University of Southern Queensland

Faculty of Health, Engineering & Sciences

# Wearable Technology and Gesture Recognition for Live Performance Augmentation

A dissertation submitted by

Sathya Smith

in fulfilment of the requirements of

ENG4111/ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Honours) (Electrical & Electronic)

Submitted: 13 October, 2016

# Abstract

The use of physical gestures within interactions between humans and computer systems is a rapidly progressing research field that find itself increasingly present in smartphone and computer applications. This dissertation intends to outline the various engineering design processes involved in the creation of a simplistic and novel gesture recognition system geared towards use in live entertainment performances. The system aims to help in increasing fluidity of human-machine interaction in the entertainment industry by providing an alternative input method for controlling other performance related systems such as mixers, monitors, digital audio workstations and stage lighting.

Electronic methods of wearable body movement tracking, gesture recognition and wireless interfacing are explored in order to determine a suitable design for the system to achieve a practical result. The resulting system consists of a wearable hardware group as well as a terminal hardware group, with associated software for each. The wearable design contains an MPU6050 motion processing unit, Arduino Uno development board and a HopeRF HM-TR wireless data link transceiver. The terminal group is responsible for receiving MIDI commands and consists of an Arduino compatible 'LeoStick' board coupled with a second transceiver. The usage of modern additive manufacturing methods was also investigated for hardware enclosure creation to allow potential for rapid prototyping.

The GR is able to accurately provide movement data to a processor, which utilises a running average based gesture recognition algorithm in an attempt to extract movement features and respond to the presence of a pre-determined gesture by generating a MIDI command that is then sent to a computer terminal for use with external applications.

Although the system did not fully live up to design objectives, it plays the role of an important stepping stone in the creation of practical, entertainment-oriented gesture recognition devices that are more accessible to the general public.

# Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Sathya Smith

0061033248

_____________

Date

_____________

# Acknowledgments

The completion of this research project would not have been possible without the support of many that are close to me. Firstly, I would like to thank my partner, Jorja Wicks, who has also completed a research project towards her Electrical and Electronic Engineering degree. Jorja is the most incredible study partner I could have ever wished for and I wouldn't want to have anyone else by my side.

I would also like to devote an enormous thank you to my wonderful parents, Katharina and Graham, for putting so much faith in my abilities to fulfil my goals and passions. Their kindness and encouragement has been crucial in motivating me to achieve the best result I possibly can. I cannot thank you enough for all you have done for me throughout my life.

Next I would like to acknowledge my close friends in no particular order – Steve, Joe, Riley, James, Matthew, Shannon and Daniel. You're all excellent lads and I might've keeled over somewhere along the way without you helping me kick back and re-focus.

Last, but not least, I would like to thank Dr. Andrew Maxwell for supervising my project and providing me with guidance along the journey towards the end result of this dissertation.

You have all been a great help to me – thank you!

Sathya Smith

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| ADC | Analog-to-Digital Converter |
| CAD | Computer Aided Drawing |
| DMP | Digital Motion Processor$^{\text{TM}}$ |
| DSP | Digital Signal Processing |
| EMG | Electromyogram |
| EOG | Electroculogram |
| FIFO | First In First Out |
| FSK | Frequency Shift Keying |
| GR | Gesture Recognition |
| GRS | Gesture Recognition System |
| HMI | Human-machine interaction |
| IDE | Integrated Development Environment |
| IMU | Inertial Measurement Unit |
| IR | Infra-red |
| MCU | Microcontroller Unit |
| MEMS | Micro-electromechanical System |
| MIDI | Musical Instrument Digital Interface |
| RAL | Remote Access Laboratory |
| RAM | Random Access Memory |
| SPI | Serial Peripheral Interface |
| TTL | Transistor-Transistor Logic |
| TWI | Two Wire Interface |
| UART | Universal Asynchronous Receiver-Transmitter |
| USB | Universal Serial Bus |

Chapter 1

# Introduction

This chapter is intended to make the reader familiar with the general concepts of gestures and their use in the computing field. The preamble contains information pertaining to the importance of gestures, as well as some examples of the most commonly experienced forms of gesture recognition within modern society. Following this, motivations for the completion of this research project are discusses, prior to the final subsections regarding project aims and objectives.

## 1.1. Preamble

Humans have long spent time analysing the ties between their thoughts and physical movements. Generally speaking, gestures are mostly widely known as physical movements; often enacted subconsciously and can be used to communicate a wide variety of information. Research into both humans and animals has shown that movement and gesture may possess a higher influence on thought than was previously expected. The unique symbolic and referential nature of human gestures is particularly as gestures such as waving, pointing at an object, and the 'OK' symbol are fast paced, easy to understand methods of communicating information and emotional state. Modern Human-Machine Interaction (HMI) systems utilise this advantageous aspect of intelligent communication to further naturalise the experience of working with computers.

HMI allows for the minimization of barriers between the human cognitive model of a gesture and the computer's understanding of the task (Kumar 2010). Advancements in computing have seen a wealth of potential applications for human movement tracking, where the aim is to use a HMI interface and a control system to allow movement trend analysis by the user, with constant feedback from the machine (Suhas & Dileep 2015)

From this user interaction arises the concept of a Gesture Recognition System (GRS); an electrical system designed to bridge the gap between a human user performing a movement, and a computer system performing a desirable or undesirable action in the response to the user.

One of the most notable examples of gesture utilisation in modern society lies in smartphones. Statista Inc. (2016) estimates a record-breaking 2.08 billion smartphone users worldwide as of 2016, meaning that every day new users are learning to Swipe, flick, pinch, tilt and shake their smartphones to control various pieces of software. This method of interaction may be highly desirable to those who find traditional computer input methods to be tedious or unintuitive. While many industries stand to benefit from advances in the HMI and GR fields, the theatre and entertainment industries are interesting stakeholders for consideration and will be the primary focus over the course of this project.

In an attempt to widen the availability and usefulness of work previously completed in the gesture recognition field, this research project involves the development of a simplistic wearable GR system for the control of various theatrical technology elements using off-the-shelf hardware and software components. Such a system aims to acquire the gestural movement data of a performer in real-time through the use dynamic sensors. If a gestural movement is detected, a Musical-Instrument-Digital-Interface (MIDI) command can then be created to act as a controlling mechanism for systems that utilise the MIDI protocol.

Such a system possesses a range of uses in a theatre/stage setting, where the MIDI language is commonly used. MIDI is most commonly used in digital audio workstations such as Ableton Live and Reaper for live and/or recorded music production; as well as for control of DMX lighting using a system such as 'Lightjams'.

## 1.2. A Brief Historical Summary

Wearable technology has been an important asset to performers since the 1840s, one of the first times that arc lamps were used to create special effect lighting for performance enhancement (Sjuve 2008). These analog systems usually involved the use of incandescent light bulbs, such as the 'Electrical Diadem' (See *Figure 1*) developed for a dancer who performed as a part of the opera ballet 'Farandole' in 1884. The diadem operated very simply and consisted of incandescent bulbs, a two part accumulator and a switch to allow the dancer to control the light (Sjuve 2008).

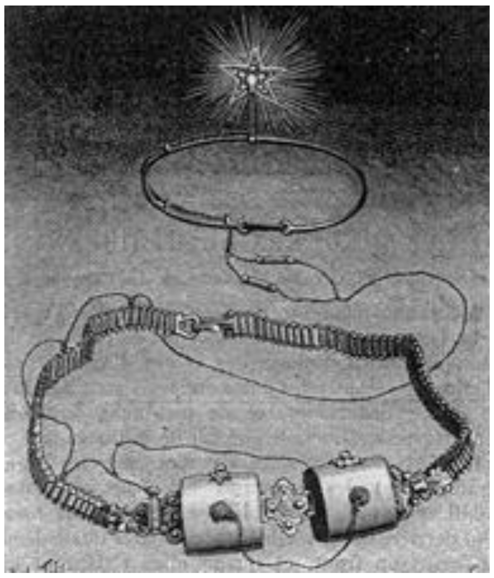


Figure 1 - Electrical Diadem (Sourced from Sjuve 2008)

Systems such as the electrical diadem required manual control via the performer or a stagehand, outlining the main difference between older wearable systems and those used in the field today, which are intended to be more automated and exist as a more fluid method of control over using, for example buttons or touchpads

Although wearable technology was present such a long time ago, HMI became much more prominent since the beginning of the digital era. In 1963, Ivan Sutherland demonstrated the first instance of directly manipulating visible on-screen objects using a light-pen, which included grabbing objects, moving them, changing size, and using constraints (Myers 1998). Following this, many more instances of HMI research began to emerge as the computational needs for more demanding purposes were met by new hardware developments. At this point in time, HMI is an element of everyday life and is under constant development.

## 1.3. Project Aim and Objectives

### 1.3.1. Aim

This project aims to investigate the design of a novel GR system using off-the-shelf components, which would allow performers to control external performance related systems in a reliable, functional and easily setup manner for a reasonably low cost.

### 1.3.2. Objectives

The completion of this project requires the definition of the following key objectives:

- Design and develop a suitable data acquisition system.
- Implement a suitable processing algorithm to perform gesture classification and recognition by extracting data features.
- Determine a viable method for creating a suitable MIDI command in response to the presence of a gesture.
- Determine a suitable method of wirelessly transmitting the MIDI command to a personal computer.
- Evaluate the performance of the system with respect to various requirements.

## 1.4. Motivation and Problem Statement

Various motivating factors have formed the foundations of the work completed in this project, with the intent to solve a particular engineering problem. For a theatre performer, the search for new and exciting methods of extending creative capability is never-ending. This curious and explorative nature is key to preventing artistic expression from becoming stale. The more tools at a performer's disposal, the greater their potential to express their creative talents and messages. Lindsay (2013) stated that:

> "*The audience needs to see, hear and feel a performance as fully as possible so that it is a rich, emotional, and unforgettable event. Modern technologies have given us the tools to enrich the whole of this experience.*"

In addition to the importance of augmenting artistic capabilities, the financial expense of current gesture recognition technology is a limiting factor placed on many who may be able to utilise the system for profit. Wearable gesture recognition devices are still emerging technologically, resulting in many prototypical systems and few commercially refined instances. With regards to this project, the intention is to prove in concept that a novel gesture recognition system can be created at a low cost using easily obtained hardware and software elements.

Certain developers are well on their way to developing highly functional gesture recognition devices, a good example of which being the "Gest" glove, which can be used for generalised computer control (see *Figure 2*). Other notable examples include the Myo armband from Thalmic Labs and the Leap Motion controller. Commercial GRSs such as these are intended as generalised gesture controllers to be interfaced with smartphones and computers; however, there exists very little in the way of live entertainment focused systems that have been well established.

Figure 2 - 'gest' gesture recognition system concept (Sourced from New Atlas 2016)

## 1.5. Dissertation Overview

This subsection outlines the composition of this dissertation:

Chapter 2 presents an analysis of relevant literature and background information.

Chapter 3 explains the chosen methodology for undertaking the research project

Chapter 4 presents the design and build process for the system in regards to data acquisition, gestural processing and data conversion and transmission.

Chapter 5 outlines the testing process for the system, and presents and discusses the results of system testing.

Chapter 6 concludes the dissertation and includes discussion of outcomes and recommendations for potential future work pertaining to the project.

Chapter 2

# Literature Review

A comprehensive technical project requires extensive consideration of background information pertaining to all relevant areas in order to ensure adequate knowledge of subject matter. Publicly available GR and HMI research covers a broad range of specific information areas, not all of which are relevant to this project; therefore, careful selection of literature must be carried out.

## 2.1. Motion Measurement

The creation of a GR system necessitates a method of reading the physical movements of an individual. In doing this, the sensors used for this purpose are effectively the computer's 'eyes and ears'. Searching the literature made it evident that a number of different types of sensors are useful for this application including electric, optic, magnetic and mechanical. Berman & Stern (2012) stated that despite the great number of GR system reviews present in the literature, comprehensive analysis of sensor types for GR systems was lacking. Their research involved investigation of different types of sensor stimulus, usage context and sensor platform. The study is useful for this project; however, there is a main focus on optic sensors, which are not suitable for this project in that the focus for this project is on using wearable technology to achieve movement tracking.

## 2.1.1. Electric Sensing

Electric sensing is the most commonly used type of sensing throughout modern day to day life (Berman & Stern, 2012), which accounts for touch screens, keyboards, computer mice and systems that function based on reading bodily electrical signals such as electromyogram (EMG) and electroculogram (EOG). Touch screen interfaces such as those found in tablets and smartphones (See *Figure 3*), while being extremely responsive, generally do not fit with the system model desired in this research project. It was evident that for an application such as live stage performance, the requirement of wearing and constantly touching a display would become cumbersome and due to the fragility of such displays, vigorous movement or mechanical shock may render the system useless in the event of fatal damage.

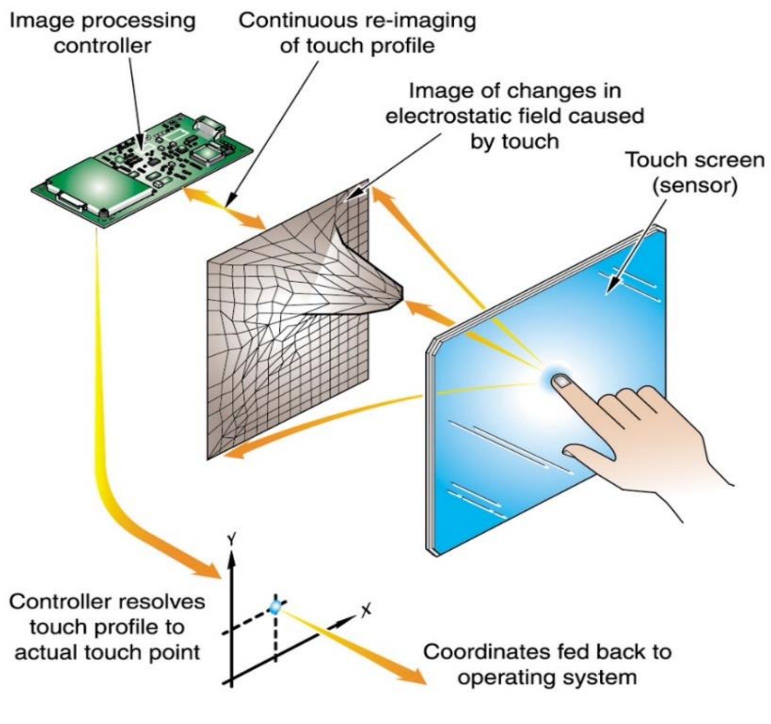


Image processing controller
Continuous re-imaging of touch profile
Image of changes in electrostatic field caused by touch
Touch screen (sensor)
Y
X
Controller resolves touch profile to actual touch point
Coordinates fed back to operating system

Figure 3 - Capacitive Touch Screen (Sourced from Electrotest Pty Ltd.)

## 2.1.2. Mechanical Sensing

Mechanical sensing appears to be the most commonly utilised approach for movement detection in a performance environment. The most notable sensors used in this way are microelectromechanical system (MEMS) based accelerometers and gyros. Berman & Stern (2012) included a description of their functionality, where single and multiple axis accelerometers are used to determine acceleration along predetermined axes in space by way of a tiny shifting mass system. A change in acceleration causes a minute mechanical shift inside the accelerometer itself, pictured below in *Figure 4*, which translates to a change in the output signal of the relevant directional axis, so that varying levels of acceleration in multiple directions can be tracked.

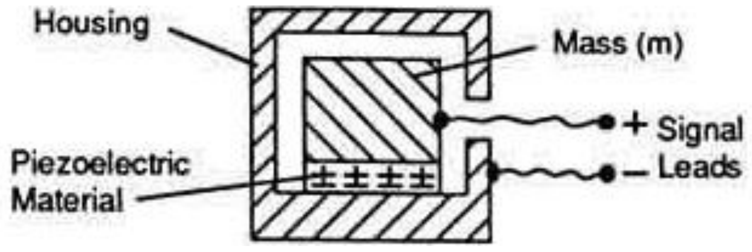


Figure 4 - Piezoelectric Accelerometer (Sourced from industrial-electronics.com)

Gyros are similar in operation to accelerometers; however, they are used to detect angular velocity, usually with an output of degrees per second. This functionality makes them very useful in conjunction with accelerometers for a system that requires reliable motion tracking. Gyros operate by creating a low-current electrical signal in response to the vibration of a small mass within the MEMS unit, which is placed under a Coriolis force orthogonal to the vibrating mass (Wikipedia 2016). These two components have been widely used in previous research on the topic area and are able to operate to extremely high degrees of accuracy. As such, these two components seem to be the most suitable for this project.

Suhas & Dileep (2015) demonstrated the use of the embedded Digital Motion Processor$^{TM}$ (DMP), located within an MPU-6050 unit that has an overall purpose of reading sensor data, which can be read from registers or buffered in a FIFO. The DMP is extremely useful for such an application as it eliminates much of the necessity for sensor data pre-processing by automatically performing those tasks for the developer. The MPU-6050 appeared to be the most suitable motion tracking unit available for a simple application such as this project.

## 2.2. Processing and Control

### 2.2.1. Existing Usage

The second element requiring thorough understanding before selection is that of a hardware controller for the wearable system. The microcontroller is responsible for acquiring sensor data, implementing GR algorithms, generating a MIDI command, and finally transmitting the MIDI command via wireless serial link. This was predicted to require extensive computations; therefore, a microcontroller capable of performing these tasks efficiently and accurately was necessary. Predko (1998) detailed the aspects of different types of microcontrollers, which usually fit into three main areas: Embedded 8-bit microcontrollers, 16-32 bit microcontrollers, and digital signal processors. Further investigation was required to determine the best route to follow for this decision.

An extremely relevant study performed by Xu et al. (2012) involved the creation of a portable gesture recognition system consisting of a tri-axial accelerometer, C8051F206 microcontroller and a ZigBee 2.4 GHz Wi-Fi module. Unfortunately, it was not particularly evident within the study whether feature extraction computations were enacted by the chip itself, or by the PC once pre-processed gesture data had been received from the ZigBee module.

It was somewhat difficult to find a great deal of previous work pertaining to GR systems that possess the ability to detect motion and carry out signal processing algorithms on the wearable module itself. Benbasat & Paradiso (2001) outlined this importance, stating that 'The authors argue that the most interesting devices are those which incorporate enough processing power to perform the software functions of the framework (recognition and matching) on-board'. Throughout their research an Analog Devices ADuC812 microcontroller with a 12-bit ADC and an 8051 microprocessor core was utilised, which allowed them to successfully implement their atomized GR framework.

### 2.2.2. Arduino

One of the most appealing control units for development of the system is the Arduino family of development boards. Arduino is an open-source development platform that contains hardware and software elements that are specifically tailored towards simplicity of use, allowing for rapid project prototyping. Arduino boards have great development potential and a wide range of features for conjunctive use with sensors and data transceivers – the other components of interest for this project.

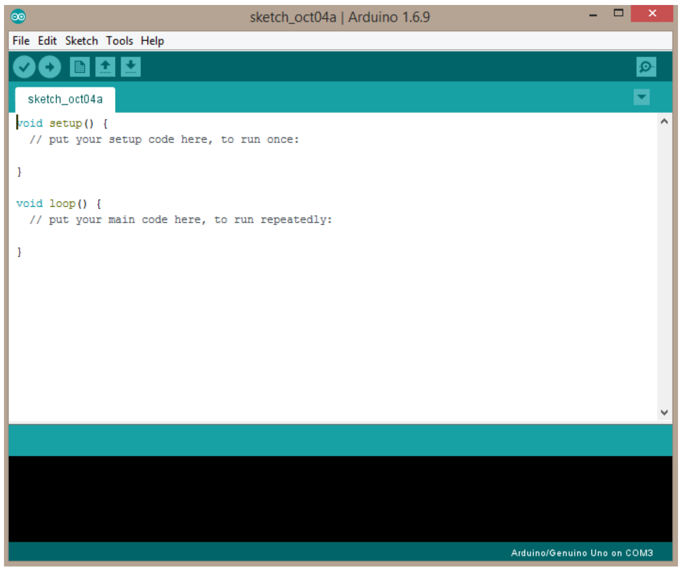
Potential choices of Arduino board include the Lilypad, Uno, Pro, Mega and Zero. Each of these boards differ in number of features, power requirements and processing power. *Table 1Table 1 - Arduino Comparison Table* on the following page shows the differences between each of the board models.

Table 1 - Arduino Comparison Table

| *Arduino model* | *Lilypad* | *Uno* | *Pro* | *Mega* | *Zero* |
|---|---|---|---|---|---|
| Microcontroller | ATmega 32u4 | ATmega32 8P | ATmega328 | ATmega256 0 | ATSAMD21 G18 |
| Operating Voltage | 3.3V | 5V | 3.3/5V | 5V | 3.3V |
| Input Voltage | 3.8V-5V | 7-12V | 3.35-12V or 5-12V | 7-12V | 7-12V |
| Digital I/O Pins | 9 | 14 | 14 | 54 | 20 |
| Analog Inputs | 4 | 6 | 6 | 16 | 6 |
| Flash Memory | 32 kB | 32 kB | 32 kB | 256 kB | 256 kB |
| SRAM | 2.5 kB | 2 kB | 2 kB | 8 kB | 32 kB |
| EEPROM | 1 kB | 1 kB | 1 kB | 4 kB | None. |
| Clock Speed | 8 MHz | 16 MHz | 8 or 16 MHz | 16 MHz | 48 MHz |
| Physical Dimensions | 50mm Diameter | 68.6mm × 53.4mm | 2.05 in × 2.1 in | 101.52 mm × 53.3 mm | 68 mm × 30 mm |
| Approximate Cost | $24.95 USD | $24.95 USD | $14.95 USD | $45.95 USD | $49.90 USD |

It is evident from the comparison table that all of the compared boards have similar suitable specifications, with the main differences being trade-offs between cost, physical size, number of pins and memory. Arduino has placed emphasis on the LilyPad as being useful for wearable technology applications, making it an attractive choice. Taking into account the developmental nature of this project, it was decided that it may be more prudent to use a board that does not require soldered connections and can easily be connected to a PC via USB for programming purposes, such as the Uno.

This leads into another advantageous aspect of Arduino boards, which is the Arduino Integrated Development Environment (IDE) (See *Figure 5*). This IDE possesses a text editor, message area, text console, toolbar and menus. In addition it contains a serial monitor for real or virtual data monitoring, which is highly useful for this project. Arduino programs, also known as sketches utilise mostly C++ language code, as well as some functions specific to the Arduino platform. The simplicity of the Arduino development process was deemed to make it a solid choice for this project and was chosen for further investigation.


sketch_oct04a | Arduino 1.6.9
File Edit Sketch Tools Help
sketch_oct04a
Arduino/Genuino Uno on COM3

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 5 - Arduino IDE

## 2.3. Recognition Methods

Once methods of detecting motion of the body have been achieved, the gesture data must be manipulated in various ways to eliminate problems and improve data reliability. There are many different digital signal processing techniques used to optimise sensor data and extract classified gestures from that data. As such, not all of these techniques are relevant to this project, as well as some being far too complex to suit the given time frame. Generally speaking, the computations involved appear to fall into three categories: Pre-processing, feature extraction or a similar technique, and gesture classification. This algorithm focused subsection aims to detail the current status of algorithms used in simple GR systems that operate using inertial measurement units.

A survey conducted by LaViola (1999) provided useful summaries of the various methods used to recognise and react to a gesture for applications similar to stage performance. Although the study is over a decade old, technology has progressed relatively slowly in the GR world as it is not a crucial research and development field, and the majority of the techniques used during this time are still highly relevant.

The techniques analysed were suitable for the use of both instrumented gloves and vision based GR systems. The three categories described for GR throughout the study were feature extraction, learning algorithms, and miscellaneous techniques. A more detailed overview of GR techniques can be found within *Table 2* on the following page, which provides good insight into the relevance of various methods to this project.

**Table 2** - Comparison of various GR techniques (Sourced from LaViola 1999)

| | Vison | Glove | Postures-Size-Accuracy | Gestures-Size-Accuracy | Training | Previous Work | Adv. Knowledge |
|---|---|---|---|---|---|---|---|
| Template Matching | Yes | Yes | Complex-Small-98% | Simple-Small-96% | Minimal | Extensive | No |
| Feature Extraction | No | Yes | Complex-N/A-N/A | Complex-N/A-N/A | None | Moderate | No |
| Active Shape Models | Yes | No | Simple-Small-N/A | Simple-Small-N/A | None | Minimal | No |
| Principal Components | Yes | Yes | Complex-Large-99% | N/A-N/A-N/A | Moderate | Moderate | No |
| Linear Fingertip Models | Yes | No | Complex-Small-90% | N/A-N/A-N/A | Minimal | Minimal | No |
| Causal Analysis | Yes | No | N/A-N/A-N/A | Simple-Small-N/A | Minimal | Minimal | No |

Upon perusing the table, it is apparent that template matching is a technique of great interest due to its compatibility with IMUs, excellent accuracy for complex postures and simple gestures, minimal training requirements and extensive previous work to look on. Another option is a feature extraction focused method, which is suited to more complex gestures and does not require training prior to recognition and classification.

LaViola (1999) stated that: "template matching determines whether a given data record can be classified as a member of a set of stored data records". This process involves creating desired templates (postures or movements), and then comparing the sensor input of the system to those templates. The system can be configured so that when a match is found, an arbitrary output is performed. The study further describes limitations and advantages of using a template matching algorithm in such a system.

An example of a template matching algorithm was presented by Benbasat & Paradiso (2001), which included a more refined approach involving 'atomic' gestures. The concept of atomised gestures is that of defining simple and common movements as gestural 'atoms', which can then be combined to form more complex gestures, without having to store that complex gesture template itself. This means that the comparisons made are between the input and a set of simple atomised gestures, rather than attempting to compare the input to an extremely large array of complex gesture patterns, which may even be difficult to reproduce.

Feature extraction is an alternative method to template matching and while being considered as a more complex route, the potential for advanced GR is greater. Feature extraction and analysis involves analysis of raw sensor data in order to determine specific information about the input, which may include properties such as mean signal level, standard deviation, variance, velocity and acceleration (LaViola 1999). Also mentioned is the computational burden for such a method.

It is important to note that GR techniques should also be accompanied by DSP techniques intended for heightening system accuracy and reliability. One approach to this, conducted by Xu et al. (2012) involved calibration, a moving average filter, a high-pass filter and normalization. Calibration involves the removal of drift errors and offsets from sensor input data, while the two filters are used to reduce high frequency noise, as well as the effects of gravitational acceleration. Finally, normalization involves the division of the signal input value over the entirety of that sensor's numerical output range to give a value that can be compared with other sensors, which may have differing voltage ranges or baselines.

## 2.4. MIDI

Musical Instrument Digital Interface (MIDI) is an important communications language and is still widely used in live music and theatre. The provision of this standard has allowed for many different types of musical instruments, home computers and multimedia equipment to communicate with ease. Background knowledge and literature review of this protocol was required in order to implement a suitable output for the GR system that can be easily read and responded to by computer software.

MIDI messages are sent asynchronously in serial format at approximately 31.25 kbps, and contain ten bits per message – A start bit, eight data bits, and a stop bit (Anderton, 1986). The protocol involves the transmission and reception of messages, which are composed of a status byte and one or two data bytes. *Figure 6* below depicts a typical midi message containing a status byte and two data bytes (note and velocity).

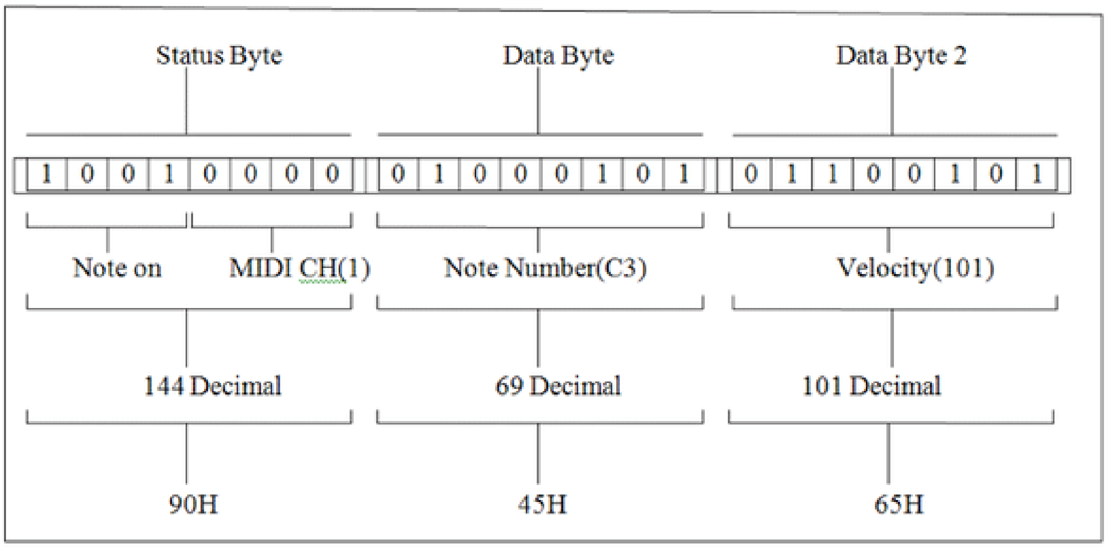


Figure 6 - Breakdown of MIDI message components

The purpose of the status byte is to identify the message type and/or the purpose of the data byte/s following the status byte. The status byte is capable of defining incoming instructions such as note on/off events, velocity, polyphonic key pressure, control/program changes, system related messages and various other commands. Data bytes contain the value of the instruction itself as a value from 0-127. The receiver must then interpret these data values depending on the status byte to perform certain functions. Appendix E contains a table detailing the conventions for transmitting messages with MIDI.

## 2.5. Data Transmission

Methods for moving data from the wearable system to a computer application needed to be explored extensively to ensure that an efficient and reliable method is used. There is a plethora of different ways to create a communications link in this way including Wi-Fi, Bluetooth, Radio frequency communication and Ethernet.

For a wearable application, the aim is to eliminate any physical connections between the wearer and external systems, so wired connections are somewhat impractical, with the exception of using wired connections for prototyping purposes. Therefore, only wireless links were to be considered, with this subsection detailing the methods that have been used in previous research. Note that the most desirable types of data link for this purpose are simple FIFO serial data.

The first transmission method to be considered is 2.4GHz Wi-Fi, which is commonly used with smartphones and personal computers. Xu, Zhou & Wen (2012) utilised a 2.4 GHz ZigBee wireless transceivers for data transfer, which are known for their excellent price point – e.g. the SparkFun nRF24L01+ Transceiver breakout at only $19.95. This cost can be reduced by opting for the transceiver chip itself rather than a breakout; however, for the purposes of the project it is far more viable to use a breakout board for prototyping and development.

The second method to consider is that of other RF modules, which operate within frequency bands from about 300 MHz to 1 GHz. Benbasat & Paradiso (2001) opted for an RF Monolithics transmitter module with a maximum transmission rate of 19.2 kbps, and transmission bands of 315 MHz and 916 MHz. Similar to this transceiver unit is the HopeRF HM-TR wireless data link. These transceivers were offered to me by my project supervisor Dr. Maxwell, as using them would mitigate the cost of the project itself. The choice of wireless link is not crucial to the outcome of the project, but the chosen link does need to possess reasonably high transmit/receive rate capabilities, as well as high reliability, i.e. ensure FIFO packet integrity and an extremely low or non-existent potential for transmission errors.

## 2.6. 3D Printing

Also known as 'additive layer manufacturing' or occasionally 'rapid prototyping', 3D printing is a method of manufacturing physical objects from various materials for many different purposes. In a world where new gadgets are dismissed in a fraction of the time they took to develop, the ability to quickly build small objects is convenient and attractive. 3D printing was considered for this project when determining the best method to produce hardware enclosures that are specifically tailored to the components at hand. Dr. Maxwell possesses enthusiasm in the field of 3D printing and suggested that it may be utilised to accompany the project to deliver a more customised solution for enclosures.

# Chapter 3

# Methodology

The methodology chapter intends to outline the engineering processes undertaken towards the completion of this project. The sections covered in this chapter are of utmost importance when completing any major engineering task, as they contain much of the planning, organisation and consideration of consequential effects necessary to ensure that no issues will arise during the course, or after the completion of the research project.

## 3.1. Research Methodologies

To gain an understanding of how the research goal can be achieved, it is necessary to consider various suitable research paradigms. A research review conducted by Borrego et al. (2009) provided a good starting point for determining adequate engineering research techniques. This review involved analysis of quantitative, qualitative and mixed research methods in engineering education; however, the knowledge contained within can be applied to engineering research in a similar manner.

### 3.1.1. Quantitative Research

Quantitative research involves tackling a research problem from an objective point of view. The aim of such practice is to attempt to reduce the possible causes of an outcome to a bare minimum in an attempt to determine a cause-effect relationship between a theory or hypothesis and the variables that contribute to the outcome. Quantitative methods usually rely on data collection through surveys or experimentation, followed by statistical analysis (Borrego et al, 2009). The results of this analysis can then be used to make generalisations about the topic, which have been supported by research findings. There will be a substantial amount of quantitative analysis involved within this project due to the quantitative nature of electrical systems. Most, if not all of the elements of a GR system can be parametrised and represented as a mathematical model. The presence of movement data from sensors, algorithmic processing and MIDI representation as numerically based system components means that mathematical and quantitative methods will need to be utilised.

### 3.1.2. Qualitative Research

Qualitative research differs from its quantitative counterpart in that it is a more subjective method of analysing a system or trend. Qualitative analysis is subjective in nature, meaning that it is often down to the discretion of the researcher as to whether a project element is described in a certain way. Whether a GRS is considered to be functional or not can be considered qualitative and arbitrary, as one person may believe that a GRS that is able to recognise one gesture and perform one output is suitably functional, whereas another may feel that while the system technically 'works', it does not possess the ability to process many gestures and is of low functionality.

## 3.2. Project Task Plan

In order to successfully complete this research project it was required to establish a solid project task plan to ensure that each necessary aspect is given an adequate amount of attention. Planning in this way helps to distribute workload evenly throughout the project completion period and eases the potential adverse effects of unforeseen circumstances.

### 3.2.1. Background Research and Theory

The first established task involved in the project was determined to be the performance of relevant background research and investigation into the mathematical and engineering concepts involved. Adequate preparation of research material assists in mitigating the necessity to determine information that is not absolutely relevant to the project itself. The project in its entirety required the asking of many research questions. *Table 3* on the following page indicates some of the more important research questions that required investigation.

Table 3 - Key research and background questions

| Project Section | Relevant Key Questions |
|---|---|
| Motivation and Problem Statement | Which engineering problem does the creation of this system solve?<br>What are the system requirements to achieve this purpose?<br>How will the project build upon existing technology? |
| Gestures | What are they?<br>Which of them are most commonly used?<br>What role do they play within human communication?<br>How are they usually incorporated into HMI? |
| Hardware | How can bodily movement be measured?<br>Which existing systems may be useful?<br>Which sensors are commonly used?<br>Which microcontrollers are commonly used?<br>What methods of wireless data transmission exist?<br>What is the most ideal method of 'wearing' this system?<br>Can 3D printing be used to create enclosures? |
| Gesture Recognition Methods | Which gestures should be added/tested with the system?<br>How will the system scan data in real-time?<br>How will the real-time data be analysed for significance?<br>How will a MIDI command be assigned to a particular gesture?<br>What will the MIDI command be used to control? |
| System Applications | Where can the MIDI protocol be usefully applied?<br>Can the system be configured for any other purposes?<br>Who will be the main users of such a system? |
| Resources and Equipment | What types of equipment will be required during the project?<br>Will any facilities be required? |
| Consequential Effects | What are the important safety practices involved in electronic system design?<br>What are the ethical implications of completing this project? What about after completion?<br>Are there any legal issues involved? |

Following the completion of the research and theoretical background section, the next step was to use the new information acquired through literature review to begin designing various aspects of the system. Although the previous section aimed to cover a broader range of crucial questions, additional questions were required as a part of the design phase in order to consolidate selection of components and planning for their assembly as a complete system. Many of the issues in designing a system such as this tend to arise in the design and build phases themselves and are difficult to pre-emptively cover.

### 3.2.2. Hardware Design

The hardware design phase involved the selection of necessary components - ensuring their compatibility with other chosen components, suitability for gesture recognition applications and reliability in terms of accuracy, precision and efficiency.

Prior to beginning thorough system design, a full risk assessment needed to be completed to prevent any serious health & safety issues as electrical design projects typically involve working with amounts of energy that can be extremely detrimental the body, equipment and facilities.

The tasks to be completed for the hardware section were as follows:

1. Design of a suitable sensing setup using accelerometer/gyroscope combination sensors for the measurement of rotational velocity and real world acceleration.
2. Select a suitable microcontroller for performing gestural data processing, MIDI output creation and any other elements of control required.
3. Select a suitable data transmission system for communicating MIDI data between the microcontroller and a personal computer.
4. Design of suitable enclosures for the sensor/s, microcontroller, transmitter and battery to produce a reasonably protected and secure system.

### 3.2.3. Software Design

The next step in task planning for this project was to develop an idea of the requirements of the software element of the system. First and foremost, the method of creating program code needed to be designated based on the chosen microcontroller and sensor. There exists a wide variety of potential candidates for programming languages that are capable of performing GR related computations, but the chosen language is largely dependent on the requirements of the ideal microcontroller.

The next task was to develop an idea of how gestures look when reading sensor data so that an apt method of analysis and computation could be implemented. Once this was determined, the focus was shifted to creating a pre-processing algorithm in an attempt to make gestural data easier to work with when performing gesture classification and recognition.

A robust way of classifying gestures, recognising them and then providing the user with an output is the core of the system and provides its true functionality. The GR program needed to be designed taking different techniques from literature, as well as knowledge of data analysis methods from courses completed by the student.

### 3.2.4. Build and Prototyping

The build and prototyping phase was planned to involve the assembly of the hardware system in addition to the software program. This process first involved the assembly of the sensing unit, microcontroller, transmitter and power source as a complete unit, preferably within 3D printed enclosures and with body attachment capability. The hardware needed to be assembled as quickly as possible to allow adequate time for software program attention. Following hardware assembly it was deemed necessary to perform testing to confirm sensor and microcontroller efficacy.

The implementation of the GR algorithm in the form of program code was a crucial element of the system, which had obviously lengthy time requirements due to likely nuances occurring. The software program could then be tested using a predetermined set of gestures to determine the accuracy of the system. Throughout this prototyping process careful notice needed to be taken of any issues preventing the system from resolving to the desired output.

### 3.2.5. Design Evaluation

As with any engineering problem, a structured process needs to be used in order to ensure that the design created is as well planned and produces optimal results. In most engineering circles it is common to finalise a system design by reflecting and re-evaluating various aspects of the design. These aspects were determined to include criteria such as functionality, cost, efficiency, ergonomics and potential for future development. At the methodology development stage, the system components that may need re-design were not evident and it was decided that the matter would be better dealt with after completing the prototype.

## 3.3. Resource Requirements and Equipment

This research project was deemed to require quite a few different resources and pieces of equipment. The tables below indicate the items that were required; however, this list aimed to be quite broad and specifics regarding approximate system costs were not clear until later in the project where the design was finalised.

Table 4 - Research and dissertation resources

| Initial Research | Necessity | Cost | Availability | Comment |
|---|---|---|---|---|
| Literature | High | Very low | High – internet, libraries | There is an extensive amount literature available for no cost, including articles present within USQ's databases. |
| Personal computer | High | Very low | In possession | Both a desktop and laptop computer are already possessed and available. |
| Word processing software | High | Very low | In possession | Microsoft Word 2013 currently installed on both PCs. |

Table 5 - System hardware and facility resources

| System Resource | Cost | Availability | Comment |
|---|---|---|---|
| 9V battery | None | In possession | 9V batteries are used for standalone Arduino operation |
| IMU | Low – roughly $10-20 | High – online | Accelerometer + Gyroscope |
| Transceiver | None – Dr. Maxwell | In possession | Two required, one for transmission, one for reception |
| Micro-controller | None – Dr. Maxwell | In possession | Arduino Uno |
| Wiring | Low – online, electronics shop | High | Adequate wiring for sensor to Arduino, as well as internal Arduino wiring. |
| Enclosures | Low | Moderate | Access to 3D printers through Dr. Maxwell, or commercially produced if required. |
| Mounting | Low | High - online | Adequate structures for bodily mounting of hardware |
| Soldering Iron | Low-moderate | In possession | Soldering iron present in home working area. |
| Safety equipment | Low | In possession | Assorted safety equipment such as safety goggles, fire extinguishing method and enclosed shoes. |

Table 6 - Software and staff resources

| System Resource | Cost | Availability | Comment |
|---|---|---|---|
| Arduino IDE | None | Free download | Software for Arduino programming |
| Analysis software (MATLAB) | Moderate | In possession | MATLAB is to be used for analysing data sets and performing simulations and calculations |
| MIDI testing software (Reaper) | Low – evaluation version is free for 60 days | Downloadable from developer website | A commonly used DAW, with which I have prior experience with music production. |
| Supervisor Contact and Feedback | N/A | As often as possible, within reason. | Notional 21 hours of contact + Dr. Maxwell's time to use at his discretion |

## 3.4. Timeline

Creating a timeline is extremely helpful when planning a research project and assists in effective utilisation of time to achieve goals before the deadline. A simple timeline was developed as a part of the initial project proposal submitted early in the project period; however, various factors caused these time constraints to become unrealistic as problems arose, and so it was advisable to create another Gantt chart to give a better representation of how the project had progressed and which tasks remained to be completed. The final version of the Gantt chart can be viewed in Appendix B.

## 3.5. Consequential Effects

The next aspect of this project's methodology to cover is the assessment of consequential effects that will arise throughout, or following the completion of the project. As an engineering student it is important to gain an appreciation of how one's work in various area may have a wider effect on society and the environment. Safety, sustainability, legality and ethics are all contenders for consideration before initiating project work to ensure that the project falls within various guidelines.

### 3.5.1. Safety

To ensure the safety of all personnel involved in the development of this research project, adequate assessment of risks and hazards needs to take place. Engineers Australia (2013) provide a useful summary of guidelines to follow when planning a research project; recognising that both professional engineers and students possess a duty of care to all people in the workplace, wherever that work may be conducted. A formal risk assessment was prepared (See Appendix C) prior to commencement of practical tasks in an attempt to minimise risk of injury or equipment damage.

### 3.5.2. Sustainability

A part of the background research involved in preparing for the project was to take environmental impacts into account. With the advent of consumer electronics, e-waste has become a topic of much discussion amongst engineers and researchers. Electronic waste comes in many forms and can contain highly hazardous materials such as lead, mercury, arsenic and antimony trioxide (Clean Up Australia 2015). These substances eventually contaminate local soil and groundwater causing a multitude of problems. In preparation for the project, methods of mitigating contribution to the e-waste dilemma needed to be considered. It is important that any electronics used for the purposes of the project are disposed of carefully, if necessary.

# Chapter 4

# Design

This chapter describes the processes that were undertaken in the development of the GR system concept. Design objectives, hardware specifications, enclosure design, software requirements, algorithm design, data transmission and system assembly will be covered, so that the reader may gain an appreciation of the steps taken in developing an optimal design for the purposes of this project.

## 4.1. Design Objectives and Requirements

Before the commencement of system design, objectives and requirements needed to be declared to ensure an end result that is capable of performing the tasks outlined in the objectives section of Chapter. The following design objectives were developed after conducting literature review and determining appropriate project methodology.

| | |
|---|---|
| Reliability | The final version of the system should produce consistent results and should not be prone to error. In addition, there should be little requirement for maintenance. |
| Accuracy | The system should be able to accurately detect when a gesture has been enacted, and should provide feedback to the user to indicate that this recognition has been successful. |
| Latency | The time delay between a gestural action and the production of an output should be sufficiently small that the user feels as though they are controlling external systems in (almost) real time. |
| Ergonomics | The wearable section of the system should be comfortable for long periods of wear, light-weight and should be adjustable so that a performer of any size or shape is able to make good use of it. |
| Ease of Use | Ideally the system should be operable by almost anyone who possesses basic computer skills and musical competency. This design requirements takes system setup, interaction and troubleshooting into account. |
| Simplicity | Due to time, manufacturing and financial constraints, the aim is to have a simple, functional system that is able to recognise between one and a handful of gestures accurately and produce a usable MIDI output for each. |
| Safety | At no point during the use of the system should the wearer feel at risk of electric shock or physical damage from hardware components. |

## 4.2. Component Selection

After examining the various hardware components available during chapter 2, final decisions needed to be made on the hardware components that would be used to build a prototype system. The components chosen within this subsection are in no way the 'perfect' components for this purpose; however, adequate consideration was taken into account regarding pricing and functionality to ensure that each of the chosen components is capable of being combined into a fully functional system. Component optimisation is to be considered later within this dissertation while performing system evaluation

### 4.2.1. Inertial Measurement Unit – MPU6050

Detection of movement was deemed to require the tracking of both acceleration and angular velocity data in three dimensions so that the position and movement of the wearer's wrist may be tracked in real time. This is possible using a combination of an accelerometer and a gyroscopic sensor. Single-axis variants of these sensors are available commercially, but would not be suitable for tracking linear motion across multiple axes. Tri-axis accelerometers and gyroscopes are far more suitable for such an application, where acceleration and angular velocity in three degrees of freedom each can be tracked with ease. A combination of tri-axis accelerometer and gyroscope provides six degrees of motion detection: X, Y and Z axis acceleration, as well as yaw, pitch and roll velocity (*Figure 7*).

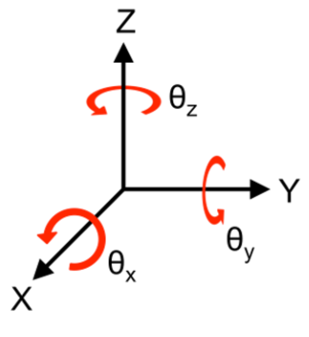


Figure 7 - Six Degrees of Freedom

The MPU6050 is a combination inertial movement sensor with a key focus on low power requirements, low cost and high performance. The MPU6050 consists of a 3-axis gyroscope and a 3-axis gyroscope with an additional digital motion processor (DMP), which acts as a preliminary signal processor and is capable of processing sensor data using 'MotionFusion' algorithms. DFRobot Australia created a breakout board for the MPU6050 sensor, pictured below in *Figure 8*.

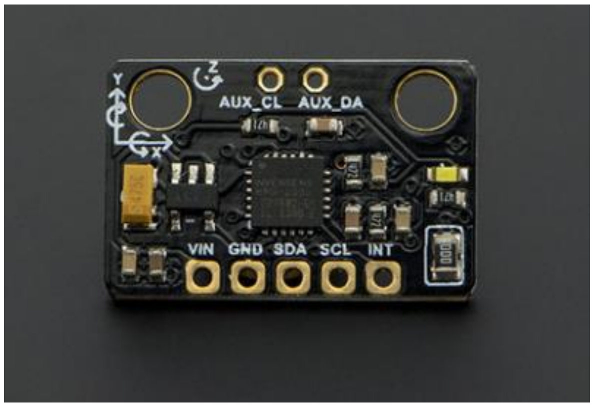


Figure 8 - DFRobot 6 DOF sensor

This 6 DOF sensor breakout includes a low noise 3.3V regulator for supply to the MPU6050 and pull-up resistors for the I2C bus. The breakout board makes connection to a microcontroller easy provided that the controller possesses I2C capabilities. I2C is a serial communication protocol for a two-wire interface (SCL and SDA) developed by Philips and is used my most major IC manufacturers for interfacing microcontrollers, memory, ADCs, I/O interfaces and other peripherals (I2C Info 2016).

The desired data format is selected within controller program code and the MPU6050 can be configured to produce sensor outputs across the two-wire interface in multiple forms including raw data, quaternion and Euler values.

## 4.2.2. Processing Unit – Arduino Uno R3

Potential options for a processing unit were briefly discussed within the literature. The result of those deliberations was the selection of the Arduino Uno as a suitable development board for this project. The Uno pictured in *Figure 9* has suitable features and processing capability for simple on-board gesture recognition. Ease of development was a large contributing factor in utilising this component, which was supplied by Dr. Maxwell. Much of the technical specifications are covered within *Table 1 - Arduino Comparison Table* in chapter 2; however, some of the more specific features still need to be covered.

The Uno is capable of performing transmission and reception of TTL level serial data using digital I/O pins 0 (RX) and 1 (TX), which are connected to an ATmega8U2 USB-to-TTL Serial chip. The board also comes with external interrupt, SPI and Twin Wire Interface (TWI) capabilities, which will be further detailed later in the chapter. The TWI allows simple connection of the aforementioned MPU6050 to the Uno board. Note also that the Arduino Uno can be powered by a 9V battery due to its generous allowable input voltage range, and in addition, the board is able to supply both 5V and 3.3V output to external components, meaning that both the transmission unit and the sensor may be connected to the Uno for power without requiring a separate supply

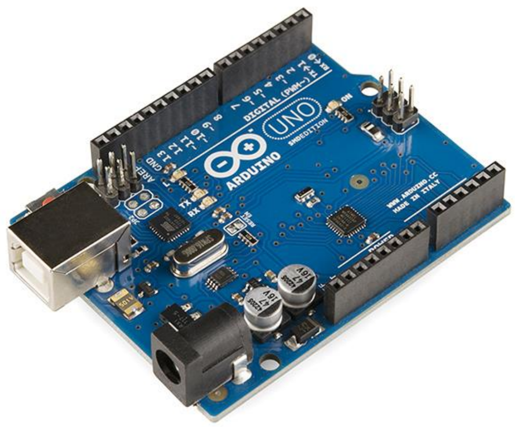


Figure 9 - Arduino Uno R3

### 4.2.3. Data Transmission – HopeRF HM-TR

This subsection details the selection of the data transmission system that transfers MIDI commands generated by the development board wirelessly to a PC terminal. The HopeRF HM-TR wireless data link transceiver supplied by Dr. Maxwell was determined to be the best immediate choice of transceiver for the application due to its simple operation, ease of availability and compatibility with the Arduino Uno. The transceivers feature high data rate and long transmission distance, both of which are advantageous in an entertainment setting.

The HM-TR operates by means of frequency shift keying (FSK) in half duplex mode. FSK involves the modulation of data onto a carrier waveform by altering the frequency of the carrier over time, so that the data can be extracted at the receiver end simply by filtering out the carrier frequency, producing the input data in its original form. The transceiver's standard universal asynchronous receiver-transmitter (UART) interface can be used in combination with the UART on the Uno making communications between transceiver and controller simple to setup and operate.

One transceiver is connected to the Arduino Uno, which receives a MIDI data stream in the event of gesture recognition. The transceiver then modulates onto a 915 MHz carrier ready for transmission. At the receiver, another identical transceiver is operated via a Freetronics LeoStick, which reads received data and then outputs the data stream to the serial monitor in the Arduino IDE. The Freetronics LeoStick is an Arduino compatible development board approximately the size of a standard USB flash drive and has similar specifications to the Arduino Uno.

## 4.3. Enclosure Design and Wearability

After selection of components, the next step was to decide on a suitable method of attaching them to the wearer, as well as shielding said components from external damage. This subsection provides details on positioning of hardware on the body in addition to the cases that will be used to store the system hardware.

### 4.3.1. Hardware Positioning

The use of this system within a live entertainment setting means that many of those who are likely to use it will be in motion frequently. This motion may be vigorous, therefore the fastening mechanisms need to provide adequate adhesion to prevent wiring and enclosures from detaching from their default positions and causing damage to enclosures or wearer. The necessity for restrictive adhesion can be minimised by reducing enclosure weight; however, based on the components selected it was unlikely that the wearable components of the system would be bulky or of any significant weight.

Firstly, the sensing component of the system needed to be positioned on the wrist of the wearer for recognition of simple arm gestures such as lateral and rotating arm movements. As such, the wrist was deemed a likely position for the sensor, preferably on the upper side to simulate the feeling of wearing a wristwatch. Wireless communication was considered for use between the sensor and development board, but was not pursued as the requirement for a separate power supply and transmitter for the sensor was not attractive. Instead, a simple ribbon cable would be used for the pin connections between the sensor and the Uno (Vin, GND, SCL, SDA and INT).

Following the positioning of the sensing component, an ideal method of attaching the development board, transceiver and power supply to the wearer needed to be determined. It was assumed that each of these components would be combined into a singular enclosure, which was to be positioned on the back of the wearer as this is the least likely place for bodily movement to influence positioning during a performance.

This central enclosure could then be mounted onto an elastic harness such as the GoPro harness depicted in *Figure 10*, which is easily adjustable and would allow most body types to easily wear the system.

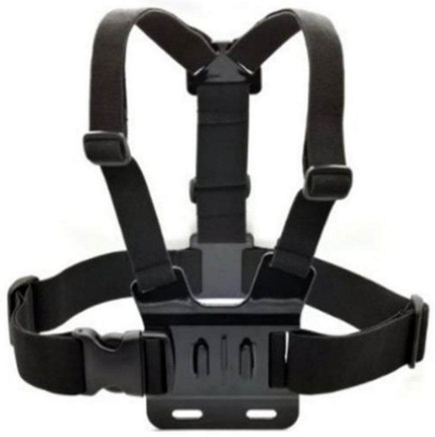


Figure 10 - GoPro Harness

### 4.3.2. OpenSCAD

OpenSCAD is an open-source computer aided drawing (CAD) program oriented towards those who are proficient in the writing of software syntax. The primary selling point of the software is that it focuses on creating accurate computerised 3D drawings rather than attractive graphical representations (Kintel 2016). This aspect of the software makes it useful for the design of machine parts, or in the case of this project, models for 3D printed hardware enclosures.

The program acts as a 3D compiler rather than an interactive modeller, allowing the user to create complicated 3D models by forming union, differences, translations and/or rotations between basic prism shapes such as rectangular boxes and cylinders. The simplicity of the program means that models can be developed rapidly with knowledge of only a few commands, thus the program was selected for use in the development of enclosures for the sensing unit and the main system circuit.

## 4.3.3. Wrist Brace

The wrist brace for the sensing unit needed to be designed ergonomically so that the wearer's wrist movements are not impeded by the enclosure. For the fastening mechanism, a simple Velcro strap allows usage by a range of wrist diameters while still being strong enough to hold the enclosure against the limb. For the main section of the wrist brace, a 3D CAD model needed to be created, which may then be utilised with a 3D printer to create a physical model of the enclosure. An overall enclosure size of approximately 40mm x 30mm was deemed appropriate for most wrist sizes while keeping the design compact.

A 3D OpenSCAD model was developed for the wrist brace with the following features:

- Rounded wrist contact surface for shape tapering.
- Two 2.5mm slots with rounded corners for strap insertion.
- A 10mm x 2mm slot for the ribbon cable.
- An open upper side for sensor access with a cover for when the system is in use.
- Pillar structures to fit into the mounting holes of the sensor board.

OpenSCAD program code for the wrist brace can be viewed in Appendix G

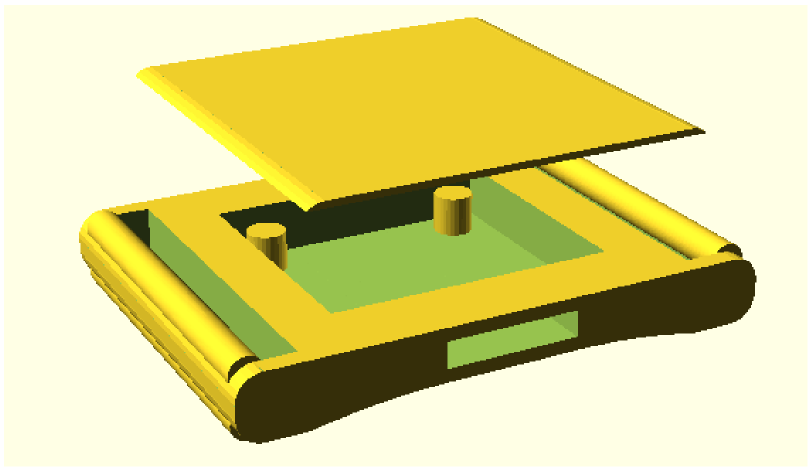


Figure 11 - Sensor wrist brace

### 4.3.4. Main System Housing

The next enclosure design task was to design a housing for the main section of hardware, which includes the development board, transmitter and power supply as well as any wiring and routing using Vero board if necessary. The power supply used to power the entire main section is a 9V battery, chosen for simplicity's sake; however, a rechargeable battery system is something to consider for further system refinement in the future. Firstly, each of the aforementioned needed to be accurately measured, which was done using Vernier callipers. A simple arrangement of the components situated side by side was applied and a mock-up of a simple housing was created in OpenSCAD, visible within *Figure 12* and *Figure 13*.

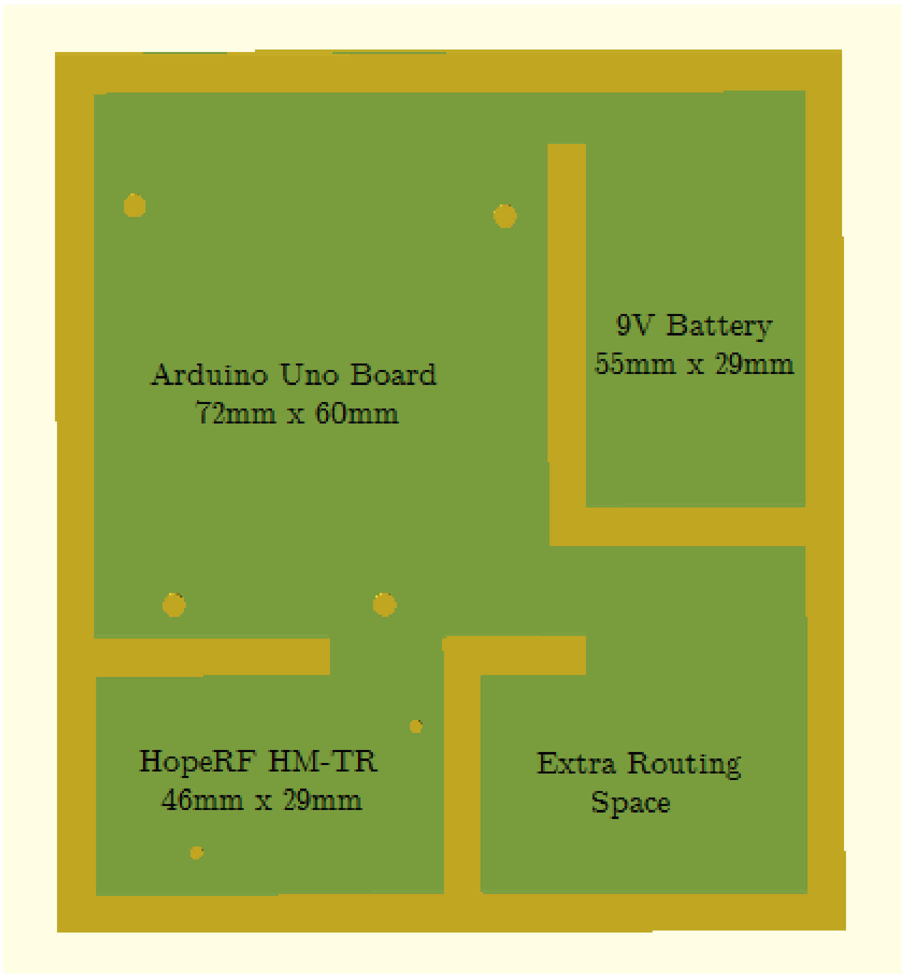


Figure 12 – Main Enclosure Top View

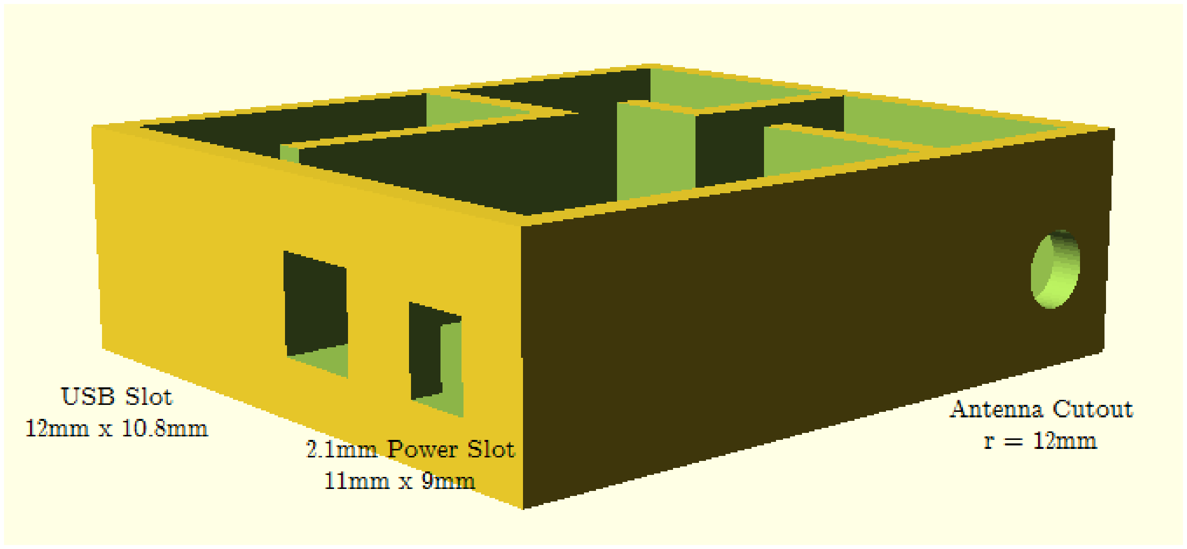


Figure 13 - Main enclosure side view

Unfortunately time was short when designing this enclosure and although it is suitable for this research project, more optimal approaches would exist. Regardless of this fact, special care was taken in the design of the enclosure with the intention of fabrication later in the project period once prototyping had been completed. Special care was taken to ensure that each component had a reasonable amount of space by concatenating an extra few millimetres onto size measurements for each of the components. Mounting pylons were also added to the design for securing the components that have mounting holes, which was foreseen to negate the requirement for other adhesion methods and allow easy removal of components.

The design has final length, width and height parameters of 116mm, 104mm and 35mm respectively, making it a reasonably sized enclosure for a performance application. Potential suggestions for further optimisation of the enclosure include removal of the compartmentalisation walls to allow further compaction of components, a general shape that is better tapered to the components being used and the possibility of a soft enclosure that would completely minimise the volumetric requirements of the chosen hardware. This concludes the design of enclosures, which ideally were to be prototyped through additive manufacturing.

## 4.3.5. Hardware Assembly

To ensure correct operation of hardware, planning was undertaken in order to understand the physical connections that are necessary between each of the components. These components can be divided into two subgroups – those that are connected to the Arduino Uno development board, and those that are connected to the PC. From this point onwards, these groups will be referred to as the wearable group and the terminal group. The wearable group consists of the sensor, Arduino, RF transmitter and battery; while the terminal group consists of the LeoStick and the RF receiver.

The first group to be assembled was the wearable group. The connection of the 6 DOF sensor to the Arduino required a 5-wire ribbon cable so that it would be ready for wear once enclosures had been produced. Correct wiring connections were determined by inspecting the MPU6050 datasheet supplied online by InvenSense. These connections are shown in *Table 7* below. The MPU6050 Vin pin requires an input voltage between 2.375 V and 3.46 V, hence it was connected to the 3.3 V output of the Arduino Uno, which is produced by an on-board voltage regulator. As for the SDA and SCL lines that form the I2C/TWI interface, connections were made between these pins and analog pins A4 and A5 on the Arduino, which correspond to the Arduino's own I2C lines. Note that third party library functions are required when programming the Arduino in order to utilise this communications protocol.

Table 7 - Sensor to Arduino connections

| 6 DOF Sensor Pin | Arduino Uno Pin |
|---|---|
| Vin | Regulated 3.3V supply |
| GND | GND |
| SDA | Analog pin A4 |
| SCL | Analog pin A5 |
| INT | Digital pin 2 |

A sensor without a suitable enclosure for performing arm movements was an unattractive prospect and so a prototype of the wrist brace was fabricated by Dr. Maxwell using his personal 3D printer, which although imperfect, would provide a suitable mounting medium to provide true sensor positioning before the enclosure designs had be further refined. This enclosure is depicted in *Figure 14* and is of a prototypical nature with no lid, created purely for testing purposes.

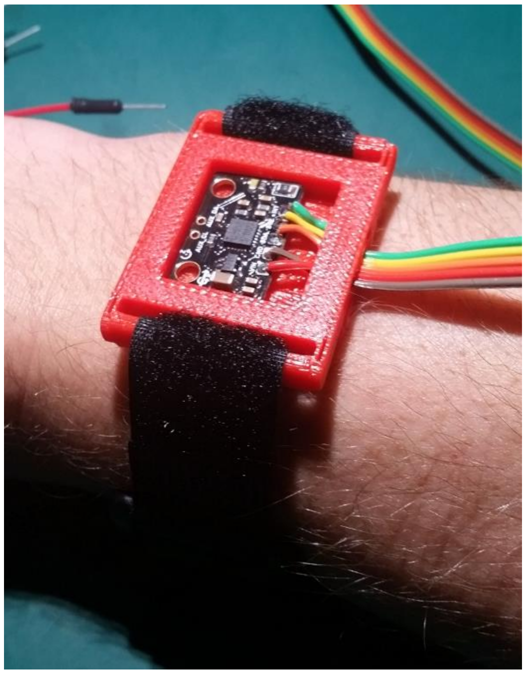


Figure 14 – Wrist brace prototype

With the required sensor connections to the Arduino finalised, the next step was to determine connections for the transceiver. The HM-TR wireless data link only required connection of four pins to the development board: Vcc, GND, DTX, and DRX. The transceiver requires a Vcc supply of approximately 5 V and an electrical grounding connection, both of which were easily connected to the relevant pins on the Arduino board. DRX and DTX are the pins responsible for UART based serial communications and were subsequently connected to the Arduino hardware UART on digital pins 0 and 1. The transparent nature of the transceiver allows for easy communications setup as the chip performs its own self-controlled protocol translation, allowing for communication with the transceiver using Arduino serial functions. As has been previously mentioned, the Arduino Uno is able to supply 3.3 V to the sensor and 5 V to the transceiver without the requirement of an external voltage regulator. A 9 V battery can be connected to the Arduino; however, this was not the most ideal choice of power supply for the system. For testing purposes the 9 V battery sufficed, but a replacement power supply for future project work is discussed in the next chapter. A battery case with a 2.1 mm jack was chosen to allow easy connection and disconnection of the battery during prototyping.

It was predicted that once the main housing had been created, this battery case would be eliminated and connections would be made directly to the Vin and GND along with a switch to allow disconnection of the battery power without having to remove the battery itself. This concluded the necessary planning for assembling the wearable group and a simple wiring diagram was created to aid in physical assembly, which lies on the following page (See *Figure 15*). 'Fritzing' software was utilised to create the diagram as it contains 2D models of the Arduino Uno in addition to various other components commonly used with such development boards. Note that the sensor and transceiver components used within the diagram are not the correct models, but still possess the same pins as the components used, hence they were still adequate for visualisation purposes.

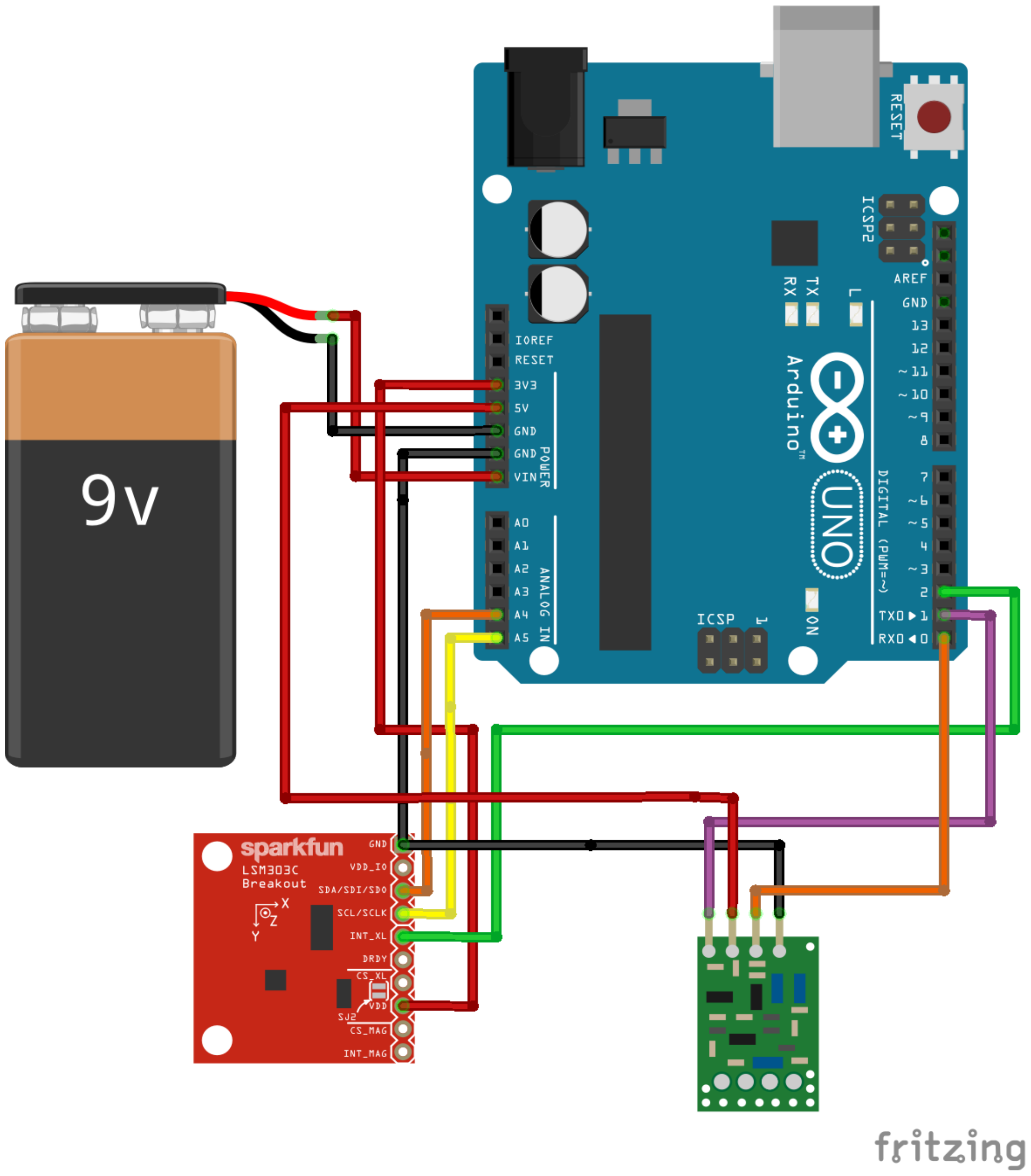


Figure 15 - Wearable group wiring diagram (Produced with fritzing)

Following the completion of the wiring diagram, physical assembly of the components as discussed was carried out. The ribbon cable used for the sensor required the addition of small screw-clamp headers to the Arduino pin headers to create a solid connection. Solderless breadboard wires were used to connect the transceiver to the Arduino as much of the Uno programming would not require the transceiver to be connected. At this point all of the components had been physically interfaced and were ready for software development. The physical assembly of the wearable group electronics is presented below in *Figure 16*.

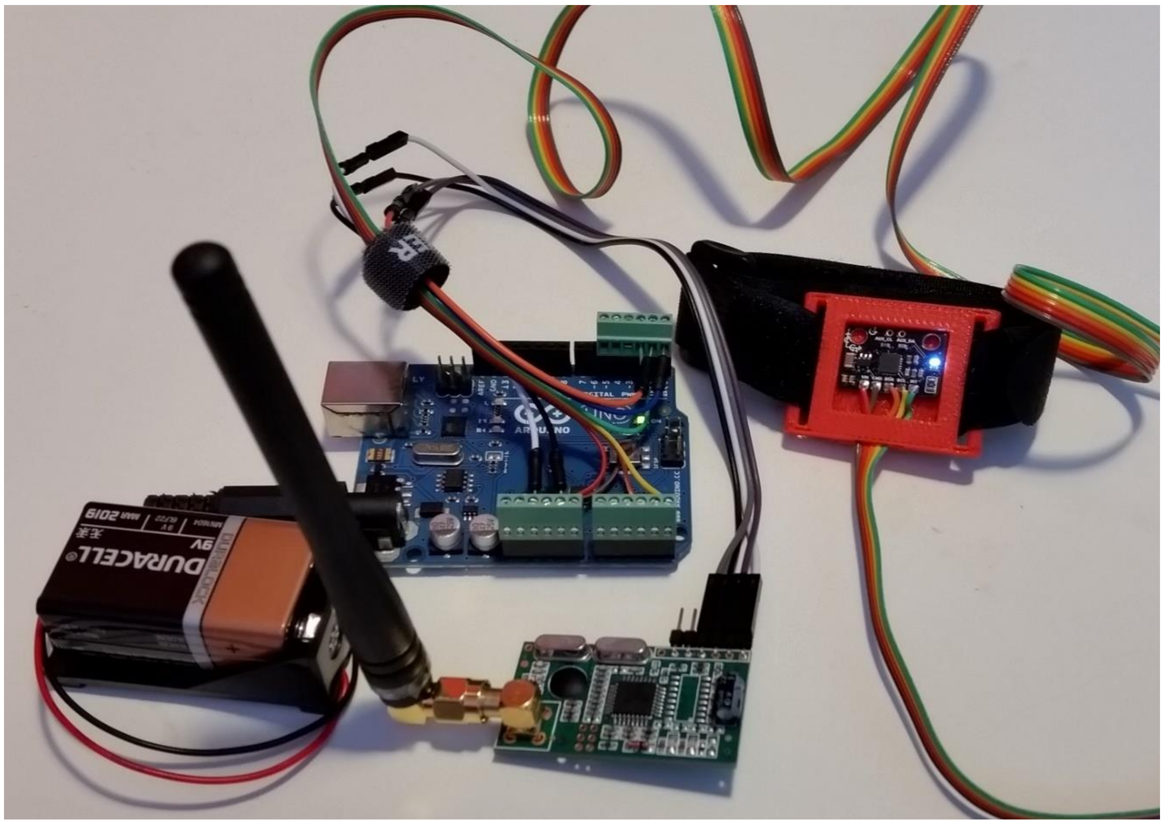


Figure 16 - Completed wearable group assembly

Following the completion of the wearable group assembly, development of the terminal group commenced. This was a simple process and only required another transceiver be connected to the Freetronics LeoStick. In this case the transceiver was connected to the LeoStick in much the same way as was the case in the wearable group, however this time a small breadboard was used to mount both components and attach the four necessary breadboard wires (See *Figure 17*). Creation of an enclosure for the terminal group was considered, but was avoided due to time constraints and was considered to be unnecessary as there would be little threat of physical damage when plugged into a laptop USB port.

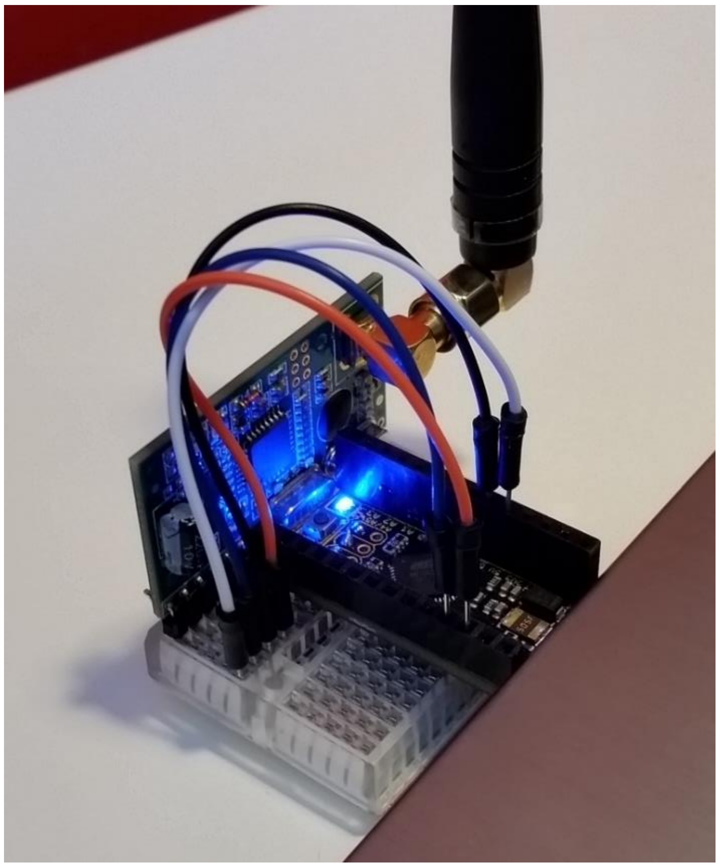


Figure 17 - Completed terminal group assembly

An important design consideration to mention is the fact that the LeoStick only possesses one hardware UART, meaning that serial communications between the board and the transceiver would not be possible at the same time as communications between the board and the computer USB port. This problem could have been solved by either scheduling UART usage so that the transceiver and the USB port share usage of the hardware UART or by utilising SoftwareSerial. SoftwareSerial is an Arduino library that contains functions allowing the use of digital pins (other than pins 0 and 1) as a virtual UART, meaning that the hardware UART would be fully devoted to presenting received data to the PC. Further details on the usage of SoftwareSerial for this purpose are detailed later in this chapter.

The hardware assembly section was concluded resulting both the wearable group and the terminal group having been assembled and in a suitably ready state for application of a GR program and to cover transmission of generated MIDI commands over the transceiver link.

## 4.4. Software requirements and objectives

Prior to carrying out any writing of program code, software design objectives and requirements needed to be developed to ensure system functionality corresponding with the project objectives mentioned in Chapter 1. The software objectives and requirements were needed to divide the intended program into sections so that a function or program element could be devoted to each major aspect of the overall system process. The flow chart presented below shows the significant steps involved in transforming the input, an arm movement, into a MIDI output that is ready to be utilised by external hardware or software.

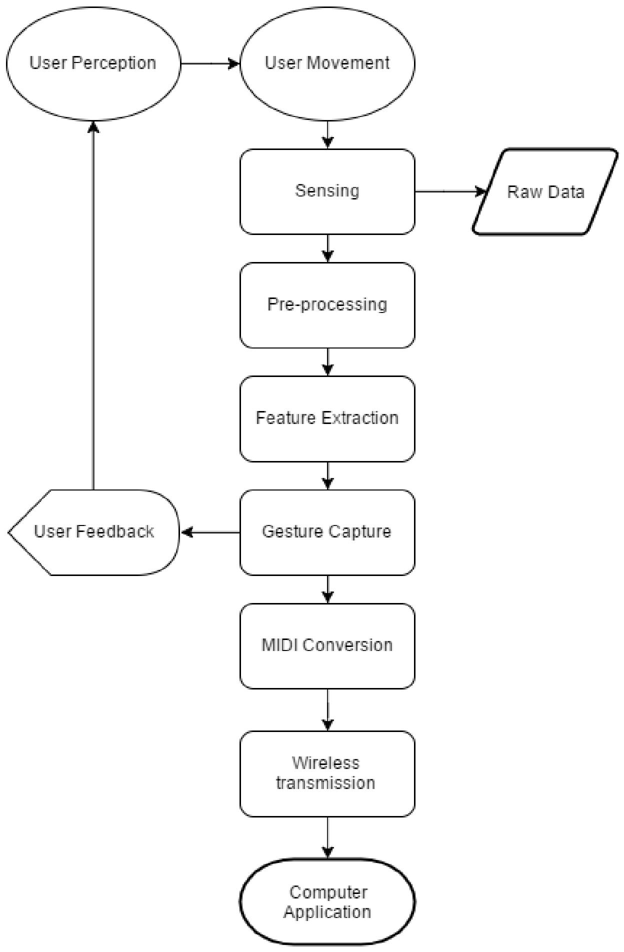


Figure 18 - GR system process flow

### 4.4.1. Operation Modes

For the GR system to be capable of remembering a gesture and recognising that gesture when it is enacted, a minimum of two operation modes are required. A training mode, where the user may choose a gesture to be recognised, which is then saved in system memory and a running mode that carries out comparison of current movement data with the trained gesture template/s and produces an output. In order to implement an algorithm based on template matching, the training mode must be capable of storing an accurate rendition of a desired gesture, which has a low probability of unintentional recognition and a high rate of intentional recognition. Therefore, the training process should involve a reasonable number of repetitions of the gesture to be trained so that a precise template can be constructed.

### 4.4.2. Data Acquisition

The MPU6050 provides highly useful positional data in three dimensional space, which can be sampled at a high rate through its FIFO buffer using the Arduino Uno (Up to about 400 Hz). Sampling rate is an important aspect of data acquisition as sample rates that are too low tend to produce unsavoury results. This sampling rate must be high enough that reasonably fast arm movements are tracked in their entirety.

In addition, the acquired data must be in a suitable format for mathematical manipulation, as any computations that are performed on error-ridden data only exaggerate the issue. The trade-off that needed to be made here was between accuracy of time based movement data and the amount of data that can be stored for computation. The compact nature of the Arduino Uno means that there is a very limited amount of memory with which to store movement data that requires further computation. With only 2 kB of SRAM for variable storage, completely filling the available memory is no difficult task when using data forms that take up many bytes per variable. Ideally 8 bit integers are used over longer data formats such as 16 bit integers or 32 bit long integers whenever possible.

### 4.4.3. Pre-processing

Fortunately the use of the MPU6050 as the sensor for this project negates much of the need for pre-processing of data. Pre-processing acquired data would usually involve the use of techniques such as low pass filtering, normalization and drift correction; however, implementation of these techniques in code was not required as the MPU6050 performs much of this processing before the acquired data is read by the Arduino. Hence, it was decided that rather than performing such pre-processing, this section would only consist of forming a sliding window of data to allow real time movement tracking, which will be discussed further later.

### 4.4.4. Feature Extraction

Feature extraction aims to perform various statistical computations on incoming data in an attempt to isolate data elements that can be used to distinguish different types of movements from positional data. In order to carry out this process, a running average filter needed to be applied to the data (further details within the 'Arduino Program Creation' section) as a simple method of differentiating the data produced by different movements, which is in turn used in the creation of a gesture template. The feature extraction aspect of the program needed to be comprehensive enough to be able to recognise similar movements with a high degree of accuracy, while using minimal amounts of processing power and memory.

### 4.4.5. Gesture Capture

The gesture capture section is responsible for comparing incoming movement data with a stored template and determining whether a match exists between the two. The comparison process needed to be performed at a high speed to ensure low latency between wearer movements and their observation of the output. The most suitable method of comparison for the system was deemed to be a simple threshold comparison of running average trends, which involves comparing the current running average to a trend template by determining whether the current position is within a certain distance of the trend template. This method would require a comparison for each positional and rotational vector, and if the comparison is returned as true for most or all of the vectors, a gesture is present and would warrant the transmission of a MIDI command.

### 4.4.6. MIDI Command Transmission

It was predicted that the creation of GR program code would take an extensive period of time, and so the requirements of the system to produce a MIDI response to a gesture needed to be kept relatively basic. Once a gesture had been captured and confirmed, the system would then need to determine which MIDI command to send to the PC. Ideally many commands would be possible, with a range of different movements triggering the output of each; however, for conceptual proofing purposes, all that would be required of this software section is the definition of an arbitrary command, followed by its transmission to the PC. Additional considerations included the frequency of transmission.

Based on the assumption that the vast majority of arm gestures are not usually performed in less than one second, the Arduino would not be required to send more frequently than at this speed. To ensure low latency between system input and output, the Arduino would need to check more frequently than once per second to determine whether a gesture has been recognised, and if so, send the MIDI data to the PC.

### 4.4.7. MIDI Command Reception

The location of the terminal group at the PC means that the receiver simply needs to check for transceiver data repeatedly at a rate that is fast enough to keep system latency low. The receiver then needed to be able to setup and perform serial communications compliant with the MIDI protocol to allow the data to be utilised by external applications.

## 4.5. Arduino Program Creation

Once the software objectives and requirements had been defined, work could be commenced on writing Arduino software that carries out each of the necessary functions. Arduino software was separated into three program listings – The main GR program to be uploaded to the Arduino Uno, the MIDI transmission program to be appended to the main program once complete, and finally the MIDI reception program to be uploaded to the LeoStick.

This section covers the software functions used to achieve the design requirements. The 'skeleton' of the program stems from a piece of Arduino code developed by Jeff Rowberg that demonstrates the use of I2C with the MPU6050 sensor. This code example, 'MPU6050_DMP6.ino', was freely accessible to the public at the time of 11 October 2016 and is likely still available at the following address:

< https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050 >

The full main program listing can be found in Appendix H.

### 4.5.1. Library Inclusions and Variables

The main program required the inclusion of a number of software libraries to provide additional functionality on top of what the Arduino was already capable of achieving. Further details on the functions contained within libraries will be described when detailing program functions of the GR system. A list of utilised libraries is provided on the following page.

Table 8 - Applicable Arduino libraries

| Arduino Library | Description |
|---|---|
| `I2Cdev.h` | I2C communications library. Used to create interfacing between the Arduino and the MPU6050 sensor. |
| `MPU6050_6Axis _MotionApps20.h` | MPU6050 library. Used to setup the sensor for movement tracking and to read positional data from FIFO buffer |
| `Wire.h` | TWI library. Only required in the event that the `I2Cdev.h` library needs definitions from this header file. |
| `elapsedMillis.h` | Timing library. Useful for tracking the amount of time a section of program takes to execute. |
| `EEPROM.h` | Electronically erasable programmable read only memory library. Used to allow more permanent storage of variables, which will not be lost after system shutdown. |

### 4.5.2. Initialisation

The first initialisation step of the main program is the definition of test integers `ArrayIndex`, `ArrayIndexDelay` and `TrainingCount`, as well as modal definitions used for testing – `TestMode`, `DispMode` and `WindowArraySize`. `WindowArraySize` defines the size of the sliding window of data to be analysed by the recognition algorithm and was selected to be 30 samples.

General variable definition follows, which defines necessary variables and containers for use with the I2C and MPU6050 libraries. These include the Boolean variable `dmpReady`, integers for I2C communications such as `mpuIntStatus`, `devStatus`, `packetSize` and `fifoCount` as well as orientation and motion variables for use the with DMP including a quaternion container, vector integers for acceleration values and floating point numbers for euler angles and yaw/pitch/roll values.

### 4.5.3. Sliding Window Tracking

Sliding window implementation necessitated the definition of many variables specific to the sliding window. Window arrays containing 30 indices needed to be created for yaw, pitch and roll as well as x, y and z axis accelerations. With the use of a running average in mind for later in the program, 32 bit integers needed to be created to hold the sums of window values for each of the degrees of freedom. 32 bits were required as each of the numbers can potentially be in the thousands in decimal format, so a large binary representation was necessary. Following this, 16 bit integer running average variables were created to hold calculate averages for the window; and finally 16 bit integer arrays needed to be defined for the storage of multiple window averages during training mode for later comparison by recognition code.

The setup procedure for the program loop involved utilisation of I2C and MPU6050 library functions to initialise and ready the MPU6050 for data acquisition. This was achieved using MPU6050 example code, which begins serial communications at a baud rate of 38400 bps and initialises the MPU with `mpu.initialise()` and `mpu.testConnection()`. The DMP is then loaded and configured using `mpu.dmpInitialise()`, and finally offsets for each of the gyro axes and the z axis accelerometer are set. Default values for offsets were used, which are provided within the MPU6050 datasheet. Provided there are no issues initialising the MPU, the DMP is enabled and flagged as such, followed by determination of expected incoming data packet size using `mpu.dmpGetFIFOPacketSize()`. Window arrays are then initialised so that all 30 values of each array are equal to zero, concluding the setup process.

The main program loop, `loop()` as well as the aforementioned setup procedure include a number of prompts that are sent along the serial connection to the PC when connected using a USB cable. These prompts are viewed using the built-in serial monitor in the Arduino IDE. At the beginning of the main program loop, `elapsedMillis` is called to begin a millisecond timer that is used to track the time taken for the program to iterate through a single window of data, i.e. 30 samples. I2C library functions are called to check interrupt status, and to read the FIFO buffer from the MPU in order to retrieve positional data.

Once FIFO data is retrieved, each of the six positional vector values are extracted and stored into window arrays. The following code section carries out this process, in addition to calculating the current running average of data.

```
// Populate yaw, pitch and roll window arrays
        Window_Yaw[ArrayIndex] = ypr[0] * 180/M_PI;
        Window_Pitch[ArrayIndex] = ypr[1] * 180/M_PI;
        Window_Roll[ArrayIndex] = ypr[2] * 180/M_PI;

// Populate window array with current accel vals
        Window_Accel_X[ArrayIndex] = aaReal.x;
        Window_Accel_Y[ArrayIndex] = aaReal.y;
        Window_Accel_Z[ArrayIndex] = aaReal.z;

 // Add fetched positional values to respective sum
        Sum_Yaw = Sum_Yaw + Window_Yaw[ArrayIndex];
        Sum_Pitch = Sum_Pitch + Window_Pitch[ArrayIndex];
        Sum_Roll = Sum_Roll + Window_Roll[ArrayIndex];
        Sum_Accel_X = Sum_Accel_X + Window_Accel_X[ArrayIndex];
        Sum_Accel_Y = Sum_Accel_Y + Window_Accel_Y[ArrayIndex];
        Sum_Accel_Z = Sum_Accel_Z + Window_Accel_Z[ArrayIndex];

// Calculation of averages
        Window_Average_Yaw = Sum_Yaw/WindowArraySize;
        Window_Average_Pitch = Sum_Pitch/WindowArraySize;
        Window_Average_Roll = Sum_Roll/WindowArraySize;
        Window_Average_X = Sum_Accel_X/WindowArraySize;
        Window_Average_Y = Sum_Accel_Y/WindowArraySize;
        Window_Average_Z = Sum_Accel_Z/WindowArraySize;
```

Once the running average has been calculated, each of the values are displayed depending on the chosen display mode (`DispMode`), which can be set low for direct yaw, pitch, roll and x, y and z acceleration values, or high for running averages of each. The loop iteration is then concluded and will either stop the program, displaying elapsed run time, or will continue to run depending on the chosen `TestMode` value at the beginning of the program.

## 4.5.4. Training Mode

There are three possible running modes that can be used with the program by altering the definition of `TestMode` in the test variables section of program code. `TestMode 0` acts as a limited running mode, which will read positional values until the window array has been filled and will then halt, waiting for user input to begin this process again. This mode was useful in determining ideal window size and elapsed time for filling the window array. `TestMode 1` is a full running mode, where running averages are calculated continuously and are compared to a saved gesture template stored in EEPROM. The final mode, `TestMode 2` engages training mode, where the user is able to store a desired gesture by enacting it five times before a template is generated based on the five movements.

## 4.5.5. MIDI Transmission and Reception

Two programs were developed for dealing with the MIDI output required of the system, one for transmission and one for reception. The goals for the design of the transmission program were to select a typical MIDI command and send it across the wireless link so that it could be accurately reproduced at the receiving end. No library functions were required for the HM-TR transceiver due to its own data conversion and modulation capabilities. The transmission code begins by opening serial connections at a baud rate of 9600 bits per second:

```
void setup()
{
  Serial.begin(9600);  // Monitoring via USB
}
```

The continuous loop for the transmitter program involve a simple process of tracking elapsed time so that the MIDI command as well as a new line are sent every second using `Serial.print()`. The MIDI command to be sent was a standard 'Note On' command of decimal value 144, or binary value 10010000. The full program listing for command transmission can be viewed in Appendix H.

An accompanying receiver program was also created to receive the incoming command. The LeoStick required the use of the `SoftwareSerial.h` library, which allowed the creation of a virtual UART on digital pins 10 and 11 for Rx and Tx respectively. The program begins in a similar way to the transmission program firstly by opening serial connections between the computer USB port and the LeoStick using `Serial.begin(9600)`, setting the data rate to 9600 bits per second. The controller then waits for the serial connection, followed by opening virtual serial communications with the transceiver using:

```
mySerial.begin(9600).
```

The process of data retrieval involves a looped process that first checks for serial data availability, and if available, reads the `mySerial` data into an integer. A check is then performed to determine whether the retrieved data is a digit with `isDigit` and if so, appends the digit onto an output string. The output string is presented over the serial monitor once a new line is detected, thereby showing the binary form of the MIDI command using the following function:

```
Serial.println(inputstring.toInt(), BIN);
```

Each of the aspects of the developed Arduino code have now been covered, with further discussion of functionality in the results and testing chapter. Unfortunately, due to the nebulous nature of the project earlier in the research period, the functionality of the software section is somewhat lacking, which will also be discussed in the next chapter.

# Chapter 5

# Testing and Results

This chapters aims to summarise the end-game functionality of the system design outlined in the previous chapter. Following this, implementation issues will be detailed and analysis of the system's fulfilment of the project objectives will be presented. Much of this section involves qualitative analysis of the system, with the aim being to determine the current stage of development including the aspects that still require implementation and the feasibility of the system as a whole.

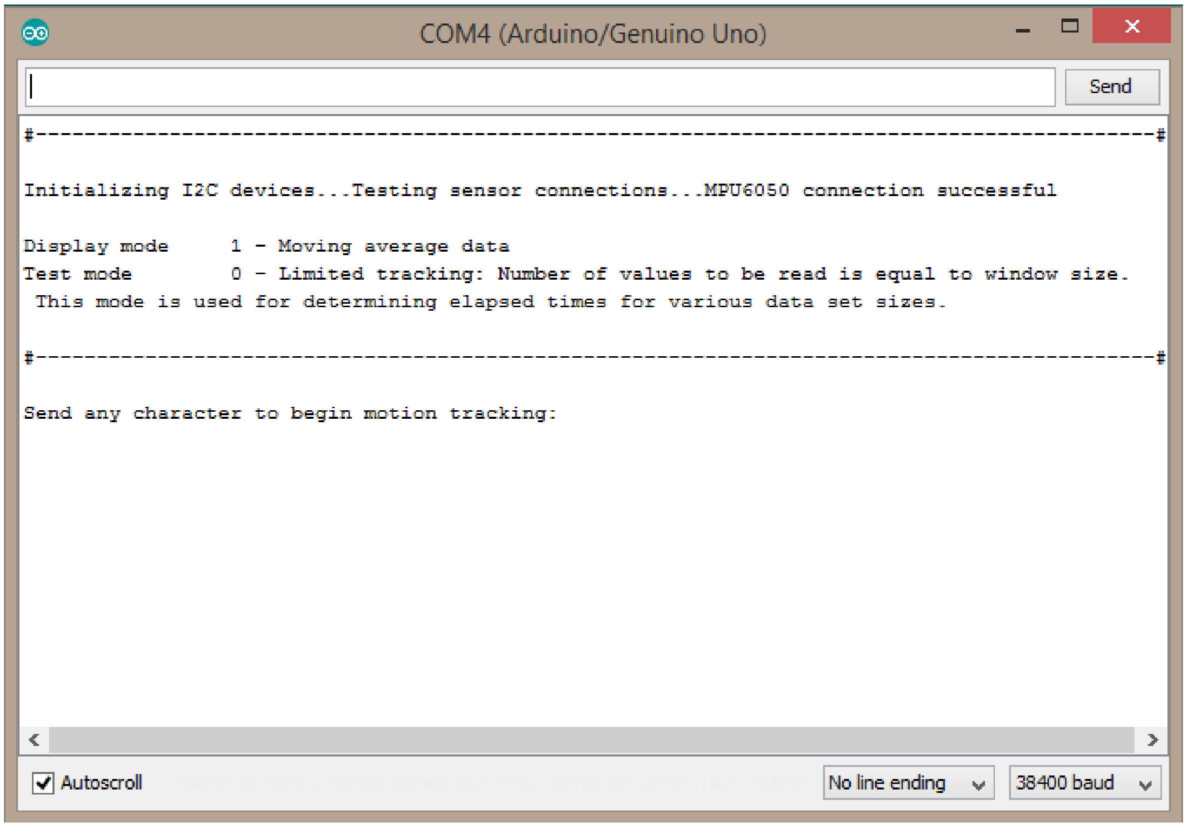
## 5.1. System Function Demonstration

To gain an appreciation of the system development progress thus far, the resulting system interface and output data is presented in the following subsections.

### 5.1.1. Gesture Recognition

The first system output to be demonstrated was the use of the limited tracking mode (`TestMode 0`) along with the running average display mode. Software functionality still needed to be added to allow the switching of modes 'on the fly', which would not only require addition of a mode switching method in software, but also a method of interfacing with the

GR system without personal computer access. In its current state, the software only allows the selection of one mode at any one time, with a requirement for program re-upload if the user wishes to change modes. The limited tracking mode that is set to display running average data required 51% of the 32 kB program storage space and utilised 64% of dynamic memory for global variables, leaving the rest for use with local variables. The initial output when opening the serial monitor is shown in below.

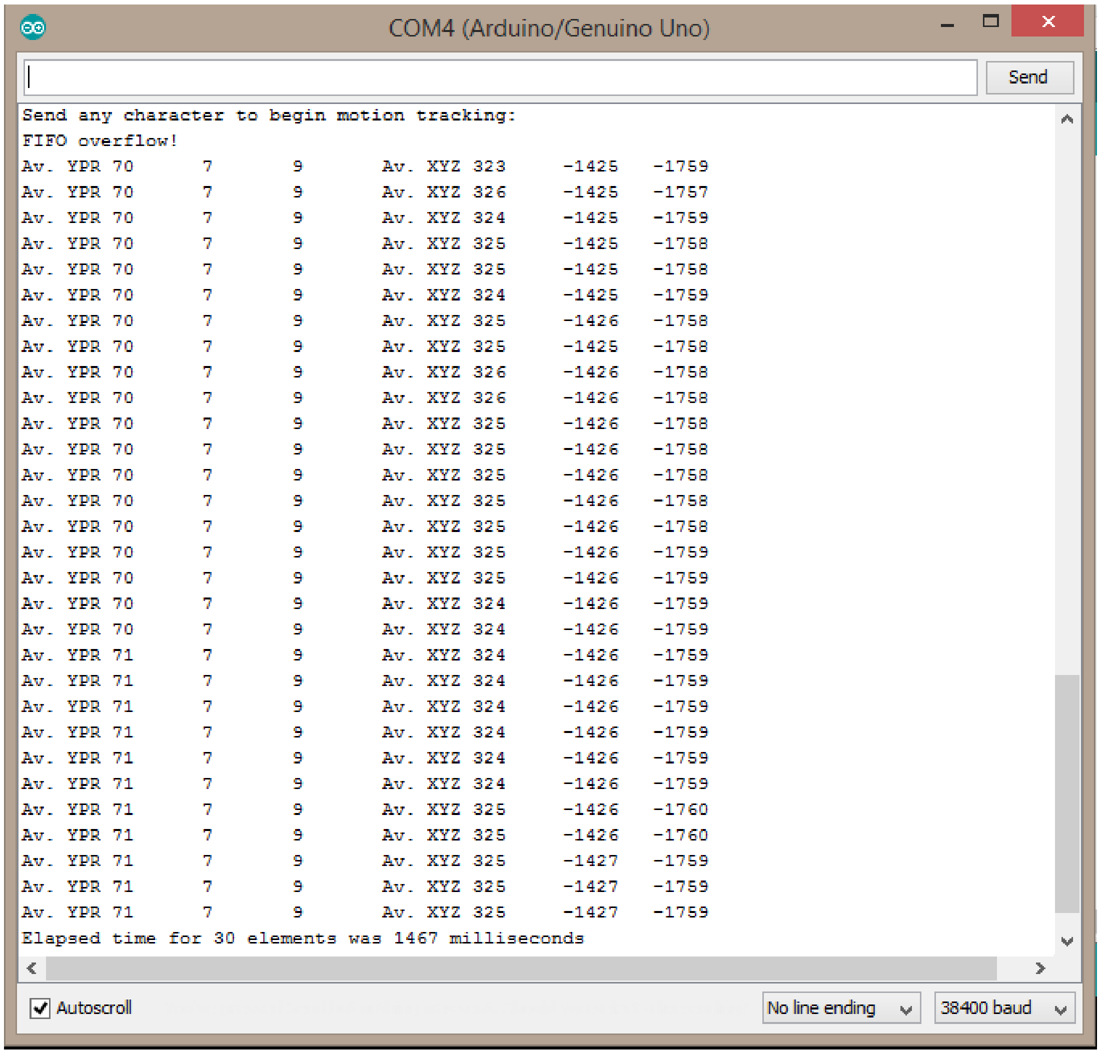


```
#----------------------------------------------------------------------------------------------#

Initializing I2C devices...Testing sensor connections...MPU6050 connection successful

Display mode      1 - Moving average data
Test mode         0 - Limited tracking: Number of values to be read is equal to window size.
 This mode is used for determining elapsed times for various data set sizes.

#----------------------------------------------------------------------------------------------#

Send any character to begin motion tracking:
```

Figure 19 - Limited tracking mode initial output

The output aims to provide a reasonable serial user interface, primarily for development purposes; however, anything much more extensive would require the use of other software such as ‘Processing’.

The serial monitor window below shows a typical running average output sequence in limited tracking mode while the sensor is at rest. The limited tracking output of 30 samples takes approximately 1.5 seconds each time to complete tracking and it is worth noting that the data is seemingly fairly consistent over the window period. When conducting testing with the system in full tracking mode; however, the presence of significant yaw drift was evident and occurred at a rate of approximately one degree per second. This drift could be remedied either by manually decreasing the yaw value by one degree each second, however, this method would not be as reliable as determining the root of the issue.

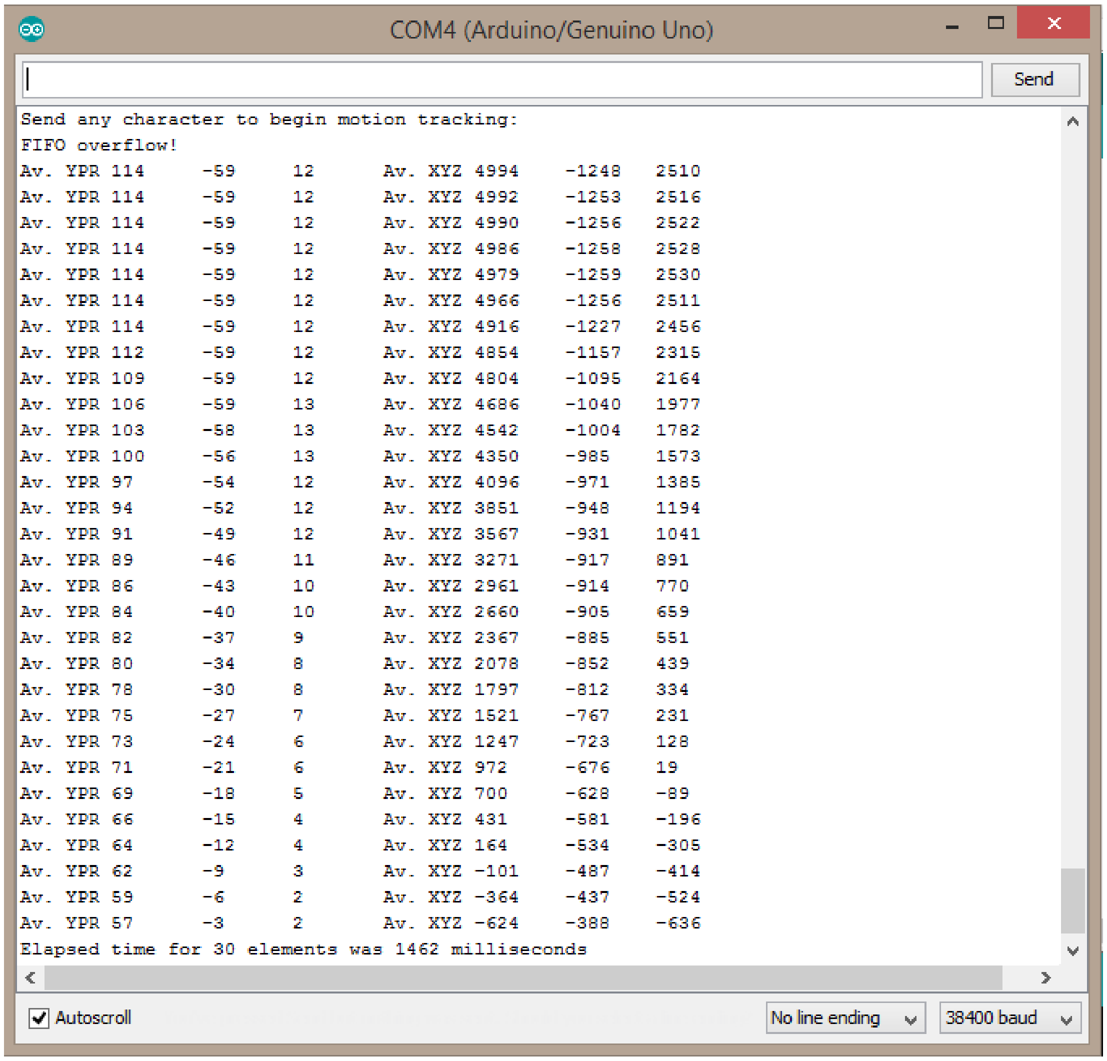


```
Send any character to begin motion tracking:
FIFO overflow!
Av. YPR 70      7       9       Av. XYZ 323     -1425   -1759
Av. YPR 70      7       9       Av. XYZ 326     -1425   -1757
Av. YPR 70      7       9       Av. XYZ 324     -1425   -1759
Av. YPR 70      7       9       Av. XYZ 325     -1425   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1425   -1758
Av. YPR 70      7       9       Av. XYZ 324     -1425   -1759
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1425   -1758
Av. YPR 70      7       9       Av. XYZ 326     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 326     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1758
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1759
Av. YPR 70      7       9       Av. XYZ 325     -1426   -1759
Av. YPR 70      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 70      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 324     -1426   -1759
Av. YPR 71      7       9       Av. XYZ 325     -1426   -1760
Av. YPR 71      7       9       Av. XYZ 325     -1426   -1760
Av. YPR 71      7       9       Av. XYZ 325     -1427   -1759
Av. YPR 71      7       9       Av. XYZ 325     -1427   -1759
Av. YPR 71      7       9       Av. XYZ 325     -1427   -1759
Elapsed time for 30 elements was 1467 milliseconds
```

[✓] Autoscroll    No line ending    38400 baud

Figure 20 - Limited tracking mode resting state

Issues are introduced when attempting to track a movement using the running average filter. The monitor output in *Figure 21* shows an output data stream for a twisting punch movement, with the wrist placed against the hip palm side up and then extended to a forward punch position by rotating the fist. The issue found here was that the running average data took far too long to change significantly, and so the latency between performing the gesture and seeing the resultant change in output was very high.



COM4 (Arduino/Genuino Uno)

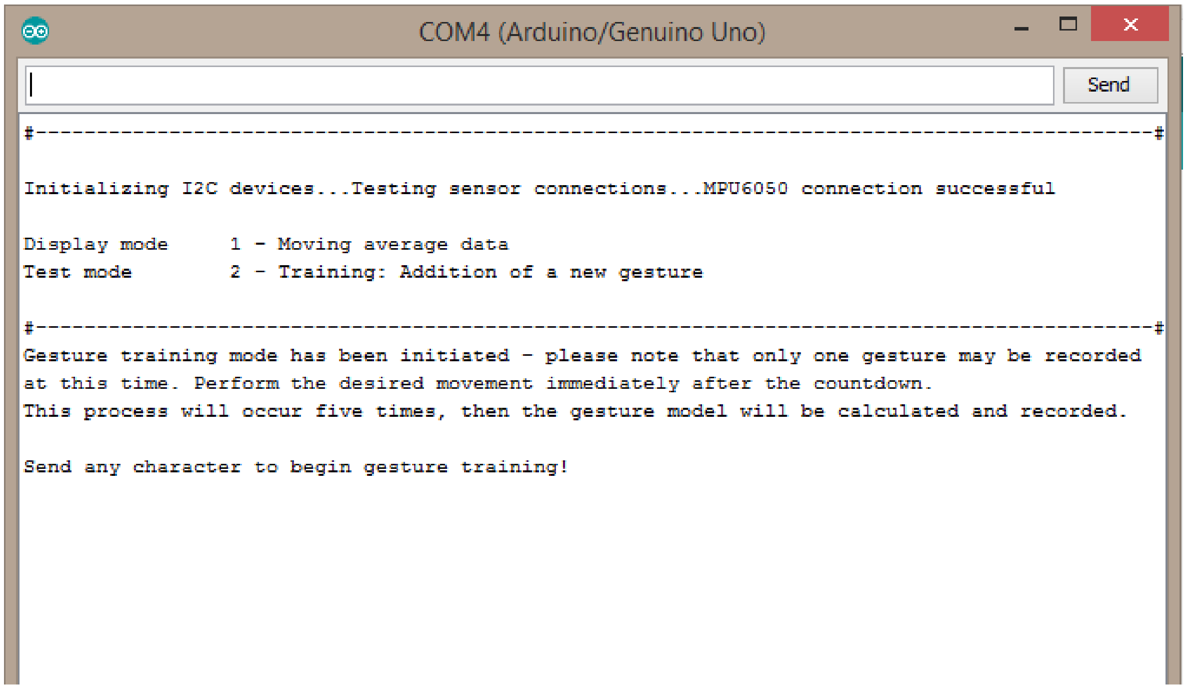
Send

```
Send any character to begin motion tracking:
FIFO overflow!
Av. YPR 114     -59     12      Av. XYZ 4994    -1248   2510
Av. YPR 114     -59     12      Av. XYZ 4992    -1253   2516
Av. YPR 114     -59     12      Av. XYZ 4990    -1256   2522
Av. YPR 114     -59     12      Av. XYZ 4986    -1258   2528
Av. YPR 114     -59     12      Av. XYZ 4979    -1259   2530
Av. YPR 114     -59     12      Av. XYZ 4966    -1256   2511
Av. YPR 114     -59     12      Av. XYZ 4916    -1227   2456
Av. YPR 112     -59     12      Av. XYZ 4854    -1157   2315
Av. YPR 109     -59     12      Av. XYZ 4804    -1095   2164
Av. YPR 106     -59     13      Av. XYZ 4686    -1040   1977
Av. YPR 103     -58     13      Av. XYZ 4542    -1004   1782
Av. YPR 100     -56     13      Av. XYZ 4350    -985    1573
Av. YPR 97      -54     12      Av. XYZ 4096    -971    1385
Av. YPR 94      -52     12      Av. XYZ 3851    -948    1194
Av. YPR 91      -49     12      Av. XYZ 3567    -931    1041
Av. YPR 89      -46     11      Av. XYZ 3271    -917    891
Av. YPR 86      -43     10      Av. XYZ 2961    -914    770
Av. YPR 84      -40     10      Av. XYZ 2660    -905    659
Av. YPR 82      -37     9       Av. XYZ 2367    -885    551
Av. YPR 80      -34     8       Av. XYZ 2078    -852    439
Av. YPR 78      -30     8       Av. XYZ 1797    -812    334
Av. YPR 75      -27     7       Av. XYZ 1521    -767    231
Av. YPR 73      -24     6       Av. XYZ 1247    -723    128
Av. YPR 71      -21     6       Av. XYZ 972     -676    19
Av. YPR 69      -18     5       Av. XYZ 700     -628    -89
Av. YPR 66      -15     4       Av. XYZ 431     -581    -196
Av. YPR 64      -12     4       Av. XYZ 164     -534    -305
Av. YPR 62      -9      3       Av. XYZ -101    -487    -414
Av. YPR 59      -6      2       Av. XYZ -364    -437    -524
Av. YPR 57      -3      2       Av. XYZ -624    -388    -636
Elapsed time for 30 elements was 1462 milliseconds
```

✓ Autoscroll    No line ending    38400 baud

Figure 21 - Limited tracking mode twisting punch data

As for the training mode, the interface is much the same; however, rather than simply prompting the user to begin movement tracking, an explanation of the training procedure is provided with a prompt to begin the countdown before performing the gesture. The countdown counts down from 3, taking four seconds to reach 'Go!', which triggers movement tracking for one window period (approximately 1.5 s) and then saves the resultant trend. Another prompt is then provided, followed by a countdown for the next practice run.



```
#-------------------------------------------------------------------------------------------#

Initializing I2C devices...Testing sensor connections...MPU6050 connection successful

Display mode      1 - Moving average data
Test mode         2 - Training: Addition of a new gesture

#-------------------------------------------------------------------------------------------#
Gesture training mode has been initiated - please note that only one gesture may be recorded
at this time. Perform the desired movement immediately after the countdown.
This process will occur five times, then the gesture model will be calculated and recorded.

Send any character to begin gesture training!
```

Figure 22 - Training mode initial output

The intention was then to use these window trends to produce a final gesture trend, which could then be used for real time matching using correlation. Time constraints did not allow for this implementation, and as such, no actual gesture recognition is able to take place within the system; however, implementation of these features would not require a significant amount of time on top of the program code that had already been produced to complete this dissertation.

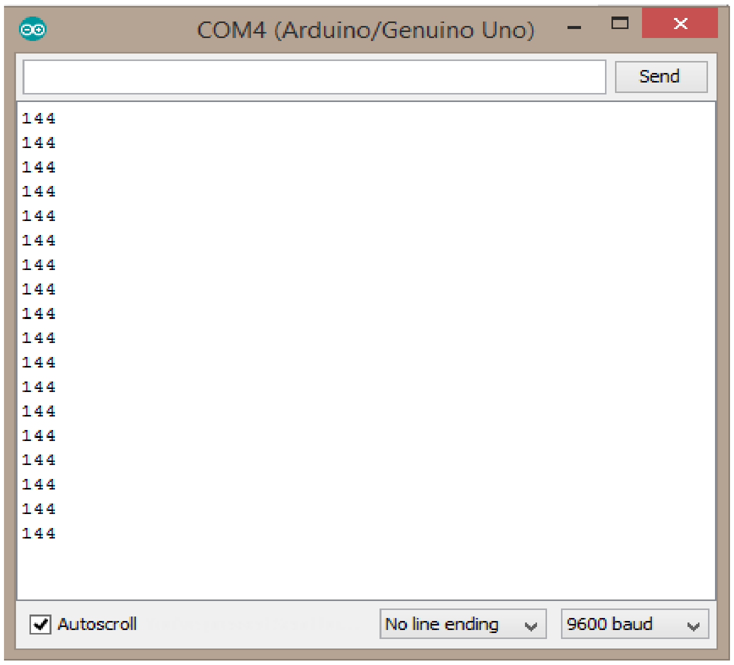
A major issue with the training mode program is the lack of memory present for extensive computations. Upon compilation of the training mode it was evident that the storage memory requirements for all of the necessary variables was becoming an issue. The compiler noted that 75% of dynamic needed to be allocated for global variables, leaving only a small amount to local variables. This huge amount of global variable allocation suggests that SRAM usage could have been substantially reduced if more care was taken in defining variables locally rather than globally, so that largely unused variables are replaced when necessary.

Another cause of this memory issue was the fact that string printed on the serial data line are stored in SRAM, unless specified otherwise by enclosing the quoted string within brackets and placing a leading 'F'. This function causes the string to be stored within flash memory rather than SRAM and helped to reduce the amount of SRAM wastage. SRAM expansion is possible using various SRAM chips such as the 23K256, which can interface with the Arduino via Serial Peripheral Interface (SPI) protocol.

### 5.1.2. Wireless Link Testing

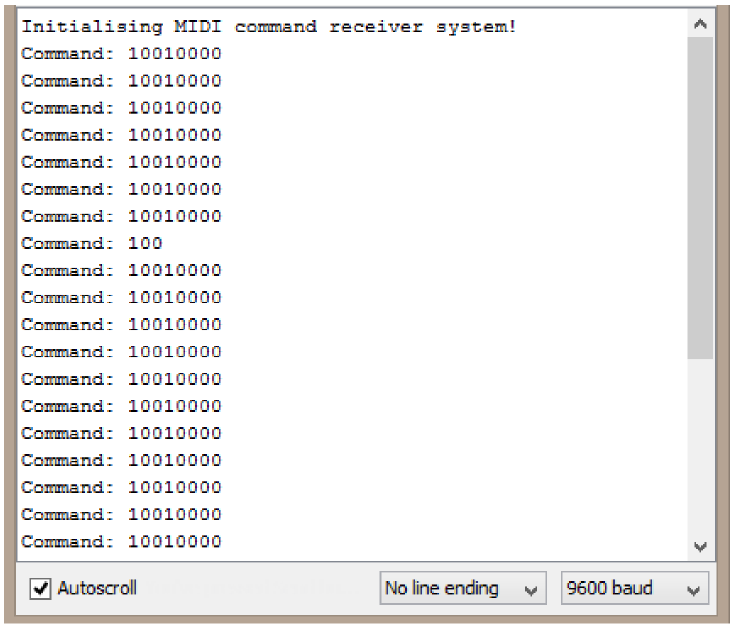
Testing of the wireless data link was a simple process and worked as intended. The transmission and reception programs needed to be uploaded to the Arduino Uno and the Freetronics LeoStick respectively. The LeoStick would then remain connected to the PC and the Uno would be disconnected and attached to the 9 V battery. Transmission began automatically after the Arduino's bootloader sequence and was confirmed by a periodic red LED flash on the transmitter every second. The receiver flashed green at the same rate showing that a connection had been made and that data would be visible on the IDE serial monitor.

The correct transmission value of `144` was confirmed by observing the serial monitor while the Arduino was connected to the PC. Serial data was sent at a baud rate of 9600 bits per second, with the correct output appearing in the monitor each second (*Figure 23*). Reception was also observed to be correct, with the monitor displaying a binary value of `10010000` each second (*Figure 24*).



```
144
144
144
144
144
144
144
144
144
144
144
144
144
144
144
144
144
144
```

Figure 23 - MIDI command transmission in decimal format



```
Initialising MIDI command receiver system!
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 100
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
Command: 10010000
```

Figure 24 - MIDI command reception shown through Arduino serial monitor

## 5.2. Design Issues and System Feasibility

The last stage of the results and testing phase of the project was to assess the current state of the conceived design in relation to project and design objectives and to determine the issues that impeded project efficacy.

### 5.2.1. Current Design Feasibility

In its current state, the system is somewhat lacking due to various design issues. The intended use of the GR system within a performance environment would require would require alterations to the structure of the system to improve its capability of real time external system control. The system design is able to acquire suitable movement data such that there were no issues regarding the sensing system, which has potential that exceeds the abilities of the associated software created in the research project.

The system's ability to process movement data and extract mathematical meaning from that data is minimal when compared to GR systems that utilise much more powerful recognition algorithms and hardware, and could be improved significantly by applying additional computations to determine movement vectors and correlation for vector matching. In addition, reconsideration of the development board used for the project would allow an increase in system complexity.

### 5.2.2. Design Issue Evaluation

The GR system's abilities are impeded by design issues that were not evident until design had already taken place, and thus it is necessary to detail those issues so that they may be taken into account when conducting future work on the system.

The first issue to mention is that of the battery used to supply power to the wearable system. Notably power drain was observed late in the project when diagnosing a malfunctioning transceiver that was attached to the Arduino Uno. Monitoring of the battery voltage during system operation made it apparent that the wearable system was draining the battery at a rate of approximately 0.01 V every few seconds. Problems were observed with functionality once the battery voltage level dropped to roughly 6.5 V. Hence, a 9 V battery connected to the Arduino would not hold a usable voltage for long enough to endure the duration of a performance that lasts more than a few minutes.  While being convenient for use with the Arduino board utilised for this project due to their fast drain were definitely a mitigating factor in the usability of the system.

According to Cybergibbons (2013), the suitability of a 9 V for powering Arduino boards is a common misconception and they are often not recommended. The main reason for this being the inefficiency of the on-board linear voltage regulator, which has a reputation for high power consumption. The use of an external switch-mode voltage regulator circuit to supply the necessary 5 V operating voltage of the Arduino may have allowed extended battery life. The other obvious choice for improving battery life is to opt for a higher capacity battery.

Another issue affecting the ability of the system to fulfil design requirements is that of its minimal algorithmic complexity. Analysis of running average trends was theoretically feasible for simple feature extraction during the design process; however, modern commercial gesture recognition systems such as those investigated within the literature review chapter have multiple significant layers of mathematical complexity, using probability based methods such as hidden markov models for gesture prediction, data correlation for comparison of movement data as well as peak detection, calculation of standard deviation and gestural velocity to further parametrise the movements.

# Chapter 6

# Conclusions and Further Work

## 6.1. Conclusions

The aim of this research project was to investigate the design of a novel GR system using off-the-shelf components, which would create a medium for the augmentation of entertainment oriented performances such as live music shows and theatre productions with an emphasis placed on system reliability, functionality and easy setup for a low price point. The project work involved resulted in the creation of a highly functional hardware system with accompanying enclosure designs and simple gesture recognition software.

Early stages of the project involved research into the current industry state of gesture recognition in terms of both hardware and software elements. Methods of bodily movement tracking, gesture recognition algorithms, MIDI interfacing, data transmission and additive manufacturing were investigated in depth to gain an appreciation of typical system elements. Review of commercial gesture recognition systems, as well as those in research and development was conducted to assist in defining the scope of the project and to develop realistic project goals.

Research methodology was developed for the project was developed by considering qualitative and quantitative research techniques and a project task plan was developed to ensure that all important aspects of engineering design were covered. A timeline was then created to assist in the time keeping aspects of the project and to provide guidelines for the importance of each project section. Assessment of consequential effects involved taking potential safety, ethics and sustainability issues into account to minimise any adverse circumstances that may have arisen over the course of the project, or following its completion.

System design involved selection of appropriate hardware components, design of hardware enclosures and wearable mounting gear, and finally the development of suitable software for performing gesture recognition and creating a wirelessly transmittable instruction based on the MIDI protocol. The resulting hardware system consisted of an acceleration and rotational velocity sensor, Arduino compatible development boards and radio frequency transceivers that complied with hardware requirements.

Enclosure design involved using OpenSCAD software to design enclosures tailored to the chosen hardware. A prototype of the sensor wrist brace was created for development; however, the main enclosure design remained simply that due to time constraints and nuances involved in the 3D printing process.

An attempt was made at creating running average based gesture recognition software that could be self-contained within the Arduino Uno board, but proved to require further complexity to be able to perform the functions that were outlined in the software requirements. The last design step involved producing software that would allow transmission of a MIDI instruction from the wearable system to the PC situated receiver.

The functionality of the system was reviewed after testing and suggestions were made regarding remedies to design issues that arose late in the piece. The current state of the design was also evaluated. In summary, a simple system was created that met, or at least partially met each of the project objectives outlined at the beginning of the project.

## 6.2. The Learning Experience

The most important end result of this project's completion has been a vast attainment of knowledge that, for the most part, had not been presented within courses completed as part of the Bachelor of Engineering program. This knowledge gained throughout the project included, but was not limited to:

- Arduino based hardware and software development.
- Inertial measurement based data acquisition systems.
- Practical usage of radio frequency communications hardware.
- 3D CAD design for parts produced through additive manufacturing.
- Gesture recognition techniques and algorithms

In choosing to carry out my own research project I brought it upon myself to create a challenge in the sense of learning to work with multiple types of technology that I had not previously worked with in the slightest. In doing so, I have gained extremely advantageous knowledge of the aforementioned subject areas, which are of great interest to me and are becoming increasingly relevant.

The vast majority of projects offered by the faculty did not feel suitable for my interests as I wished to incorporate research interests that I am particularly passionate about into my project work, such as the use of electronics to increase the fluidity of human-machine interaction, in addition to incorporating my hobby-based interest in music production.

This project has been an interesting journey and although much work was put into creating the GR system, the outcome is lacklustre to some extent. Due to this fact, I will endeavour to continue my development of the GR system following the completion of the research project, so that I may eventually produce a system capable of adding personalisation and an interesting new medium with which to express my musical interests.

## 6.3. Future Work and Potential Uses

The completion of this research project has revealed that further design of new system elements and re-design of certain existing elements would be fruitful in attaining an end product that has expansive practical uses.

Future work on this current system would include:

- Re-structuring of program code to improve the system's overall efficiency and to create a more modular approach to the problem.
- The creation of computer software that would allow easy interfacing
- Consideration of a controller that is more capable in relation to processing power and available memory without increasing physical footprint and without sacrificing the low cost nature of the system.
- Selection of a more adequate power supply for the wearable system with a higher capacity and stable voltage and current levels.
- Consideration of alternate recognition algorithms to provide a user with more options for applications.

Potential uses for the system in a more refined state and potentially with other modifications include:

- Applying the system to a performance as a GR based MIDI controller with various movements triggering virtual instrument samples at various velocities, or even the selection and control of audio filters.
- Using the system to control DMX stage lighting through performer movements, reducing the need for manual lighting control.
- Using the system to control other technical aspects of a performance such as mixer channel gain, in-ear monitor volume and/or instrument volume.

# References

Anderton, C 1986, *MIDI for Musicians*, Amsco Publications, New York.

Azad, R, Azad, B, Khalifa, NB & Jamali, S 2014, 'Real-time Human-computer Interaction Based on Face and Hand Gesture Recognition', *International Journal in Foundations of Computer Science & Technology (IJFCST)*, vol 4, no. 4, pp. 37-48.

Benbasat, A & Paradiso, J 2001, 'An Inertial Measurement Framework for Gesture Recognition and Applications', *Gesture and Sign Language in Human-machine Interaction*, pp. 9-20.

Benbasat, A & Paradiso, J 2001, 'Compact, Configurable Inertial Gesture Recognition', *Extended Abstracts on Human Factors in Computing Systems*, pp. 183-184.

Berman, S & Stern, H 2012, 'Sensors for Gesture Recognition Systems', *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol 42, no. 3, pp. 277-290.

Borrego, M, Douglas, E & Amelink, C 2009, 'Quantitative, Qualitative, and Mixed Research Methods in Engineering Education', *Journal of Engineering Education*, vol 98, no. 1, pp. 53-66.

Camurri, A, Mazzarino, B, Ricchetti, M, Timmers, R & Volpe, G 2003, 'Multimodal Analysis of Expressive Gesture in Music and Dance Performances', *International Gesture Workshop*, Springer Publishing.

Chinmaya, KT, Amal, S, Aswath, S & Ganesha, U 2014, 'Human Gesture Recognition for Real-Time Control of Humanoid Robot', *International Journal of Advances in Mechanical & Automobile Engineering (IJAMAE)*, vol 1, no. 1, pp. 96-100.

Clay, A, Couture, N, Nigay, L, De La Riviere, J-B, Martin, J-C, Courgeon, M, Desainte-Catherine, M, Orvain, E, Girondel, V & Domenger, G 2012, 'Interactions and Systems for Augmenting a Live Dance Performance', *International Symposium on Mixed and Augmented Reality (ISMAR-AMH)*, pp. 29-38.

CyberGibbons 2013, *Arduino misconceptions 6: a 9V battery is a good power source*, viewed October 2016, <cybergibbons.com/uncategorized/arduino-misconceptions-6-a-9v-battery-is-a-good-power-source/>.

Fang, Y, Wang, K, Cheng, J & Lu, H 2007, 'A Real-Time Hand Gesture Recognition Method', *2007 IEEE International Conference on Multimedia and Expo*, pp. 995-998.

Gupta, N, Singh, R & Bhatia, S 2014, 'Hand Gesture Recognition using Ultrasonic Sensor and ATmega128 Microcontroller', *International Journal of Research in Engineering and Technology (IJRET)*, vol 3, no. 6.

I2C Info 2016, *I2C Info - I2C Bus, Interface and Protocol*, viewed Sep 2016, <http://i2c.info>.

Jessop, EN 2010, 'A Gestural Media Framework: Tools for Expressive Gesture Recognition Method', Massachusetts Institute of Technology.

Kintel, M 2016, *About OpenSCAD*, viewed Sep 2016, <http://www.openscad.org/about.html>.

Kumar, AEA 2010, 'Human Computer Interface Using EMG Signals: Hand Gesture Based Manipulator Control', Amrita School of Engineering, Coimbatore.

LaViola, J 1999, 'A Survey of Hand Posture and Gesture Recognition Techniques and Technology', Department of Computer Science, Brown University, Rhode Island.

Leroi-Gourhan, A 1993, *Gesture and Speech*, MIT Press.

Liu, J, Pan, Z & Li, X 2010, 'An Accelerometer-based Gesture Recognition Algorithm and its Application for 3D Interaction', *Comput. Sci. Inf. Syst.*, vol 7, no. 1, pp. 177-188.

Lyons, KEA 2007, 'GART: The Gesture and Activity Recognition Toolkit', Springer Publishing, USA.

Market Reserach Man, LLC. 2015, *Quantitative vs. Qualitative Research: What's the Difference?*, viewed July 2016, <http://www.mymarketresearchmethods.com/quantitative-vs-qualitative-research-whats-the-difference/>.

Myers, BA 1998, 'A Brief History of Human-machine Interaction Technology', *Interactions*, vol 5, no. 2, pp. 44-54.

Nallaperumal, K 2013, *Engineering Research Methodology A Computer Science and Engineering and Information and Communication Technologies Perspective.*

Parab, PEA 2014, 'Hand Gesture Recognition Using Microcontroller & Flex Sensor', *International Journal of Scientific Research and Education*, vol 2, no. 3, pp. 518-522.

Patel, B, Shah, V & Kshirsagar, R 2011, 'Microcontroller Based Gesture Recognition System for the Handicap People', *Proceedings of Journal of Engineering Research and Studies, Surat, India*, 2011.

Predko, M 1998, *Handbook of Microcontrollers*, McGraw-Hill, New York, USA.

Simmonds, A 1997, *Data Communications and Transmission Principles*, MacMillan Press Ltd, Great Britain.

Sjuve, E 2008, 'Gestures, Interfaces and Other Secrets of the Stage', in *Transdisciplinary Digital Art. Sound, Vision and the New Screen*, Springer Publishing, USA.

Starner, T, Weaver, J & Pentland, A 1998, 'Real-time American Sign Language Recognition Using Desk and Wearable Computer Based Video', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 20, no. 12, pp. 1371-1375.

Statista Inc. 2016, *Number of smartphone users worldwide from 2014 to 2019 (in millions)*, viewed 15 June 2016, <http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>.

Suhas, S & Dileep, MK 2015, 'Gesture Controlled User Interface Using Inertial Measurement Unit', *International Journal of Engineering Research and Technology*, vol 4, no. 5.

Wheeler, KR, Chang, MH & Knuth, KH 2006, 'Gesture-based Control and EMG Decomposition', *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol 36, no. 4, pp. 503-514.

Wikipedia, TFE 2016, *Coriolis force*, viewed July 2016, <https://en.wikipedia.org/wiki/Coriolis_force>.

Xu, R, Zhou, S & Wen, L 2012, 'MEMS Accelerometer Based Nonspecific-user Hand Gesture Recognition', *IEEE Sensors Journal*, vol 12, no. 5, pp. 1166-1173.

# Appendix A

# Project Specification

ENG4111/4112 Research Project

**Project Specification**

| | |
|---|---|
| For: | Sathya Smith |
| Title: | Utilisation of wearable actuators and gesture recognition to augment live stage performances |
| Major: | Electrical/Electronic Engineering |
| Supervisors: | Andrew Maxwell |
| Enrolment: | ENG4111 – ONC S1, 2016<br>ENG4112 – ONC S2, 2016 |
| Project Aim: | To develop an efficient and reliable system for recognising gestures created by the human body, with the aim of determining the feasibility of using such a system as a tool in live stage performances. |
| **Programme:** | **Issue B, 16th March 2016** |

1. Review existing gesture control systems, as well as systems used to control various elements of live stage performance.

2. Research relevant data logging and gesture capture methods and software.

3. Research and select suitable microcontroller/s, WiFi/Radio transmission module/s and sensors to be potentially used in the physical wearable sensing system design.

4. Prepare a suggested design for the physical wearable sensing system to be suitable for interfacing with the computer-based system.

5. Prepare a suggested design for the PC based system implementing a gesture capture method for basic performance gestures.

6. Investigate methods of increasing the reliability of captured gesture data.

7. Investigate and implement methods of outputting gestures as MIDI commands.

*If time and resources permit:*

8. Produce a simple plugin to allow the system to be used in a digital audio workstation and demonstrate potential uses.

# Appendix B

# Project Timeline

## Smith_S_Maxwell_Timeline

| Background Reserach & | 0% | | Start | Due | Assigned |
|---|---|---|---|---|---|
| Develop understanding of problem | 0% | 🟦 | Mar 8, 2016 | Mar 15, 2016 | |
| Seek literature | 0% | 🟦 | Mar 15, 2016 | Mar 29, 2016 | |
| Conduct literature review | 0% | 🟦 | Mar 29, 2016 | Apr 21, 2016 | |
| Develop project methodology | 0% | 🟦 | Apr 21, 2016 | May 12, 2016 | |
| Complete preliminary report | 0% | 🟦 | May 12, 2016 | May 25, 2016 | |
| **Hardware Design** | **0%** | | **Start** | **Due** | **Assigned** |
| Consult additional hardware | 0% | 🟥 | May 25, 2016 | Jun 3, 2016 | |
| Decide on components to be used | 0% | 🟥 | Jun 3, 2016 | Jun 17, 2016 | |
| Outline specifications and plan for | 0% | 🟥 | Jun 17, 2016 | Jun 24, 2016 | |
| Evaluate efficacy of chosen | 0% | 🟥 | Jun 24, 2016 | Jul 1, 2016 | |
| Begin designing device enclosures | 0% | 🟥 | Jul 1, 2016 | Jul 12, 2016 | |
| **Software Design** | **0%** | | **Start** | **Due** | **Assigned** |
| Develop general model of system | 0% | 🟩 | Jul 12, 2016 | Jul 19, 2016 | |
| Outline real-time data analysis | 0% | 🟩 | Jul 19, 2016 | Jul 26, 2016 | |
| Develop algorithmic plan for GRS | 0% | 🟩 | Jul 26, 2016 | Aug 15, 2016 | |
| **Build & Prototyping** | **0%** | | **Start** | **Due** | **Assigned** |
| Conduct hardware assembly | 0% | 🟪 | Jul 12, 2016 | Jul 19, 2016 | |
| Print enclosures and fit hardware | 0% | 🟪 | Jul 19, 2016 | Jul 26, 2016 | |
| Conduct initial sensor testing | 0% | 🟪 | Jul 26, 2016 | Jul 30, 2016 | |
| Implement GR program | 0% | 🟪 | Jul 30, 2016 | Aug 20, 2016 | |
| Conduct recognition testing | 0% | 🟪 | Aug 20, 2016 | Aug 27, 2016 | |
| Re-evaluate design efficacy | 0% | 🟪 | Aug 27, 2016 | Sep 4, 2016 | |
| Re-design elements if required | 0% | 🟪 | Aug 27, 2016 | Oct 1, 2016 | |
| **Results & Design evaluation** | **0%** | | **Start** | **Due** | **Assigned** |
| Analyse results of testing in | 0% | 🟧 | Sep 4, 2016 | Sep 20, 2016 | |
| Prepare results to present in | 0% | 🟧 | Sep 20, 2016 | Sep 27, 2016 | |
| Discuss results | 0% | 🟧 | Sep 27, 2016 | Oct 1, 2016 | |

| Documentation | 0% | | Start | Due | Assigned |
|---|---|---|---|---|---|
| Dissertation write-up | 0% | | Aug 15, 2016 | Oct 13, 2016 | |
| Partial dissertation draft due | 0% | | Sep 7, 2016 | Sep 7, 2016 | |
| Final dissertation due | 0% | | Oct 13, 2016 | Oct 13, 2016 | |

# Appendix C

# Risk Assessment

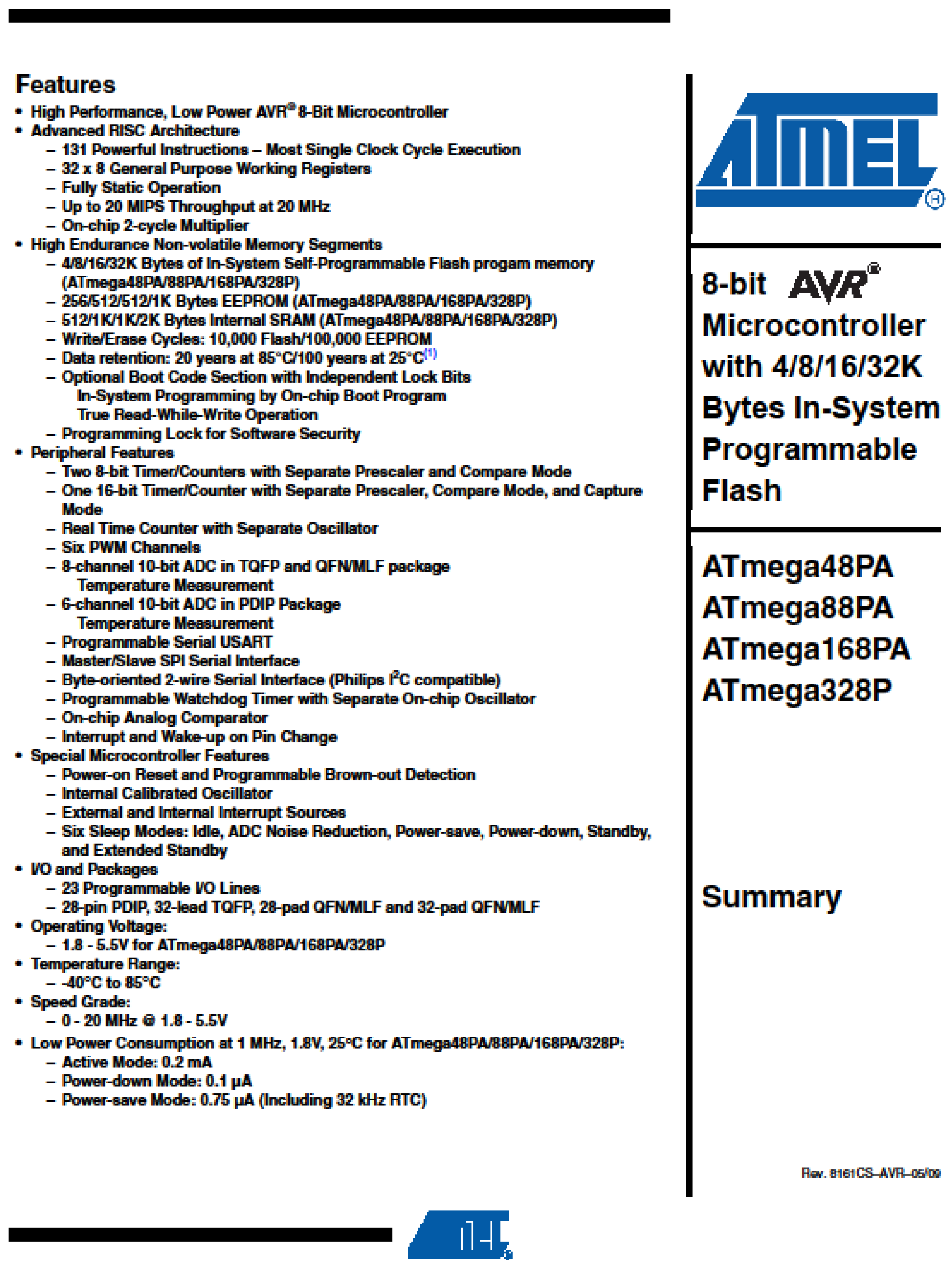
Table 9 - Project Risk Assessment

| Identified Hazard | Significance of risk | Likelihood of exposure | Consequences | Control measures |
|---|---|---|---|---|
| *Risk of electric shock from power supplies* | High – 240V AC mains electrocution can cause extensive bodily damage and can be potentially fatal. | Very low - the only power supplies to be used was a 9V battery and a laptop charger. | Potential for severe bodily damage in the event of shock from mains supply. | Ensure caution is taken when connecting the laptop charger to 240V mains supply. |
| *Risk of damage to electronic equipment* | Moderate – damage to project equipment would extend time required for completion | Moderate – A great deal of handling of equipment is required throughout the project | Damage to equipment required for project completion, which would need replacing. | Double check when connecting power to a piece of equipment and make sure all instructions for use are followed. |
| *Risk of eye strain* | Moderate – Much of the project requires significant computer usage | High – Looking at computer screens for extended periods of time while word processing etc. | Eye fatigue or soreness as well as the possibility of headaches or blurred vision | Take regular breaks when working with screens to allow eyes to readjust |

Appendix D

# Component Datasheets

## D.1. ATMega328 Datasheet (page 1)



**Features**

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips $I^2C$ compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

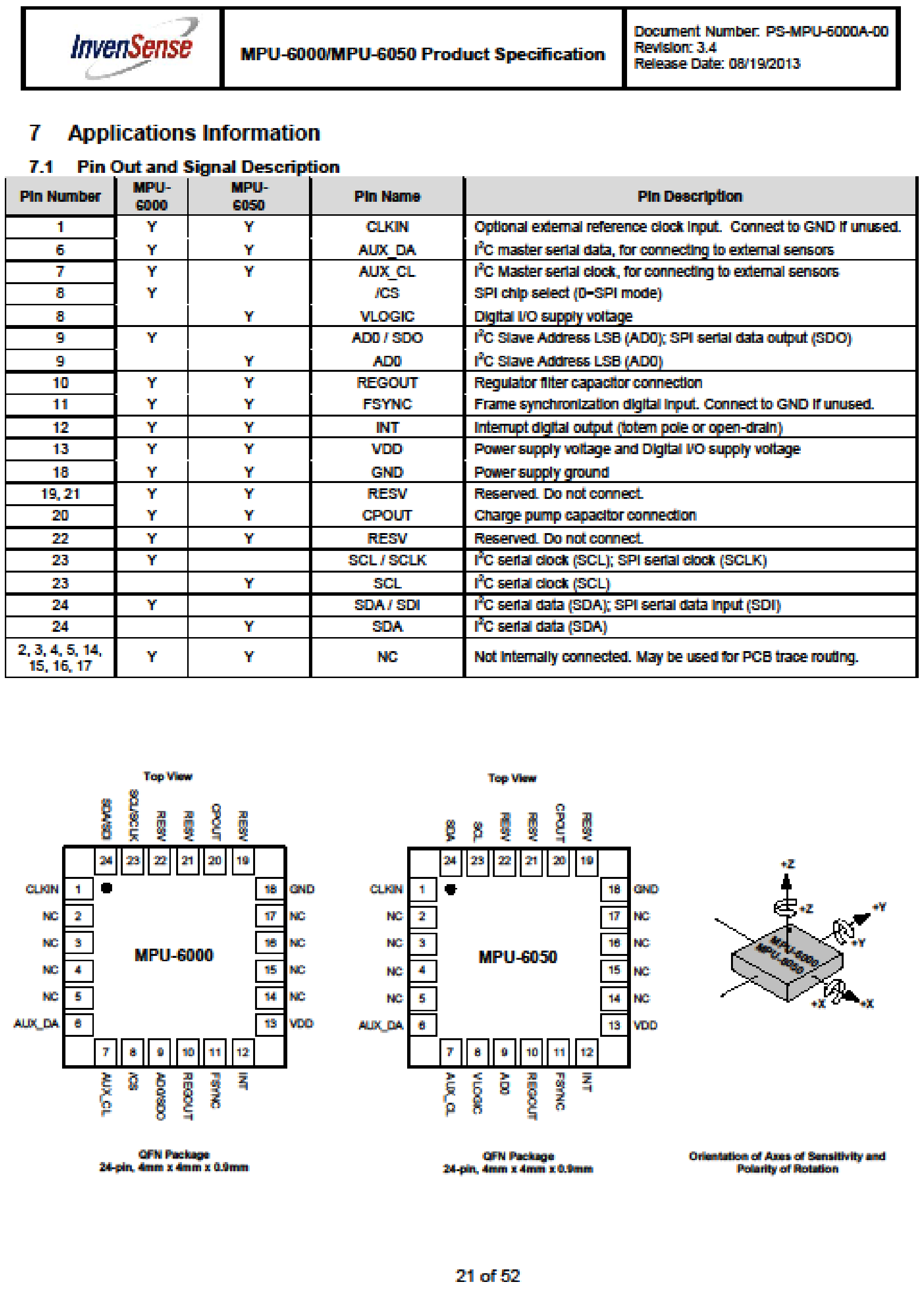
8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Summary

Rev. 8161CS–AVR–05/09

## D.2. MPU6050 Pin Descriptions

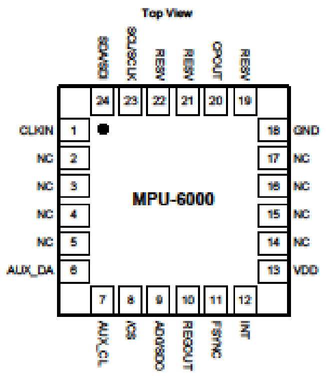


| InvenSense | MPU-6000/MPU-6050 Product Specification | Document Number: PS-MPU-6000A-00<br>Revision: 3.4<br>Release Date: 08/19/2013 |
|---|---|---|

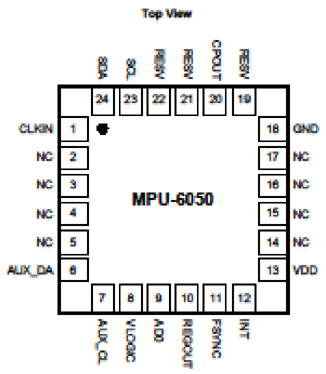
### 7 Applications Information

#### 7.1 Pin Out and Signal Description

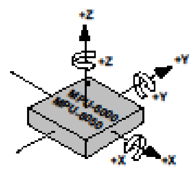
| Pin Number | MPU-6000 | MPU-6050 | Pin Name | Pin Description |
|---|---|---|---|---|
| 1 | Y | Y | CLKIN | Optional external reference clock input. Connect to GND if unused. |
| 6 | Y | Y | AUX_DA | I$^2$C master serial data, for connecting to external sensors |
| 7 | Y | Y | AUX_CL | I$^2$C Master serial clock, for connecting to external sensors |
| 8 | Y | | /CS | SPI chip select (0=SPI mode) |
| 8 | | Y | VLOGIC | Digital I/O supply voltage |
| 9 | Y | | AD0 / SDO | I$^2$C Slave Address LSB (AD0); SPI serial data output (SDO) |
| 9 | | Y | AD0 | I$^2$C Slave Address LSB (AD0) |
| 10 | Y | Y | REGOUT | Regulator filter capacitor connection |
| 11 | Y | Y | FSYNC | Frame synchronization digital input. Connect to GND if unused. |
| 12 | Y | Y | INT | Interrupt digital output (totem pole or open-drain) |
| 13 | Y | Y | VDD | Power supply voltage and Digital I/O supply voltage |
| 18 | Y | Y | GND | Power supply ground |
| 19, 21 | Y | Y | RESV | Reserved. Do not connect. |
| 20 | Y | Y | CPOUT | Charge pump capacitor connection |
| 22 | Y | Y | RESV | Reserved. Do not connect. |
| 23 | Y | | SCL / SCLK | I$^2$C serial clock (SCL); SPI serial clock (SCLK) |
| 23 | | Y | SCL | I$^2$C serial clock (SCL) |
| 24 | Y | | SDA / SDI | I$^2$C serial data (SDA); SPI serial data input (SDI) |
| 24 | | Y | SDA | I$^2$C serial data (SDA) |
| 2, 3, 4, 5, 14, 15, 16, 17 | Y | Y | NC | Not internally connected. May be used for PCB trace routing. |



QFN Package
24-pin, 4mm x 4mm x 0.9mm



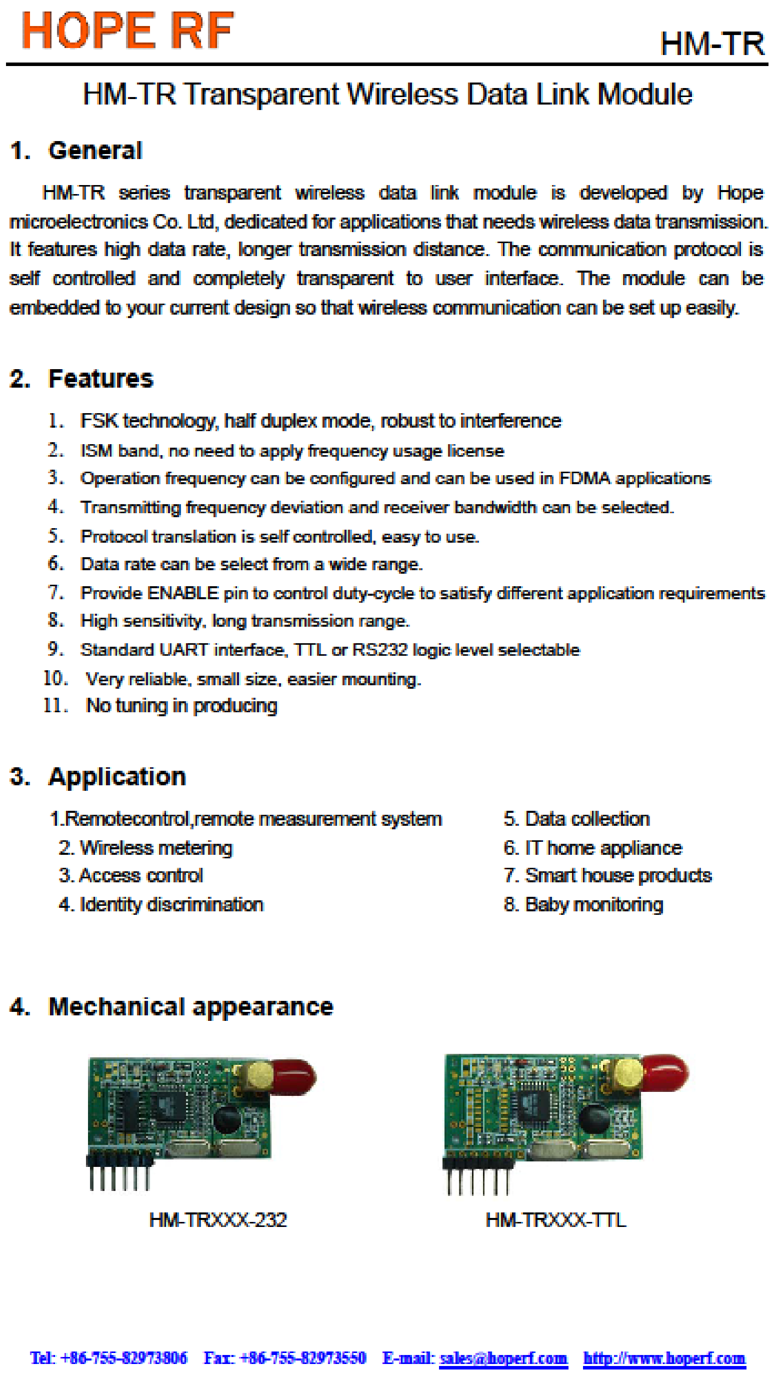
QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity and Polarity of Rotation

21 of 52

## D.3. HopeRF HM-TR Datasheet (page 1)



HOPE RF HM-TR

HM-TR Transparent Wireless Data Link Module

**1. General**

HM-TR series transparent wireless data link module is developed by Hope microelectronics Co. Ltd, dedicated for applications that needs wireless data transmission. It features high data rate, longer transmission distance. The communication protocol is self controlled and completely transparent to user interface. The module can be embedded to your current design so that wireless communication can be set up easily.

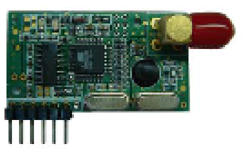
**2. Features**

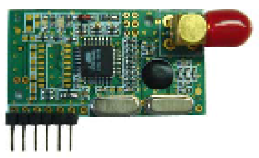
1. FSK technology, half duplex mode, robust to interference
2. ISM band, no need to apply frequency usage license
3. Operation frequency can be configured and can be used in FDMA applications
4. Transmitting frequency deviation and receiver bandwidth can be selected.
5. Protocol translation is self controlled, easy to use.
6. Data rate can be select from a wide range.
7. Provide ENABLE pin to control duty-cycle to satisfy different application requirements
8. High sensitivity, long transmission range.
9. Standard UART interface, TTL or RS232 logic level selectable
10. Very reliable, small size, easier mounting.
11. No tuning in producing

**3. Application**

| | |
|---|---|
| 1.Remotecontrol,remote measurement system | 5. Data collection |
| 2. Wireless metering | 6. IT home appliance |
| 3. Access control | 7. Smart house products |
| 4. Identity discrimination | 8. Baby monitoring |

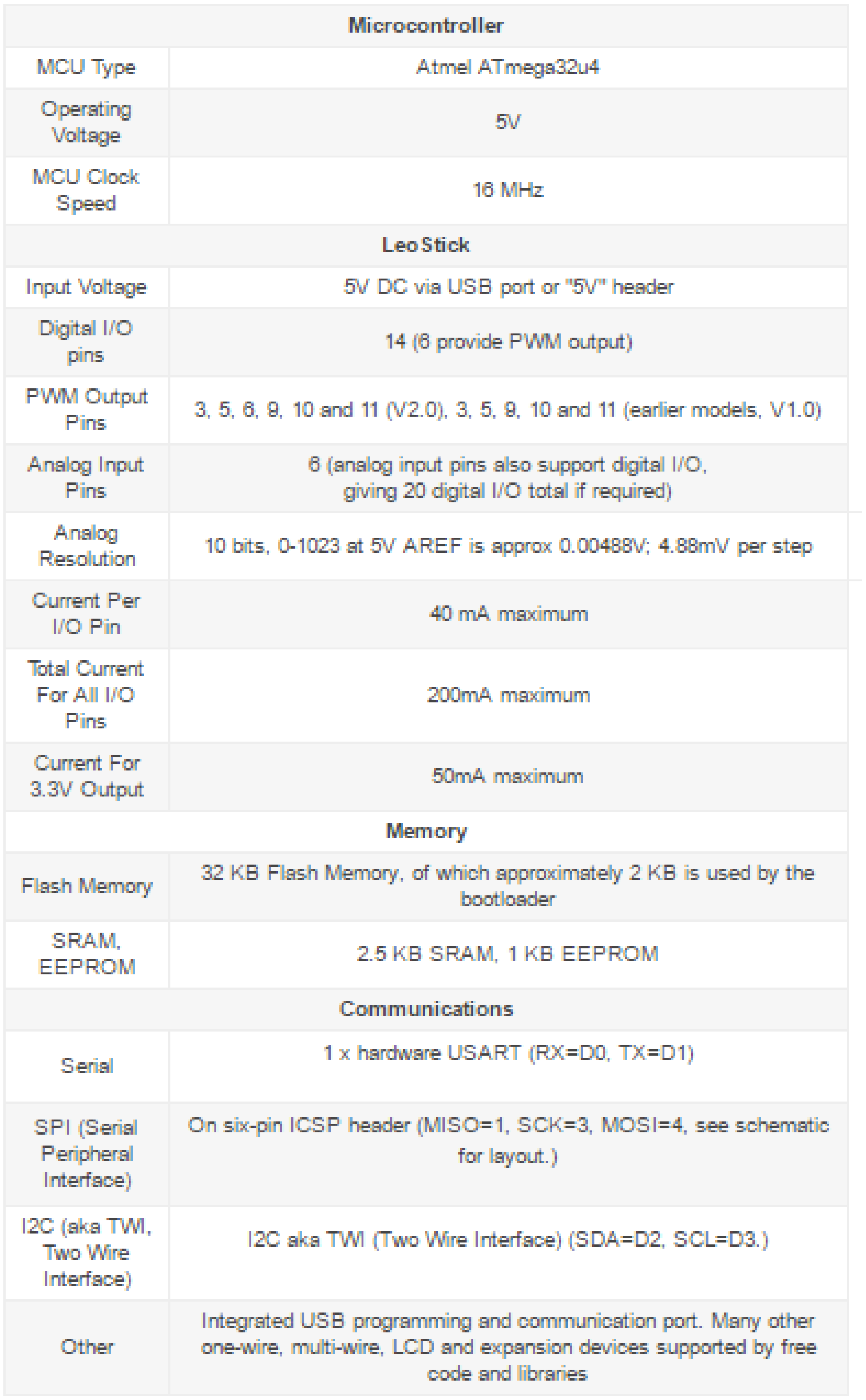
**4. Mechanical appearance**


HM-TRXXX-232


HM-TRXXX-TTL

Tel: +86-755-82973806 Fax: +86-755-82973550 E-mail: sales@hoperf.com http://www.hoperf.com

## D.4. Freetronics LeoStick Specifications



| | |
|---|---|
| **Microcontroller** | |
| MCU Type | Atmel ATmega32u4 |
| Operating Voltage | 5V |
| MCU Clock Speed | 16 MHz |
| **LeoStick** | |
| Input Voltage | 5V DC via USB port or "5V" header |
| Digital I/O pins | 14 (6 provide PWM output) |
| PWM Output Pins | 3, 5, 6, 9, 10 and 11 (V2.0), 3, 5, 9, 10 and 11 (earlier models, V1.0) |
| Analog Input Pins | 6 (analog input pins also support digital I/O, giving 20 digital I/O total if required) |
| Analog Resolution | 10 bits, 0-1023 at 5V AREF is approx 0.00488V; 4.88mV per step |
| Current Per I/O Pin | 40 mA maximum |
| Total Current For All I/O Pins | 200mA maximum |
| Current For 3.3V Output | 50mA maximum |
| **Memory** | |
| Flash Memory | 32 KB Flash Memory, of which approximately 2 KB is used by the bootloader |
| SRAM, EEPROM | 2.5 KB SRAM, 1 KB EEPROM |
| **Communications** | |
| Serial | 1 x hardware USART (RX=D0, TX=D1) |
| SPI (Serial Peripheral Interface) | On six-pin ICSP header (MISO=1, SCK=3, MOSI=4, see schematic for layout.) |
| I2C (aka TWI, Two Wire Interface) | I2C aka TWI (Two Wire Interface) (SDA=D2, SCL=D3.) |
| Other | Integrated USB programming and communication port. Many other one-wire, multi-wire, LCD and expansion devices supported by free code and libraries |

Appendix E

# MIDI Summary Table

| MIDI 1.0 Specification Message Summary (Sourced from The MIDI Association) | | |
|---|---|---|
| **Status**<br>**D7----D0** | **Data Byte(s)**<br>**D7----D0** | **Description** |
| Channel Voice Messages [nnnn = 0-15 (MIDI Channel Number 1-16)] | | |
| 1000nnnn | 0kkkkkkk<br>0vvvvvvv | Note Off event. This message is sent when a note is released (ended). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity. |
| 1001nnnn | 0kkkkkkk<br>0vvvvvvv | Note On event. This message is sent when a note is depressed (start). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity. |
| 1010nnnn | 0kkkkkkk<br>0vvvvvvv | Polyphonic Key Pressure (Aftertouch). This message is most often sent by pressing down on the key after it "bottoms out". (kkkkkkk) is the key (note) number. (vvvvvvv) is the pressure value. |
| 1011nnnn | 0ccccccc<br>0vvvvvvv | Control Change. This message is sent when a controller value changes. Controllers include devices such as pedals and levers. Controller numbers 120-127 are reserved as "Channel Mode Messages" (below). (ccccccc) is the controller number (0-119). (vvvvvvv) is the controller value (0-127). |
| 1100nnnn | 0ppppppp | Program Change. This message sent when the patch number changes. (ppppppp) is the new program number. |
| 1101nnnn | 0vvvvvvv | Channel Pressure (After-touch). This message is most often sent by pressing down on the key after it "bottoms out". This message is different from polyphonic after-touch. Use this message to send the single greatest pressure value (of all the current depressed keys). (vvvvvvv) is the pressure value. |
| 1110nnnn | 0lllllll<br>0mmmmmmm | Pitch Bend Change. 0mmmmmmm This message is sent to indicate a change in the pitch bender (wheel or lever, typically). The pitch bender is measured by a fourteen bit value. Center (no pitch change) is 2000H. Sensitivity is a function of the transmitter. (llllll) are the least significant 7 bits. (mmmmmm) are the most significant 7 bits. |
| Channel Mode Messages (See also Control Change, above) | | |

| | | |
|---|---|---|
| 1011nnnn | 0ccccccc<br>0vvvvvvv | Channel Mode Messages. This the same code as the Control Change (above), but implements Mode control and special message by using reserved controller numbers 120-127. The commands are: |
| | | All Sound Off. When All Sound Off is received all oscillators will turn off, and their volume envelopes are set to zero as soon as possible. c = 120, v = 0: All Sound Off |
| | | Reset All Controllers. When Reset All Controllers is received, all controller values are reset to their default values. (See specific Recommended Practices for defaults).<br>c = 121, v = x: Value must only be zero unless otherwise allowed in a specific Recommended Practice. |
| | | Local Control. When Local Control is Off, all devices on a given channel will respond only to data received over MIDI. Played data, etc. will be ignored. Local Control On restores the functions of the normal controllers. c = 122, v = 0: Local Control Off<br>c = 122, v = 127: Local Control On |
| | | All Notes Off. When an All Notes Off is received, all oscillators will turn off. c = 123, v = 0: All Notes Off (See text for description of actual mode commands.) c = 124, v = 0: Omni Mode Off<br>c = 125, v = 0: Omni Mode On<br>c = 126, v = M: Mono Mode On (Poly Off) where M is the number of channels (Omni Off) or 0 (Omni On)<br>c = 127, v = 0: Poly Mode On (Mono Off) (Note: These four messages also cause All Notes Off) |
| System Common Messages | | |
| 11110000 | 0iiiiiii<br>[0iiiiiii<br>0iiiiiii] | System Exclusive. This message type allows manufacturers to create their own messages (such as bulk dumps, patch parameters, and other non-spec data) and provides a mechanism for creating additional MIDI Specification messages. The |

| | | |
|---|---|---|
| | 0ddddddd<br>---<br>---<br>0ddddddd<br>11110111 | Manufacturer's ID code (assigned by MMA or AMEI) is either 1 byte (0iiiiiii) or 3 bytes (0iiiiiii 0iiiiiii 0iiiiiii). Two of the 1 Byte IDs are reserved for extensions called Universal Exclusive Messages, which are not manufacturer-specific. If a device recognises the ID code as its own (or as a supported Universal message) it will listen to the rest of the message (0ddddddd). Otherwise, the message will be ignored. (Note: Only Real-Time messages may be interleaved with a System Exclusive.) |
| 11110001 | 0nnndddd | MIDI Time Code Quarter Frame. nnn = Message Type dddd = Values |
| 11110010 | 0lllllll<br>0mmmmmmm | Song Position Pointer. This is an internal 14 bit register that holds the number of MIDI beats (1 beat= six MIDI clocks) since the start of the song. l is the LSB, m the MSB. |
| 11110011 | 0sssssss | Song Select. The Song Select specifies which sequence or song is to be played. |
| 11110100 | | Undefined. (Reserved) |
| 11110101 | | Undefined. (Reserved) |
| 11110110 | | Tune Request. Upon receiving a Tune Request, all analog synthesizers should tune their oscillators. |
| 11110111 | | End of Exclusive. Used to terminate a System Exclusive dump (see above). |
| System Real-Time Messages | | |
| 11111000 | | Timing Clock. Sent 24 times per quarter note when synchronization is required (see text). |
| 11111001 | | Undefined. (Reserved) |
| 11111010 | | Start. Start the current sequence playing. (This message will be followed with Timing Clocks). |
| 11111011 | | Continue. Continue at the point the sequence was Stopped. |

|  |  |  |
|---|---|---|
| 11111100 |  | Stop. Stop the current sequence. |
| 11111101 |  | Undefined. (Reserved) |
| 11111110 |  | Active Sensing. This message is intended to be sent repeatedly to tell the receiver that a connection is alive. Use of this message is optional. When initially received, the receiver will expect to receive another Active Sensing message each 300ms (max), and if it does not then it will assume that the connection has been terminated. At termination, the receiver will turn off all voices and return to normal (non- active sensing) operation. |
| 11111111 |  | Reset. Reset all receivers in the system to power-up status. This should be used sparingly, preferably under manual control. In particular, it should not be sent on power-up. |

Appendix F

# Engineers Australia Code of Ethics – p. 1

# *our* CODE *of* ETHICS

ENGINEERS AUSTRALIA

**As engineering practitioners, we use our knowledge and skills for the benefit of the community to create engineering solutions for a sustainable future. In doing so, we strive to serve the community ahead of other personal or sectional interests.**

Our ***Code of Ethics*** defines the values and principles that shape the decisions we make in engineering practice. The related Guidelines on Professional Conduct provide a framework for members of Engineers Australia to use when exercising their judgment in the practice of engineering.

As members of Engineers Australia, we commit to practise in accordance with the *Code of Ethics* and accept that we will be held accountable for our conduct under Engineers Australia's disciplinary regulations.

**In the course of engineering practice we will:**

## 1. DEMONSTRATE INTEGRITY

1.1 Act on the basis of a well-informed conscience

1.2 Be honest and trustworthy

1.3 Respect the dignity of all persons

## 2. PRACTISE COMPETENTLY

2.1 Maintain and develop knowledge and skills

2.2 Represent areas of competence objectively

2.3 Act on the basis of adequate knowledge

## 3. EXERCISE LEADERSHIP

3.1 Uphold the reputation and trustworthiness of the practice of engineering

3.2 Support and encourage diversity

3.3 Communicate honestly and effectively, taking into account the reliance of others on engineering expertise

## 4. PROMOTE SUSTAINABILITY

4.1 Engage responsibly with the community and other stakeholders

4.2 Practise engineering to foster the health, safety and wellbeing of the community and the environment

4.3 Balance the needs of the present with the needs of future generations

Continued over >>

Appendix G

# OpenSCAD Program Code and Models

## G.1. Wrist Brace OpenSCAD listing

```
//------------------------------------------------------------
//  ~ Student Details ~
//
//      Sathya Smith - Engineering Research Project 2016
//      Supervisor: Dr. Andrew Maxwell
//      Smith_armbraceV2.scad
//
//------------------------------------------------------------
//  ~ Description ~
//
//      This CAD file contains the intended 3D design for
//      the inertial measurement unit housing. The device
//      to be housed is the MPU6050 6 DOF IMU.
//
//------------------------------------------------------------
//  ~ Specifications~
//      Device length - 21mm
//      Device width  - 14mm
//      Desired housing length - 40mm
//      Desired housing width  - 30mm
//------------------------------------------------------------
//  ~ Variables ~
MPUlength = 21.5; // Added 0.5mm to ensure fit
MPUwidth = 14.5;  // Added 0.5mm to ensure fit
MPUheight = 5;
//----------------------------------------------------------
```

```
//------------------------------------------------------------
//  ~ Specify desired module ~
//MPU650_Blank();
bracket();
cover();
//------------------------------------------------------------

//------------------------------------------------------------
//   The Cover module creates a lid for the wristbound sensor
housing.

module cover(){
    union(){
    translate([0,5,9.25])                       rotate([0,0,90])
cube([MPUwidth+5,MPUlength,1]);

    difference(){

    rotate([-90,0,0])          translate([-10.75,-10,10.75+4.25])
roundCornersCube(31,1,30,2);
    translate([-30,-0.5,8.5]) cube([40,31,1.5]);
    }
}
}
//------------------------------------------------------------
```

```
module bracket(){

    difference(){

        translate([-MPUlength/2,15,2.5])        rotate([90,0,0])
roundCornersCube(40,5,30,2);
        translate([0,5,1]) MPU650_Blank();

        rotate([90,0,180])        translate([MPUlength/2,-49,-1])
cylinder(r = 100/2, h=60,$fn=100);

        rotate([90,0,180])      translate([MPUlength/2+3,-49,-1])
cylinder(r = 100/2, h=60,$fn=100);
        rotate([90,0,180])      translate([MPUlength/2+5,-49,-1])
cylinder(r = 100/2, h=60,$fn=100);

        rotate([90,0,180])      translate([MPUlength/2-3,-49,-1])
cylinder(r = 100/2, h=60,$fn=100);
        rotate([90,0,180])      translate([MPUlength/2-5,-49,-1])
cylinder(r = 100/2, h=60,$fn=100);

        rotate([90,0,180])              translate([-6.75,-5,1.5])
cube([2.5,11,27]);
        rotate([90,0,180])              translate([25.75,-5,1.5])
cube([2.5,11,27]);
   }

   //  Draw cylinders for strap hole smoothing

    rotate([90,0,180])  translate([-7.75,4.2,1.5])  cylinder(r  =
1.3, h=27,$fn=100);
    rotate([90,0,180])  translate([-7.75,0.8,1.5])  cylinder(r  =
1.3, h=27,$fn=100);

    rotate([90,0,180])  translate([29.25,4.2,1.5])  cylinder(r  =
1.3, h=27,$fn=100);
    rotate([90,0,180])  translate([29.25,0.8,1.5])  cylinder(r  =
1.3, h=27,$fn=100);
}
```

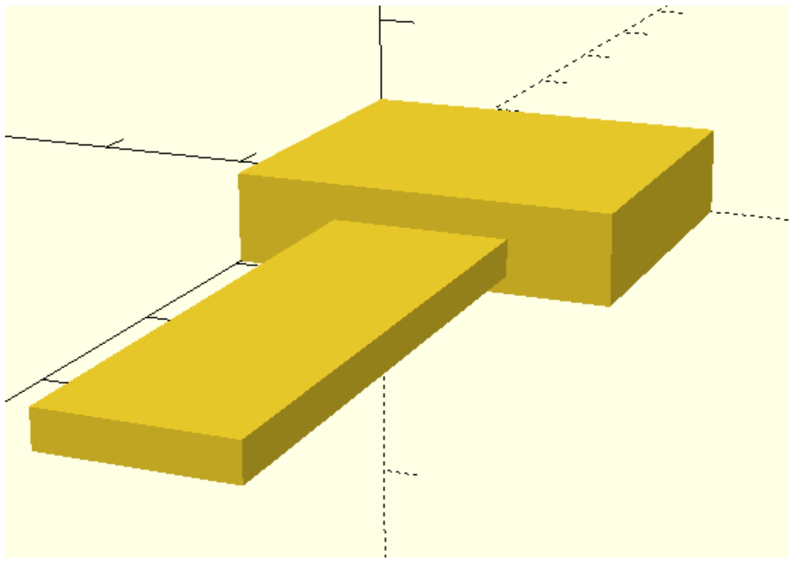
```
module MPU650_Blank(){

    difference(){

        union(){
            rotate([0,0,90]) cube([MPUwidth,MPUlength,5]);
            rotate([0,0,90])           translate([MPUwidth,0,0])
cube([5,MPUlength,5]);
            rotate([0,0,90])     translate([0,(MPUlength/2)-5,1])
cube([50,10,2]);
        }
        union(){
                rotate([0,0,90]) translate([1.5+0.8, 1.5+1.3, 0])
cylinder(r = 3/2, h = 3, $fn=30);
                rotate([0,0,90])  translate([1.5+0.8,  MPUlength-
1.5-1.3,0]) cylinder(r = 3/2, h = 3, $fn=30);
        }
    }
}
//----------------------------------------------------------
// DESIGN CODE CREATED BY SATHYA SMITH CONCLUDES HERE
// The function that follows was used to generate rounded
// corners.
//----------------------------------------------------------
```

```
// The following section includes modules for the creation of a
cube with rounded corners, Copyright is held by Sergio Vilches
2011

/*
http://codeviewer.org/view/code:1b36
Copyright (C) 2011 Sergio Vilches
This program is free software: you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details. You should have received
a copy of the GNU General Public License along with this program.
If not, see <http://www.gnu.org/licenses/>.
Contact: s.vilches.e@gmail.com

-----------------------------------------------------------
                Round Corners Cube (Extruded)
      roundCornersCube(x,y,z,r) Where:
         - x = Xdir width
         - y = Ydir width
         - z = Height of the cube
         - r = Rounding radious

      Example: roundCornerCube(10,10,2,1);
     *Some times it's needed to use F6 to see good results!
      ----------------------------------------------------------
-
*/
// Test it!
// roundCornersCube(10,5,2,1);
```

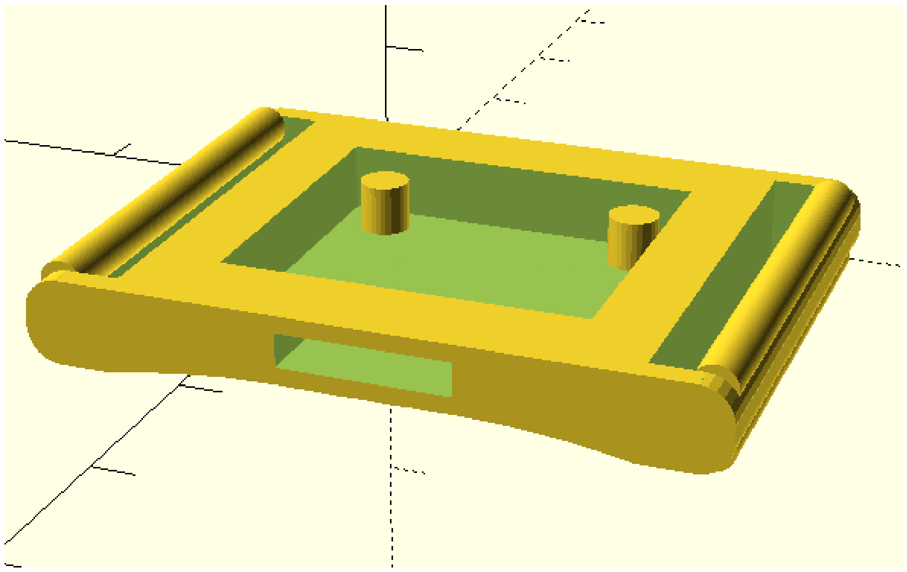
```
module createMeniscus(h,radius)
// This module creates the shape that needs to be substracted from
a cube to make its corners rounded.
difference(){
//This shape is basically the difference between a quarter of
cylinder and a cube

   translate([radius/2+0.1,radius/2+0.1,0]){
      cube([radius+0.2,radius+0.1,h+0.2],center=true);
 // All that 0.x numbers are to avoid "ghost boundaries" when
substracting

   }
   cylinder(h=h+0.2,r=radius,$fn = 25,center=true);
}

module roundCornersCube(x,y,z,r)
// Now we just substract the shape we have created in the four
corners

difference(){
   cube([x,y,z], center=true);

translate([x/2-r,y/2-r]){  // We move to the first corner (x,y)
      rotate(0){
         createMeniscus(z,r); // And substract the meniscus
      }
   }
translate([-x/2+r,y/2-r]){
// To the second corner (-x,y)
      rotate(90){
         createMeniscus(z,r);
// But this time we have to rotate the meniscus 90 deg
      }
}
translate([-x/2+r,-y/2+r]){ // ...
      rotate(180){
         createMeniscus(z,r);
      }
   }
      translate([x/2-r,-y/2+r]){
      rotate(270){
         createMeniscus(z,r);
      }
   }
}
```

## G.2. Wrist Brace Modular Views

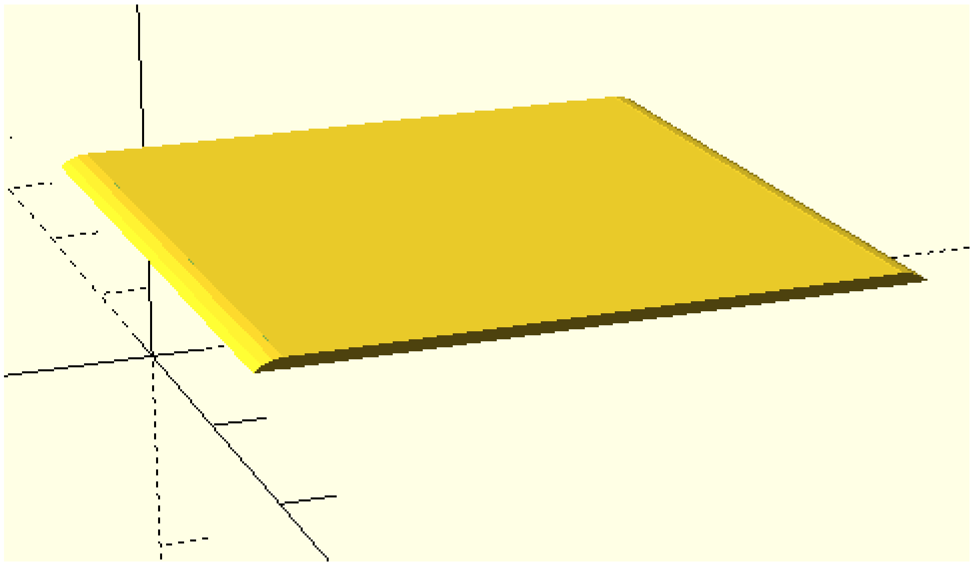
### G.2.1. MPU-6050 Model



### G.2.2. Wrist Brace Main View



## G.2.3. Wrist Brace Cover View



## G.3. Main System Housing Program Listing

```
//------------------------------------------------------------
//  ~ Student Details ~
//
//  Sathya Smith - Engineering Research Project 2016
//  Supervisor: Dr. Andrew Maxwell
//  Smith_armbraceV2.scad

//------------------------------------------------------------
//  ~ Description ~
//
//  This OpenSCAD file contains the intended 3D design for the
main hardware housing adhered to the back of the wearer with an
elastic harness
//
//------------------------------------------------------------
//  ~ Specifications ~
//
//  Arduino:
        ArdLen = 72;    // Actual length = 69, 3.2mm extra
        ArdWid = 60;    // Actual width = 54, 6mm extra
        Ardhole = 1.5;  // Arduino mounting hole radius
//
//  Battery:
        BatLen = 55;    // Actual length = 52.2, 2.8mm extra
        BatLenEx = 2.8;
        BatWid = 29;    // Actual width = 25.9, 3.1mm extra
        BatWidEx = 3.1;
//
//  Transceiver:
        TranLen = 46;   // Actual length = 43.6, 2.4mm
```

```
extra
        TranLenEx = 2.4;
        TranWid = 29;   // Actual width = 23.7, 5.3mm extra
        TranWidEx = 5.3;
        TranHole = 1.7/2;
        AntHole = 12;
        TranBoardThick = 3.3;
//
//  Ports/holes:
        USBLen = 12;    // Actual length = 11.8mm
        USBWid = 10.8;  // Actual width = 10.6mm
        USBDep = 6.3;
        PowerLen = 9;   // Actual len = 8.6mm
        PowerWid = 11;  //Actual wid = 10.7mm

//  Housing:
        HouseLen = 116;
        HouseWid = 104;
        HouseBaseThick = 5;

//  Misc.:
        WallThick = 5;  // Wall/divider thickness
        BoardThick = 4; // Approximate board thickness, taking
solder joints into account
        CompHeight = 31;
        HouseHeight = 35;
        PillarHeight = CompHeight/4;
//
```

```
//  ~ Module Selection ~

//Transceiver();
//Arduino();
//Battery();
//ExtraSpace();
Housing();
Lid();

module Arduino(){

    // This module creates an appropriate model of the space that
will be taken up within the housing by the Arduino Uno board. This
is achieved by forming an appropriately sized rectangular block
and subtracting cylinders for mounting posts, as well as creating
a union between the main block and others in order to create port
holes in the housing.

    difference(){

        union(){

            // Main Arduino block
            cube([ArdWid,ArdLen,CompHeight]);

            // USB port block
            translate([3.17+31.6-3.2,ArdLen-1,BoardThick])
cube([15,CompHeight,15]);

            // 2.1mm power port block
            translate([3.17+3.6,ArdLen-1,BoardThick])
cube([11,CompHeight,13]);
        }
```

```
// Cylinders to create mounting pillars in housing
        translate([10.47+0.05,0.9+1.5+Ardhole,0])  cylinder(h  =
PillarHeight, r = Ardhole, $fn = 50);
        translate([10.47+0.05+24.8+3,0.9+1.5+Ardhole,0])
cylinder(h = PillarHeight, r = Ardhole, $fn = 50);
        translate([3.97+1.5,ArdLen-15.3,0])     cylinder(h      =
PillarHeight, r = Ardhole, $fn = 50);
        translate([ArdWid-(0.9+3.17+1.5),ArdLen-16.8,0])
cylinder(h = PillarHeight, r = Ardhole, $fn = 50);
    }

module Battery(){

    // The battery module creates a simple cube to represent the
appropriately measured size of the 9V battery.

    cube([BatWid,BatLen,CompHeight]);
}

module Transceiver(){

    // The Transceiver module creates a suitable model for the HM-
TR transceiver that is to be used to transmit serial data
wirelessly. Careful notice is taken of the size of the antenna
hole as it will need to be tightened following insertion of the
transceiver board.

    difference(){

union(){

            // Main transceiver block
            cube([TranLen,TranWid,CompHeight]);
// 915MHz antenna hole
            rotate([0,-90,0])
translate([TranBoardThick+6.7,TranWidEx/2+5.3,-1])  cylinder(h  =
PillarHeight,r = AntHole/2, $fn = 50);
        }
// Transceiver mounting post cylinders

translate([11.3+TranHole+TranLenEx/2,TranWidEx/2+1.7+TranHole,0]
) cylinder(h = PillarHeight, r = TranHole, $fn = 50);
        translate([TranLen-TranLenEx/2-1.6-TranHole,TranWid-
TranWidEx/2-3.7-TranHole,0])  cylinder(h  =  PillarHeight,  r  =
TranHole, $fn = 50);
    }
}

module ExtraSpace(){

    // The ExtraSpace module forms a cavity in the south-eastern
section of the housing, which is to be used for wire routing
purposes.

    union(){

        translate([WallThick,0,0])    cube([HouseWid-3*WallThick-
TranLen,TranWid,CompHeight]);
        translate([ArdWid-TranLen+WallThick,HouseLen-
2*WallThick-ArdLen-WallThick-1,0])    cube([HouseWid-3*WallThick-
ArdWid,HouseLen-TranWid-3*WallThick-BatLen+1,CompHeight]);

    }
}
```

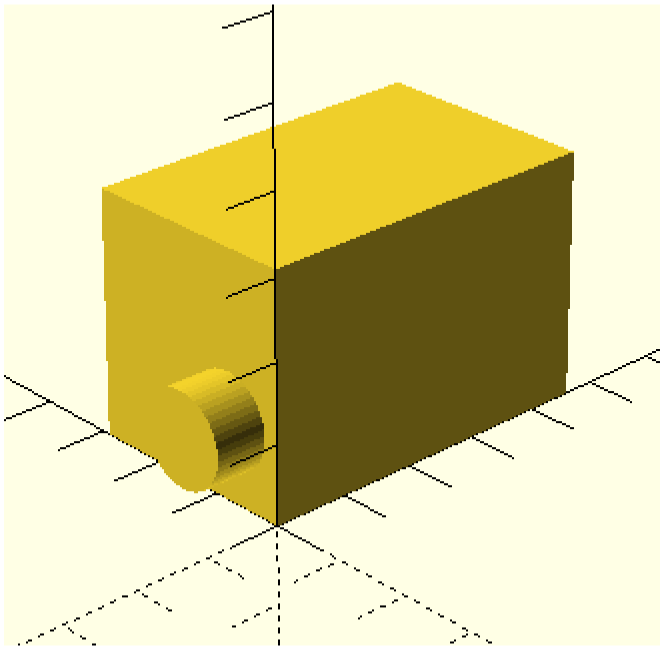
```
module Housing(){

    // The Housing module builds the housing by forming a main
block with x,y,z elements of HouseWid, HouseLen and HouseHeight,
followed by the subtraction of each of the Arduino, Transceiver,
Battery and ExtraSpace modules, forming suitable spaces for each
to sit within the housing. In addition, wall sections have been
removed for wiring access purposes.

difference(){

        // Main housing block
        cube([HouseWid,HouseLen,HouseHeight]);

        // Subtraction of Arduino

translate([WallThick,2*WallThick+TranWid,HouseBaseThick])
Arduino();

        // Subtraction of Transceiver
        translate([WallThick,WallThick,HouseBaseThick])
Transceiver();

        // Subtraction of Battery

translate([2*WallThick+ArdWid,2*WallThick+TranWid+(HouseLen-
3*WallThick-BatLen-TranWid),HouseBaseThick]) Battery();

        // Subtraction of ExtraSpace

translate([1*WallThick+TranLen,WallThick,HouseBaseThick])
ExtraSpace();
// Wiring space allowance
        translate([WallThick+ArdWid-1,HouseLen-2*WallThick-
2,HouseBaseThick]) cube([WallThick+2,WallThick+2,CompHeight]);
        translate([WallThick+ArdWid-1,HouseLen-4*WallThick-2-
BatLen,HouseBaseThick])
cube([WallThick+2,2*WallThick+2,CompHeight]);
        translate([TranLen-2*WallThick,WallThick+TranWid-
1,HouseBaseThick]) cube([3*WallThick,WallThick+2,CompHeight]);
    }
}

module Lid(){

    // The Lid module is the final module created for the hardware
housing and creates a smooth fitted lid by subtracting the top of
the housing from a cube with rounded ends.

    difference(){

        rotate([-90,0,0])                 translate([HouseWid/2,-
100,HouseLen/2])  roundCornersCube(HouseWid-1,WallThick,HouseLen-
1,WallThick);

        translate([0,0,65]) Housing();
    }
}
//------------------------------------------------------------
```

```
// The following section includes modules for the creation of a
cube with rounded corners, Copyright is held by Sergio Vilches
2011

/*
http://codeviewer.org/view/code:1b36
Copyright (C) 2011 Sergio Vilches
This program is free software: you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details. You should have received
a copy of the GNU General Public License along with this program.
If not, see <http://www.gnu.org/licenses/>.
Contact: s.vilches.e@gmail.com

-----------------------------------------------------------
                 Round Corners Cube (Extruded)
      roundCornersCube(x,y,z,r) Where:
         - x = Xdir width
         - y = Ydir width
         - z = Height of the cube
         - r = Rounding radious

      Example: roundCornerCube(10,10,2,1);
     *Some times it's needed to use F6 to see good results!
      ----------------------------------------------------------
-

*/
// Test it!
// roundCornersCube(10,5,2,1);
```

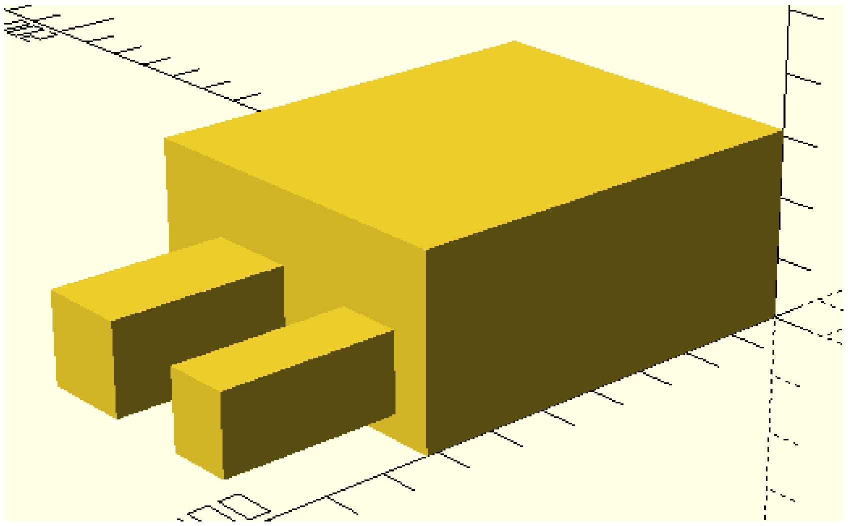
```
module createMeniscus(h,radius) // This module creates the shape
that needs to be substracted from a cube to make its corners
rounded.
difference(){
   translate([radius/2+0.1,radius/2+0.1,0]){
      cube([radius+0.2,radius+0.1,h+0.2],center=true);         //
All that 0.x numbers are to avoid "ghost boundaries" when
substracting
   }
   cylinder(h=h+0.2,r=radius,$fn = 25,center=true);
}
module roundCornersCube(x,y,z,r)   // Now we just substract the
shape we have created in the four corners
difference(){
   cube([x,y,z], center=true);
translate([x/2-r,y/2-r]){  // We move to the first corner (x,y)
      rotate(0){
         createMeniscus(z,r); // And substract the meniscus
      }
   }
   translate([-x/2+r,y/2-r]){ // To the second corner (-x,y)
      rotate(90){
         createMeniscus(z,r); // But this time we have to rotate
the meniscus 90 deg
      }
   }
      translate([-x/2+r,-y/2+r]){ // ...
      rotate(180){
         createMeniscus(z,r);
      }
   }
      translate([x/2-r,-y/2+r]){
      rotate(270){
         createMeniscus(z,r);
}}}
```

## G.4. Main Housing Model Views

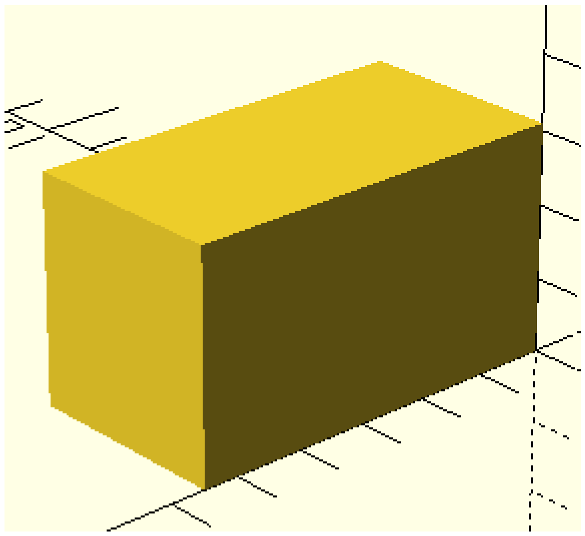
### G.4.1. Transceiver Shape Model



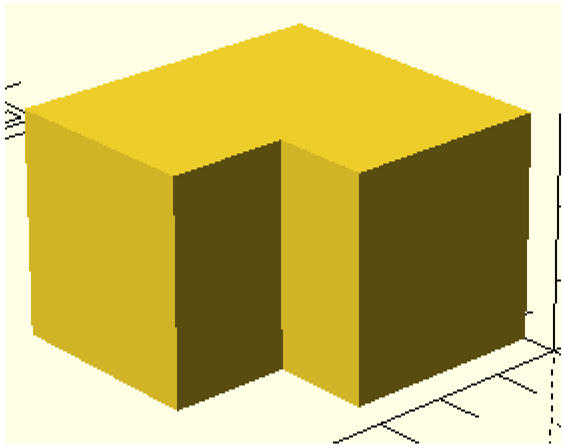
### G.4.2. Arduino Uno Shape Model



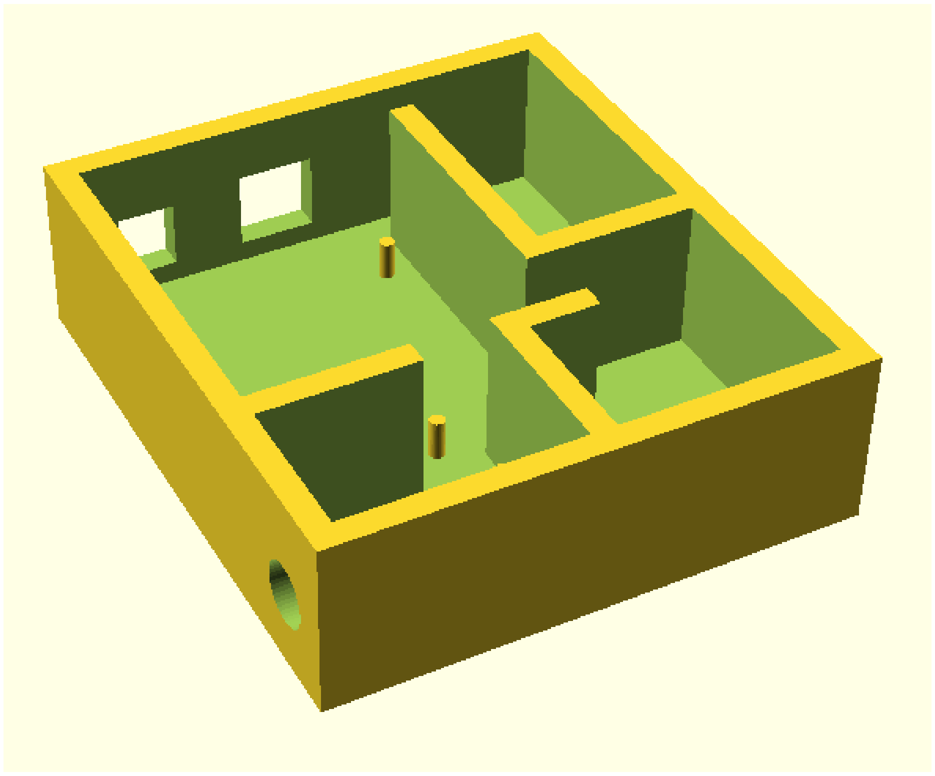
### G.4.3. Battery Shape Model



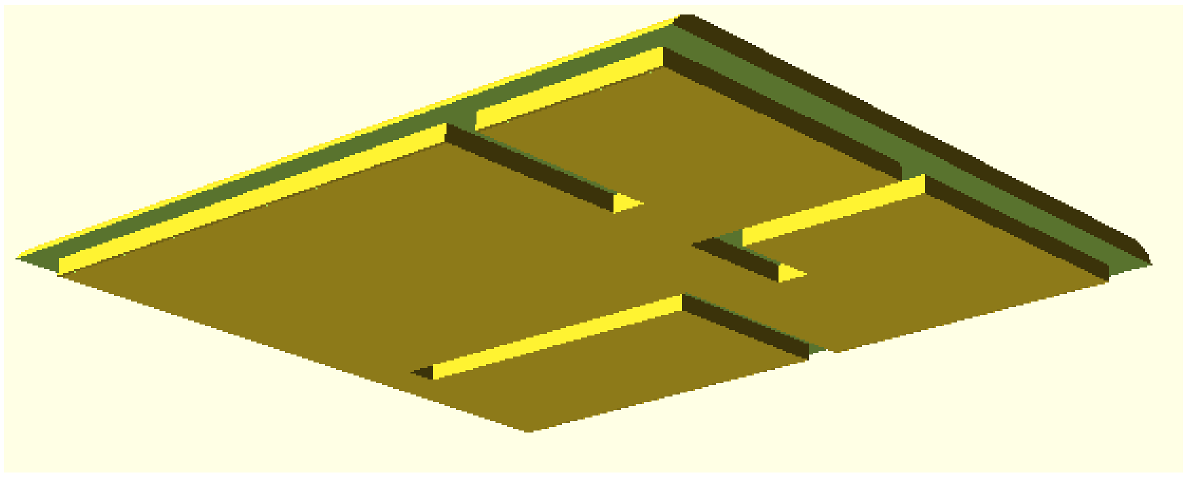
### G.4.4. Extra Space Shape Model



### G.4.5. Main Housing Model



### G.4.6. Main Housing Cover View



Appendix H

# Arduino Program Code

## H.1. Main Arduino Program

```
//----------------------------------------------------------------------//
// Engineering Research Project 2016
// Student:   Sathya Smith
// Supervisor:  Andrew Maxwell
// University of Southern Queensland - Faculty of Health, Engineering and
Sciences
// Program code for Arduino based gesture recognition system.
//----------------------------------------------------------------------//
//
// *** Disclaimer ***
//
// This program contains sections of code utilised with permission from the
// creators, with licensing information presented within the program. The
creator
// of this software program does not take credit for the development
// of various libraries and pieces of example code, as they are simply
// basic building blocks used to quicken the process of system development
// for this engineering research project.
//
//----------------------------------------------------------------------//

// ================================================================
// ===                    INCLUDE LIBRARIES                     ===
// ================================================================

// Include header file of I2C communications
    #include "I2Cdev.h"

// Include header file for MPU6050 6-axis DMP
    #include "MPU6050_6Axis_MotionApps20.h"

// Include header file Wire.h if I2Cdev.h utilises Arduino Wire
   #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
    #endif

// Include elapsedMillis header file for tracking program output rate
    #include "elapsedMillis.h"
```

```
// Include EEPROM capabilities
    #include "EEPROM.h"

// MPU6050 address - 0x68
    MPU6050 mpu;

// ================================================================
// ===                      TEST VARIABLES                      ===
// ================================================================

// Setup Test integers for development
    int ArrayIndex = 0;         // Counter for test purposes
    int ArrayIndexDelay = 0;    // Delay counter for skipping readings
    int TrainingCount = 0;      // Training mode counter for adding values
to arrays

// Mode definition
    #define TestMode 2         // 0 - limited tracking, 1 - full tracking,
2 - training
    #define DispMode 1          // Set low for YPR and XYZ data, set high
for running averages
    #define WindowArraySize 30  // Sliding window size

// ================================================================
// ===                    GENERAL VARIABLES                     ===
// ================================================================


    #define LED_PIN 13        // Definition of LED_PIN for simplicity
    bool blinkState = true;    // Set true to allow Arduino LED blinking
(indicative of activity)

// MPU control/status vars
    bool dmpReady = false;  // set true if DMP init was successful
    uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
    uint8_t devStatus;      // return status after each device operation
(0 = success, !0 = error)
    uint16_t packetSize;    // expected DMP packet size (default is 42
bytes)
    uint16_t fifoCount;     // count of all bytes currently in FIFO
    uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
```

```
    Quaternion q;           // [w, x, y, z]         quaternion container
    VectorInt16 aa;         // [x, y, z]            accel sensor measurements
    VectorInt16 aaReal;     // [x, y, z]            gravity-free accel
sensor measurements
    VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
measurements
    VectorFloat gravity;    // [x, y, z]            gravity vector
    float euler[3];         // [psi, theta, phi]    Euler angle container
    float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container
and gravity vector


// ================================================================
// ===                SLIDING WINDOW VAR CREATION               ===
// ================================================================

/* Yaw, Pitch and Roll arrays */
    int Window_Yaw[WindowArraySize];       // Yaw value array - rotation
about z-axis
    int Window_Pitch[WindowArraySize];    // Pitch value array - rotation
about y-axis
    int Window_Roll[WindowArraySize];     // Roll value array - rotation
about x-axis

/* X, Y and Z accelerometer arrays */
    int Window_Accel_X[WindowArraySize]; // x-axis acceleration array
    int Window_Accel_Y[WindowArraySize]; // y-axis acceleration array
    int Window_Accel_Z[WindowArraySize]; // z-axis acceleration array

/* Sum arrays - 4 bytes per sum */
    long Sum_Yaw = 0;
    long Sum_Pitch = 0;
    long Sum_Roll = 0;
    long Sum_Accel_X = 0;
    long Sum_Accel_Y = 0;
    long Sum_Accel_Z = 0;

/* Running Average array */
    int Window_Average_Yaw = 0;
    int Window_Average_Pitch = 0;
    int Window_Average_Roll = 0;
    int Window_Average_X = 0;
    int Window_Average_Y = 0;
    int Window_Average_Z = 0;
```

```
/* Template average array - length = 5 for five gesture movement average
*/
    int Gesture_Average_Yaw[4];
    int Gesture_Average_Pitch[4];
    int Gesture_Average_Roll[4];
    int Gesture_Average_X[4];
    int Gesture_Average_Y[4];
    int Gesture_Average_Z[4];

/* Gesture template array */
    int GestureTemplate[5]; // 6 value array containing mean values for
each degree of freedom for a gesture


// ================================================================
// ===                      INTERRUPT DETECTION                   ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt
pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

// ================================================================
// ===                      INITIAL SETUP                         ===
// ================================================================

void setup() {

    // join I2C bus (I2Cdev library doesn't do this automatically)
      #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
          Wire.begin();
          TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)

      #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
          Fastwire::setup(400, true);
   #endif

    // initialize serial communication
      Serial.begin(38400);
      while(!Serial); // wait for Leonardo enumeration, others continue
immediately
```

```
    // initialize device
      Serial.println("#---------------------------------------------------
-----------------------------------------#");
      Serial.println(" ");
      Serial.print(F("Initializing I2C devices..."));
      mpu.initialize();

    // verify connection
      Serial.print(F("Testing sensor connections..."));
      Serial.println(mpu.testConnection()    ?    F("MPU6050    connection
successful") : F("MPU6050 connection failed"));
      Serial.println(" ");

    // Choose display mode
      if(DispMode == 0){
        Serial.println("Display mode\t 0 - YPR and RealAccel data");
      }

      else if(DispMode == 1){
        Serial.println("Display mode\t 1 - Moving average data");
      }

    // Choose test mode
      if(TestMode == 0){
        Serial.println("Test mode\t 0 - Limited tracking: Number of values
to be read is equal to window size.\n This mode is used for determining
elapsed times for various data set sizes.");
      }
      else if(TestMode == 1){
        Serial.println("Test  mode\t  1  -  Tracking:  Running  averages  are
displayed as they are calculated\n\t\t\t\tand are compared to gesture data
in memory.");
      }
      else if(TestMode == 2){
        Serial.println("Test  mode\t  2  -  Training:  Addition  of  a  new
gesture");
      }

      Serial.println(" ");
      Serial.println("#---------------------------------------------------
-----------------------------------------#");
```

```
    // wait for ready
    if(TestMode != 2){
      Serial.println("\nSend any character to begin motion tracking: ");
      while (Serial.available() && Serial.read()); // empty buffer
      while (!Serial.available());                 // wait for data
      while (Serial.available() && Serial.read()); // empty buffer again
    }

    // load and configure the DMP
      devStatus = mpu.dmpInitialize();

    // Gyro and Accel offsets
      mpu.setXGyroOffset(220);
      mpu.setYGyroOffset(76);
      mpu.setZGyroOffset(-85);
      mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0)
    {
        // turn on the DMP, now that it's ready
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's
okay to use it
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();

    } else {
        // ERROR!
     // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);

    // Initialise Window arrays at 0
    for (int i = 0; i < WindowArraySize; i++){
      Window_Yaw[i] = 0;
      Window_Pitch[i] = 0;
      Window_Roll[i] = 0;
      Window_Accel_X[i] = 0;
      Window_Accel_Y[i] = 0;
      Window_Accel_Z[i] = 0;
    }
}

// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {

  // If TestMode was set to 2 at the beginning of the program display
training mode information
  // Only display this information on the first training run

  if(TestMode == 2 && TrainingCount == 0){
   Serial.print("Gesture training mode has been initiated - please note
that only one gesture may be recorded\nat this time. ");
   Serial.print("Perform the desired movement immediately after the
countdown.\nThis process will occur five times, then the gesture model will
be calculated and recorded.\n");
   Serial.println("\nSend any character to begin gesture training! ");
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

      Serial.print("3...");
      delay(1000);
      Serial.print("2...");
      delay(1000);
      Serial.print("1...");
      delay(1000);
      Serial.print("Go!");
      delay(1000);
  }

  // For consecutive training runs - prompt to send another character for
next run

  else if(TestMode == 2 && TrainingCount != 0)
  {
    Serial.print("Training run ");
    Serial.print(TrainingCount+1);
    Serial.print(F(" completed, send any character to begin the next
countdown."));
  }

  elapsedMillis ElapsedTime;  // Begin timer to track practice run period

  // while loop will iterate infinitely in full tracking mode, but
  // only for WindowArraySize no. of values for limited tracking mode
  // as well as training mode

  while(ArrayIndex < WindowArraySize) {

    // if dmp programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too
inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));
    // otherwise, check for DMP data ready interrupt (this should happen
frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short
wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

    // Subtract old values from sum
        Sum_Yaw = Sum_Yaw - Window_Yaw[ArrayIndex];
        Sum_Pitch = Sum_Pitch - Window_Pitch[ArrayIndex];
        Sum_Roll = Sum_Roll - Window_Roll[ArrayIndex];
        Sum_Accel_X = Sum_Accel_X - Window_Accel_X[ArrayIndex];
        Sum_Accel_Y = Sum_Accel_Y - Window_Accel_Y[ArrayIndex];
        Sum_Accel_Z = Sum_Accel_Z - Window_Accel_Z[ArrayIndex];


    // Get positional angle in degrees
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    // Populate yaw, pitch and roll window arrays
        Window_Yaw[ArrayIndex] = ypr[0] * 180/M_PI;
        Window_Pitch[ArrayIndex] = ypr[1] * 180/M_PI;
        Window_Roll[ArrayIndex] = ypr[2] * 180/M_PI;

    // Get adjusted real acceleration
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetAccel(&aa, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);

    // Populate window array with current accel vals
        Window_Accel_X[ArrayIndex] = aaReal.x;
        Window_Accel_Y[ArrayIndex] = aaReal.y;
        Window_Accel_Z[ArrayIndex] = aaReal.z;
    // Add fetched positional values to respective sum
        Sum_Yaw = Sum_Yaw + Window_Yaw[ArrayIndex];
        Sum_Pitch = Sum_Pitch + Window_Pitch[ArrayIndex];
        Sum_Roll = Sum_Roll + Window_Roll[ArrayIndex];
        Sum_Accel_X = Sum_Accel_X + Window_Accel_X[ArrayIndex];
        Sum_Accel_Y = Sum_Accel_Y + Window_Accel_Y[ArrayIndex];
        Sum_Accel_Z = Sum_Accel_Z + Window_Accel_Z[ArrayIndex];

    // Increment to next index for next positional readings
        ArrayIndex = ArrayIndex + 1;

    // Circular array population - index returns to 0 after final value
        if(ArrayIndex >= WindowArraySize && TestMode == 1){
          ArrayIndex = 0;
        }

    // Calculation of averages
        Window_Average_Yaw = Sum_Yaw/WindowArraySize;
        Window_Average_Pitch = Sum_Pitch/WindowArraySize;
        Window_Average_Roll = Sum_Roll/WindowArraySize;
        Window_Average_X = Sum_Accel_X/WindowArraySize;
        Window_Average_Y = Sum_Accel_Y/WindowArraySize;
        Window_Average_Z = Sum_Accel_Z/WindowArraySize;

    // If in training mode and not the last practice run, add window average
to gesture array
        if (TestMode == 2 && TrainingCount <= 4)
        {
        // Populate average positional values for trained gesture
          Gesture_Average_Yaw[TrainingCount] = Window_Average_Yaw;
          Gesture_Average_Pitch[TrainingCount] = Window_Average_Pitch;
          Gesture_Average_Roll[TrainingCount] = Window_Average_Roll;
          Gesture_Average_X[TrainingCount] = Window_Average_X;
          Gesture_Average_Y[TrainingCount] = Window_Average_Y;
          Gesture_Average_Z[TrainingCount] = Window_Average_Z;

        // Incrememnt training counter until gesture has been repeated 5
times
          TrainingCount++;
        }

        // Display real time positional values if DispMode is set low
          if (DispMode == 0)
          {
            Serial.print("YPR\t");
            Serial.print(Window_Yaw[ArrayIndex]);
            Serial.print("\t");
            Serial.print(Window_Pitch[ArrayIndex]);
            Serial.print("\t");
            Serial.print(Window_Roll[ArrayIndex]);
            Serial.print("\t");

            Serial.print("XYZ\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
          }

        // Display running average positional values if DispMode is set
high
          if (DispMode == 1)
          {
            Serial.print("Av. YPR\t");
            Serial.print(Window_Average_Yaw);
            Serial.print("\t");
            Serial.print(Window_Average_Pitch);
            Serial.print("\t");
            Serial.print(Window_Average_Roll);
            Serial.print("\t");

            Serial.print("Av. XYZ\t");
            Serial.print(Window_Average_X);
            Serial.print("\t");
            Serial.print(Window_Average_Y);
            Serial.print("\t");
            Serial.println(Window_Average_Z);
          }
    }
  }

  // Display elapsed running time
    Serial.print(F("Elapsed time for "));
    Serial.print(WindowArraySize);
    Serial.print(F(" elements was "));
    Serial.print(ElapsedTime);
    Serial.print(F(" milliseconds"));
  // Reset elapsed time
    ElapsedTime = 0;

  if(TestMode == 0){
  // Begin next motion tracking loop
    Serial.println(F("\nSend any character to begin motion tracking: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again
    ArrayIndex = 0;
  }
}

//--------------------------------------------------------------------//
//                    LICENSING INFORMATION                           //
//--------------------------------------------------------------------//

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a
copy
of this software and associated documentation files (the "Software"), to
deal
in the Software without restriction, including without limitation the
rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================*/
```

## H.2. MIDI Command Transmission Code

```
//--------------------------------------------------------------------
----//
// Engineering Research Project 2016
// Student:   Sathya Smith
// Supervisor:  Andrew Maxwell
// University of Southern Queensland - Faculty of Health, Engineering
and Sciences
// Program code for Arduino based gesture recognition system.
//--------------------------------------------------------------------
----//

long  lastSendTime = 0; // Initialise sending period counter
int   NoteOn = 144;     // Decimal MIDI command 'Note On' - 10010000

void setup()
{
    Serial.begin(9600);  // Monitoring via USB
}

void loop()
{
  // Track elapsed program run time
  long thisTime = millis();

  // If one second elapsed - send
  if (thisTime > lastSendTime + 1000)
  {
    Serial.print(NoteOn);     // Send MIDI command
    Serial.print("\n");       // Send new line
    lastSendTime = thisTime;  // Reset sending period counter
  }
}
```

## H.3. MIDI Command Receiver Code

```
//--------------------------------------------------------------------
----//
// Engineering Research Project 2016
// Student:   Sathya Smith
// Supervisor:  Andrew Maxwell
// University of Southern Queensland - Faculty of Health, Engineering
and Sciences
// Program code for Arduino based gesture recognition system.
//--------------------------------------------------------------------
----//

#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // Create software serial port with
RX - 10, TX - 11

String inputstring = "";

void setup() {

  // Open serial communications
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect.
  }

  Serial.println("Initialising MIDI command receiver system!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(9600);
  mySerial.println("Data incoming...");

}

void loop() {
  // Read receiver data and present in binary format
  while (mySerial.available() > 0) {
    //Serial.print("\nReceived MIDI command: ");
    int inputchar = mySerial.read();
    if (isDigit(inputchar)) {
      inputstring += (char)inputchar;
    }

    if (inputchar == '\n') {
      Serial.print("Received MIDI Command: ");
      Serial.println(inputstring.toInt(), BIN);
      inputstring = "";
    }
  }
}
```