	0	0	0	0
Flop	1	8	1	0	0	0
Jiggle	0	0	9	1	0	0
Pickup	0	0	1	9	0	0
Left to Right	0	0	0	0	9	1
Right to Left	0	0	0	0	2	8

Chapter 6

Future Work

6.1 Sensor Fusion Using the Wii Motion Plus

iGesture currently uses the WiiGee library for the communication with the Wii Remote. Besides communication, WiiGee encapsulates the low level Wii Mote memory registry readings one has to preform in order to get the data. In the beginning of the WiiGee development, only the Wii Remote was supported. Afterwards, the sensor bar and Wii Motion Plus was added. WiiGee does not do a good job in processing WiiMotion Plus data. The continuous integration that is preformed to turn the angular velocity into a angle is drifting. The data coming from the other sensors should be used to correct this. It would be best if this sensor fusion is added to the WiiGee library.

We only use WiiGee to communicate with the Wii Remote. The gesture recognition capabilities of WiiGee are not used. It could be worth it to implement the Wii Remote communication directly in iGesture without using the WiiGee library. It would then be possible to implement Wii Motion Plus sensor fusion in iGesture.

When implementing sensor fusion, Kalmann filters [11] are widely used. Complementary filters which are a simpler version of Kalmann filters could be an interesting alternative to this problem.

6.2 Extending and Adding Recognition Algorithms

We have implemented a version of the Dynamic Time Warping algorithm. This is a very flexible algorithm for which numerous variations exist [18, 9, 21]. Some of these variations would

probably result in better recognition rates and lower resource usage.

Besides Dynamic Time Warping, other algorithms should be implemented. Hidden Markov Models are another technique used in gesture recognition [3, 5, 22]. It is even possible to combine Dynamic Time Warping with Hidden Markov Models [15].

6.3 Processing of Data Streams

When using 2D input devices, the start and stop values of the performed gesture are clearly defined. For instance, when performing a gesture with a mouse, the gesture starts when pressing the left mouse button and stops when releasing the button.

When using motion sensing input devices, this is no longer the case. They provide an endless stream of sensor data which could contain preformed gestures that should be classified. How does one find these gestures in an efficient way and can this be optimised based on training gesture analysis? The process of finding gestures in a stream of sensor data is called gesture spotting. Possible solutions to this problem can be found in [7, 24]

Chapter 7

Conclusion

We began this thesis with an in depth investigation of common sensors and devices. We explained how to handle certain sensor data and how sensors can be combined to improve sensor accuracy. We identified the common pitfalls of double integration and tilt, and we explained why this will result in inaccurate data. This background study enabled us, and should help future developers to process sensor data from 3D motion sensing devices in the most optimal way.

Secondly, we introduced a new IP-based communication protocol for iGesture. This message protocol enables developers to easily integrate new motion sensing devices with iGesture. The protocol can be used one all devices with a built-in WiFi connection. We have implemented an application that can be installed on Android devices, augmenting them with iGesture IP support. This application turns Android devices into iGesture input devices

Finally, using our initial research and our Android implementation, we have been able to develop a new 3D recognition algorithm. This new algorithm, which is based on the k-nearest neighbour and Dynamic Time Warping algorithms, will help iGesture users to record, test and deploy 3D gesture recognition in their own applications. We have performed multiple tests with different datasets to measure the performance of our algorithm. These tests have proven that Dynamic Time Warping is a good choice for 3D gesture recognition in iGesture.

Appendix A

Wii Remote

A.1 BlueCove

To communicate with a WiiMote, a JSR82 compatible library is needed in combination with a L2CAP compatible Bluetooth stack. WiiGee relies on the BlueCove JSR82 implementation to handle this. iGesture uses the WiiGee library to communicate with the Wii Remote. BlueCove is freely available for Windows, Mac OSX and Linux. BlueCove 2.1 released 2008-12-26 is currently the latest stable BlueCove version. This version has no support for x64 windows systems[1]. In windows 7 (x32 and x64), Microsoft did a redesign of its Bluetooth stack. BlueCove does claim to support x64 operation systems. In the 2.1.1 BlueCove SNAPSHOT release, a x64 DLL is included which should be able to communicate with Bluetooth devices on x64 based windows systems. A Wii Remote needs the L2CAP protocol to communicate which is only present on WIDCOMM[2] (nowadays broadcom) devices. Unfortunately, in the 2.1.1 version of BlueCove the Bluetooth module registers as a Microsoft generic Bluetooth adapter disabling L2CAP support. BlueCove does support x64 windows systems, but does not support L2CAP on x64 windows systems eliminating the support to connect a Wii Remote on a x64 windows system.

[1] http://code.google.com/p/bluecove/issues/detail?id=109

[2] http://www.broadcom.com/support/bluetooth/update.php

A.2 WiiGee

The WiiGee library is used to communicate with a Wii Remote and Wii Motion Plus. Support for the Wii Motion Plus was added to WiiGee at a later stage. The latest version available for download on the WiiGee website[3], is not the latest SVN version. A Wii Motion Plus bug is present in this prebuilt version of the WiiGee library. When experiencing the following error:

Listing A.1: Wii motion plus error

```
1 Unknown data retrieved.
2  A1 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Try checking out the latest WiiGee development version from SVN and build your own custom version of WiiGee.

[3]http://www.wiigee.org/

Appendix B

Source Code

Listing B.1: Dynamic Time Warping shortest path algorithm

```
1  /**
2   * Calculate shortest path in the matrix from 0;0 to i;j
3   * (top left -> down right)
4   *
5   * The algorithm starts at i;j and works its way up to 0:0 with the
6   * following constraints:
7   *
8   * 1. diagonal move counts as 0.5 whereas down or left move counts as 1
9   * 2. only go forward in time: down and right move is not allowed
10  * (0:0 is up/left and i:j down/right)
11  * 3. the path must go from 0:0 to i:j
12  *
13  * @param matrix
14  * @return
15  */
16 private float calculateShortestPath(Float[][] matrix) {
17     int rowCount = matrix.length - 1;
18     int columnCount = matrix[rowCount].length - 1;
19
20     // traverse the matrix from right->left
21     for (int column = columnCount; column >= 0; column--) {
22         // down->up
23         for (int row = rowCount; row >= 0; row--) {
24
25             if (column == columnCount && row == rowCount) {
```

```
26                    // last point in path, the lowest remaining cost till the end is
27                    // this cost
28
29                }
30                else if (column == columnCount) {
31                    // last column, the lowest cost to the end is on down 1 position
32                    matrix[row][column] += matrix[row + 1][column];
33
34                }
35                else if (row == rowCount) {
36                    // last rows, the lowest cost to the end is on it's right side
37                    matrix[row][column] += matrix[row][column + 1];
38
39                }
40                else {
41                    // pick the lowest values down/right/(diagonal*0.5)
42                    float down = matrix[row + 1][column];
43                    float right = matrix[row][column + 1];
44                    float diagonal = (float)(matrix[row + 1][column + 1] * 0.5);
45
46                    matrix[row][column] += min(down, right, diagonal);
47                }
48            }
49        }
50
51        return matrix[0][0];
52    }
```

Listing B.2: Dynamic Time Warping Euclidean Norm algorithm

```
1  /**
2   * Calculates AccelerationSample EuclideanNorm:
3   * sqrt(x^2+y^2+z^2)
4   *
5   * This technique eliminates device rotation and gravity
6   *
7   * @param challenge
8   * @return
9   */
10 private double euclideanNorm(AccelerationSample a, AccelerationSample b) {
```

```
11     return Math.sqrt((a.getXAcceleration() * a.getXAcceleration())
12                 + (a.getYAcceleration() * a.getYAcceleration())
13                 + (a.getZAcceleration() * a.getZAcceleration()))
14           - Math.sqrt((b.getXAcceleration() * b.getXAcceleration())
15                 + (b.getYAcceleration() * b.getYAcceleration())
16                 + (b.getZAcceleration() * b.getZAcceleration()));
17 }
```

Listing B.3: Dynamic Time Warping difference algorithm

```
1  /**
2   * Calculate the most naive difference between two AccelerationSamples
3   *
4   * |x1-x2| + |y1-y2| + |z1-z1|
5   *
6   * @param base
7   * @param challenge
8   * @return double plainDiff
9   */
10 private double plainDiff(AccelerationSample base, AccelerationSample challenge) {
11     return Math.abs(base.getXAcceleration() - challenge.getXAcceleration())
12             + Math.abs(base.getYAcceleration() - challenge.getYAcceleration())
13             + Math.abs(base.getZAcceleration() - challenge.getZAcceleration());
14 }
```

Listing B.4: iGesture IP device discovery

```
1  package org.ximtec.igesture.tool.view.devicemanager.discoveryservice;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.lang.reflect.Constructor;
7  import java.lang.reflect.InvocationTargetException;
8  import java.net.InetAddress;
9  import java.net.ServerSocket;
10 import java.net.Socket;
11 import java.net.UnknownHostException;
12 import java.util.HashSet;
13 import java.util.Set;
14 import java.util.logging.Level;
```

```
15 import java.util.logging.Logger;
16
17 import org.ximtec.igesture.io.AbstractGestureDevice;
18 import org.ximtec.igesture.io.DeviceDiscoveryService;
19 import org.ximtec.igesture.io.android.AndroidReader3D;
20 import org.ximtec.igesture.util.Constant;
21
22
23 /**
24  * An IP 3D device discovery service. It extends
25  * {@link org.ximtec.igesture.tool.view.devicemanager.discoveryservice.
26  * AbstractTuioDeviceDiscoveryService}
27  *
28  * @author Johan Bas
29  *
30  */
31 public class Android3DDeviceDiscoveryService implements DeviceDiscoveryService {
32
33    private static final Logger LOGGER = Logger
34          .getLogger(Android3DDeviceDiscoveryService.class.getName());
35    private Set<AbstractGestureDevice< ? , ? >> devices;
36
37
38    /**
39     * Constructor
40     */
41    public Android3DDeviceDiscoveryService() {
42       devices = new HashSet<AbstractGestureDevice< ? , ? >>();
43    }
44
45
46    @Override
47    public Set<AbstractGestureDevice< ? , ? >> discover() {
48
49       LOGGER.log(Level.INFO, "Android Device discovery started!");
50
51       try {
52          InetAddress addr = InetAddress.getLocalHost();
53          String ipAddr = addr.getHostAddress();
```

```
54
55          LOGGER.log(Level.INFO, "Connect to: " + ipAddr.toString());
56          LOGGER.log(Level.INFO, "Starting Android Gesture Server on port 80");
57
58          ServerSocket serverSocket = null;
59          try {
60              serverSocket = new ServerSocket(80);
61          }
62          catch (IOException e) {
63              System.out.println("Could not listen on port: 80");
64              System.exit(1);
65          }
66
67          LOGGER.log(Level.INFO, "Socket created.");
68          LOGGER.log(Level.INFO, "Listening to socket on port 80");
69
70          Socket clientSocket = null;
71          try {
72              clientSocket = serverSocket.accept();
73              @SuppressWarnings("rawtypes")
74              Constructor ctor;
75              try {
76                  ctor = AndroidReader3D.class.getConstructor(Socket.class,
77                          BufferedReader.class);
78                  try {
79                      BufferedReader in = null;
80                      String name = "";
81                      try {
82                          in = new BufferedReader(new InputStreamReader(
83                                  clientSocket.getInputStream()));
84                          name = in.readLine();
85                      }
86                      catch (IOException e1) {
87                          System.out.println("Error:" + e1);
88                      }
89
90                      AbstractGestureDevice< ? , ? > device = (AbstractGestureDevice< ? , ?
91                              .newInstance(clientSocket, in);
92                      String[] temp = name.split(":");
```

```
93                          device.setName(temp[0]);
94                          device.setDeviceType(Constant.TYPE_3D);
95                          device.setConnectionType(Constant.CONNECTION_IP);
96                          device.setIsConnected(true);
97                          device.setDeviceID(temp[1]);
98                          devices.add(device);
99                      }
100                     catch (IllegalArgumentException e) {
101                         e.printStackTrace();
102                     }
103                     catch (InstantiationException e) {
104                         e.printStackTrace();
105                     }
106                     catch (IllegalAccessException e) {
107                         e.printStackTrace();
108                     }
109                     catch (InvocationTargetException e) {
110                         e.printStackTrace();
111                     }
112                 }
113                 catch (SecurityException e) {
114                     e.printStackTrace();
115                 }
116                 catch (NoSuchMethodException e) {
117                     e.printStackTrace();
118                 }
119             }
120             catch (IOException e) {
121                 System.exit(1);
122             }
123         }
124         catch (UnknownHostException e) {
125         }
126
127         return devices;
128     }
129
130
131     @Override
```

```
132     public void dispose() {
133         devices.clear();
134     }
135 }
```

Listing B.5: Android application used to send data to iGesture

```
1
2
3 package org.ximtec.igesture;
4
5 import java.io.BufferedWriter;
6 import java.io.OutputStreamWriter;
7 import java.io.PrintWriter;
8 import java.net.InetAddress;
9 import java.net.Socket;
10 import java.util.List;
11
12 import android.app.Activity;
13 import android.content.Context;
14 import android.hardware.Sensor;
15 import android.hardware.SensorEvent;
16 import android.hardware.SensorEventListener;
17 import android.hardware.SensorManager;
18 import android.os.Bundle;
19 import android.util.Log;
20 import android.view.View;
21 import android.view.View.OnClickListener;
22 import android.widget.Button;
23 import android.widget.TextView;
24
25
26 /**
27  * When loaded, the Activity will try to connect to the given server IP. The
28  * button click listener will enable and disable recording mode. Sensor changes
29  * will be sent to the server if the connection is alive and the app is in
30  * recording mode.
31  *
32  * @author Johan Bas
33  *
```

```
34  */
35 public class Application extends Activity implements SensorEventListener {
36
37     private static final String ANDGESTURE_STOP = "Q - End of Gesture!";
38     private static final String ANDGESTURE_OFF = "OFF";
39     private static final String ANDGESTURE_START = "S - Start of Gesture!";
40     private static final String ANDGESTURE_ON = "ON";
41     private String serverIP = "127.0.0.1";
42     private final static int serverPort = 80;
43     private Socket socket = null;
44     private boolean connected = false;
45     private PrintWriter socketWriter = null;
46
47     private SensorManager mgr;
48     private List<Sensor> sensorList;
49     private Sensor sensor = null;
50
51     private boolean record = false;
52     private final static double TRESHOLD = 0.1;
53
54
55     /**
56      * Called when the activity is first created.
57      */
58     @Override
59     public void onCreate(Bundle savedInstanceState) {
60         super.onCreate(savedInstanceState);
61         setContentView(R.layout.application);
62         serverIP = getIntent().getExtras().getString("serverIP");
63         connectToServer();
64
65         mgr = (SensorManager)getApplicationContext().getSystemService(
66                 Context.SENSOR_SERVICE);
67         for (Sensor sensor : mgr.getSensorList(Sensor.TYPE_ACCELEROMETER)) {
68             if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
69                 this.sensor = sensor;
70             }
71         }
72
```

```
73          mgr.registerListener(this, sensor, SensorManager.SENSOR_DELAY_FASTEST);
74
75          final Button capture = (Button)findViewById(R.id.Capture);
76
77          capture.setOnClickListener(clickListenerForCaptureButton(capture));
78
79      }
80
81
82      /**
83       * Try to connect to a given server IP
84       */
85      private void connectToServer() {
86          try {
87              InetAddress serverAddr = InetAddress.getByName(serverIP);
88              Log.d("IGesture", "Connecting... " + serverAddr + ":" + serverPort);
89              socket = new Socket(serverAddr, serverPort);
90
91              try {
92                  Log.d("IGesture", "Sending command.");
93                  socketWriter = new PrintWriter(new BufferedWriter(
94                          new OutputStreamWriter(socket.getOutputStream())), true);
95
96                  connected = true;
97                  socketWriter.println(android.os.Build.MODEL + ":"
98                          + android.os.Build.ID);
99                  TextView msg = (TextView)findViewById(R.id.Msg);
100                 msg.setText("CONNECTED");
101             }
102             catch (Exception e) {
103                 connected = false;
104                 Log.e("IGesture", "Error: ", e);
105             }
106         }
107         catch (Exception e) {
108             connected = false;
109             Log.e("IGesture", "Error: ", e);
110         }
111     }
```

```
112
113
114     /**
115      * Handle the button clicks -> enable and disable recording state of the
116      * application
117      *
118      * @param Button The capture button on the activity
119      * @return OnClickListener
120      */
121     private OnClickListener clickListenerForCaptureButton(final Button capture) {
122         return new View.OnClickListener() {
123
124             @Override
125             public void onClick(View v) {
126                 record = !record;
127                 TextView msg = (TextView)findViewById(R.id.Msg);
128                 msg.setText("Connected:" + connected + " - recording:" + record);
129
130                 if (record) {
131                     capture.setText(ANDGESTURE_ON);
132                     socketWriter.println(ANDGESTURE_START);
133                 }
134                 else {
135                     capture.setText(ANDGESTURE_OFF);
136                     socketWriter.println(ANDGESTURE_STOP);
137                 }
138             }
139         };
140     }
141
142
143     @Override
144     public void onSensorChanged(SensorEvent event) {
145         // http://developer.android.com/reference/android/hardware/SensorEvent.html#values
146         switch (event.sensor.getType()) {
147
148             case Sensor.TYPE_LINEAR_ACCELERATION: // sensor fusion
149                 if (record)
150                     socketWriter.println("A - " + event.values[0] + ":"
```

```
151                         + event.values[1] + ":" + event.values[2] + ":"
152                         + event.timestamp);
153
154                 break;
155
156             default:
157                 break;
158
159         }
160     }
161
162
163     /**
164      * Idle filter
165      *
166      * @param event
167      * @return true if idle, false if in motion
168      */
169     private boolean idle(SensorEvent event) {
170         if (event.values[0] < TRESHOLD && event.values[0] < TRESHOLD
171                 && event.values[0] < TRESHOLD)
172             return true;
173         else
174             return false;
175     }
176
177
178     @Override
179     protected void onResume() {
180         super.onResume();
181
182         sensorList = mgr.getSensorList(Sensor.TYPE_ALL);
183         for (Sensor sensor : sensorList) {
184             if (sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION
185                     || sensor.getType() == Sensor.TYPE_ACCELEROMETER
186                     || sensor.getType() == Sensor.TYPE_GYROSCOPE) {
187                 mgr.registerListener(this, sensor,
188                         SensorManager.SENSOR_DELAY_FASTEST);
189             }
```

```
190         }
191     }
192
193
194     @Override
195     protected void onPause() {
196         super.onPause();
197         // Stop updates to save power while the app is paused
198         mgr.unregisterListener(this);
199     }
200
201
202     @Override
203     public void onAccuracyChanged(Sensor sensor, int accuracy) {
204         if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
205             if (accuracy == SensorManager.SENSOR_STATUS_ACCURACY_HIGH)
206                 Log.d("accuracy", "ACCELEROMETER Accuracy is high");
207             else if (accuracy == SensorManager.SENSOR_STATUS_ACCURACY_LOW)
208                 Log.d("accuracy", "ACCELEROMETER Accuracy is low");
209             else if (accuracy == SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM)
210                 Log.d("accuracy", "ACCELEROMETER Accuracy is medium");
211         }
212     }
213 }
```

Listing B.6: Android application used to send data to iGesture connection screen code

```
1
2
3 package org.ximtec.igesture;
4
5 import android.app.Activity;
6 import android.content.Intent;
7 import android.os.Bundle;
8 import android.util.Log;
9 import android.view.Menu;
10 import android.view.MenuInflater;
11 import android.view.MenuItem;
12 import android.view.View;
13 import android.widget.Button;
```

```
14 import android.widget.TextView;
15
16
17 /**
18  *
19  * @author Johan Bas
20  *
21  */
22 public class Connection extends Activity {
23
24     /**
25      * Called when the activity is first created.
26      */
27     @Override
28     public void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.connection);
31         Log.d("IGesture", "Start app.");
32
33         final Button random = (Button)findViewById(R.id.Connect);
34         random.setOnClickListener(new View.OnClickListener() {
35
36             public void onClick(View v) {
37                 TextView ipv = (TextView)findViewById(R.id.ip);
38                 Intent intent = new Intent(v.getContext(), Application.class);
39                 intent.putExtra("serverIP", ipv.getText().toString());
40                 startActivity(intent);
41             }
42         });
43     }
44
45
46     /**
47      * Add menu to main view. Menu has a quit button.
48      */
49     @Override
50     public boolean onCreateOptionsMenu(Menu menu) {
51         MenuInflater inflater = getMenuInflater();
52         inflater.inflate(R.menu.menu, menu);
```

```
53         return true;
54     }
55
56
57     /**
58      * Handle quit button if pressed.
59      */
60     @Override
61     public boolean onOptionsItemSelected(MenuItem item) {
62         switch (item.getItemId()) {
63             case R.id.quit:
64                 this.finish();
65                 return true;
66             default:
67                 return super.onOptionsItemSelected(item);
68         }
69     }
70 }
```

Listing B.7: Process data coming from an IP device according to the protocol

```
1
2
3 package org.ximtec.igesture.io.android;
4
5 import java.io.IOException;
6
7
8 /**
9  *
10  * @author Johan Bas
11  *
12  */
13 public class AndroidStreamer extends Thread {
14
15     private static final String PARSE_ACCELERATION = "A";
16     private static final String PARSE_IDLE = "I";
17     private static final String PARSE_QUIT = "Q";
18     private static final String PARSE_STOP = "S";
19     private static final String PARSE_BYE = "Bye.";
```

```
20    private boolean recording = false;
21    private AndroidReader3D device;
22
23
24    public AndroidStreamer(AndroidReader3D device) {
25        this.device = device;
26    }
27
28
29    @Override
30    public void run() {
31        String inputLine;
32
33        try {
34            while ((inputLine = device.getBufferedReader().readLine()) != null) {
35
36                if (inputLine.equals(PARSE_BYE)) {
37                    System.out.println(PARSE_BYE);
38                    break;
39                }
40                else if (inputLine.startsWith(PARSE_STOP)) {
41                    this.device.startGesture();
42                    recording = true;
43                }
44                else if (inputLine.startsWith(PARSE_QUIT)) {
45                    // end gesture
46                    this.device.stopGesture();
47                    recording = false;
48                }
49                else {
50                    if (recording) {
51                        if (inputLine.startsWith(PARSE_IDLE)) {
52                            // idle state
53                        }
54                        else if (inputLine.startsWith(PARSE_ACCELERATION)) {
55                            inputLine = inputLine.substring(4);
56                            String[] coordinated = inputLine.split(":");
57
58                            double x = Double.parseDouble(coordinated[0]);
```

```
59                      double y = Double.parseDouble(coordinated[1]);
60                      double z = Double.parseDouble(coordinated[2]);
61                      long time = Long.parseLong(coordinated[3]);
62
63                      this.device.addAccelerationValues(x, y, z, time);
64
65                  }
66              }
67          }
68      }
69  }
70  catch (IOException e) {
71      e.printStackTrace();
72  }
73  }
74 }
```

Listing B.8: Proof of concept application

```
1
2
3 package be.johanbas;
4
5 import java.io.BufferedReader;
6 import java.io.FileNotFoundException;
7 import java.io.FileReader;
8 import java.io.IOException;
9 import java.util.LinkedList;
10 import java.util.List;
11
12 import android.app.Activity;
13 import android.content.Context;
14 import android.graphics.Color;
15 import android.hardware.Sensor;
16 import android.hardware.SensorEvent;
17 import android.hardware.SensorEventListener;
18 import android.hardware.SensorManager;
19 import android.os.Bundle;
20 import android.os.PowerManager;
21 import android.view.Menu;
```

```
22 import android.view.MenuInflater;
23 import android.view.MenuItem;
24 import android.widget.ScrollView;
25 import android.widget.TextView;
26
27 import com.androidplot.Plot;
28 import com.androidplot.xy.BoundaryMode;
29 import com.androidplot.xy.LineAndPointFormatter;
30 import com.androidplot.xy.SimpleXYSeries;
31 import com.androidplot.xy.XYPlot;
32
33
34 /**
35  *
36  * @author Johan Bas
37  *
38  */
39 public class Main extends Activity implements SensorEventListener {
40
41     private SensorManager sensorMgr = null;
42     private List<Sensor> sensorList;
43     private XYPlot xPlot = null;
44     private XYPlot yPlot = null;
45     private XYPlot zPlot = null;
46     private SimpleXYSeries xSeries = new SimpleXYSeries("X Levels");
47     private SimpleXYSeries ySeries = new SimpleXYSeries("Y Levels");
48     private SimpleXYSeries zSeries = new SimpleXYSeries("Z Levels");
49     private LinkedList<Number> xHistory = new LinkedList<Number>();
50     private LinkedList<Number> yHistory = new LinkedList<Number>();
51     private LinkedList<Number> zHistory = new LinkedList<Number>();
52     private Sensor sensor = null;
53     private static final int HISTORY_SIZE = 20;
54
55     private static final double RECOGNITION_TRESHOLD_FLIP = 4;
56     private static final double RECOGNITION_TRESHOLD_SIDEWAYS = 5;
57     private static final int ACCELERATION_LIST_SIZE = 20;
58     private static final double IDLE_TRESHOLD = 0.3;
59     private static final int DTW_INTERVAL = 4;
60
```

```
61      private TextView textView;
62
63      private static List<Gesture> flip = new LinkedList<Gesture>();
64      private static List<Gesture> flop = new LinkedList<Gesture>();
65      private static List<Gesture> right = new LinkedList<Gesture>();
66      private static List<Gesture> left = new LinkedList<Gesture>();
67
68      private static List<Gesture> fullList = new LinkedList<Gesture>();
69
70      private DTW dtw;
71      private List<Acceleration> accelerations = new LinkedList<Acceleration>();
72
73      private PowerManager.WakeLock wl;
74
75
76      @Override
77      public void onCreate(Bundle savedInstanceState) {
78          super.onCreate(savedInstanceState);
79          setContentView(R.layout.main);
80
81          PowerManager pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
82          wl = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK, "DoNotDimScreen");
83
84          xPlot = (XYPlot)findViewById(R.id.xPlot);
85          yPlot = (XYPlot)findViewById(R.id.yPlot);
86          zPlot = (XYPlot)findViewById(R.id.zPlot);
87
88          setupChart(xPlot, xSeries);
89          setupChart(yPlot, ySeries);
90          setupChart(zPlot, zSeries);
91
92          sensorMgr = (SensorManager)getApplicationContext().getSystemService(
93                  Context.SENSOR_SERVICE);
94          for (Sensor sensor : sensorMgr
95                  .getSensorList(Sensor.TYPE_LINEAR_ACCELERATION)) {
96              if (sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {
97                  this.sensor = sensor;
98              }
99          }
```

```
100
101         sensorMgr.registerListener(this, sensor, SensorManager.SENSOR_DELAY_GAME);
102
103         setupText();
104
105         readAccelerations();
106
107         if (fullList.size() == 0) {
108             fullList.addAll(flip);
109             fullList.addAll(flop);
110             fullList.addAll(left);
111             fullList.addAll(right);
112         }
113
114         textView.append("Gestures loaded " + fullList.size() + "\n");
115
116         dtw = new DTW(0);
117     }
118
119
120     private static void readAccelerations() {
121         flip = getContents("/sdcard/gestures/androidflip.txt", "Flip");
122         flop = getContents("/sdcard/gestures/androidflop.txt", "Flip");
123         right = getContents("/sdcard/gestures/androidright.txt", "Sideways");
124         left = getContents("/sdcard/gestures/androidleft.txt", "Sideways");
125     }
126
127
128     static public List<Gesture> getContents(String filepath, String type) {
129         BufferedReader input = null;
130         try {
131             input = new BufferedReader(new FileReader(filepath));
132         }
133         catch (FileNotFoundException e) {
134             e.printStackTrace();
135         }
136
137         List<Gesture> gestures = new LinkedList<Gesture>();
138         List<Acceleration> accelerations = new LinkedList<Acceleration>();
```

```
139         try {
140
141             String line = input.readLine();
142
143             while (line != null) {
144
145                 if (line.trim().equals("")) {
146                     Gesture gesture = new Gesture();
147                     gesture.setAccelerations(accelerations);
148                     gesture.setType(type);
149                     gestures.add(gesture);
150                     accelerations = new LinkedList<Acceleration>();
151                 }
152                 else {
153                     Acceleration acc = new Acceleration();
154                     String[] split = line.split(":");
155
156                     acc.setX(Float.parseFloat(split[0]));
157                     acc.setY(Float.parseFloat(split[1]));
158                     acc.setZ(Float.parseFloat(split[2]));
159                     acc.setTimestamp(Long.parseLong(split[3]));
160                     accelerations.add(acc);
161                 }
162
163                 line = input.readLine();
164             }
165
166             Gesture gesture = new Gesture();
167             gesture.setAccelerations(accelerations);
168             gestures.add(gesture);
169             gesture.setType(type);
170             accelerations = new LinkedList<Acceleration>();
171         }
172         catch (IOException ex) {
173             ex.printStackTrace();
174         }
175
176         return gestures;
177     }
```

```
178
179
180     private void setupText() {
181         textView = (TextView)findViewById(R.id.textView1);
182         textView.setTextColor(Color.WHITE);
183         textView.setSelected(true);
184         textView.setText("");
185     }
186
187
188     private void scroll() {
189         final ScrollView sv = (ScrollView)findViewById(R.id.start_scroller);
190         sv.post(new Runnable() {
191
192             public void run() {
193                 sv.fullScroll(ScrollView.FOCUS_DOWN);
194             }
195         });
196     }
197
198
199     private void setupChart(XYPlot plot, SimpleXYSeries series) {
200         plot.addSeries(
201                 series,
202                 new LineAndPointFormatter(Color.argb(100, 255, 255, 255), Color.rgb(
203                         255, 255, 255), Color.argb(0, 0, 0, 0)));
204         plot.setBackgroundColor(Color.BLACK);
205         plot.setBorderStyle(Plot.BorderStyle.NONE, 0f, 0f);
206         plot.setRangeBoundaries(-15, 15, BoundaryMode.FIXED);
207         plot.setBorderPaint(null);
208         plot.disableAllMarkup();
209         plot.setPlotMargins(0, 0, 0, 0);
210         plot.setPlotPadding(0, 0, 0, 0);
211         plot.getLegendWidget().setVisible(false);
212         plot.setDomainLabel("");
213     }
214
215
216     @Override
```

```
217     public void onAccuracyChanged(Sensor arg0, int arg1) {
218         // TODO Auto-generated method stub
219 
220     }
221 
222 
223     @Override
224     public void onSensorChanged(SensorEvent sensorEvent) {
225         if (sensorEvent.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {
226 
227             xHistory.addLast(sensorEvent.values[0]);
228             yHistory.addLast(sensorEvent.values[1]);
229             zHistory.addLast(sensorEvent.values[2]);
230 
231             // get rid the oldest sample in history:
232             if (xHistory.size() > HISTORY_SIZE) {
233                 xHistory.removeFirst();
234                 yHistory.removeFirst();
235                 zHistory.removeFirst();
236             }
237 
238             xSeries.setModel(xHistory, SimpleXYSeries.ArrayFormat.Y_VALS_ONLY);
239             ySeries.setModel(yHistory, SimpleXYSeries.ArrayFormat.Y_VALS_ONLY);
240             zSeries.setModel(zHistory, SimpleXYSeries.ArrayFormat.Y_VALS_ONLY);
241 
242             xPlot.redraw();
243             yPlot.redraw();
244             zPlot.redraw();
245 
246             processAcceleration(sensorEvent);
247         }
248     }
249 
250     private int count = 0;
251 
252 
253     private void processAcceleration(SensorEvent sensorEvent) {
254         count++;
255 
```

```
256            Acceleration acc = new Acceleration();
257            acc.setX(sensorEvent.values[0]);
258            acc.setY(sensorEvent.values[1]);
259            acc.setZ(sensorEvent.values[2]);
260
261            accelerations.add(acc);
262
263            if (accelerations.size() > ACCELERATION_LIST_SIZE) {
264                accelerations.remove(0);
265            }
266
267            if (count % DTW_INTERVAL == 0) {
268                Gesture g = new Gesture();
269                g.setAccelerations(accelerations);
270                g = dtw.findBestMatch(g, fullList);
271
272                if (g.getType().equals("Flip")
273                        && g.getMaxDistanceToOtherGestures() < RECOGNITION_TRESHOLD_FLIP) {
274                    textView.setText(g.getType() + ":"
275                            + g.getMaxDistanceToOtherGestures());
276                }
277                else if (g.getType().equals("Sideways")
278                        && g.getMaxDistanceToOtherGestures() < RECOGNITION_TRESHOLD_SIDEWAYS) {
279                    textView.setText(g.getType() + ":"
280                            + g.getMaxDistanceToOtherGestures());
281                }
282            }
283        }
284
285
286        private boolean idle(SensorEvent sensorEvent) {
287            if (sensorEvent.values[0] > IDLE_TRESHOLD
288                    || sensorEvent.values[1] > IDLE_TRESHOLD
289                    || sensorEvent.values[2] > IDLE_TRESHOLD)
290                return false;
291            else
292                return true;
293        }
294
```

```
295
296     @Override
297     protected void onResume() {
298         super.onResume();
299         wl.acquire();
300
301         sensorList = sensorMgr.getSensorList(Sensor.TYPE_ALL);
302         for (Sensor sensor : sensorList) {
303             if (sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {
304                 sensorMgr.registerListener(this, sensor,
305                         SensorManager.SENSOR_DELAY_GAME);
306             }
307         }
308     }
309
310
311     @Override
312     protected void onPause() {
313         super.onPause();
314         // Stop updates to save power while the app is paused
315         sensorMgr.unregisterListener(this);
316         wl.release();
317     }
318
319
320     /**
321      * Add menu to main view. Menu has a quit button.
322      */
323     @Override
324     public boolean onCreateOptionsMenu(Menu menu) {
325         MenuInflater inflater = getMenuInflater();
326         inflater.inflate(R.menu.menu, menu);
327         return true;
328     }
329
330
331     /**
332      * Handle quit button if pressed.
333      */
```

```
334     @Override
335     public boolean onOptionsItemSelected(MenuItem item) {
336         switch (item.getItemId()) {
337             case R.id.quit:
338                 this.finish();
339                 sensorMgr.unregisterListener(this);
340 
341                 return true;
342             default:
343                 return super.onOptionsItemSelected(item);
344         }
345     }
346 }
```

Listing B.9: Proof of concept application Dynamic Time Warping algorithm used

```
 1 
 2 
 3 package be.johanbas;
 4 
 5 import java.util.LinkedList;
 6 import java.util.List;
 7 
 8 
 9 /**
10  *
11  * @author Johan Bas
12  *
13  */
14 public class DTW {
15 
16     private double treashold;
17 
18 
19     public DTW(double treashold) {
20         this.treashold = treashold;
21     }
22 
23 
24     public Gesture findBestMatch(Gesture challenge, List<Gesture> bases) {
```

```
25
26          Gesture g = new Gesture();
27          g.setMaxDistanceToOtherGestures(50);
28          for (Gesture base : bases) {
29              double path = warpInTimeDynamicallyPlainDiff(challenge, base);
30              if (path < g.getMaxDistanceToOtherGestures()) {
31                  g = base;
32                  g.setMaxDistanceToOtherGestures(path);
33              }
34          }
35
36          return g;
37      }
38
39
40      /**
41       * Dynamic Time Warping algorithm to calculate the difference between two
42       * gestures
43       *
44       * @param sampleAccelerations
45       * @param challengeGesture
46       *
47       * @return boolean if the shortest path is beneath the threshold
48       */
49      public boolean warpInTimeDynamically(Gesture challenge, Gesture base) {
50          Float[][] matrix = generateMatrixPlainDiff(challenge, base);
51          return calculateShortestPath(matrix) < treashold;
52      }
53
54
55      /**
56       * Dynamic Time Warping algorithm to calculate the difference between two
57       * gestures
58       *
59       * @param sampleAccelerations
60       * @param challengeGesture
61       *
62       * @return shortest path
63       */
```

```
64     public double warpInTimeDynamicallyEuclideanNorm(Gesture challenge,
65             Gesture base) {
66         Float[][] matrix = generateMatrixEuclideanNorm(challenge, base);
67         return calculateShortestPath(matrix);
68     }
69
70
71     /**
72      * Dynamic Time Warping algorithm to calculate the difference between two
73      * gestures
74      *
75      * @param sampleAccelerations
76      * @param challengeGesture
77      *
78      * @return shortest path
79      */
80     public double warpInTimeDynamicallyPlainDiff(Gesture challenge, Gesture base) {
81         Float[][] matrix = generateMatrixPlainDiff(challenge, base);
82         return calculateShortestPath(matrix);
83     }
84
85
86     /**
87      * Dynamic Time Warping algorithm to calculate the difference between two
88      * gestures 3 times for each axis
89      *
90      * @param challenge
91      * @param base
92      * @return Shortest path (3 warpes combined)
93      */
94     public double warpInTimeDynamicallyThreeTimes(Gesture challenge, Gesture base) {
95         List<Float> baseValuesX = new LinkedList<Float>();
96         for (Acceleration acc : base.getAccelerations()) {
97             baseValuesX.add(acc.getX());
98         }
99
100        List<Float> baseValuesY = new LinkedList<Float>();
101        for (Acceleration acc : base.getAccelerations()) {
102            baseValuesY.add(acc.getY());
```

```
103         }
104 
105         List<Float> baseValuesZ = new LinkedList<Float>();
106         for (Acceleration acc : base.getAccelerations()) {
107             baseValuesZ.add(acc.getZ());
108         }
109 
110         List<Float> challengeValuesX = new LinkedList<Float>();
111         for (Acceleration acc : challenge.getAccelerations()) {
112             challengeValuesX.add(acc.getX());
113         }
114 
115         List<Float> challengeValuesY = new LinkedList<Float>();
116         for (Acceleration acc : challenge.getAccelerations()) {
117             challengeValuesY.add(acc.getY());
118         }
119 
120         List<Float> challengeValuesZ = new LinkedList<Float>();
121         for (Acceleration acc : challenge.getAccelerations()) {
122             challengeValuesZ.add(acc.getZ());
123         }
124 
125         Float[][] matrixx = generateMatrix(baseValuesX, challengeValuesX);
126         Float[][] matrixy = generateMatrix(baseValuesY, challengeValuesY);
127         Float[][] matrixz = generateMatrix(baseValuesZ, challengeValuesZ);
128 
129         return calculateShortestPath(matrixx) + calculateShortestPath(matrixy)
130                 + calculateShortestPath(matrixz);
131     }
132 
133 
134     /**
135      * Most basic way to generate a matrix: between two Lists of double values
136      *
137      * @param baseValues
138      * @param challengeValues
139      * @return matrix
140      */
141     private Float[][] generateMatrix(List<Float> baseValues,
```

```
142            List<Float> challengeValues) {
143
144        Float[][] matrix = new Float[baseValues.size()][challengeValues.size()];
145        for (int i = 0; i < baseValues.size(); i++) {
146            for (int j = 0; j < challengeValues.size(); j++) {
147                matrix[i][j] = Math.abs(baseValues.get(i) - challengeValues.get(j));
148            }
149        }
150
151        return matrix;
152    }
153
154
155    /**
156     * Generate a matrix with the differences between two gestures:
157     *
158     * sqrt(x1^2+y1^2+z1^2) - sqrt(x2^2+y2^2+z2^2) for all acceleration values
159     *
160     * @param sampleAccelerations
161     * @param challengeGesture
162     * @return
163     */
164    private Float[][] generateMatrixEuclideanNorm(Gesture a, Gesture b) {
165        Float[][] matrix = new Float[a.getAccelerations().size()][b
166                .getAccelerations().size()];
167
168        for (int i = 0; i < a.getAccelerations().size(); i++) {
169            for (int j = 0; j < b.getAccelerations().size(); j++) {
170                double temp = euclideanNorm(a.getAccelerations().get(i))
171                        - euclideanNorm(b.getAccelerations().get(j));
172                matrix[i][j] = (float)Math.sqrt(temp * temp);
173            }
174        }
175
176        return matrix;
177    }
178
179
180    private Float[][] generateMatrixPlainDiff(Gesture a, Gesture b) {
```

```
181         Float[][] matrix = new Float[a.getAccelerations().size()][b
182                 .getAccelerations().size()];
183 
184         for (int i = 0; i < a.getAccelerations().size(); i++) {
185             for (int j = 0; j < b.getAccelerations().size(); j++) {
186                 double temp = plainDiff(a.getAccelerations().get(i), b
187                         .getAccelerations().get(j));
188                 matrix[i][j] = (float)Math.sqrt(temp * temp);
189             }
190         }
191 
192         return matrix;
193     }
194 
195 
196     /**
197      * Calculate AccelerationSample eigen value: sqrt(x^2+y^2+z^2)
198      *
199      * This techniques eliminates device rotation and gravity
200      *
201      * @param challenge
202      * @return
203      */
204     private double euclideanNorm(Acceleration a) {
205         return Math.sqrt((a.getX() * a.getX()) + (a.getY() * a.getY())
206                 + (a.getZ() * a.getZ()));
207     }
208 
209 
210     private double plainDiff(Acceleration a, Acceleration b) {
211         return Math.abs(a.getX() - b.getX()) + Math.abs(a.getY() - b.getY())
212                 + Math.abs(a.getZ() - b.getZ());
213     }
214 
215 
216     /**
217      * Calculate shortest path in the matrix from 0;0 to i;j (top left -> down
218      * right)
219      *
```

```
220     * The algorithm starts at i;j and works its way down to 0:0 with the
221     * following constraints:
222     *
223     * 1. diagonal move counts as 0.5 whereas down or left move counts as 1
224     * 2. only go forward in time: down and right move is not allowed
225     * (0:0 is up/left and i:j down/right)
226     * 3. the path must go from 0:0 to i:j
227     *
228     * @param matrix
229     * @return
230     */
231    private float calculateShortestPath(Float[][] matrix) {
232       int rowCount = matrix.length - 1;
233       int columnCount = matrix[rowCount].length - 1;
234
235       // traverse the matrix from right->left
236       for (int column = columnCount; column >= 0; column--) {
237          // down->up
238          for (int row = rowCount; row >= 0; row--) {
239
240             if (column == columnCount && row == rowCount) {
241                // last point in path, the lowest remaining cost till the end is
242                // this cost
243
244             }
245             else if (column == columnCount) {
246                // last column, the lowest cost to the end is on down 1 position
247                matrix[row][column] += matrix[row + 1][column];
248
249             }
250             else if (row == rowCount) {
251                // last rows, the lowest cost to the end is on it's right side
252                matrix[row][column] += matrix[row][column + 1];
253
254             }
255             else {
256                // pick the lowest values down/right/(diagonal*0.5)
257                float down = matrix[row + 1][column];
258                float right = matrix[row][column + 1];
```

```
259                 float diagonal = (float)(matrix[row + 1][column + 1] * 0.5);
260
261                 matrix[row][column] += min(down, right, diagonal);
262             }
263         }
264     }
265
266     return matrix[0][0];
267 }
268
269
270 private Float min(float down, float right, float diagonal) {
271     return Math.min(Math.min(down, right), diagonal);
272 }
273 }
```

Listing B.10: Proof of concept application Gesture class

```
1
2
3 package be.johanbas;
4
5 import java.util.LinkedList;
6 import java.util.List;
7
8
9 /**
10  *
11  * @author Johan Bas
12  *
13  */
14 public class Gesture implements Cloneable {
15
16     List<Acceleration> accelerations = new LinkedList<Acceleration>();
17     private double maxDistanceToOtherGestures;
18     private String type = "";
19
20
21     public String getType() {
22         return type;
```

```
23      }
24
25
26      public void setType(String type) {
27          this.type = type;
28      }
29
30
31      public List<Acceleration> getAccelerations() {
32          return accelerations;
33      }
34
35
36      public void setAccelerations(List<Acceleration> accelerations) {
37          this.accelerations = accelerations;
38      }
39
40
41      @Override
42      public String toString() {
43          return "Gesture [accelerations=" + accelerations + "]";
44      }
45
46
47      public boolean equals(Gesture gesture) {
48          return gesture.getAccelerations().get(0).getTimestamp() == this.accelerations
49                  .get(0).getTimestamp();
50      }
51
52
53      public double getMaxDistanceToOtherGestures() {
54          return maxDistanceToOtherGestures;
55      }
56
57
58      public void setMaxDistanceToOtherGestures(double maxDistanceToOtherGestures) {
59          this.maxDistanceToOtherGestures = maxDistanceToOtherGestures;
60      }
61
```

```
62
63      public Object clone() throws CloneNotSupportedException {
64          Gesture gesture = new Gesture();
65
66          List<Acceleration> accs = new LinkedList<Acceleration>();
67          for (Acceleration acceleration : accelerations) {
68              Acceleration acc = (Acceleration)acceleration.clone();
69              accs.add(acc);
70          }
71          gesture.setAccelerations(accs);
72
73          return gesture;
74      }
75  }
```

Listing B.11: Proof of concept application Acceleration class

```
1
2
3  package be.johanbas;
4
5  /**
6   *
7   * @author Johan Bas
8   *
9   */
10 public class Acceleration implements Cloneable {
11
12     private float x;
13     private float y;
14     private float z;
15     private float timestamp;
16
17
18     public float getX() {
19         return x;
20     }
21
22
23     public void setX(float x) {
```

```
24          this.x = x;
25      }
26
27
28      public float getY() {
29          return y;
30      }
31
32
33      public void setY(float y) {
34          this.y = y;
35      }
36
37
38      public float getZ() {
39          return z;
40      }
41
42
43      public void setZ(float z) {
44          this.z = z;
45      }
46
47
48      public float getTimestamp() {
49          return timestamp;
50      }
51
52
53      public void setTimestamp(float values) {
54          this.timestamp = values;
55      }
56
57
58      @Override
59      public String toString() {
60          return "Acceleration [x=" + x + ", y=" + y + ", z=" + z + ", timestamp="
61                  + timestamp + "]";
62      }
```

```
63
64
65    public Object clone() throws CloneNotSupportedException {
66       return super.clone();
67    }
68
69 }
```

Listing B.12: Example of a batch processing configuration File

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <iGestureBatch>
3    <algorithm name="org.ximtec.igesture.algorithm.dtw.DynamicTimeWarpingAlgorithm">
4      <parameter name="TRESHOLD">
5        <for start="1" end="10" step="1" />
6      </parameter>
7      <parameter name="METRIC">
8        <for start="1" end="3" step="1" />
9      </parameter>
10   </algorithm>
11 </iGestureBatch>
```

Bibliography

[1] Ahmad Akl. A Novel Accelerometer-based Gesture Recognition System. Master's thesis, University of Toronto, 2010.

[2] Miranda C. Boonstraa, Rienk M.A. van der Slikke, Noe L.W. Keijsersc, Rob C. van Lummelb, Maarten C. de Waal Malefijt, and Nico Verdonschot. The Accuracy of Measuring the Kinematics of Rising From a Chair With Accelerometers and Gyroscopes. *Journal of Biomechanics*, 39(2):354–358, 2006.

[3] Paul-Valentin Borza. Motion-based Gesture Recognition with an Accelerometer. Master's thesis, Babes-Bolyai University, Faculty of Mathematics and Computer Science, 2008.

[4] Andrea Corradini. Dynamic TimeWarping for Off-line Recognition of a Small Gesture Vocabulary. In *Proceedings of ICCV 2001, 8th International Conference on Computer Vision*, pages 82–89, Vancouver, Canada, August 2001.

[5] Chunsheng Fang. From Dynamic Time Warping (DTW) to Hidden Markov Model (HMM). Technical report, University of Cincinnati, March 2009.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, 1994.

[7] Lode Hoste and Beat Signer. Declarative Continuous Gesture Spotting Using Inferred Control Points. Submitted to SGA 2011, 2nd International Workshop on Sign, Gesture and Activity, Alicante, Spain, November 2011.

[8] Norio Ishigaki, Teiji Kimura, Yuki Usui, Kaoru Aoki, Nobuyo Narita, Masayuki Shimizu, Kazuo Hara, Nobuhide Ogihara, Koichi Nakamura, Hiroyuki Kato, Masayoshi Ohira, Yoshiharu Yokokawa, Kei Miyoshi, Narumichi Murakami, Shinpei Okada, Tomokazu Nakamura, and Naoto Saito. Analysis of Pelvic Movement in the Elderly During Walking Usinga Posture Monitoring System Equipped with a Triaxialaccelerometer and a Gyroscope. *Journal of Biomechanics*, 44(9):1788–1792, June 2011.

[9] Youngseon Jeong, Myong Kee Jeong, and Olufemi A. Omitaomu. Weighted Dynamic Time Warping for Time Series Classification. *Pattern Recognition*, 44(9):2231–2240, 2011.

[10] Eamonn J. Keogh and Michael J. Pazzani. Derivative Dynamic Time Warping. In *Proceedings of SDM 2001, 1st SIAM International Conference on Data Mining*, Chicago, USA, April 2001.

[11] Cezary Kownacki. Optimization Approach to Adapt Kalman Filters for the Real-time Application of Accelerometer and Gyroscope Signals Filtering. *Digital Signal Processing*, 21(1):130–140, 2011.

[12] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[13] Mario E. Munich and Pietro Perona. Continuous Dynamic Time Warping for Translation-invariant Curve Alignment with Applications to Signature Verification. In *Proceedings of ICCV 1999, 7th International Conference on Computer Vision*, pages 108–115, Korfu, Greece, September 1999.

[14] Steve Nasiri, David Sachs, and Michael Maia. Selection and Integration of MEMS-based Motion Processing in Consumer Apps. Technical report, InvenSense, Inc., July 2009.

[15] Tim Oates, Laura Firoiu, and Paul R. Cohen. Clustering Time Series with Hidden Markov Models and Dynamic Time Warping. In *Proceedings of IJCAI 1999, Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, pages 17–21, Stockholm, Sweden, August 1999.

[16] Alexandra Psarrou, Shaogang Gong, and Michael Walter. Recognition of Human Gestures and Behaviour Based on Motion Trajectories. *Image and Vision Computing*, 20(5–6):349–358, 2002.

[17] Björn Puype. Extending the iGesture Framework with Multi-modal Gesture Interaction Functionality. Master's thesis, Vrije Universiteit Brussel, 2010.

[18] Stan Salvadora and Philip Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[19] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture Recognition with a Wii Controller. In *Proceedings of TEI 2008, 2nd International Conference on Tangible and Embedded Interaction*, pages 11–14, Bonn, Germany, February 2008.

[20] Beat Signer, Ueli Kurmann, and Moira C. Norrie. iGesture: A General Gesture Recognition Framework. In *Proceedings of ICDAR 2007, 9th International Conference on Document Analysis and Recognition*, pages 954–958, Curitiba, Brazil, September 2007.

[21] G.A. ten Holta, M.J.T. Reinders, and E.A. Hendriks. Multi-Dimensional Dynamic Time Warping for Gesture Recognition. In *Proceedings of ASCI 2007, 13th Annual Conference of the Advanced School for Computing and Imaging*, Delft, The Netherlands, June 2007.

[22] David L. Thomson and R. Chengalvarayan. Use of Voicing Features in HMM-based Speech Recognition. *Speech Communication*, 37(3-4):197–211, 2002.

[23] Arthur Vogels. iGesture Extension for 3D Recognition: The Wiimote as an Input Device. Master's thesis, ETH Zurich, 2009.

[24] Jiahui Wu, Gang Pan, Daqing Zhang, Guande Qi, and Shijian Li. Gesture Recognition with a 3-D Accelerometer. In *Proceedings of UIC 2009, 6th International Conference on Ubiquitous Intelligence and Computing*, pages 25–38, Brisbane, Australia, August 2009.