EXPERIMENTING WITH WATCHES

MARK BYRNE
B.A. (Mod.) Computer Science, Linguistics and a Language
Final Year Project  April 2015
Supervisor: Dr. Donal E. O'Mahony

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

_________________________ April 7th 2015
Mark Byrne
Acknowledgements

I would first of all like to thank my supervisor Dr. Donal O’Mahony for all taking the time to meet with me on a regular basis to provided me with help and support throughout this project. I would also like to thank my tutor Dr. Glenn Strong for helping me out during some times of difficulty. My family and friends must also be thanked for their encouragement during this year, as well as Joseph K. who provided me with some additional guidance towards the end of the year.
“You could boil down all these notes and their relationships into a mathematical description, but actually there’s always something beyond that. And the fascinating thing for me is the way these relationships act as routes to an emotional spectrum. That absolutely fascinates me, and that emotional spectrum, the way you link through to it, is always subtly changing. But the vocabulary, if you like, of the emotions is static. There’s no new feelings, but what fascinates me is that there are new routes to them, through music.”

Tom Jenkinson - Squarepusher
# Contents

Chapter 1: Introduction 7

Chapter 2: Background 8

2.1 Inspiration 8

2.2 The Pebble Smart watch 9

2.3 Midi 10

2.4 Resources 13

2.4.1 Pebble SDK and Cloud Pebble 13

2.4.2 Processing 14

2.4.3 Ableton Live 14

2.5 Similar applications 15

2.5.1 Leap motion and Geco 16

2.5.2 Kinekt as a midi controller 16

2.5.3 Buchla Lightning 17

Chapter 3: Design 18

3.1 Watch application 18

3.2 Computer application 18

3.3 Communication between applications 20

Chapter 4: Implementation 21

4.1 Watch application 21

4.2 Computer application 22

4.3 Communication between applications 25

4.3.1 Bluetooth link - Pebble and computer 25

4.3.2 Processing and Ableton 25

Chapter 5: Conclusion 27

Bibliography 28
List of Figures

2.1 - Examples of pebble watch face variants
   http://www.mypebblefaces.com/?pebble_sdk%5B%5D=2&

2.2 - Novation Launch Key 49 Midi Keyboard

2.3 - Structure of Midi Message

2.4 - Ableton Live Session View
   Screen shot

2.5 - Buchla Lightning
   http://www.soundonsound.com/sos/dec05/articles/buchla200e.htm
   “Photo Courtesy of Don Buchla”

3.1 - Ableton Phaser effect
   Screen shot

4.1 - Pebble Accelerometer Axes
   http://developer.getpebble.com/guides/pebble-apps/sensors/accelerometer/

4.2 - controlP5 Knob
   Screenshot

4.3 - Ableton midi connection
   Screenshot
Chapter 1: Introduction

This report describes the design and implementation of a gesture based Midi controller application for the Pebble smart watch. The application is intended to be used alongside a digital audio workstation (DAW) or software music sequencer to allow the user to control parameter levels such as the velocity of notes being played or a filter frequency using natural hand gestures, rather than using a mouse and a GUI, giving electronic music production a more organic and instrumental feel.

The creation of this controller involved the design of a pebble watch application from which control data is sent. This was written in C using CloudPebble, a cloud based SDK for the Pebble watch. It was also necessary to create a computer based application to receive this information and translate it into Midi data to be sent on to the relevant software. This was created using Processing which is a java based programming language and SDK. The DAW which I used for this testing this project was Ableton Live 9 suite, a very popular and commercially successful DAW created by Ableton AG in Berlin.

The report will be structured as follows:

• Chapter 2 describes the background of this project, my inspiration for creating the application, a description of the digital resources used in it’s development. It also gives some examples of similar applications.
• Chapter 3 describes the design of the application in a general sense i.e. A more in depth description of how the app will work but without specific to any particular device or DAW
• Chapter 4 details how exactly I went about implementing this design.
• Chapter 5 contains my conclusions and final comments on this project.
Chapter 2: Background

2.1 Inspiration

Wearable technology is a category of technology which has received a large increase in interest in recent years. The concept of wearable tech covers any item of clothing or wearable accessory which incorporates computers and electronics to give it an added or increased functionality. This increased research and development in the area of wearable tech has resulted in great innovation in fields such as medicine, health and fitness as well as in the commercial technology industry with items like discreet wearable monitors to read data such as blood sugar levels or heart rate becoming more and more commonplace.

With every invention of a new piece of wearable technology a new and different platform is created which programmers can develop for. I decided that for this project I would experiment in developing an application for a piece of wearable tech and to investigate what can be achieved with a wearable device that can not be done otherwise. As smart watches are the most prominent and accessible example of wearable technology in the commercial market today I decided that I would experiment in creating a unique smart watch application and carried out the project using the Pebble smart watch.

When deciding what the application should be I needed to think about what exactly makes a smart watch application unique. As putting an application on a watch which could as easily be put on the phone it is paired to would serve no real purpose I was required to ask myself what could be done with a watch on my wrist that couldn’t as easily be done with a phone or other device. What does the “wearable” quality of a device bring to it in terms of app development. I concluded that, in the case of the pebble, having an accelerometer directly attached to your wrist at all times would allow for natural motion capture beyond which could be achieved on a non wearable device. 

PebblePong, created by Kim Jeker of Switzerland ([http://kije.github.io/PebblePong-2/](http://kije.github.io/PebblePong-2/)), is a game which uses this feature to allow the user to control the paddle in a traditional Pong game using tilts of the wrist. SwimIO (Active in Time ltd) is an innovative swim tracker app which takes advantage of the Pebble’s water resistance. Although still currently in BETA, this app uses motion capture to provide the user with data such as their stroke count, lap count, total distance and lap time information. The user can then store this data on their phone in order to keep track of their progress over time.

I wanted to use the Pebble’s motion capabilities for a slightly different reason to the apps mentioned above however. As I am an avid music enthusiast and hobbyist producer I wanted to
create an application which could use gestures interact with music software to enhance the electronic production experience. A natural user interface (NUI) is an interface which allows the user to interact in an instinctive and seemingly direct way with whatever they are interfacing with. It aims to create the illusion that there is, in fact, no interface at all through the use of natural gestures for interaction. The *natural* in this case refers to the users behaviour and feeling during the experience rather than to the design of the interface itself (Widgor, Wixon, 2011). The process of producing music within a DAW can often seem quite unintuitive and unnatural. Musical parameters often have to be adjusted and fine tuned through constant clicking of the mouse in order to achieve the perfect sound meaning producers generally can spend more time in silence tapping on their QWERTY keyboard and mouse than they do actually playing instruments. I wanted to use the pebble to create a NUI to allow users to control audio parameters in real time through motions of the wrist to give the process of a production a more intuitive and instrumental feel. An app which would allow the user to play a melody on a keyboard with one hand while controlling factors like the note velocity or sustain through gestures with the other is an example of how this idea is intended to be applied.

### 2.2 The Pebble Smart watch

The Pebble smart watch, developed by *Pebble Technology* in California, was released in July 2013 after a highly successful crowdfunding campaign on the Kickstarter platform. It’s comparatively low price, e-paper display and reported seven day battery life make it very popular in the mainstream smart watch market. It also has a strong programming community associated with it with many app development guides and tutorials, highly active developer forums and a cloud-based SDK all available on the Pebble website ([http://developer.getpebble.com/](http://developer.getpebble.com/)). It is compatible with both Android and iOS mobile operating systems.

The standard Pebble has a 30 x 50 mm 144 x 168 pixel e-paper display with an LED backlight. It’s sensors include a 3D accelerometer, magnetometer, light sensor and it is also equipped with Bluetooth 4.0. It also has four buttons, *Up, Down, Back,* and *Select,* which can be assigned functions by a developer. The Pebble has 786kb of storage space which allows for a maximum of 8 applications to be stored at any time. Applications for the pebble come in two forms; *Watchfaces* and *Watchapps.* *Watchface* applications are generally just stylistic variants of the traditional analog or digital watch face but can also have additional features such as displaying weather or news notifications. They are purely for displaying information and do not take any input from the buttons. (see fig.01 for some examples of pebble watch face variants). *Watchapp* applications, on the other
hand have no restrictions, they can be interactive or non-interactive and all buttons can be assigned functions.

Since the release of the Pebble there have been two newer models released; The Pebble Steel and, more recently, the Pebble Time. The Pebble Steel is essentially just a more premium version of the original Pebble watch. It has the same hardware and software specifications but comes with a steel wristband instead of rubber and has a tougher gorilla glass screen. The Pebble Time is a more recent variant Pebble watch and features a full colour screen with a higher resolution than the others and a microphone. For this project I have been using the original Pebble watch.

2.3 Midi

Midi, which stands for Musical Instrument Digital Interface, is a set of specifications agreed upon by the electronic music industry in the early 1980s that describes what internal circuitry electronic instruments and related devices must have in order to communicate with each other and the form that this information will take. The invention of midi coincided with the creation and popularisation of synthesisers and electronic instruments in the seventies and eighties. It became necessary to have a means of communication between these instruments to allow for things such as keeping time between automated instruments such as drum machines and sequencers etc. or for the pressing of a key on one keyboard to trigger sounds from many different synthesisers at once to allow for the creation and playing of layered sounds from just one console. Midi instruments have
specially designed inputs and outputs, each of which has 16 different midi channels along which messages can be carried. (Casabona, Frederick, Hurtig, 2012). Midi channels can only carry one message at a time so in order to send messages to multiple devices at once, each recipient device must be assigned a separate channel. This means that for every output messages can be sent to 16 other devices.

A Midi controller is a device which does not itself generate any sounds but can transmit Midi messages. While these can be used for controlling hardware synthesisers and Midi instruments in live performance settings they are more often used in the production process alongside a DAW to control Virtual Studio Technology instruments (VSTis) or to be used as a simpler and more tactile way of navigating the software. In this case the Midi data can be transferred to the computer via USB. Controllers generally come in the form of a standard piano-style keyboard with some additional buttons, knobs and faders, (all of which can be assigned to whatever function the user specifies within the DAW), but many other styles some consisting of just knobs, pads for drumming on or even touch screens are also available. Fig 2.2 is an image of a typical Midi controller keyboard.

Midi messages are the medium in which information is transferred between Midi devices. They do not contain descriptions of sound waves but rather binary codes that specify particular events such as the turning of a knob or the pressing of a key. Every event which is being described is assigned

Fig.2.2 - Novation Launch Key 49 Midi Keyboard
- All keys, faders, knobs and pads are user-assignable
a message number and messages can fall into different categories depending on the type of
instruction they represent. “Note on”, which triggers the specified note, and “program change
which causes a change such as the loading of a different preset sound in the designated
synthesiser, are examples of different types of Midi messages (Rothstein, 1995).

All messages are made up of a Status Byte followed by one or more Data Bytes. The Status byte
is used to identify what type of event is being transmitted as well as the Midi channel that it is being
transmitted on. For example the status byte to send a note on message along channel 8 would
contain ‘1001 0100’ with 1001 being the sequence that indicates the note on message type.
Similarly a status byte to send a “program change” message along channel 16 would be ‘1100
1111’ as 1100 is the sequence that indicates program change. The Data byte portion of the
message tells the receiving device what values are associated with the event specified in the
status byte e.g. note and velocity information. Status bytes range in value from 128 to 255 and
Data bytes from 0 to 127 (these numbers are represented in their binary form). This means that the
first bit in a Status byte will always be ‘1’ and in a Data byte ‘0’. This allows for the receiving midi
device to instantly distinguish between the different byte types via the most significant bit. The Midi
message to trigger the middle C (C3) note with a velocity of 100 (medium-high velocity. - range
between 0 and 127) along Midi channel 16 would contain ‘1001 1111 00111100 01100100’. (Guérin,
2006).
Despite being developed for audio purposes, over the years the use of Midi technology for control has expanded into other fields. Many modern applications such video sequencers and visual displays, light shows and even robotics allow for Midi control. *Resolume* and *DMX* are examples of popular live visual mixing and stage lighting software (respectively) which allow for control via Midi. *Music for robots*, an album by electronic music producer Tom Jenkinson (aka *Squarepusher*), is an example of some very experimental use of Midi. Every track on the album was performed on live musical instruments by a Midi controlled robotic band known as *Z-machines*. Due to the standardised and relatively format of Midi messages, it is only necessary to change how messages are interpreted on the receiving end to allow the use of Midi control outside of audio.

### 2.4 Resources

Outlined below is a description of each of the digital resources that I used for this project.

#### 2.4.1 Pebble SDK and Cloud Pebble

*Pebblekit*, the pebble SDK, consists of different frameworks that are organised according to their functionality. Each framework includes an API and access to the software libraries supported in Pebble OS. These APIs are used in the C programming language to create applications. Frameworks are grouped hierarchically into Foundation, Graphics, Standard C and User Interface. ([http://developer.getpebble.com/](http://developer.getpebble.com/))

Cloud Pebble is a cloud based version of the pebble SDK. It allows developers to write and compile C code within a browser. Compiled code can be run in an on-screen Pebble emulator allowing users to see their app as it would appear on the watch itself. Apps can also be pushed directly from Cloud Pebble onto the watch itself provided that the mobile phone it is paired to is connected to wifi and the watch and phone are paired via Bluetooth. The creation of individual user profiles means that users can store all of their different pebble projects online as well as any additional resources they may need. Because of this the Cloud Pebble would likely be sufficient for most developer's needs.

Cloud pebble also allows for the writing of Javascript pebble apps but I was not required to use this for this project.
2.4.2 Processing

Processing is a free and open source java-based programming language and IDE created by Ben Fry and Casey Reas. It is aimed towards digital artists and designers as it promotes teaching programming “within a visual context” and is described on it’s website as a “software sketchbook and production tool”. Applications in Processing differ from regular Java classes as they all follow a stripped back format. All processing “sketches” begin with a void “setup” method to declare window size, background colour etc. and to create static variables. This is then followed by a void “draw” method in which the main body of code is written and repeatedly looped. The reason for this structure seems to simplify the coding to allow more focus to be put into creative use of the many tools and libraries that are available.

I chose to use Processing for this project because of the large variety of libraries available and the strong user community and forum. The libraries which I used for this project were:

- The MidiBus - A simple real-time focused Midi I/O library which allows for interaction with pre-installed Midi Services. I used this library to handle all Midi data transmission. (http://www.smallbutdigital.com/themidibus.php)
- blueTooth Desktop - Contains tools to allow Processing sketches to send and receive data via bluetooth wireless networks. I used this library to establish a connection between the Pebble watch and the computer application. (http://www.extrapixel.ch/processing/bluetoothDesktop/)
- controllerP5 - A GUI and controller library for Processing. Allows for creation of buttons, knobs and faders etc. I used this library to create visual aids to allow me to more easily see if the Midi messages were sending the appropriate information. (http://www.sojamo.de/libraries/controlP5/)

2.4.3 Ableton Live

*Ableton Live* is one of the leading DAWs on the market for both professional and hobbyist music producers. It is popular due to it’s relatively simple interface and it's ability to be used as both a live performance tool and as a studio workstation. It's interface is based around a grid of Midi and audio clips of variable length. Each column represents a different instrument or audio track and each row represents a “scene”. Different “scenes” can represent different sections of a song (verse, chorus etc.). Clips can be triggered individually or entire scenes can be triggered at once. This ability to trigger individual elements of a song is what makes live so popular for use in live performances. In a live
performance Ableton would almost always be accompanied by a Midi controller to allow for more tactile clip triggering and parameter changing.

I chose to use Ableton for this project as it is a program I use regularly so I am very familiar with its interface and have experience in using various Midi controllers with it.

2.5 Similar applications

There are currently no existing Smart watch applications for the Pebble or any other smart watch. There are, however, numerous examples gesture controlled midi devices which fall under the heading of natural user interfaces and wearable technology. I shall now discuss some notable examples these.

Fig 2.4 - Ableton Live Session View
Each Column represents an individual instrument or audio sample and each row represents a ‘scene’ or segment of a song.

Coloured boxes are clips which contain melodies in the form of sequences of Midi data
2.5.1 Leap motion and Geco

The *leap motion* 3D controller is a portable infrared sensor device produced by *leap motion inc* in California. When connected to a computer via USB it can detect and process movements of the fingers and hands without any physical contact. The device has its own dedicated app store in which a wide range of different applications can be found. These vary from simple navigation apps allowing the leap motion to be used as an alternative to a mouse to more intricate games such as “robot chess” in which the individual chess pieces are moved purely with gesture control. There are also applications which are designed to be used alongside the *Occulus Rift* to allow users to manipulate a 3D environment to create a fully immersive 3D experience.

*Geco* is a Midi controller application created by *Uwyn Software* for the Leap Motion. It has the ability to track the movement of the left and right hands simultaneously and independently as it’s input and the precision of the Leap Motion also allows it to detect spreading of the fingers on each hand. It works by specifying nine hand gestures which are based around the movement of and rotation of the hand, for example “left and right rotation”, “back and forth movement” and “inclination of palm”, as it’s varying control inputs. These each of these gestures can be assigned a specific Midi function and as the gestures work separately for fingers spread and fingers together and for both hands separately meaning that a total of 36 separate Midi functions can be assigned. For example a rotation of the left hand with fingers closed could cause a change in filter frequency while an inclination of the palm of the right hand with fingers spread could modulate the pitch. This wide variety of assignable gesture combinations makes Geco one of the most intricate and articulate gesture controlled Midi applications available. Since it's release Geco has received very positive feedback from the Leap Motion community with the few negative reviews concerning troubleshooting difficulties rather than the application itself.

2.5.2 Kinekt as a midi controller

The Kinekt is a natural user interface created by Microsoft for the Xbox 360 game platform. It consists of a camera intended to be placed on top of a television which is used to capture motion from a player allowing them to control and interact with games without the need for a physical controller. In a project entitled “Creating Musical Expression using Kinect”, Min-Joon Yoo, Jin-Wook Beak and In-Kwon Lee of Yonsei university in South Korea aimed to utilise the motion capturing abilities of the Kinekt to create a Midi Controller based on gestures from the entire body. In order to achieve this they split the reading of a
the human body into 24 different sections and then further categorised these into five more general sections, center, left arm, right arm, left leg and right leg. Each of these five sections was assigned a particular sound, and then the smaller sections within that section control the parameters of that sound. For example, one sound corresponding to the left leg had four parameters, and these parameters were controlled by the movements of the left hip, left knee, left ankle and left foot. The program also allowed for the velocity of movements of joints to be used as a parameter. (Yoo, Wook, Lee, 2011).

2.5.3 Buchla Lightning

The Buchla Lightning is a very early and influential example of the use of gestures to control a midi device. Released in 1991 the Lightning consists of two hand held wireless wands which contain infrared transmitters, a remote infrared tracking head that senses the positions of gestures made with the wands and a box containing the electronic circuits. The receiver which tracks the wands can be programmed to carry out functions like note selection, pitch bends, stereo panning and volume (Vail, 2014).

Fig 2.5 - Buchla Lightning
Chapter 3: Design

The design of this project can be divided into three main subsections.

- The watch application - Captures the motion data and transmits it to the computer in real time.
- The computer application - Receives motion data, translates it into the relevant midi messages and passes these messages onto a DAW or other software to be controlled.
- The way in which data is transmitted between the applications.

In this chapter I will be discussing these sections from a more general point of view as a design which could be applied to any smart watch device.

3.1 Watch application

The watch application design is relatively simple as it only really has one function; To capture and transmit motion data. All processing and interpretation of this data will be done on the by the computer application. When paired and activated the watch application should begin capturing motion data along each axis and transmitting it to the computer application.

Buttons can also be used to send Midi information. Switching between different instruments, or effects to be controlled by motion or any operation that has a binary “on/off” value associated with it (e.g. Arpeggiator ‘On’) is better suited to be controlled using a button rather than a gesture in order to allow the full extent of each axis to be reserved solely for functions which can utilise the gradually increasing and decreasing range of values which a gesture can achieve. Again, this kind of gesture processing will be discussed further in the computer application design. It is assumed that within the SDK of any smart watch there will be some operator to deal button presses, such as a ‘buttonPressed’ boolean. This can be used to trigger the transmission of a some data which will allow the pressing of a particular button to be registered within the computer application. Separate operators will be required for each button (e.g. ‘Up’, ‘Down’, ‘Select’ and ‘Back’ on the Pebble).

3.2 Computer application

The function of the computer application is to receive the incoming accelerometer data, recognise individual hand gestures based on the data received and then translate that data into the relevant Midi messages. The application then automatically forwards these messages on to the DAW or
other software to be controlled. The input gestures in this design have been kept relatively simple; movements of the hand horizontally left and right or vertically up and down, in an orchestra conductor like fashion. It is restricted in this way as the app is intended to provide as natural a user experience as possible and to be used while playing melodies with the opposite hand so the incorporation of complex hand gestures could complicate this and take away from the seemingly organic musical experience which was intended. Also, seeing as the application is not intended to actually trigger notes but rather imitate the function of knobs or faders, the restriction to horizontal and vertical movement seems suitable.

With gestures performed only along the X and Y axis we can imagine the ‘gesture area’ as a two dimensional square space. This was partially inspired by the graphical interface used to control effects within Ableton live. Fig 3.1 shows a screenshot taken of a phaser effect in Live. The black square in the centre of this image represents the area within which the effect can be controlled and the position of the green circle determines the current values of the parameters assigned to each axis. In this case the value for frequency is controlled along the X axis and the value for feedback along the Y axis, and the precise value for each is displayed underneath. Therefore, by assigning the horizontal and vertical movement gestures of the Pebble to frequency and feedback respectively the app could be used to fully control control the parameters of this phaser effect by moving the green circle around in a natural and intuitive manner. Alternatively the user could only assign one axis to only allow for control of one parameter at a time (e.g. only assign the horizontal gesture to frequency and nothing to vertical gesture to keep feedback at a fixed level while modulating frequency or vice versa.).

Fig 3.1 - Ableton Phaser effect
Frequency modulated along X axis and Feedback along Y.
As mentioned in section 2.3 a Data byte can range in value from 0 to 127. By setting a maximum possible value for the X and Y coordinates, for example 4000 for both (based on 4000 milli-Gs which is the maximum acceleration the pebble is calibrated to measure, (developer.getpebble.com)), and allowing them correspond with the data byte 127 and setting minimum values for both, for example -4000, which correspond to data byte 0, a ratio is created which allows for the translation of coordinate values into Midi Data bytes. As the status byte is determined based on what event the message is associated with, or rather what the user has assigned the controller to perform, the translation from coordinate data to midi message is complete. The methods required to assign an event type and actually transfer the midi messages to the designated software can be found in any Midi library or API. (for example Java Sound API, Oracle’s Midi library, or, in my case, the midiBus).

Translating data from buttons is far more straightforward due to their binary nature. When data indicating a specific button is received, the relevant assigned midiMessage will be transmitted. Examples of the type of messages buttons can be used to transmit include ‘device on’ and ‘device off’ for the switching on and off of effects, or messages which would allow for stepping through presets or midi channels by incrementing the current value with each press.

3.3 Communication between applications

Communication between the watch app and the computer app is achieved via bluetooth. It is necessary to provide code to assign a bluetooth server role to the computer app to allow it to listen for devices and to assign the role of the client to the watch app to allow it to initiate the connection. The methods required for assigning these roles and handling the transfer of data can be found in JSR-82, the Java Bluetooth API.

In order to transfer midi messages from the computer application to a DAW a virtual loopback driver must be used. This essentially creates a virtual Midi device which can be detected by the DAW in exactly the same manner as if a physical controller had been connected. This virtual device is used as the medium through which midi messages as sent from the computer application to the DAW.
Chapter 4: Implementation

In this chapter I will be discussing the implementation of my design. Similar to chapter three, I shall discuss the watch application, computer application and connections in separate subsections.

4.1 Watch application

In order to create the watch application I was required to familiarise myself with Pebble’s accelerometer API.

When looking at Pebble, axes, as shown in 4.1, are standardised to the following co-ordinates:

- X+ Left to right axis
- Y+ Bottom to top axis
- Z+ Coming up out of the watch

In the API, data events along the three axes are numerated as raw x, y and z 16 bit integers with each value measured in milli-Gs and, as mentioned 3.2, the Pebble is calibrated to measure a maximum acceleration of ±4G. These coordinate integers are presented in the form of an AccelData structure which contains the three integer values as well as a timestamp of when the first measurements were made and a boolean object called did_vibrate which tells whether or not the watch was vibrating while accelerometer data was being recorded (This is useful because the
vibrations of the watch could have an effect on the values measured.). It is possible to implement the accelerometer in three different ways; using “tap events” when it is only necessary to now if the watch has been tapped or shaken, processing data in batches for recognition of detailed gesture control and analysis of user behaviour and then real time use (developer.getpebble.com).

In my case I require real time use of the accelerometer in order to allow for smooth and continuous control. As part of “Sensors” section of the “Writing Pebble apps” Guide available on the Pebble website there is example code for implementing an application to capture motion data. I was able to use this code with just a few minor adjustments to suit my needs.

The code contains a boolean object called TAP_NOT_DATA which when set to true activates the Tap handler services. I set this value to false as I will only be requiring actual data samples. I was also able to remove any tap handler specific code from the example. Also, given that I only intended to process gestures within a 2D square X/Y axis I was able to eliminate the reading of what would normally be considered a Z axis. As the pebble describes horizontal left right motion as being on the Y axis, and vertical up and down motion as being on the Z axis I was able to eliminate readings from the X axis which is reserved for forwards and backwards motion. See Fig. 4.1.

4.2 Computer application

The computer application was created using Processing along with the midiBus and controlp5 libraries discussed in 2.4.

The application begins by calling the midiBus.list() method which lists all midi devices (both physical and virtual) which are currently available in the console window along with a number corresponding to each of their Midi channels and their name respective names. Provided that there is a virtual loopback driver running, a virtual device should show up under a user specified name. As I was carrying out this project using a Macbook, activating the loopback driver was very simple as all Apple computers have one pre-installed as standard. I was simply required to open the midi studio which is located under audio settings and create a new virtual device with a specified name which I called “FYP BUS”.

Within the “setup” component of the Processing application the constructor for the midiBus is then called which takes the channel number and device name as it’s arguments. This constructor basically creates an object which can be used to send Midi messages along the specified channel number under the guise of my virtual Midi device “FYP BUS” which DAWs will recognise as a physical controller.
In the “draw” component of the application two midiBus methods are called; sendNoteOn and sendNoteOff. As can be gathered from the name, the sendNoteOn method sends a ‘note on’ Midi message to the DAW. This takes an output channel, an integer to represent a velocity value and an integer value to represent a pitch value as it’s arguments. This is followed by a sendNoteOff which must take the exact same arguments as the the sendNoteOn. The sendNoteOFF method is extremely important as without it the note will continue to play indefinitely. Between the sendNoteOn and sendNoteOff method there is also a delay method called, which is a native processing method. This takes an integer as its argument to specify amount of milliseconds to delay for and is used in this case to determine how long the note will be played for before it is switched off.

I should point out that these send methods could be replaced by a more general “sendMessage” which is provided by the midiBus library. This method can send any type of Midi message and therefore be used to convey any Midi event. It can take either an array of bytes or one to four integers to represent the status byte and up to three data bytes as it’s arguments. However, as Midi messages are take values in binary an additional operation would be required to convert values for pitch and velocity etc. into binary before transmission.

As explained in 2.4.2 the draw component of a processing application works by repeatedly and unconditionally looping through the code within it. In this way the values for velocity and pitch within the sendNoteOn method can be modified with each loop. Ideally each loop would begin with the updating of integer variables for pitch and velocity with values transmitted over bluetooth directly from the Pebble. These variables would then be used in the sendNote methods, unfortunately due to problems establishing a bluetooth connection which I shall discuss in 4.3 this was not possible. In order to allow for a test of my application I used the controlP5 library to create a GUI knob which I could manipulate with the mouse to obtain values.
The controlP5 library makes generation of GUls very simple. Firstly a controlP5 object called cp5 is generated which is a kind of general controller object, then a knob object called myKnobA is created. Using the “getKnob” method these on the controlP5 object with the knob object as it’s argument the knob becomes added to the controller. I used the setRange() method to set the range of values covered by the knob to be between 0 and 127 and the setDragDirection() method to specify that I wanted the knob to operate using a vertical mouse drag.

In order to actually allow the knob to control what Midi messages are sent I simply needed to replace either the velocity or pitch values within the sendNoteOn method, depending on which one I wanted to modify, with myKnobA.getValue() and the value for that parameter would be updated to the knob value with each iteration of the loop.

While this is not the the source of input which I had intended to use, the knob generated using the controlP5 library worked perfectly for Midi control and was a perfect alternative to use to simulate how the computer side of the application would have operated had I managed to successfully transmit the accelerometer data over Bluetooth. Had I been successful the computer application would have been essentially the except that the Pitch and Velocity variables would have been updated from a different source.
4.3 Communication between applications.

4.3.1 Bluetooth link - Pebble and computer

Unfortunately I had problems trying to transmit data from the Pebble to my directly to my laptop and ultimately did not succeed in establishing a connection between the two. As the Pebble is intended to be used alongside an Android or iOS device there is currently no API which contains methods to allow for a Bluetooth link to be established directly between the Pebble and a computer. It seems as if everything must be transferred via the phone and even at that there are few resources which allow for transmission of data from the Pebble to the phone, it seems that Bluetooth is primarily intended for transmitting data from the phone to the Pebble.

In my attempts at establishing a connection I used a processing library called bluetoothDesktop which I briefly mentioned in 2.4.2. By using the “simpleDesktopClient” program which comes as part of this library I was able to scan for all available Bluetooth devices and display them in the console window. With this I was able to detect the presence of the Pebble’s bluetooth connection from within processing, but this proved to be quite unreliable and the Pebble would only be detected about half of the time I tried and because of this I never managed to achieve and actual connection.

In my attempts to work around this I decided to try and establish a link from the pebble to the phone and then from the phone to processing. This required the installing a mobile version of processing, which I had to do on a borrowed Android phone as it is not available for iOS, and running a blueTooth desktop sketch called “simpleMobileServer” to allow the phone to be detectable as a bluetooth device within Processing. Unfortunately this, again, proved to be quite troublesome for me and a connection was not established. Even if this had worked out, the fact that the accelerometer data would have to travel from the watch to the phone then to the laptop and then to the DAW would likely have caused latency issues.

If I had more time for this project I likely would have found a solution to this problem but unfortunately that was not the case this time.

4.3.2 Processing and Ableton

Establishing a link between Processing and Ableton after having written the computer application was relatively straightforward as it essentially followed the same process as
linking a hardware controller. Once the “FYP BUS” virtual controller was created within the Apple Midi studio it became detectable as an available Midi device to any DAW. This is the case regardless of whether the Processing application is running or not.

In order to establish a connection all that needed to be done was to select the “FYP BUS” as a Midi source for the desired channel in Ableton from a drop down list of all available sources. As can be seen in 4.3 the virtual controller has been selected under “MIDI FROM” and the green lighting in the show that Midi data is being received in the channel. Once a connection has been established Midi data will be sent to the allocated channel as soon as the Processing application is launched. When an instrument is assigned to this channel the set notes are played.

Fig. 4.3 - Ableton midi connection
FYP BUS selected from dropdown menu of available midi devices.
Green lights indicate that midi data is being received.
Chapter 5: Conclusion

Despite the fact that I did not achieve my overall goal, I was still successful in creating a fully functioning midi control application in Processing and an app to caption motion data for the Pebble. I found this project to be a great learning experience for me and a good investigation into the unique qualities that a piece of wearable technology has.

While I am confident in my design I feel that the Pebble may not have been the appropriate device with which to implement it which is the reason why I was not successful. Despite having a strong user community forum, a large amount of online tutorials and great resources such as the cloud pebble and pebble libraries, I feel that the fact that the pebble almost always has to operate via a mobile phone definitely limits what it can be used for, especially when it comes to applications of the kind I was trying to implement.

I found processing to be an excellent resource and would definitely use it for similar projects in the future. It's clean and stripped back design is very easy to learn to use and it appears to have libraries and tools prepared to cover almost all development needs.

As I have a great personal interest in electronic music production and Midi technologies and, had I succeeded in implementing the design, it would have been the only one of it's kind available for a smart watch I definitely plan working further and this project in the future and seeing it through to completion.
Bibliography


• BluetoothDesktop Library - http://www.extrapixel.ch/processing/bluetoothDesktop/

• controlP5 Library - http://www.sojamo.de/libraries/controlP5/