The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)______________________________
Summary

This project explores the possibility of using accelerometer data from mobile phone sensors for gesture recognition, and to determine a user's musical conducting patterns in order to recognise features such as time signature and tempo. The areas of conducting and gestures to control musical performances are researched, and previously applied techniques for gesture recognition are explored and adapted so that a new solution can be designed. This system is then implemented and tested to evaluate its suitability for the problem.
I would not have been able to complete this project without the help and support of several people, who I would like to take the opportunity to thank:

- Dr Kia Ng, my project supervisor, who was helpful and supportive throughout, and gave up far more of his free time than was reasonable
- My friends Robin and Elliot, for their useful and timely advice
- My parents, who have supported me for the last 23 years
- Dr Kevin McEvoy, for his help and support during my entire degree at Leeds
## Contents

1 Introduction ..................................................... 1
   1.1 Project Aim .................................................... 1
   1.2 Objectives ...................................................... 1
   1.3 Minimum Requirements ..................................... 1
   1.4 Possible Extensions ........................................ 2

2 Background Reading ............................................. 3
   2.1 Music and technology ........................................ 3
   2.2 Mobile devices ............................................... 4
   2.3 Gesture Recognition ........................................ 5
      2.3.1 Touch-based Systems .................................... 5
      2.3.2 Vision-based Systems .................................... 6
      2.3.3 Inertia-based Systems .................................. 7
   2.4 Conducting ..................................................... 8
      2.4.1 Recognising Conducting Gestures ....................... 8
   2.5 Multimedia Interactions .................................... 10
   2.6 Dynamic Time Warping ...................................... 10

3 Planning and Methodology ...................................... 12
   3.1 Methodology ................................................... 12
      3.1.1 Waterfall Models ....................................... 12
      3.1.2 Iterative Models ....................................... 13
5.3 Time signature recognition ............................................ 33
5.4 System Latency .......................................................... 33

6 Conclusion ..................................................................... 35
6.1 Discussion ................................................................. 35
6.2 Extensions and future work .......................................... 35
1 – Introduction

1.1 – Project Aim

The aim of this project is to investigate the use of inertial sensors in a mobile device for gesture recognition, and build an application which uses the data from such sensors to recognise specific gestures from a user. The application will provide a realtime visualisation and sonification of the recorded gesture data to provide feedback to the user.

1.2 – Objectives

The project objectives are to:

- Conduct research into previous work in gesture recognition and inertial sensor data to gain understanding of the problem
- Create a mobile app to capture data from the sensors, and transmit it to a desktop via WiFi
- Design an algorithm that will recognise the beat points in the data, and segment it accordingly
- Design an algorithm which, using the segmented data, will recognise the time signature that the user is playing in
- Provide output to the Music via Motion framework via UDP
- Integrate all modules to create a live system

1.3 – Minimum Requirements

The minimum requirements for this project are:

- Capture the data from the mobile device's sensors and relay it to the device that will perform the analysis
- Display the data in a manner suitable for analysis
• Implement algorithms to analyse the data and recognise specific gestures (a beat point)
• Integrate a GUI to map the detected gestures onto configurable multimedia events

1.4 – Possible Extensions

The possible extensions are:

• Compare a user’s gestures with their previous gestures and provide feedback on how similar they are
• Allow the user to control the volume of the sonification via their gestures
In order to properly plan, design and implement a solution, it is necessary to fully understand the problem. In this chapter, each facet of the problem will be explored, by looking at relevant literature. Previous work which is closely related will be reviewed, and techniques which are applicable to the project will be identified and evaluated.

2.1 – Music and technology

Since the beginning of human culture, music has been an important method of expression. Musical instruments were created to supplement the human body’s natural ability to make sound (e.g. clapping and singing). Curt Sachs (1940) provides some insight as to the development of instruments as history progressed. As technology improved, and new techniques for working natural materials were discovered, the makers of instruments had ever more choice as to their tools and raw materials. Progression was made from simple drums used to create a rhythm to drums that could produce a melody. Stringed instruments (the lute, and the lyre) were developed, along with wind instruments such as the clarinet and pan-pipes, followed by more recent developments such as the keyboard, and the plethora of modern electrical instruments.

The exact chronology is not known, but it is clear that as science progressed, so did musical instruments, their makers driven to use the latest technologies to create the sounds demanded by popular culture. In recent years, computers have become ubiquitous, and there are countless methods for using them to create music.

2.2 – Mobile devices

The hand-held mobile phone was first demonstrated in 1973, and in 1983 the DynaTAC 8000x was the first to become commercially available (Time.com, 2010). Originally intended to be only for making and receiving phone calls via the public telephone network, mobile phone subscriptions have increased to the point where the vast majority of people in the developed world own one. With increases in technology, mobile phones have become more like hand-held computers, most
noticeably with recent “smartphone” developments.

Smartphones combine the functions of a camera phone and PDA, running a complete operating system on compact but ever more powerful hardware. With features such as a GPS locators, magnetic and light sensors, 3-axis accelerometers, and gyroscopes, smartphones are equipped to run a wide array of services and applications that the user can choose from online markets. Since their introduction to the market, smartphones are quickly becoming adopted, with 22% of UK consumers owning one – increasing to 31% in the 24-35 year-old group (olswang.com, 2011).

2.3 – Gesture Recognition

Throughout history, gestures have been an important tool for communication. By using movement of parts of the body, usually the hands, verbal communication can be enhanced or supplanted by gesturing. Gestures and speech are processed in the same area of the brain, and it has been suggested that symbolic gestures and spoken language developed together as communication evolved. (Xu et al, 2009) In everyday life, gestures are common, whether waving in greeting, pointing out something of interest, or simply embellishing speech with expressive hand movements.

As technology has progressed, existing input methods are becoming increasingly outdated and inapplicable. While a keyboard worked for command line input, a mouse allows for a more intuitive way to control a system via a graphical interface. More recently, multi-touch technology is creating opportunities for advancements and controls that would not be possible with a single pointing device like a mouse.

If we look specifically at mobile devices, they have progressed extremely rapidly in recent years. While mobile phones were originally purely for making and receiving phone calls, they have morphed into personal computing devices, first with the advent of SMS messages, then with mobile operating systems and wireless internet access, and most recently the smartphone. Throughout these advancements, the input method has remained mostly the same – the keypad. While perfect for its original intent – text input – it has been used as the sole method of input, foregoing more natural and intuitive methods.
2.3.1 – Touch-based Systems

Touch-screen technology is becoming more widely used in larger applications, such as public use terminals and screens. Multi-touch screens, recently developed, allow multiple points of contact to be registered as inputs, creating a far larger variety of potential input commands. These allow users to interact with the devices more efficiently, depending on the tasks being performed.

Many smartphones on the market are fitted with touch-screen interfaces, allowing users to interact with the operating system via gestures on the screen, rather than using the keypad. These gestures tend to be simple – either a press down at a point on the screen corresponding to a button or function, or a scroll/flick – pressing down followed by a linear movement, generally causing the screen to scroll in the direction moved. These gestures allow a user to quickly and accurately select a point on the screen in a single movement, while using a keypad may require moving through multiple intermediary points to reach the destination.

Lorenz et al (2010) conducted a formative comparison between keypad, touch-screen, and gesture inputs. The touch-screen was used for both software buttons, and for gestures, while it was also possible to make gestures by drawing in the air. They present results demonstrating that the hardware buttons are significantly quicker than other methods, and gestures in the air were slower, although using the “StaticArea” gesture (moving a set distance in a certain direction) was in fact comparable with other input methods. It is suggested that this is due to the much greater familiarity with hardware buttons that the users had (eg, not having to look at the buttons to perform tasks, whereas with software buttons they had to constantly look between the main display and the phone display). Time had to be invested in learning the gestures, but not the buttons. Their sample size and gesture set were both small, but their study seems to show that gestures can be quickly learnt and yet still be comparable in usefulness to existing input methods.

Touch-based systems have also been used for handwriting recognition and input using a tablet and stylus. A common method is to capture the location of the stylus on the tablet at points during its movement, and then to reconstruct the gesture using those points. Narayanaswamy et al (1999) describe a design for a smartphone user interface which includes a touch-screen input and handwriting recogniser, using a Hidden Markov Model (HMM). Wei et al (2008) compared touch-screen gestures to hardware buttons for a mobile phone First Person Shooter game, finding that gestures were considered easier to use.
Mobile application developers are also using the touch-screen capabilities of devices like the iPhone to create apps which simulate musical instruments. These include the iBone (a trombone simulator), Pocket Guitar, and the Ocarina by Smule, an app which also uses the microphone to create an experience close to playing a real ocarina.

For this project, touch-based interaction was considered, due to the high sample rate and accuracy of the touch-screen on most smartphones. However, the prohibitively small amount of space provided by the screen of the smartphone means that as gestures become more complex, they become more difficult to accurately reproduce on the screen. Multi-touch interaction is also not feasible on a mobile device, due to the need to hold the device in one hand.

### 2.3.2 – Vision-based Systems

Computer vision is another discipline which people have been applying to the problem of gesture recognition for some time. Solutions vary greatly in method and application. A vision-based method of determining beat points is described by Nakra *et al* (2009). Using very limited images (the video was not originally intended to be used in computer vision), the conductor’s hand is segmented from the rest of the image, and then tracked in subsequent frames. Following the y co-ordinate, they plotted a graph of the hand’s height over time, and compared this to the beats noted in the score. It was found that there is a correlation between sharp changes in height (minima and maxima on the graph) and beat points.

Kolesnik and Wanderley (2004) began by creating a HMM-based system for recognising symbols, using 2-D co-ordinates as input. Their conducting gesture recognition system includes two cameras, one directly in front of the conductor and one showing the conductor in profile. The conductor wore a coloured glove, which was used to segment the hand and provide a 2-D co-ordinate describing the position of the hand. This system could recognise 5 left handed gestures and 2 right handed beat patterns with 94.6% accuracy. Beats were segmented using the same information, by finding minima and maxima of positional values. Features of the gestures made by the conductor (such as time between beats, and position of the left hand) were used to control the tempo and volume of the audio playback respectively.

Vision-based recognition was also considered, but using a mobile phone as an input device would
require either a second person or a stand of some kind in which to place the phone, to allow the camera to remain still and frame the user properly, meaning it was less feasible for easy, one-person use. Alternatively, the camera on the phone could be used while held in the hand during the gestures, comparing subsequent images to determine the direction and distance moved. However, it was decided that this posed a greater problem than using alternative methods.

2.3.3 – Inertia-based Systems

By using inertial sensors such as accelerometers, it is possible to infer information about the forces acting on a device.

3-axis accelerometers are common among modern smartphones, and have been used before in gesture recognition. In Choi et al’s (2005) Beatbox Music Phone software, data from the accelerometer in a Samsung SCII-S310 mobile phone is used to recognise numbers and symbols drawn in the air with the phone itself, and the recognised inputs used to control the phone’s functions. By simple thresholding on the maximum value out of the three axes, they show it is possible to recognise when the phone is being shaken. By extracting feature points (minima and maxima for each axis) and using them as inputs to a Bayesian network, they demonstrate an average of 97.01% recognition across a set of 11 gestures (numbers 1-9, letters O and X). They also demonstrate that it is possible to recognise the beat point of a rhythmic shaking, using a rule-based process. This was a method that influenced the design of the algorithms in this project.

Strachan et al (2007) describe the BodySpace system, a method for using accelerometers to control music playback on a mobile phone. By comparing pitch and roll values, it is possible to differentiate between different locations in which the phone may be held, for example the ear or the hip. These are used for the “modes” of the system. They then implement a recognition system for determining when a user has flicked the phone, which is mapped to a function according to the mode – if the phone is near the ear, flicking the phone could change between tracks, while at the hip it would control the volume.

Guerreiro et al (2008) created a similar system named “Mnemonical Body Shortcuts”, which aims to allow mobile phone users to run apps on their phones using gesture shortcuts. The gestures they implemented were simple movements of the phone towards different parts of the body: mouth, ear, shoulder etc. Using Bayes classifiers, recognition rates of 80.5% while moving and 89.5%
were achieved, showing that accelerometers can be used to recognise specific movements of a mobile device.

2.4 – Conducting

A conductor is the person responsible for the direction of an ensemble of musicians during the performance of a piece of music. They must ensure that the performers are unified, that the tempo is clear, that members of the ensemble are cued in at the correct times, and that the sound of the group is cohesive. In order to do this, they utilise a number of expressive gestures using primarily their hands (with a baton held in the dominant hand), but potentially any part of their body. By doing so, they convey both technical information about the piece being played to the musicians, and have the ability to impart their own style and flair upon the piece, meaning the role of conductor is artistic and interpretive, not merely proscriptive.

To convey the tempo of the music, a conductor will move their baton in a set beat pattern, varying according to the time signature that the piece is in. Each separate beat is indicated with a sharp change in direction from downwards to upwards, called the ictus. In this manner, the musicians can see when each bar has begun, and where each beat within the bar falls. By changing the size of their gesture, the conductor can convey the volume they wish the piece to be played at – a larger, more expressive gesture conveys a louder piece. To cue elements of the ensemble to begin playing, gestures vary from eye contact to more overt movements such as a gesture with the non-dominant hand. The conductor can provide much more information to the musicians by subdividing the beat, changing the character of the ictus to be more or less staccato, and by making their entire body language quick and jerky or smooth and flowing.

2.4.1 – Recognising Conducting Gestures

Automatic recognition of conducting gestures is a problem that has been undertaken by several groups of people. The Personal Orchestra program created by Borchers et al (2002) uses an infra-red baton to capture a user’s movements. By calculating the first derivative of the y co-ordinate of the baton’s trajectory, and looking for turning points, they find the beat point. This is then used to calculate the tempo of the user’s playing, which is mapped onto a video of an orchestra being displayed to the user, giving them control over the speed at which the music is played. By pointing
at sections of the orchestra, they can also increase the volume of that section. The system is quite limited in that it can only recognise an up-down motion from a user.

Building on this paper, Lee et al (2006a) created a system called iSymphony, which was similar in concept. However, by using the conga framework, also developed by Lee et al (2006b), the system can recognise more varied gestures from the user (the same up-down motion as the Personal Orchestra, a four-beat conducting pattern, and random gestures). The users' gestures are compared to a known set of gesture profiles, as the volume of users mean that no training is possible. These are then mapped to the playback accordingly, the up-down and four-beat patterns give the beat points which can be used to find the tempo, while the speed of the random gestures is mapped to the music without finding specific beat points.

Je et al (2007) propose a different solution, using a stereo video camera. By using a face detector, they look at points closer than the depth of the user's face to find the co-ordinates of the user's hand. By tracking the co-ordinates, they capture the directions that the hand is moving in, and quantize them into 8 directions. They then create a histogram consisting of these directions, and compare it against known histograms for the recognisable time signature. By doing this, they achieve a high (86.5%) rate of success.

The Digital Baton designed by Marrin (1997) is a gestural controller which includes an infra-red LED for position tracking, three orthogonal accelerometers, and 5 pressure sensors for the fingers and palm. The infra-red sensor and accelerometers measure the position of the baton in 3D, providing data which can be used to find the tempo and beat. The pressure sensors can be used to vary other input that the user has pre-set.

Similarly, Pinocchio is a system developed by Bruegge et al (2007). It runs on a laptop, and has a control panel through which a user can remove or reposition musicians in their orchestra. The input device is either a baton or Wiimote, as chosen by the user. An easy-to-learn conducting gesture is recognised by data from the Wiimote's accelerometer sensors, or by tracking the baton via a camera on the laptop. The user can control the tempo and the volume of the virtual orchestra with their gestures. Classification of gestures is done by neural networks.

Lee et al (2005) carried out an evaluation of previous conducting-based work, and found that there were a variety of “usability breakdowns” present. For example, users who perceived that the music
was slowing down may also slow their conducting tempo, causing a feedback loop where the music would get increasingly slower. One of the more interesting of their topics of discussion was that conductors are in fact taught to conduct ahead of the orchestra, meaning that there is a latency between when a conductor gestures for a beat, and when the orchestra plays it. Their value for this latency is 152 ± 17 ms. They also found that trained conductors are far more consistent in their gestures than untrained. Both of these factors must be taken into account in this project.

2.5 – Multimedia Interactions

Ng's Music via Motion (2004) is a framework designed to “map detected changes in an environment to multimedia events”, with the purpose of creating a tool to be used for audio-visual performance and musical expression. MvM has several mapping functions, mapping movement from left-right into a change of pitch, for example. Three implementations are presented: the Coat of Invisible Notes, the Interactive Music Head, and the Augmented Drum.

For CoiN, MvM is configured to react to movement of colours. Hence by wearing a costume and dancing, musical feedback is provided. By changing the costume colour, the feedback can be altered as the performer wishes. The Interactive Music Head tracks movements of individual facial features independently, and uses them as input – the given example is of a mouth opening to apply a low pass-filter. Finally, the AugDrum system consists of a flex sensor embedded in a drum brush, allowing the drummer to control a MIDI or other multimedia event while using a drum kit to provide percussion. The highly configurable nature of both the input and output of MvM means that it will be a useful method of providing multimedia feedback to a user.

2.6 - Dynamic Time Warping

Dynamic Time Warping was first proposed by Sakoe and Chiba (1978) for use in the field of speech recognition, but it has become more widely used. It is an algorithm for comparing the similarity of two time series, by finding the minimum cost of warping one series into the other. A time series is a sequence of data points sampled at successive times, and at uniform intervals.

Hartmann and Link (2010) show an example of how well DTW can work with accelerometer data. They applied DTW to recognise a set of gestures performed by a user with an accelerometer.
strapped to their wrist. Their system had a precision of 97.35%, and a recall of 85.86%. The ability of DTW to match two sequences which are similar but vary in speed means that it will potentially be highly useful for this project, where similar beat patterns will be performed at different tempos.
3 – Methodology and Planning

3.1 – Methodology

Historically, large software engineering projects have failed to meet expectations; they are delivered late, overbudget, or lacking in functionality, if at all. The primary reason for this was a lack of acceptance of software development methodologies, which are now considered essential for improving productivity and product quality by defining repeatable and consistent processes to be used. There are many different development methods in use, suitable for different styles of project, according to the size, user requirements, schedule, and so on.

Choosing and following a development methodology is something that was clearly of great importance in keeping the project focused and ensuring that the final product was of a high standard. When deciding upon the methodology, several properties of this specific project were kept in mind: the (relatively short) time allowance, the fact that there would be only one programmer, and the lack of knowledge regarding the form the data would take.

3.1.1 – Waterfall Models

The Waterfall Model (and similar models, such as the V-Model) are rigid, sequential design processes, where development passes through defined stages (Requirements, Design, Implementation, Verification, Maintenance) linearly, completing one stage before beginning the next. This requires a large amount of knowledge up-front, and a comprehensive analysis of requirements which are unlikely to change during development. Due to the potentially variable nature of the data that would be used, it was decided that such models were unsuitable.

3.1.2 – Iterative Models

The iterative model was developed in response to perceived flaws in the Waterfall model. (Larman & Basili, 2003). Rather than the rigid format of the Waterfall, the iterative process is more flexible, providing functionality in increments. Each increment has its own requirements, design, implementation and validation procedures. This allows for learning throughout the whole project,
building on what was discovered in earlier increments, and also means backtracking is possible.

Recently, many iterative models have been proposed and become more popular. By providing working prototypes earlier on in the project, it is possible for the user to give feedback much earlier than would be possible with a more proscribed methodology.

3.1.2.1 – Agile Development

Agile methodologies are designed around delivering working software prototypes frequently, with the software itself being the measure of progress. Small teams of coders work closely with one another and the user, providing the ability to adapt very quickly to any changes in the requirements of the project. Short iterations go through a full development cycle, but may not always add significant functionality or new features.

3.1.2.2 – Feature Driven Development

Feature Driven Development is a process with short iterations that are driven by required features. An overall model is developed, and a feature list built. Each feature is planned and designed. Finally, features are built separately, and promoted to the main build as they pass testing.

3.1.3 – Conclusion

Considering the specifics of this project, it was decided that Agile Development would be the primary methodology used, although elements of Feature Driven Development would also be applied. A list of features would be created, and each assigned to an iteration of the software. Each feature would then be built and tested, then modified as necessary as later features were added. This would mean that the development process would be adaptable as more was discovered about the data. It would also be possible to evaluate each feature of the software as it was created, and observe how adding functionality changed the results of the same evaluation.

3.2 – Schedule

The originally devised schedule can be found in Appendix C. The intention was to first gain an
understanding of the problem and conduct background research to determine the best approach to take, then to design the software, and begin to write these sections of the report early on. After submitting the mid-project report, emphasis would switch to building and testing of the software, progressing through iterations while adding features, and evaluating each addition both alone and as a part of the system as a whole.

### 3.3 – Revised schedule

Heavy delays were present early on, due to medical reasons, and it was clear the schedule would need to be changed to accommodate. While most of the background reading had been done, and notes taken, little writing was complete. However with time running out, the decision was made to prioritise software development, while writing when time was available. With an extension granted, the time pressure was eased somewhat, and the schedule was amended to the one seen in Appendix C. As writing was taking place continually, background reading also continued.
4 – Design and Implementation

This chapter will describe the steps taken in designing and building the system, discussing system architecture and the planning and creation of the modules necessary.

4.1 – Choice of Mobile Platform

One of the first choices to be made in the project was to decide upon the platform to be used for development. Options included Nokia's Symbian^3 platform, Apple's iPhoneOS, Microsoft Windows Phone 7, and Google's Android.

When deciding which was the best to develop on, several factors were taken into consideration:

- Ease of learning (programming language used, available tools and resources)
- How widely used the platform was among smartphone users
- Availability of hardware for development and testing

Looking at market share, [http://www.canalys.com/pr/2011/r2011013.html], it can be seen that Google's Android platform is both the most popular and the fastest growing in popularity. Symbian has a similar share, but is gradually losing market share, while iOS is a distant third, although gaining on Symbian.

The native language for iOS is Objective-C, a language developed by Apple which adds Smalltalk features to C. Development takes place in the iOS software development kit (SDK), including an iPhone simulator. However, an iPhone Developer Program fee must be paid for the ability to install apps on physical devices.

Android also has an SDK, freely available and also inclusive of a mobile phone simulator. Development on Android is done in Java, with a full API of inbuilt functions provided.

Symbian supports several languages, including C++ and Java, and development is done using standard tools. There is a large developer community, however Nokia has announced that while
Symbian will remain supported, they will change to using Windows Phone 7 on new devices in 2011.

Before the project started, I had already gained experience in programming in Java throughout my degree course, while having no experience in C or C++. With the availability of the Android SDK, API and Emulator, as well as readily available smartphones running Android and the increasing popularity of the Android platform, the choice was made to use it for the project.

For a development environment, the Java IDE Eclipse was used, as it contains an Android plug-in which allows the emulator to be run from within the Eclipse GUI. The mobile phone that testing was done on was a HTC Desire.

The accelerometer API provided by Android gives the co-ordinates relative to the position of the screen of the phone. While looking at the front of the screen, the X axis is horizontal and positive to the right. Y is vertical, and positive upwards. Z points directly out of the screen, and is positive towards the viewer. This is illustrated in figure 4.1 (from developer.android.com).

![Android Coordinate System](image-url)
4.2 – Data Analysis

Before any design could take place, it was first necessary to collect some initial data, as there was no prior knowledge as to the form that the data from the accelerometer would take. In order to capture this initial data, a free application from the Android Market called SensorDump was used. This app allows a user to choose a location on the phone's SD card in which to save a comma separated value (CSV) file, and then begins recording. While recording is occurring, the readings for each of the X, Y and Z axes of the accelerometer are displayed on the screen, and simultaneously written to the CSV file. The CSV file consists of three header lines, followed by a line for each point at which data was captured:

"Dump of Accelerometer: BMA150 3-axis Accelerometer"

"Uptime Timestamp (ns),"Unix Epoch Timestamp (ns),"X-Accel (m/s^2),"Y-Accel (m/s^2),"Z-Accel (m/s^2)"
"661036945709000,"1303995897266709000","-0.0","2.8330324","9.193735"
"661037024689000,"1303995897345689000","0.08172209","2.6014864","9.507003"

Here you can see how the header lines are formatted, and how the data is saved. The first two sections of the data line are timestamps – the Uptime Timestamp is the the time that the phone has been switched on, in nanoseconds. The Unix Epoch Timestamp is the time since the Unix Epoch (midnight on January 1\textsuperscript{st}, 1970) in nanoseconds. All the data points are enclosed in double quotes, and separated by commas.

CSV files are easy to read using a spreadsheet program such as LibreOffice Calc. They can be imported in a manner which will use the commas to separate data between cells of the spreadsheet, and can also strip the double quotes used. Calc also provides the ability to create graphs, which was useful for finding patterns within the data. Fig. 4.2 shows a small sample of the data collected while performing a 2/4 beat pattern.

The data from the accelerometer is provided in the format of a double, and by viewing the data it was possible to see that the maximum value that the accelerometers can register is approximately 19.6 m/s\textsuperscript{2}. By comparing the graph of time and acceleration against a video recorded of the test
being performed, it was possible to observe that large peaks in the graphs were directly correlated with sharp changes in direction. The X axis in particular features very prominent and clear peaks at regular intervals, which made it appear suitable for segmenting the beats.

![Sample graph of CSV data](image.png)

**Fig 4.2** – Sample graph of CSV data

After this initial exploration of the format of the data, it was then possible to design the system architecture.

### 4.3 – System Architecture

Given the nature of the MvM output, it was always going to be necessary to communicate from the phone to a desktop. There were concerns about hardware of the mobile phone, and whether or not it would be possible to conduct all the computing on the phone without significant latency being introduced. Given the limited time provided, it was decided to develop the majority of the system on a desktop, as a proof of concept.

The decision was also made to first create a system that would use CSV files as a method for transferring data from the phone to the desktop. This would allow the test data to be recorded in one session, rather than requiring testers to continually attend testing sessions, as would be
required with a system that was live from the start. After the beat segmentation, feature extraction and recognition algorithms were working adequately using CSV inputs, then the UDP modules could be implemented, and all the modules integrated into the final live system. Figure 4.3.1 shows the initial design of the offline system, while 4.3.2 shows the final live system.

Fig 4.3.1 – System Architecture – CSV version

Fig 4.3.2 – System Architecture – live version
4.4 – Capturing Sensor Data

In order to capture the sensor data, an Android application was necessary. This app had to first communicate with the hardware sensors to retrieve the acceleration data, then either store it in a CSV file or transmit it to the desktop via a UDP socket. All Android applications which provide a window that the user can interact with are called Activities. These have some standard methods which must be implemented:

- `onCreate()` - called when the activity is first started, and is where all static set-up (creating the viewing window, for example) should take place
- `onStart()` - called just before the activity becomes visible to the user. This is not mandatory.
- `onResume()` - called whenever the activity is running in the foreground of the mobile device.
- `onPause()` - called when another activity comes into the foreground, but the original activity is still visible
- `onStop()` - called when the original activity is no longer visible

In the `onCreate()` method of the app, we first set the layout of the window. As any user would find it difficult to view feedback on a mobile that they are gesturing with, this was kept very simple in order to facilitate testing and debugging – the screen shows the current X, Y and Z acceleration values, and the filename of the capture CSV being recorded to.
Next, an instance of the SensorManager class is created. This allows interaction with the device’s sensors. As we are only interested in the accelerometer data, we only use the SensorManager to find the default accelerometer. A new File object is created for the CSV, stored on the SD card by using Environment.getExternalStorageDirectory() to find the location of the SD card. A PrintWriter is opened to the file, and the CSV headers are printed.

In onResume(), we use the SensorManager to register a SensorEventListener, which is used for receiving notifications from the SensorManager when any sensor values have changed. This class also has a method which must be implemented, onSensorChanged. This is called whenever a change in sensor value is detected, with a SensorEvent parameter. In this method, we use Calendar.getInstance(); to obtain the current system epoch timestamp, and Android.os.SystemClock.uptimeMillis() to obtain the system uptime. We then loop through the values in the SensorEvent, and write them to a new line in the file using the PrintWriter, along with the timestamp.
Finally, in onPause we unregister the SensorEventListener and close the PrintWriter, renaming the CSV file with a timestamp so it will not be overwritten.

This application records all the changes in acceleration detected by the phone's accelerometer, and stores them in a CSV file in a similar format to that used by SensorDump, although no quotes are used.

In order to modify this application to send a UDP packet with the data, first a string must be constructed using the values and timestamps. Now, we create a new thread for the socket to run in. This prevents slowdown, as the UI interactions would otherwise have to wait for the UDP to complete before continuing. A UDP packet requires a byte array as a parameter, so we construct a byte array from the value string. Now a new DatagramSocket is opened, a new DatagramPacket is created using the byte array and desktop destination URL and port, and the packet is sent by the socket.

The application can now both write to a CSV file, and send a UDP packet containing the same data to a desktop UDP server over WiFi. Either of these functions can be disabled to prevent any latency in the other. An example of the CSV file and UDP packet format are below:

Calendar time, Uptime, X, Y, Z
1308240256344,438708494,-0.14982383,8.008764,5.134871

4.5 – Beat Segmentation

Beat segmentation is a crucial part of the system, as it will be used in all the other modules – for extracting features such as the tempo, for aiding recognition, and for providing output information. Hence, it is important that the segmentation is accurate and quick. By the nature of the gestures being performed, the beat points come at the ictus – a point in the movement where there is a sharp change in direction. Figure 3.5 shows the gesture patterns for the three time signatures considered in this project – 2/4, 3/4 and 4/4 legato, as the conductor would view them (and performed right-handed), and illustrates where the ictuses (and thus beats) fall.
4.5.1 – Segmentation Algorithm 1

A change in the direction of movement is caused by acceleration. As these changes are so sharp, they should be clearly visible in the data as peaks. Two approaches were considered, and compared against one another. The first was to use the total acceleration being applied to the device as an estimate of the force being used. At beat points, it was theorised that the forces applied would be higher than during the movement between beat points. Total acceleration was calculated by taking the square of the acceleration of each of the X, Y and Z axes, and then summing them together. The second approach was to only look at the X axis, with the reasoning being that every ictus has a change from downwards to upwards motion, in the positive X direction.
Figure 3.5.1.1 shows an example graph comparing the two approaches. In the graph, the X-only values have been multiplied by 30 to scale them for viewing more suitably. In order to detect the beat points, a local maximum function was used – if a point was larger than its immediate neighbours then it was a local maximum, and if it exceeded a pre-set threshold, then it was considered to be a beat point. Figure 3.5.1.2 shows the results of this algorithm for both approaches, with beat detections indicated by a vertical line.
It can be seen that both approaches have drawbacks in this algorithm. The combined axes approach has detected a false positive, while the X axis only has plateaus rather than clear points – this is due to the value exceeding the maximum that the accelerometer can measure.

4.5.2 – Segmentation Algorithm 2

To improve upon the algorithm, it was decided to use a sliding window for finding the local maxima. Instead of only considering the adjacent values, 2 additional values to either side would be considered in determining whether a peak was a true beat point. As the number of values used increases, the amount of time spent on computation also increases. As values in front of the beat are also necessary for detection of a maximum, increasing the number of values considered effectively adds one sample’s worth of latency for each value, as the algorithm must wait for the values to arrive before considering them. In order to alleviate the computational requirements, an addition was made to the algorithm so that after detecting a beat, it would ignore the next 4 values. This number was chosen based on the average number of samples between each beat, discussed in the evaluation section of this report. Figure 3.5.2 shows the results of the new detection algorithm.
False positives were vastly reduced using the new implementation. A direct comparison of the two approaches showed that using the X axis only provided much more consistent results in terms of the number of samples between each beat detection. The double peaks seen in the summed axes result in a beat at one of the two peaks, while the X axis peak will be between the two. The accelerometer also reached its maximum value even with relatively small, gentle movements, which meant that the threshold for the beat detection could remain static despite the manner in which the user was conducting. Using the sum of the axes, the peaks were much less uniform in
height even in a single recording, and a threshold that worked for a very expressive recording would prove too high for a softer one. Implementation of these algorithms was originally done using the CSV files. The entire file was read into an ArrayList, and then the program would loop through the ArrayList, applying the algorithm to each value in turn. To modify this for use in a live setting, each value string is read into an array. The most recent 5 arrays are used for comparison, to determine whether the centre array of the 5 is a beat or not. This does mean that 2 samples of latency are added, by being forced to wait.

4.6 – Tempo Calculation

With the beat segmentation complete, tempo becomes quite simple to calculate. As each beat is determined, a timestamp is associated with it. By subtracting the timestamp of the previous beat from that of the current beat, the number of milliseconds between the beats is calculated. Dividing 1000 by this number gives the tempo in Hertz (beats per second), and multiplying this by 60 gives the beats per minute, the standard unit of tempo. This can then be added to a running average, to give a more accurate indication of the tempo that the user is conducting at.

4.7 – Time Signature Recognition

Two algorithms were considered for the recognition of the time signatures, a dynamic time warping algorithm and a simpler, rule-based algorithm.

4.7.1 – Rule-based

The rule-based algorithm was designed by looking at the movements made by the conductors. Referring again to figure 3.5, it is possible to see that in the beat patterns for 3/4 and 4/4, there is only one significant movement to the conductor's right, and the rest of the movements are to the left, or vertical. A movement to the right is in the negative Z direction, and so will appear on a graph as a negative peak (or valley). The rule-based algorithm aimed to find the occurrences of these minima, and by observing the frequency of them, determine which beat pattern was being used – if the minima were occurring every 3 beats, it would be a 3/4, and if they occurred every 4 beats, it would be a 4/4. Generally in the 2/4 playing, the minima are much smaller in size, as the movement from side to side is quite small.
The implementation was very similar to the beat segmentation algorithm, comparing a value's neighbours and applying a threshold to determine whether or not a rightward movement has taken place.

4.7.2 – Dynamic Time Warping

Consider a pair of time series $s[1...m]$ and $t[1...n]$, and a distance function $d(x,y) = |x-y|$. To compare the two time series, a distance matrix $D$ is first constructed, where $D(i,j) = d(s_i,t_j)$. Now a continuous, monotonic path is constructed from $D(0,0)$ to $D(m,n)$. These constraints on the path mean that each point $D(i,j)$ can only lie on a path coming from $(i-1,j-1)$, $(i-1,j)$ or $(i,j-1)$.

A cumulative cost matrix $C$ is constructed, where $C(i,j) = d(i,j) + \min[C(i-1,j-1), C(i-1,j), C(i,j-1)]$

This cost matrix contains the optimal path for warping $s$ into $n$, and the total cost for this is given by $C(m,n)$.

Pseudocode is given below:

```plaintext
for ( i == 1 to m )
    for ( j = 1 to n )
        DTW [ i, j ] = infinity

DTW [ 0,0 ] = 0

for ( i = 1 to m )
    for ( j = 1 to n )
        cost = distance(s[i], t[j])
        DTW [ i,j ] = cost + minimum ( DTW [ i-1,j-1 ]
                                      DTW [ i-1,j ]
                                      DTW [ i,j-1 ]
                                    )

return DTW [ n,m ]
```
In order to apply DTW to the data, a pre-existing library called FastDTW was used. This accepts a pair of CSV files by default, and returns the cost of warping one into the other, with the option to also display the warp path through the matrix. It is also possible to change the parameters to a custom object, but before this was done, the DTW was tested on the CSVs already recorded. The CSVs were segmented by beat and then by into many smaller CSVs, each containing the values of one axis during one beat segment. It was then possible to compare beat segments against one another, by comparing $x_1$ with $x_2$, $y_1$ with $y_2$, and $z_1$ with $z_2$, and then summing the costs for a total warp cost. Beats with the same time signature should have low costs, while different time signatures should have high costs. However, during this initial testing, it was found that the recognition rate was very low, and the decision was made to use the rule-based algorithm due to time constraints.

4.9 – MvM Feedback

The final section of the project is the feedback to the user via the MvM framework. MvM accepts a UDP packet containing a custom header, and uses the data from the packet as described by pre-set methods, one for each possible input header. In the case of this project, the format for the UDP packets was:

\[ 3ddata:x-value:y-value:z-value:beat?:timesignature \]

3ddata describes the type of data contained in the packet, and indicates how MvM should handle the data. The x, y and z values are the acceleration values from the sensor capture. The last two fields are a boolean, describing whether or not there was a beat at this point, and a number 2, 3 or 4 representing the time signature recognised. Note that as beat segmentation must take place before this can be sent, the set of values transmitted is 2 samples behind the actual data currently being read by the sensors.

To transmit this information from the desktop running the Java server to the desktop running MvM, a UDP socket is used, in a very similar manner to that used in the Android app. A string is constructed, using the values discussed above. A byte array is then created using the string, and finally the byte array is sent in a packet by a DatagramSocket.

The acceleration values are mapped onto a 3-dimensional grid, where the co-ordinates
[0.0,0.0,0.0] represent the centre of the grid. When a user is conducting with the phone, a moving display of their acceleration data is shown on the screen of the MvM desktop. When a beat is detected, it is displayed on the screen by changing the size and colour of the current data point, as seen in figure 4.9.

Fig 4.9 – Sample MvM output
5 – Results and Evaluation

In this chapter we will look at the results obtained by using the system, and critically evaluate them.

5.1 – Data Capture

For data capture, the primary value to observe was the sample rate of the sensors, and the variation in the sample rate. Sample rate was calculated by using the timestamps. For values $i$ and $i+1$, the time between them was $\text{timestamp}(i+1) - \text{timestamp}(i)$. This number was expressed in nanoseconds, so it was divided by $1000000000$ to get a time in seconds.

The average time between samples was 0.0804s, giving a sample rate of 12.43Hz. The standard deviation is 0.00222s, 2.76% of the mean. Thus we can see that the sample rate of the accelerometer is quite consistent.

5.2 – Beat Segmentation

In order to test the accuracy of the beat segmentation method, several sets of data were first labelled by hand with the ground truth values – ie the points at which the beats occurred. Both algorithms 1 and 2 were tested, and both approaches (sum of squared axes and X axis only) were tested. The beat segmentation was tested on each time signature separately.

<table>
<thead>
<tr>
<th>2/4</th>
<th>Beats recognised</th>
<th>Beats missed</th>
<th>False Positives</th>
<th>Success (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>Sum of axes</td>
<td>248</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>X axis only</td>
<td>253</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>Sum of axes</td>
<td>255</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>X axis only</td>
<td>258</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3/4</th>
<th>Beats recognised</th>
<th>Beats missed</th>
<th>False Positives</th>
<th>Success (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>Sum of axes</td>
<td>311</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>X axis only</td>
<td>309</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>Sum of axes</td>
<td>315</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>X axis only</td>
<td>319</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Beasts recognised</td>
<td>Beats missed</td>
<td>False Positives</td>
<td>Success (%)</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>4/4</td>
<td>Algorithm 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sum of axes</td>
<td>216</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>X axis only</td>
<td>218</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Algorithm 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sum of axes</td>
<td>219</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>X axis only</td>
<td>219</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

From the results, it is possible to see that all the beat detection algorithms were very successful. However, using the sliding window implementation on the X axis alone provides almost 100% accuracy. This is attributed to the very clear peaks on the X axis, caused by the users upwards motions at each ictus.

### 5.3 – Time Signature Recognition

To test the accuracy of time signature recognition, users first played a segment of a single time signature. This was repeated 5 times for each of 2/4, 3/4 and 4/4. Finally, the users performed 10 bars of 2/4, followed by 10 bars of 3/4 and 10 bars of 4/4. The users were of differing familiarity with conducting gestures. User 1 had not used the system before, and was not aware of conducting gestures at all before testing. User 2 had familiarity with the system, but not with real conducting. Users 3 and 4 were music students, and both quite familiar with conducting. Confusion matrices are supplied below:

<table>
<thead>
<tr>
<th></th>
<th>Actual</th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2/4</td>
<td>3/4</td>
<td>4/4</td>
</tr>
<tr>
<td>User 1</td>
<td>212</td>
<td>56</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>238</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>67</td>
<td>258</td>
</tr>
</tbody>
</table>

For user 1, the system had an accuracy of 69.8%.
For user 1, the system had an accuracy of 72.8%.

For user 1, the system had an accuracy of 82.3%.

For user 1, the system had an accuracy of 79.9%.

Unsurprisingly, the system recognises trained much more easily. Users who have experience with conducting are able to produce much more consistent gestures, and the ictuses in their gestures are much sharper, producing clearer peaks in the data, and thus making it easier for the algorithm to detect the minima and maxima. An untrained user will not only be less able to consistently reproduce a gesture, but are also prone to moving the phone in more circular motions, creating very low peaks. A sharp change in direction is required for accurate recognition.

5.4 – System Latency

Another important and relevant factor to test was the latency of using the system. As it is being used in a live format, it is important that the latency be as low as possible – if there is a significant
delay between a user making a gesture and receiving feedback for that gesture, then the usefulness of the system will be lowered. As discussed in chapter 2.4.1, a conductor in real life conducts $152 \pm 17$ ms ahead of the orchestra they are leading. If the latency of the system is close to this value, it can be considered successful.

To calculate system latency is difficult, as each of the 3 devices involved (Android phone, Java server desktop, MvM desktop) may have different timestamps at any given instant. To calculate latency of transmission between two devices, a UDP packet was first sent from device A to device B, adding a timestamp to it at the last possible instant. Device B added a timestamp to the data as soon as it was received, and then sent it back to device A, adding a third timestamp as the last command before transmission. Device A added a fourth timestamp as soon as the packet was received. By subtracting timestamp 2 from timestamp 3, the time taken for device B to handle and return the packet is known. Subtracting timestamp 1 from timestamp 4 gives the time for the whole transaction. By subtracting B's handling time from the total time, the latency of transmission can be estimated. Mean latency between the Android phone and Java desktop was 40ms. Mean latency between the Java desktop and MvM desktop was 23ms.

Latency is also introduced by the computations performed by the Java desktop – after the transmission is received from the phone, it must decipher the packet, extract the data, perform calculations on the data, and then encode a new packet to be sent to MvM. To calculate this latency, a timestamp was saved locally as soon as a packet was received. As soon as the data in this packet had been dealt with and a new packet containing the modified was sent to MvM, a second timestamp was saved. The difference between these two timestamps gives the time taken to process the data. Mean latency for processing on the Java desktop was 78ms. Some of this can be attributed to the fact that when a packet is received, it is not the next packet to be sent out, as beat and time signature recognition for any given packet cannot be calculated until two more packets have arrived.

Total latency for the system is estimated at 141ms. This is beneath the average real life latency mentioned above, and so the system can be considered suitable for emulating real life conducting. In practice for an untrained user, the latency feels somewhat unnatural, but with experience (like trained musicians and conductors have) this effect diminishes.
6 – Conclusion

6.1 – Discussion

In this project, a system has been created which can recognise the beat points and time signature of a user who is performing conducting gestures using a mobile phone in place of a baton. Beat points are recognised extremely accurately, with near 100% recognition. Time signatures are less accurately recognised, with the highest rate being 82.3% for a trained user, while a totally untrained user only achieved 69.8%. Given that the algorithm used was very simple, it seems like there is room for improvement in the algorithm. Only minima and maxima are observed, using the data inbetween the peaks could also prove to be useful.

6.2 – Extensions and Future Work

Due to time constraints, it was unfortunately not possible for me to implement any of the suggested extensions listed in the introduction. If implemented, they would certainly bring the system closer to being a training tool for budding conductors. There are other ways in which the system could potentially be improved. In

In the MvM output alone, there is a large scope for improvement. The visual feedback shown in this project is quite rudimentary, and fairly abstract, as mapping the acceleration values onto a 3-dimensional grid does not show the direction of movement like a user might anticipate. By instead using applying the acceleration vector to an object in the grid and having it move accordingly, it could be possible to show the trajectory of the user’s gesture.

Sonification could also be added, whether this be simply a noise played when a beat is detected, or a full musical track which the user could control with their gestures. The time signature detected is not used in the output; it could be used to change the display in some manner, perhaps altering the colour or theme for each time signature. Recognition of 3 time signatures, while they are the most common, is a fairly small subset, and more could be included. All performances were also legato, or fairly smooth. Often, conductors will gesture much more jerkily, to indicate a staccato piece of music. It would be interesting to see how the system fares with such an input.
Bibliography


Ha, Peter, (2010) *All-TIME 100 Gadgets* URL: [http://www.time.com/time/specials/packages/article/0,28804,2023689_2023708_2023656,00.html](http://www.time.com/time/specials/packages/article/0,28804,2023689_2023708_2023656,00.html) [16th May, 2011]


iBone (2009), URL: http://playibone.com/ [10th May, 2011]

Pocket Guitar (2008), URL: http://podmap.net/pocketguitar/ [10th May, 2011]
Appendix A – Personal Reflection

This project has been by far the largest piece of work that I have ever completed, and it has been an extremely educational and formative experience. At no point was it easy, but finishing it was an immensely satisfying moment. Throughout the project I have been learning not only about the specifics of the problem I was tackling, but also a lot of other things.

Time management is something that many people struggle with, and I am certainly no exception. Trying to schedule a project of this size is very difficult, and I doubt anyone finishes without amending their initial schedule. In my case, delays due to medical reasons created huge disruptions to my schedule, which had to be dealt with in order to continue. I urge anyone who is planning an endeavor similar to this to be as generous as possible when it comes to allowing time for themselves – there is always another paper to read, or another technique to try, or a bug that you take ages to fix, and delays are inevitable. It is only possible to minimise the disruption that they cause to your overall work, by budgeting enough time to allow for these delays.

The importance of continually writing cannot be overstated. Beginning to write is the hardest part of all in my experience, but if you can write a little every day, then what seems like an immense task at the beginning will slowly begin to get more manageable as time goes on.

Giving presentations to the research group was something that I found very daunting, but also very rewarding. The level of understanding that you must reach in order to give a good presentation means that you have a high level of engagement with the subject, and I find that explaining things to other people is an excellent way in which to also ensure one understands the subject fully. The (constructive) criticism provided by the group was also invaluable, and the discussions about related subjects and each others’ projects was helpful in offering new areas for research.

Relatively, giving constructive criticism to the members of the group (and critically evaluating the papers in the background reading) was something that I had not had a chance to do before, but was extremely useful in helping to understand how I should approach similar tasks, knowing where others had fallen short.
The most important advice I feel I can give to any other student doing a final year project is to constantly be in communication with the school as much as possible – your project supervisor, your personal tutor, the leaders of your research groups and potentially the other members of the groups – they are there to help you, and to provide assistance. If you are struggling, burying your head in the sand will not help, while asking for advice and guidance from these (vastly more experienced) staff members certainly will.
Appendix B – External Contributions

I have used multiple outside sources of software within this project. The Android Software Development Kit was provided by Google. SensorDump, the initial data collection tool, was created by “cvps” ([http://cvpcs.org/](http://cvpcs.org/)) and is available as a free download for Android phones. The DTW implementation used, FastDTW, was written by Stan Salvador and Philip Chan and is available as a free download. The MvM output framework was developed by James Leonard, in the Interdisciplinary Centre for Scientific Research in Music at Leeds University.
Appendix C – Schedules

### Original

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background Reading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial data capture</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android app</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beat Segmentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Sig Recognition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>System Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the week numbers in the schedules do not correspond to the same dates, as a large amount of time was lost due to medical reasons.

### Revised

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background Reading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial data capture</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android app</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beat Segmentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Sig Recognition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>System Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the week numbers in the schedules do not correspond to the same dates, as a large amount of time was lost due to medical reasons.

43
Appendix D – Android SDK guide

Installing the Android emulator and SDK

1 – Install the Java Platform (JDK)
   

2 – Install Eclipse (Classic is recommended)
   
   http://www.eclipse.org/downloads/

3 – Download the Android SDK installer
   

4 – Install the ADT plugin for eclipse
   
   Instructions at http://developer.android.com/sdk/eclipse-adt.html#installing

5 – Install Android platforms using the SDK and AVD manager (opens at end of Windows installation wizard automatically, also accessible via Eclipse “Window” menu option)

   2.1 Platform is used on the Samsung i5500

Example of creating an Android application and running it using the emulator:


Existing apps can be installed on the emulator using .apk files, place them in the \platform-tools\ directory, navigate to the folder via command line, and use “adb install appname.apk” while the emulator is running.