Sheffield Hallam University

Faculty of Science, Technology and Arts

Department of Computing Project (Technical Computing) [55-604708] 2019/20

Author:	**Mitchel Peake**
Student ID:	**B6017670**
Year Submitted:	**2020**
Supervisor:	**Sergio Davies**
Second Marker:	**Tom Sampson**
Degree Course:	**Computer Science (Games)**
Title of Project:	**Data Capture & Visualisation Tool**

Confidentiality Required?

[✓ **NO**]

YES ☐

2020

Data Capture & Visualisation Tool

MITCHEL PEAKE

Abstract

This project was to find a low-cost method of recording movement and positional data of an object. This data would then be used inside a game engine to visualise the movements. Inspiration was taken from motion capture, but not utilising the equipment used in that process. Instead data will be captured by a small battery-operated device that can be attached to an object. Research was focused on Arduino based devices that recorded rotational data and transmitted this data to a visualisation tool. The research proved that data can be captured and visualised, it also highlighted the equipment that would be able to perform such a task.

With research complete, a prototype device was built that was able to record data, and an application to view the data was built. During testing and development, it was found that accurate positional data was not possible without the use of further hardware, such as a camera and using filters like the Kalman filter. In the conclusion, we will look at how the device and supporting application can be improved and potential future uses for the current iteration.

Contents

Abstract 2
1 Introduction 5
1.1 Identifying the project 5
1.2 Solution to the problem 6
1.3 Aims and Objectives 7
1.3.1 Data Capture 7
1.3.2 Data Visualization 7
1.4 Project Outlines 8
2 Literature Review 9
2.1 Hardware 9
2.1.1 Arduino 9
2.1.2 Storage 10
2.2 Similar projects & motion recorders 11
2.2.1 MPU-6050 11
2.2.2 Example 1 MPU-6050 & Processing 12
2.2.3 Example 2 MPU-6050 & Uduino 12
2.2.4 Example 3 MPU9150 & Unity 13
2.2.5 Example 4 Arduino 9 axis shield 14
2.2.6 Unity 14
3 Methodology / Design 16
3.1 Hardware 16
3.1.1 Arduino Uno 16
3.1.2 Storage 17
3.1.3 IMU 18
3.2 Tools 18
3.2.1 Arduino IDE 18

3.2.2 Unity 3D 18
3.3 Techniques 18
3.3.1 Overview of Plan 19
4 Development 20
4.1 Hardware 20
4.1.1 Setting up the IMU 20
4.1.2 Storage 22
4.2 Visualisation 24
4.3 Findings 25
4.4 Gimbal Lock 26
4.5 Quaternions 27
4.6 Rotating 3D Model 27
4.7 Positional Data 31
4.8 Supporting Functionality 33
5 Testing 34
6 Evaluation 36
6.1 Product 36
6.2 Self 37
7 Conclusion 38
7.1 Limitations 38
7.2 Future implementations 38
7.3 Final Statement 39
8 Appendices 40
A Data 40
..... 40
B Project Specification 42
9 References 42

1 Introduction

This project is to record rotational and positional data from a real-life object using low cost physical computing devices, such as a gyroscope. This data will then be used to visualise that movement in a real time 3D environment. This will be done without the use of a camera, animator, or motion capture technology.

1.1 Identifying the project

Inspiration for this project presented itself when learning how to skateboard.
In skateboarding, there is one trick that when mastered unlocks the opportunity to learn countless other tricks. This trick is the ollie. In short, the ollie allows the user to propel the board, and themselves, off the ground. The ollie is a trick which can take months to learn; it requires balance, timing, and patience.

When trying to learn this trick, the author filmed an attempt. Then watched the footage back to gain insight on the movement, to see if improvements were being made.

The data received was not always suitable or accurate enough. In addition to this, the process of obtaining the data was cumbersome. It required a tripod and a phone. The phone would be set to record; I would get into position and then attempt the trick. To make it more efficient, multiple attempts of the trick would be performed. I would then return to the phone, stop recording and watch the footage (usually trimming the video, to remove the parts where the trick was not being performed).

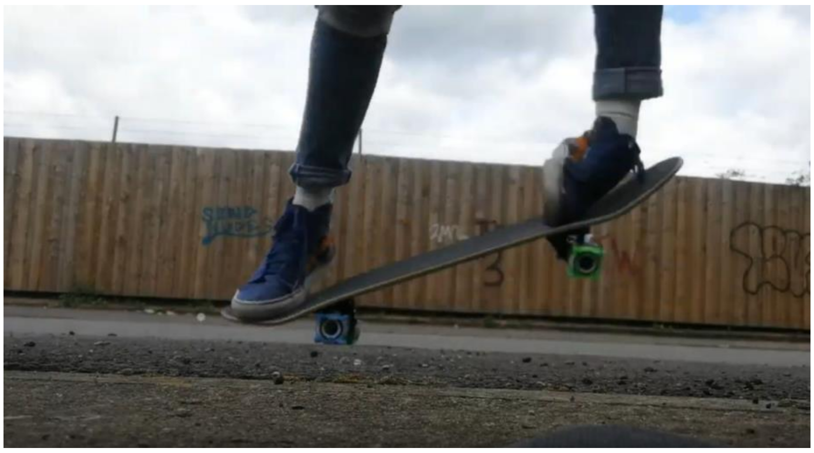


Figure 1-1 Screen shot of an ollie attempt

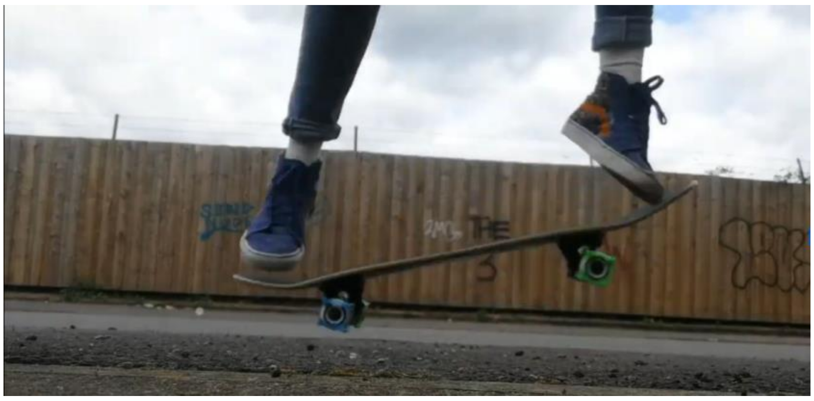


Figure 1.1-2 Screen shot of an ollie attempt

The stills above are taken from a video recorded when first learning how to perform an ollie. These frames show the information that I was wanting to obtain.

When performing the trick, the board needs to be kicked down by the back leg. By doing this, the board will tilt like a seesaw. The tail of the board will then hit the ground and bounce off it. The user will then need to lift the back leg up and flick their front foot forwards to gain height and to level the board out for descent.

What I wanted to see from the video was the board pitching up and then see the rear of the board rise and level out.

Due to the lack of experience from the user, the time that this was happening within was in milliseconds. This made it harder to view the movement, and to see if the correct forces were being applied.

To improve the data received, the slow-motion function on the hardware was used to capture the movement.
The hardware used at the time (Huawei p10) was able to capture slow motion at 240FPS (Frame Per Seconds). This made viewing the desired data easier but, in the process, created more data.
In a video which was 27 Seconds long, only 1 second’s worth was valuable data.

By creating more data, it was making it harder to find the data that would be useful.

This was when a possible solution was hypothesised.

1.2 Solution to the problem

A solution to the problem would be to track movement and rotational changes in the axis of the skateboard. This could be done by using professional equipment such as motion capture.
Then using this data to manipulate a 3D model of the object in an application which a user could view frame by frame and at any angle. The problem with using motion capture is that it is an expensive solution that requires a studio to perform the movement in. What is required here is a cheap, portable solution to record movement data.

The initial idea was inspired by a spirit level application on a smartphone and motion-controlled games. A modern smart phone has an accelerometer and gyroscope sensors (Nield, 2017), that can be used for sensing the orientation and movement of the device.

Attaching a mobile phone to a moving object would not be suitable, as the risk of damaging the device is too high. Therefore, an alternative would be required. The decision was taken to research different devices that would be able to record and transmit the data. This will be discussed in-depth in chapter 2.

1.3 Aims and Objectives

To be able to record and view data of an object moving in real time, two things will need to be done.

1. To program a device that can capture real-time physics data from a skateboard (or another object).

2. Create an application which allows the user to see a 3D model of the board (or other object), performing the recorded data (from task 1).

By achieving these aims, the project deliverable will either prove or disprove if motion data can be captured and accurately recreated in a graphical representation. Without the use of traditional motion capture technology or a video camera.

To achieve these aims, objectives will need to be met. The project consists of two parts: data capture and data visualisation. Let us have a look at the objectives for each component.

1.3.1 Data Capture

To successfully complete this tool, the following will need to be completed:

- Research the various devices available that are capable of recording and saving the required data.
- Identify which devices would be suitable for use. Looking at price, size, power, technologies required.
- Research how to save and transmit the data to an external application.
- Implement the recording and saving of the data.

<u>Once the data visualization tool can read the data</u>

- Depending on how the visualiser renders the data, make any changes to the recorder. This may be sample rate (the amount times in a second that data is recorded) or the type of data recorded.

1.3.2 Data Visualization

To successfully complete this tool the following will need to be completed:

- Identify any existing tools that will be able to visualise the data and manipulate a graphical representation of the object.
- Use this tool to read the data recorded and convert it into useable data.
- Implement a functionality to allow a 3d model of an object to be manipulated by the data recorded.
- Wrap the functionality inside an application which a user can use (i.e. open, load data, view data, save data, close.).

1.4 Project Outlines

The problem can be summarised as:
Is it possible to record movement data from an object and visualise that in a real time 3D environment, without the use of a camera, animator, or motion capture technology.

The proposed solution to this:
Is to create a device which records positional and rotational data of an object. This data is then used in an external application, which will allow the user to see the movement in real-time.

The next chapter will focus on research. This will be on current solutions to the problem and on similar projects.

2 Literature Review

Research began with trying to find hardware that would be suitable for the project. The hardware needed to meet the following criteria:

- Record rotational and acceleration data in 3-axis.
- Operate with one button. The device will not have a screen, it will need to be turned on then record.
- Save and store the data so it can be transferred to an external application.
- Be able to run off battery power

2.1 Hardware

Research was required to gather information on the equipment that was available to use. This started with the microcomputer.
Originally the Raspberry Pi was the first choice, but the rationale for this was poor. Raspberry Pis have become synonymous with home brew electronics and hobbyists; therefore, it was natural to consider this first. But there was no reason to choose this device. The Raspberry Pi is a micro-computer, this means it has all the functionality of a computer and all the complexities.
This project needed a piece of hardware with one job, this is where a microcontroller excels. A micro controller such as the Arduino Uno is designed to run one task repeatedly.
An article on Electronics Hub correlated the main differences between the Raspberry Pi and an Arduino micro controller. Declaring:

- The Arduino runs one program repeatedly.
- The device can be powered by a battery.
- Simple to interface sensors, with Arduino.
- The device can be set to run once power is supplied.
- Storage can be supplied to the device.

(Electronics Hub, 2017)

The Arduino board contains:

- Digital pins (pins 0-13) which can be set to input or output, depending on what you choose in the sketch (program).
- Analogue In pins (pins 0-5) that take analogue values from sensors and covert them to a number between 0-1023.
- Analogue Out (pins 3, 5, 6, 9, 10, 11) six digital pins that can be reprogrammed for analogue output. (Banzi, 2011)

When compared against the requirements of the hardware, the Arduino makes a compelling case. Research moved onto what was possible with an Arduino.

2.1.1 Arduino

Getting Started with Arduino by cofounder Massimo Banzi, introduces the platform, the hardware, and the code. It introduces users into physical computing gently, with simple tutorials such as blinking LEDs. Banzi states that complex sensors (sensors that contain microcontrollers which pre-processes information) such as accelerometers can be used, but tutorials are not provided in the publication. (Banzi, 2011).
Arduino produces a number of different boards for example; The Lilypad board is designed for

wearable technology, the Arduino Yun is designed for internet of things devices.
The Arduino Uno is regarded as the starting point board for electronics and coding because of its robust nature (Nayyar & Puri, 2016). The project requires the devices to be rotated and moved, therefore the device needs to be robust. The Arduino Uno is an entry level device that will be capable of performing the tasks required

Figure 2-1 Image of Arduino Uno (Arduino, 2020)

The project now had a board to enable the data capture.
Referring to the requirements of the device:

- Record rotational and acceleration data in 3-axis.
- Operate with one button. The device will not have a screen, it will need to be turned then record.
- Save and store the data so it can be transferred to an external application.
- Be able to run off battery power

The Arduino can operate with one button, when powered it begins running the sketch (program) uploaded. The device can also be powered by a battery.
What this board cannot do is record movement, rotational data and store this data.
The focus on research now is to identify sensors and hardware that can be combined with the board to achieve the remaining requirements.

2.1.2 Storage

Arduinos can have their functionality improved with peripherals called shields. A shield is a circuit board that can be attached on top of the Arduino adding additional functionality. Shields allow beginners to create devices without the need to solder components, therefore allowing fast development of prototypes.
The prototype device for this project requires the ability to store data recorded via sensors. During research on Arduino boards the ethernet shield was identified as a possible solution for storage.

(Nayyar & Puri, 2016)

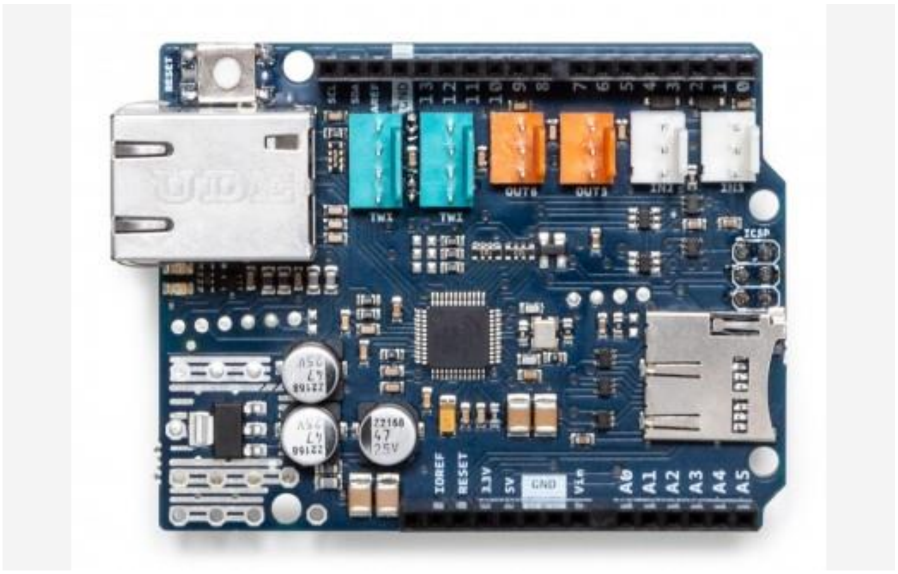


Figure 2-2 Image of Arduino Ethernet Shield (Arduino, 2020)

The Ethernet shield was created for providing internet access to an Arduino for projects such as a chat server. It features a SD card reader that can be used for storage. The shield fits directly on top of the Arduino board and the SD card can be utilised by using the SD library. To save data to the SD card, you can create a string (an object that can store an arrangement of characters) of the data you wish to store, create/locate a text file to store the data, and write the string to the file.
With a proposed solution for saving data, focus moved to finding hardware to obtain movement data.

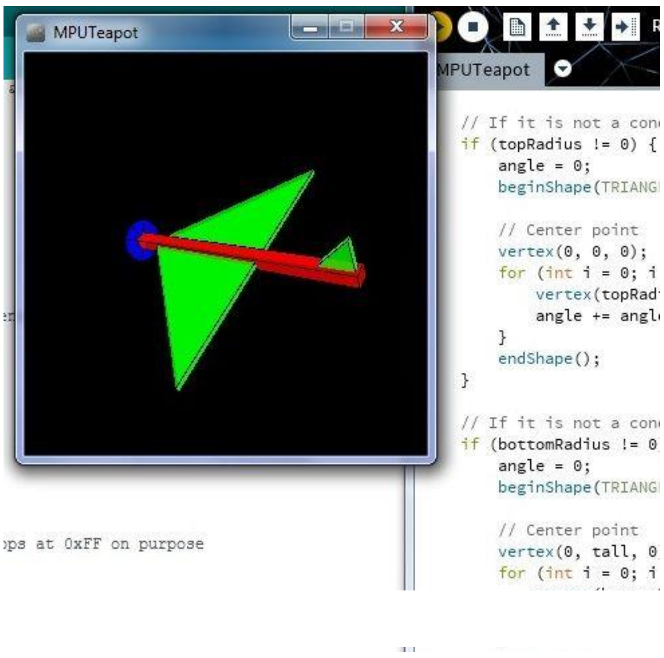
2.2 Similar projects & motion recorders

Investigating projects which utilise motion data and micro controllers will highlight devices that can be used in this project. It will also help with development by giving inspiration, libraries that can be used and common pitfalls.

2.2.1 MPU-6050

During research, the IMU (inertial measurement unit) MPU-6050 was highlighted in several examples.
The MPU6050 is a small (21.2mm x 16.4mm) IMU which has a gyro and accelerometer (plus thermometer). This device can capture acceleration and rotational changes in the X, Y, Z Axis.
However according to the official Arduino playground (A wiki for discussing all things Arduino) it states that “Reading raw values is easy, the rest is not” (Arduino, 2020) Although this can be negated by utilising external libraries, such as the Jeff Rowberg’s I2C library .

2.2.2 Example 1 MPU-6050 & Processing



```
// If it is not a con
if (topRadius != 0) {
    angle = 0;
    beginShape(TRIANG

    // Center point
    vertex(0, 0, 0);
    for (int i = 0; i
        vertex(topRad
        angle += angl
    }
    endShape();
}

// If it is not a con
if (bottomRadius != 0
    angle = 0;
    beginShape(TRIANG

    // Center point
    vertex(0, tall, 0
    for (int i = 0; i
```

```
ps at 0xFF on purpose
```

Figure 2-3 image showing example 1

In an example from instructables.com, a member has used the MPU-6050 and an Arduino to capture and recreate the live rotational data from the sensor (HobbyTransform, 2020).
As mentioned on the Arduino playground this user utilises the I2C Library created by Jeff Rowberg https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050 (Rowberg, 2020). The user states that if the I2C library wasn't used "it would take ages" and would require writing the whole thing from scratch. (HobbyTransform, 2020).The guide shows how to wire the IMU to the Arduino and how to use the information received to output a graphical representation in processing ("flexible software sketchbook and a language for learning how to code within the context of the visual arts") (Processing, 2020).

The model of the plane can be manipulated by moving the IMU sensor attached to the Arduino, it records the Roll, Pitch and Yaw, and displays this for the user.

This tutorial was useful because it showed that the project (the author is writing about) can be done, and others have successfully recorded movement data and output a graphical representation of this.

Where the example fails to meet the requirements for the project specification are in the way the data is transmitted and the GUI (graphical user interface). A video of the above example can be found here: https://www.youtube.com/watch?v=FYf59J0YUDE&feature=youtu.be

2.2.3 Example 2 MPU-6050 & Uduino

This example used the same IMU, but a different GUI and a plugin created by Marc Teyssier. This example used the I2C library (Like the previous example), but instead of using processing it uses the game engine Unity. (Teyssier, 2020)
An Example of the project can be seen in action here: https://youtu.be/8YP8HrcGr9M. Marc Teyssier has created a plugin for Unity called Uduino which allows a simplified way to communicate data from an Arduino to the game engine Unity.

What is interesting about this project, is the use of Unity. Unity is a game engine, it can manipulate an object with simulated physics. The Author has experience with using Unity in the past and found it is fast to prototype in. What Unity can offer is physics-based movements, reactions, camera and lighting effects (to improve the graphical representation, something processing does not offer as easily.) and a quick and efficient way to make UI.

2.2.4 Example 3 MPU9150 & Unity

In the paper Micro-IMU-based motion tracking system for virtual training, the authors attempt to create a “low cost wireless real-time inertial body tracking system” (Zhang, Fei, Xu, & Sun, 2015). The project used multiple IMUs to track the positions of a person’s arms wrists and hands. This would then be used in a training simulation in Unity 3D. The MPU9150 is the IMU used in this example and data is sent live via Zigbee wireless network to Unity.

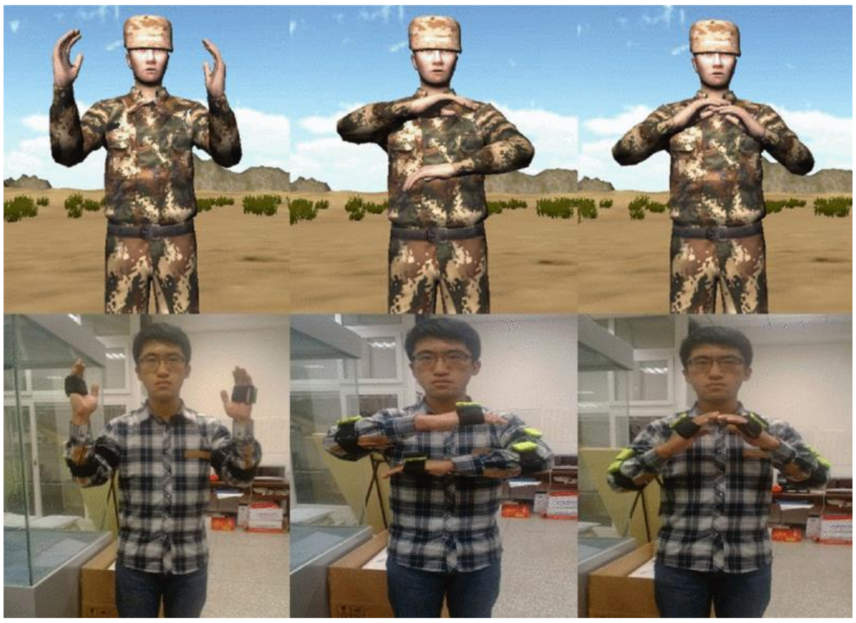


Figure 2-4image of working prototype IMU motion capture (Zhang, Fei, Xu, & Sun, 2015)

The results showed capture of rotational changes by IMUs located on the body, with the mean errors of pitch, roll and yaw showing 0.368°, 0.193° and 2.277° (Zhang, Fei, Xu, & Sun, 2015). The Euler angles are converted to quaternions to avoid gimble lock.
This means that capture of rotational data can be achieved by low cost IMUs (at time of writing the IMU was priced at less than £10 exc. VAT).
What this paper does not discuss is changes to position i.e. moving in x, y, and z axis. This was noticeable in other examples.

2.2.5 Example 4 Arduino 9 axis shield.

This next example is for a very simple application, however the tools used are interesting as it opens the idea to a plug and play IMU. The IMU used in this tutorial is the 9 Axis Motion Shield created by Arduino. The application is a test to see if the user holding the IMU is moving or not. (chauhannaman98, 2020)

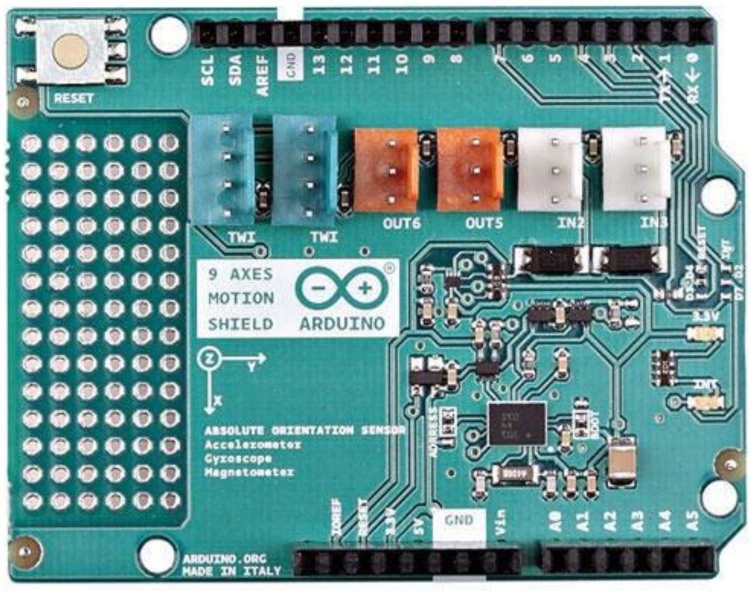


Figure 2-5 Image of 9 axis IMU (chauhannaman98, 2020)

This IMU is an absolute orientation that has a 14-bit accelerometer, a triaxial 16-bit gyroscope with a range of ±2000 degrees per second (chauhannaman98, 2020). This device comes with a library which allows the user to read Quaternions, Euler angles, and Linear acceleration quite easily. Another benefit is that it is a shield, which means it does not require soldering and stacks directly onto the Arduino Shield.

2.2.6 Unity

Unity is a game engine developed by unity technologies, that can be used for visualisation of data films, and other non-game related digital media.

During research, processing was used in numerous examples to visualise data. Processing can provide a suitable visualisation for data, but the visualisation can look basic.
When looking at the examples above, two use the game engine unity. Unity will allow the visualisation to look more realistic (depending on the models chosen) and will make it easy to add physics and camera options, due to the inbuilt physic options. Tools such as processing will require the user to create a physics system, which would take up precious time.
Other game engines are available however the user has more experience with Unity therefore leading to progress in development. Unity also makes it easy to develop for mobile and desktop options quickly.

After researching these tutorials and reports, we can see that it is possible to record movement data from an IMU and display it graphically within Unity and other applications. Two IMUs have been highlighted for potential use, this is the MPU-6050 and the Arduino 9 axis shield.

Example	**IMU**	**GUI**	**Advantages Discovered**	**Disadvantages Discovered**
1	MPU6050	Processing	Yaw pitch and Roll changes can be displayed in a graphic simulation.	The IMU can be difficult to get readings from and relies heavily on external libraries. Will need to be soldered. Uses Euler angles which can cause gimble lock. Processing does not look as polished as other alternatives such as unity. No positional data.
2	MPU6050	Unity	Seeing the movements in Unity looks better than processing and opens opportunities for physics camera effects and UI	Relies heavily on third party plugin created by the author of this example. Uses the MPU6050 IMU has been quoted as difficult to work with help from external libraries Streams live data to Unity. No positional data.
3	MPU9150	Unity	Shows that movement can be reliably recorded from IMUs and output into Unity, with little drift. Wireless transmission of data.	Streams live data to Unity. No positional data.
4	9Axis Motion	N/A	Uses a 9axis IMU which is a shield, so can be used in a plug and play style. The shield comes with its own library which makes obtaining the data easier. Can record quaternions instead of Euler angles.	The scope of the project is too small to be compared with what this project is aiming to achieve.

The examples explored sent data live to a graphical reconstruction. In this project the aim is to record the data and view separately at a different time, hence the need for storage.
Seeing similar examples showed that it is possible and gave inspiration for tools and hardware that will be considered for use.
In the next section we will look at what tools and equipment will be used and the justification for that.

3 Methodology / Design

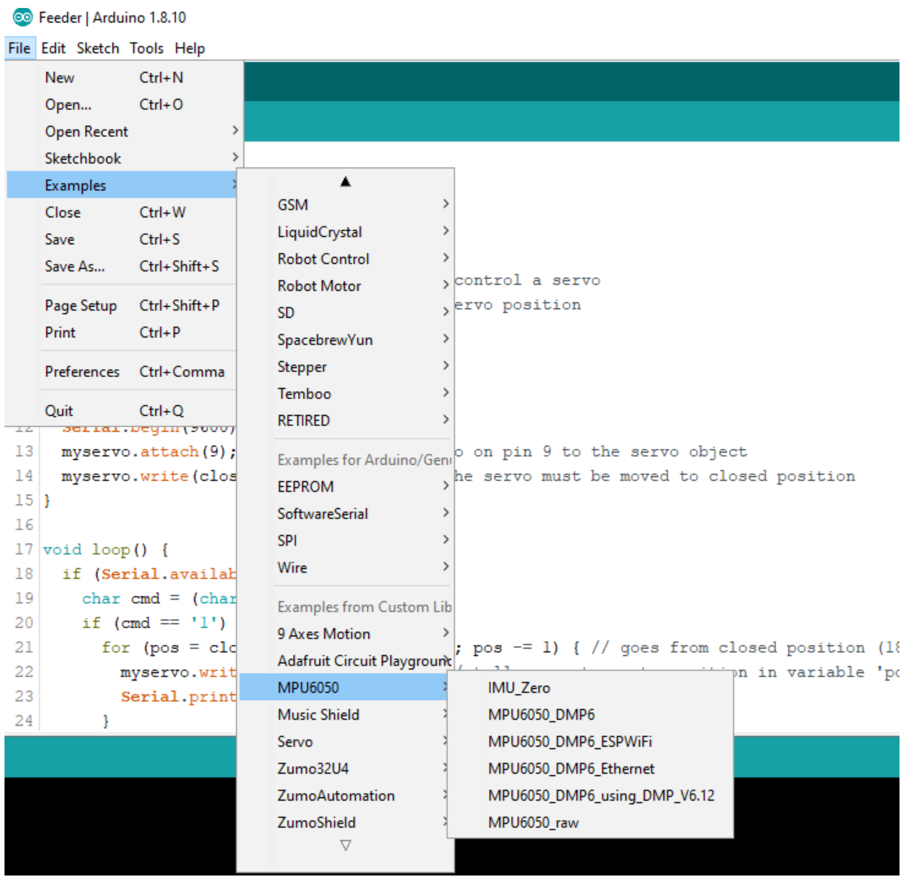
In this section we will be discussing the equipment, tools and techniques chosen for the project, and the justification for some of these decisions.

3.1 Hardware

Hardware was researched in the previous chapter, so this section will give a brief overview of the equipment that will be used in the project and why. During development, decisions were taken to change equipment this will be discussed in depth in the next chapter, so multiple pieces of hardware will be listed.

3.1.1 Arduino Uno

The Arduino Uno was chosen due to it being an entry level device that would be capable of performing the single task that was required. Arduino is open source with a plethora of resources such as the official forum and tutorials. The Arduino IDE (Integrated Drive Electronics) has libraries for most modules and shields created, this allows users to load example sketches which showcase the functionality of a device.



```
                                        control a servo
                                        ervo position

  myservo.attach(9);                    o on pin 9 to the servo object
  myservo.write(clos                    he servo must be moved to closed position
}

void loop() {
  if (Serial.availab
    char cmd = (char
    if (cmd == '1')
      for (pos = clc                    ; pos -= 1) { // goes from closed position (1
        myservo.writ                                 n in variable 'pc
        Serial.print
      }
```

Figure 3-1 Screenshot from Arduino IDE

The above image shows the examples imported for the MPU-6050. These examples show different uses for the device, these examples are open source and be used and edited to suit. It allows the user to test the device quickly and expand on the code provided.

The Uno has the capability to stack components together. Shield stacking is when a shield (external board that has specialist functionality) can be placed directly on top of the Arduino board. The images below show how easy this process is. The ethernet shield pins are lined up with the Arduino Uno then pushed into place.

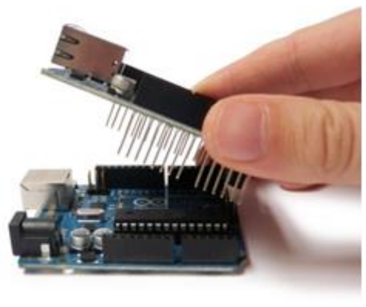


Figure 3-2Image showing boards (mathworks, 2020)

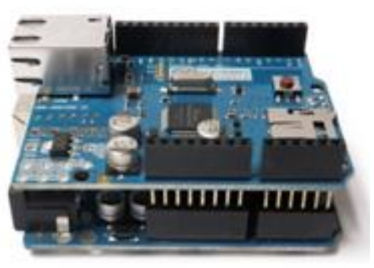
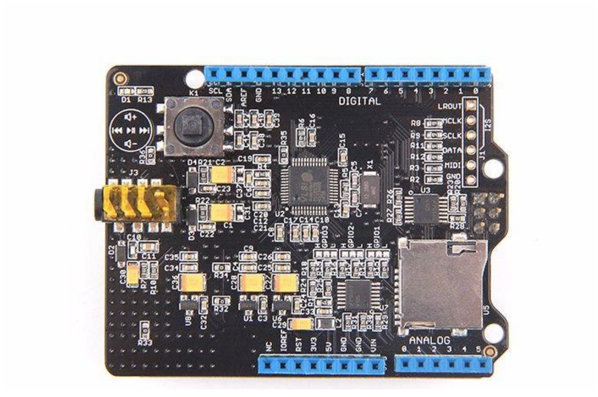


Figure 3-3 Image showing boards stacked (mathworks, 2020)

Once stacked, the examples located in the IDE can be uploaded and utilised instantly. This plug and play style will help with prototyping. The alternative would be to solder devices to the board, this can improve stability but requires experience and can easily render a device unusable if not done correctly.

3.1.2 Storage

As identified in Chapter 2, to store the data recorded from the IMU the prototype will require a component that uses a SD card reader. The ethernet shield was identified as being capable of performing this task.



After finishing research and before development began, Sheffield Hallam University provided equipment that had the potential to be used. One of these devices was the Music Shield V2.0. This device features a SD reader which can be accessed via the SD library, it also features a small joystick (located left hand corner in the image). This joystick can be accessed like a button, providing additional functionality that the ethernet shield could not. For example: instead of running data capture once powered, the button could give the user the ability to start and stop recording data. Unfortunately, the device was faulty when received so development continued with the ethernet shield.

3.1.3 IMU

The decision was taken to use the Arduino 9 Axis Motion Shield. The reason for this was due to the ability to stack the device on top of the Arduino and Ethernet shield.
During research and testing with the MPU6050, it was clear that a lot of noise would be recorded. The MPU6050 would require soldering wires to the board. Because of this, the wires would elevate the IMU and allow it to sway (when in motion.) In hindsight, this problem would be negated if the IMU was soldered directly to the board, but this would be a permanent feature and development progressed with the 9-axis motion shield.

3.2 Tools

With the hardware selected, the attention will move onto the tools required for development.

3.2.1 Arduino IDE

The Arduino IDE is required for creating programs (Sketches) for Arduino boards, this will be used for creating code on the hardware side of the project. Sketches are created in C++ and once loaded to the board run automatically when supplied power.
Due to the open source nature of Arduino, examples can be imported via the library management tool inside the IDE. This will help get an understanding of how the different components work and help combine them to create an overall solution to the data capture objective.

3.2.2 Unity 3D

Unity was considered due to the author's previous experience with the engine and in C# (examples in processing were in Java which the author had not used).
The ability to quickly import models, add lighting, cameras, and to potentially use inbuilt physics also favoured the choice for Unity. For example: in Unity a user can import a model, then assign a Rigidbody. A Rigidbody will give Unity's physics engine control over the model. Therefore, if the application were running and the model was placed off the ground, it would fall due to gravity. This can be implemented without any code. Whereas in processing external libraries would need to be researched and implemented.
Applications created in Unity can be built for multiple platforms, be it game consoles, Mac, PC and Android. Adding UI is seamless by creating new scenes and adding a canvas to add buttons to navigate these scenes.

3.3 Techniques

The Project has two parts: data capture and data visualisation, this next section will discuss how the project was planned.

3.3.1 Overview of Plan

1. Research into devices completed.
2. Configure the IMU sensor to output data.
3. Configure the Storage device to save data.
4. Combine the functionality of Steps 1 and 2.
5. Record the data registered from the IMU in a text document.
6. In the visualisation tool, write a function to extract data that has been saved.
7. Format the data that it can be used in Unity.
8. Manipulate an object with the data recorded, identifying if the data is suitable or not.
    a. If the data is not suitable go back to step 5 to make changes i.e. sample frequency, different types of data (Euler/Quaternion/Acceleration.)
9. Once data is suitable add additional functionality.
    a. Move camera.
    b. Stop playback.
    c. Extract data.
    d. Add UI.
10. Build Project.
11. Reassess to make sure objects have been met.

The first task for the project is to get the multiple hardware components to work together and to meet the aims set out in chapter one (record and capture movement data).
The IMU can record data in Euler angles, quaternions, acceleration in rotation and more, the decision was to begin with Euler angles and assess the decision when the data could be visualised.

Saving data will require using the SD library, this library allows reading and writing of SD cards in the ethernet shield.
The Sd Card will need to be formatted as 32 Fat and be no larger than 2gb in size.
To save the data, a file will be created during setup (a function which is called on start and initialises variables). During the capture of data, the results will be stored in a string and added to the text document via println. The file will need to be opened before writing can commence and to save the data the file must be closed.
To begin with, the results that will be recorded will be a time stamp, yaw, pitch, and roll readings.

Now that data has been saved to a text document on the SD card, the focus will be moved towards visualisation.
The first action will be accessing the file on the SD card and reading each line of text into a string array. Once complete, the string array will need to be looped over to split each line of data into relevant data types i.e. floats and integers. After splitting, the data can be looped over again to form vectors of Euler or quaternions.
To visualise the data, a model will be inserted into the unity scene and its transform (a matrix which represents the object's position, rotation and scale) will be fed the extracted data via a loop.
Development will now consist of checking if the data recorded is valid (manipulates the model as expected).
Changes can now be made to the data capture device, with testing taking place in the visualisation tool. When the data is visualised as intended, development can now move to additional elements such as UI and additional functionality such as buttons for changing the view and pausing the animation.

In the next section we will investigate how development took place and how the project progressed.

4 Development

4.1 Hardware

4.1.1 Setting up the IMU

Development started with the Arduino UNO and the 9-axis shield. The first objective was to get the sensor to work. The IMU arrived with no documentation, therefore it relies on the user to find suitable instructions. As discussed in the chapter 2, the Arduino IDE allows the user to import libraries for modules and shields, therefore importing the NineAxesMotion library from GitHub was a priority.

The examples provided by Arduino had options for using the accelerometers , motion and Euler. Having seen Euler angles working in projects found during research, this was the first example to use.
The plan at this point was to see if the example worked then start adapting the code to suit the project.

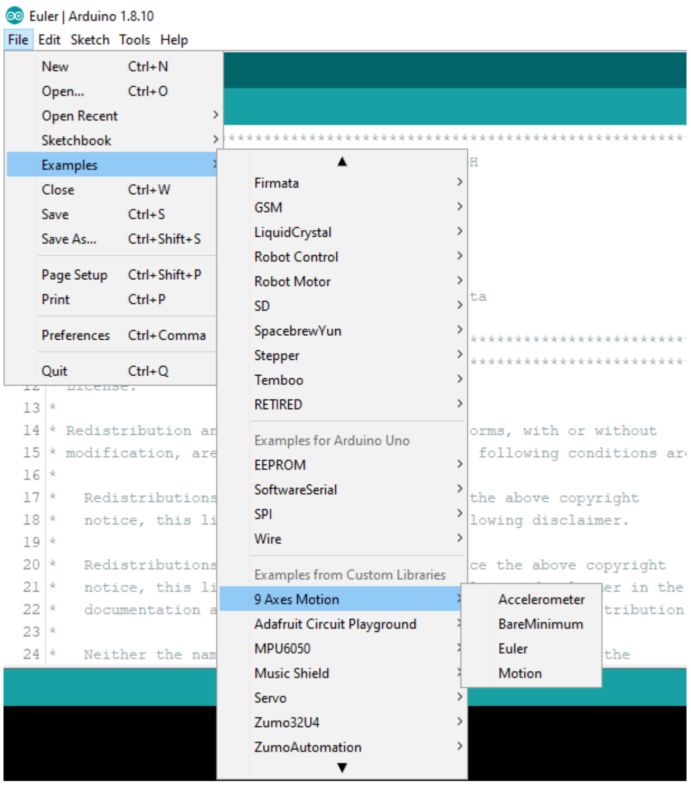


Figure 4-1 Image showing Arduino examples

After loading the code onto the Arduino and running the sketch, the IMU output no readings. Below is a copy of the readings received.

What these readings are showing are:

- Timestamp.
- Heading, Roll, and Pitch in degrees.
- Accelerometer, Magnetometer, Gyro and system calibration settings.

The readings received were all 0 bar the timestamp taken. This meant that there may be an issue

```
Time: 4765ms  H: 0.00deg  R: 0.00deg P: 0.00deg  A: 0 M: 0 G: 0 S: 0

Time: 4805ms  H: 0.00deg  R: 0.00deg P: 0.00deg  A: 0 M: 0 G: 0 S: 0

Time: 4845ms  H: 0.00deg  R: 0.00deg P: 0.00deg  A: 0 M: 0 G: 0 S: 0

Time: 4925ms  H: 0.00deg  R: 0.00deg P: 0.00deg  A: 0 M: 0 G: 0 S: 0
```

with either the board or the IMU. The board being used at the time was a Sainsmart UNO, which is an alternative to the Arduino UNO. The example code was uploaded to a brand-new Arduino Uno and the same outcome occurred.
After researching the lack of readings online, it was advised to run a I2C scanner. What this scanner does is scan the I2C-bus for devices connected to the Arduino. If a device is found, it notifies the user via the Serial window. (Arduino, 2018). After running his sketch, it stated that no device was found. At this stage of development, it was clear that physical computing was not as "plug and play" as the author once thought.
Whilst trying to research the problem, a discussion on the Arduino forum presented the same problem. In the forum, users suggested the same steps that were taken in this project, which had no success. The original poster made a complaint with customer services at Arduino and received a response which had updated code in it. The examples hosted on GitHub by Arduino were out of date and needed updated libraries for the BNO055 sensor (the sensor which the IMU is built on) (Hardach, 2019). The poster uploaded a direct link to a ZIP file which had the updated libraries. Problems like this were common during development, but resources such as the Arduino forum helped greatly.
A link to the discussion can be found here https://forum.arduino.cc/index.php?topic=627645.0

Armed with updated libraries, work could commence on receiving readings from the IMU.
The example provided by Arduino allowed readings of Euler angles by calling:

- readEulerHeading();
- readEulerRoll();
- readEulerPitch();

The page below shows the code that will be ran in the loop function. This code will be started after set-up when power has been supplied to the board. Set-up of the IMU has not been included.

```
void loop() //This code is looped forever{
 if (updateSensorData)  //Keep the updating of data as a separate task
 {
  mySensor.updateAccel();        //Update the Accelerometer data
  mySensor.updateLinearAccel();  //Update the Linear Acceleration data
  mySensor.updateGravAccel();    //Update the Gravity Acceleration data
  mySensor.updateCalibStatus();  //Update the Calibration Status
  updateSensorData = false;
 }
 if ((millis() - lastStreamTime) >= streamPeriod)
 {
  lastStreamTime = millis();

  Serial.print("Time: ");
  Serial.print(lastStreamTime);
  Serial.print("ms ");
  mySensor.updateEuler();        //Update the Euler data into the structure of the object
    mySensor.updateCalibStatus();  //Update the Calibration Status

    Serial.print(" H: ");
    Serial.print(mySensor.readEulerHeading()); //Heading data
    Serial.print("deg ");

    Serial.print(" R: ");
    Serial.print(mySensor.readEulerRoll()); //Roll data
    Serial.print("deg");

    Serial.print(" P: ");
    Serial.print(mySensor.readEulerPitch()); //Pitch data
    Serial.print("deg ");

  Serial.println();
  updateSensorData = true;
 }
}
```

This code looped 25 times per second and will output readings in degrees to the user. At this point all other outputs were tested to verify that the device worked as intended.
With the ability to record rotational and positional data now possible, saving the data is became the next task.

4.1.2 Storage

Storage will be provided by the Ethernet shield. The Music Shield V2.0 was the first choice, but unfortunately this device was not working.
As with the IMU, development started with looking at the examples provided by Arduino. The read write example showed how to create a file and write to it.

With a working example of storage, the next step was to combine the functionality of the two shields.

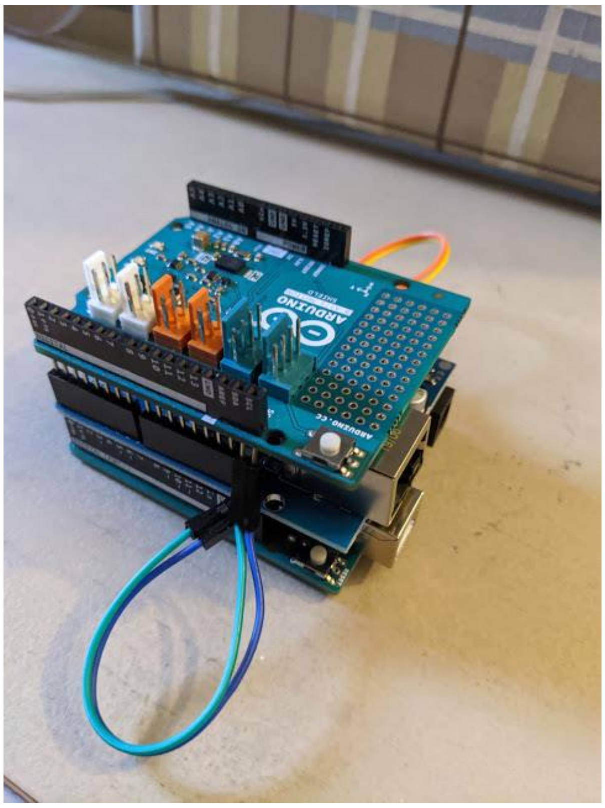


Figure 4-2 Image showing the stacked shields

When stacked, the shields required wires to link the Arduino Uno and IMU. Functionality however was simple to combine. After outputting the readings on screen, we simply combine each component into a string then print that to the text document.

```
    Serial.println();
    String dataString = String(lastStreamTime) + ","
                  + double(mySensor.readQuatX()) + ","
                  + double(mySensor.readQuatY()) + ","
                  + double(mySensor.readQuatZ()) + ","
                  + double(mySensor.readQuatW()) + ","
                  + double(mySensor.readLinearAccelZ()) + ","
                  + double(mySensor.readSystemCalibStatus());
    Serial.println(dataString);
    myFile = SD.open(filename, FILE_WRITE);
    if (myFile)
    {
      myFile.println(dataString);
      myFile.close();
    }
    else
    {
      Serial.println("Could not acess file");
    }
    updateSensorData = true;
  }
```

Once printed to the text file the raw data looks like this:

```
4681,-872.00,-8.00,0.00,16361.00,-0.27,0.00
4725,-872.00,-8.00,0.00,16361.00,-0.26,0.00
4765,-872.00,-8.00,0.00,16361.00,-0.30,0.00
4805,-872.00,-8.00,0.00,16361.00,-0.26,0.00
4845,-872.00,-8.00,0.00,16361.00,-0.25,0.00
4885,-872.00,-8.00,0.00,16361.00,-0.27,0.00
4925,-872.00,-8.00,0.00,16361.00,-0.27,0.00
```

Figure 4-3 raw data from imu

Data is separated via commas with no white space to make it easier to split the data in the visualisation tool.
Now that the hardware can successfully record and save data, the data needs to be visualised. By visualising the data, it will highlight any changes that will need to be made in the capture part of the project.

4.2 Visualisation

To use the data inside Unity it first needs to be converted.

```
void Start()
{
  string path = "F:/test.txt";
  lines = System.IO.File.ReadAllLines(@path);
  times = new int[lines.Length];
  quaternions = new Quaternion[lines.Length];
  accelerationXYZ = new Vector3[lines.Length];
  Vector4[] vector4 = new Vector4[lines.Length];

  for (int i = 0; i < lines.Length; i++)
  {
    var line = lines[i];
    var v = line.Split(',');
    string[] result = v;
    int time = int.Parse(result[0]);
    float fltX = float.Parse(result[1]);
    float fltY = float.Parse(result[2]);
    float fltZ = float.Parse(result[3]);
    float fltW = float.Parse(result[4]);
    float accelZ = float.Parse(result[5]);

    times[i] = time;
    var quat = new Vector4(fltX, fltY, fltZ, fltW);
    Quaternion quaternion = new Quaternion(quat.x,
                                quat.z,
                                quat.y,
                                quat.w);
    Vector3 vector3 = new Vector3(0, accelZ + 0.28f, 0);
    accelerationXYZ[i] = vector3;
    quaternions[i] = quaternion;
  }
```

Figure 4-4 Part of the conversion code for Quaternions

The code on the left is how this is done. On start-up the text file saved to the SD card is located, we then create a sting array of each line in the text file.
After this, the arrays of data types required to convert to are made.

Inside the loop the data is split by the comma, this creates an array of all the comma separated data for that line. i.e.

Line 1:
4681,-872.00,-8.00,0.00,16361.00,-0.27

becomes

Result 1:
4681
-872.00
-8.00
0.00
16361.00
-0.27

These results are then converted from strings to floats or integers.
Finally the floats are combined to create larger data types such as Vectors or quaternions.

*Note the code above, is for converting strings to quaternions. An error in Euler angles was discovered before visualisation was fully implemented. This error will be discussed in the next sections.

4.3 Findings

The project can now record, save, and view data. During this point of development, a problem was encountered. The IMU was reading the heading, pitch, and yaw in degrees (Euler). At first glance this was working correctly.
The sample frequency of the IMU was reduced to make the readings more visible. Below we can see the first set of readings.

```
Time: 4584ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
Time: 4704ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
Time: 4824ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
Time: 4944ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
Time: 5064ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
Time: 5184ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
Time: 5304ms  H: 359.94deg  R: -0.25deg P: 7.50deg  A: 0 M: 0 G: 3 S: 0
```

Figure 4-5 reading 1 IMU sat on table

Here we can see the heading is at 360/0, roll and pitch are offset from 0° because of the position of the IMU when stacked. The accelerometer, magnetometer, and system calibration are not yet calibrated. The gyroscope has been calibrated.
The last four readings are the calibration status. “3” represents calibrated “0” represents not calibrated.
The next set of results **Appendix A Figure 8-1** show the changes in heading over the course of a second and a half. We can see changes from 360° to 322°, roll and pitch also change due to the sensitivity of the device.

Heading operates on a range of 0°– 360°. Pitch operates on a range of -180°-180° which totals to 360 so it can give a full range of movement. Roll however has a range of -90°-90°, which will make it more difficult to take readings from.

The results displayed in **Appendix A Figure 8-2** show what happens to the readings when the IMU is rotated past 90° in the X-axis (roll.)

The negative numbers shown for Roll indicate that the device is being rotated away from the user (although this depends on the way that the user is holding the device), positive numbers indicate rotation towards the user. The readings for pitch are also affected by the changes or Roll. Therefore, a pattern can be devised for reading the change in roll, by factoring in the change in pitch.
Continuing to look at the **Appendix A Figure8-1** a bigger issue has been made present. Once the device has been rotated 90° in the X-axis (Roll), the heading (Z-Axis) loses its reference and resets to ~90°.
When the device has been placed back in its original starting position (on top of a table), the readings in heading are dramatically different.

When comparing theses readings to the ones taken in **Appendix A Figure8-3**, we can see that Roll has changed by 0.06° and Pitch by 3°, which could be due to the placement of the IMU. Heading (Z-axis) now shows a reading of around 87° which is a change of 273°.

With further investigation, it was found that that even though the device was reading a different heading to when powered on, it still incremented/decremented in a full range (0-360°).

4.4 Gimbal Lock

Recording rotational data in the form of Euler highlighted a common problem with using IMUs, Gimbal Lock. Gimbal Lock is a problem when two out of three axes align on a plane. This happens when two gimbals rotate around the same axis. When aligned, a degree of freedom is lost. The diagram below shows what happens axes are aligned.

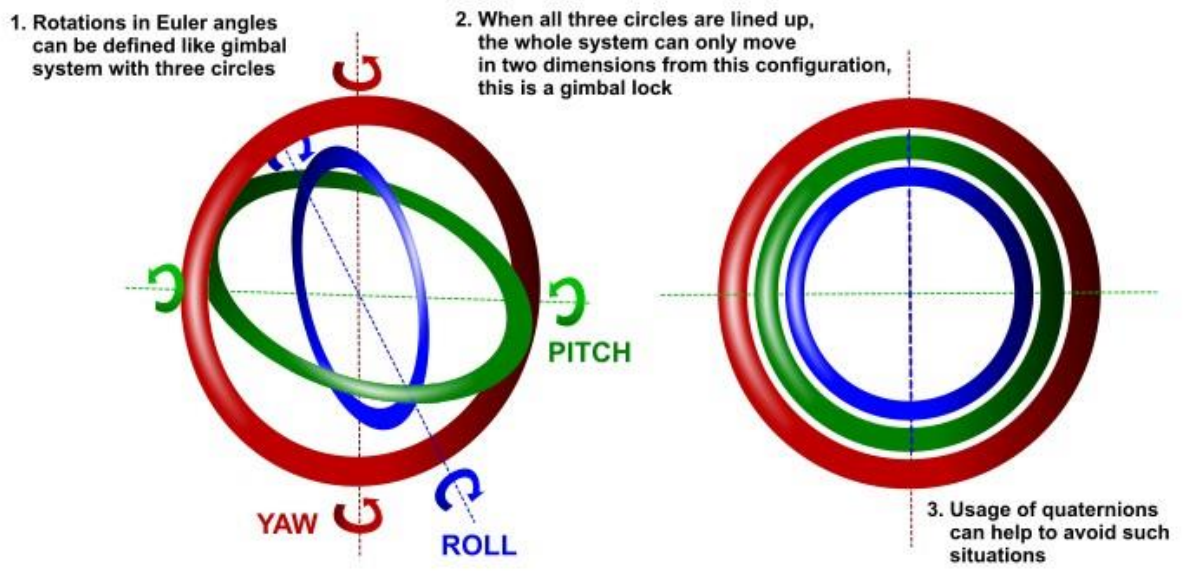
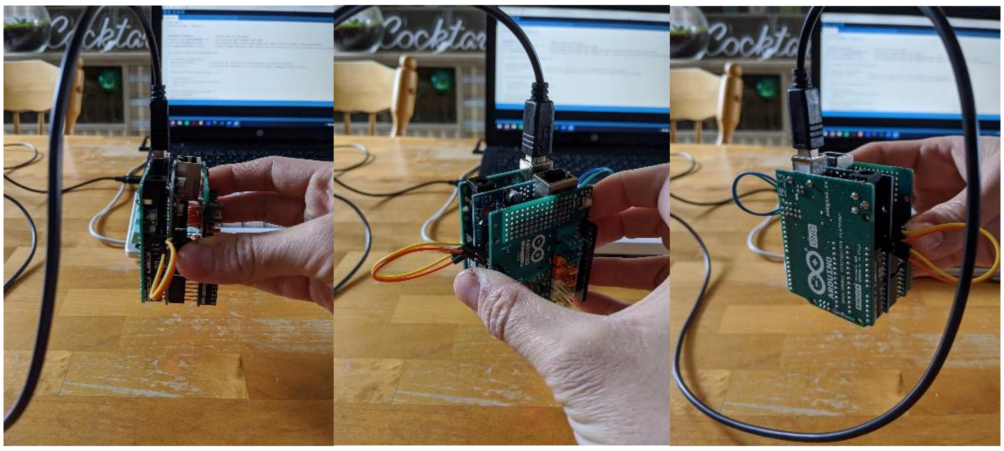

1. Rotations in Euler angles can be defined like gimbal system with three circles
2. When all three circles are lined up, the whole system can only move in two dimensions from this configuration, this is a gimbal lock
PITCH
YAW
ROLL
3. Usage of quaternions can help to avoid such situations

Figure 4-5 Image showing the alignment of axes (Fauvel, 2012).

To illustrate the problem, we will look at the readings received, when the device is rotated 90° in Y-axis and then rotated in X-Axis.



The images above show the IMU rotated 90° in the Y-Axis the rotated clockwise then anti clockwise in X-axis which is now aligned with Y.

Appendix A Figure 8-4 shows the readings from the device during this rotation. Roll (X-axis) does register changes, but not enough to replicate the movements made. Pitch (Y-axis) indicates that the device is being held at roughly 90°. Heading (Z-axis) shows the changes in degrees that are affecting the X-Axis as they are now aligned.

The readings in **Appendix A Figure 8-5** show what happens to roll once the device was rotated back to 0° in pitch, here you can clearly see changes in degrees in the X-axis.
In conclusion, Euler angles cause gimbal lock. They can still be used for devices that have a limited range of motion. For this project, the aim is to get full range of motion.

4.5 Quaternions

Whilst researching the issues with gimbal lock, a solution was found. This was to use quaternions. (Mark, 2017)
Quaternions were discovered in 1843 by William Hamilton (1805-1865) when trying to find a solution to multiplying/dividing 3-dimensional points. A quaternion is a four-dimensional vector i.e. x, y, z, w that will not encounter gimbal lock (If not derived from Euler angles.).
The IMU that is being used can output quaternions, as they are already defined in the NAxesMotion library provided by Arduino, however they are not implemented. After researching, to update the quaternion data from the sensor we use the function “updateQuat();”. To read the quaternion we simply use “readQuatX();” (Where X can be replaced by Y,Z,W).

Below **Figure 4-6** shows the output from a reading of quaternions.

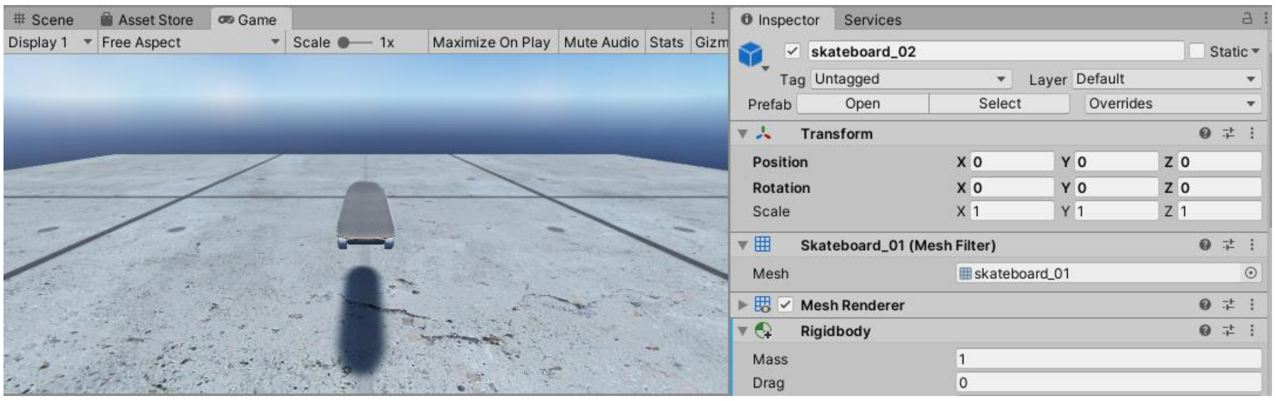
```
4681,-872.00,-8.00,0.00,16361.00
4725,-872.00,-8.00,0.00,16361.00
4765,-872.00,-8.00,0.00,16361.00
4805,-872.00,-8.00,0.00,16361.00
4845,-872.00,-8.00,0.00,16361.00
4885,-872.00,-8.00,0.00,16361.00
```

Figure 4-6 Readings from IMU displaying Quaternions

The first comma separated data is a timestamp, the rest of the data shows values for X, Y , Z, W.
One downside to using quaternion data is, it is hard to read what is happening to the device from the figures alone. To see if these figures are accurate or not, they will need to be loaded into Unity and to manipulate an object.

4.6 Rotating 3D Model

Unity stores all GameObject rotations internally as Quaternions (Unity Technologies, 2020) but converts them to Euler angles for readability in GUI. With this knowledge, it was now time to manipulate the game object with data recorded.
The image below shows the scene (a level of a game) and model of a skateboard, as well as the inspector for the GameObject (Skateboard model in this case.)



skateboard_02

Tag Untagged   Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Skateboard_01 (Mesh Filter)

Mesh skateboard_01

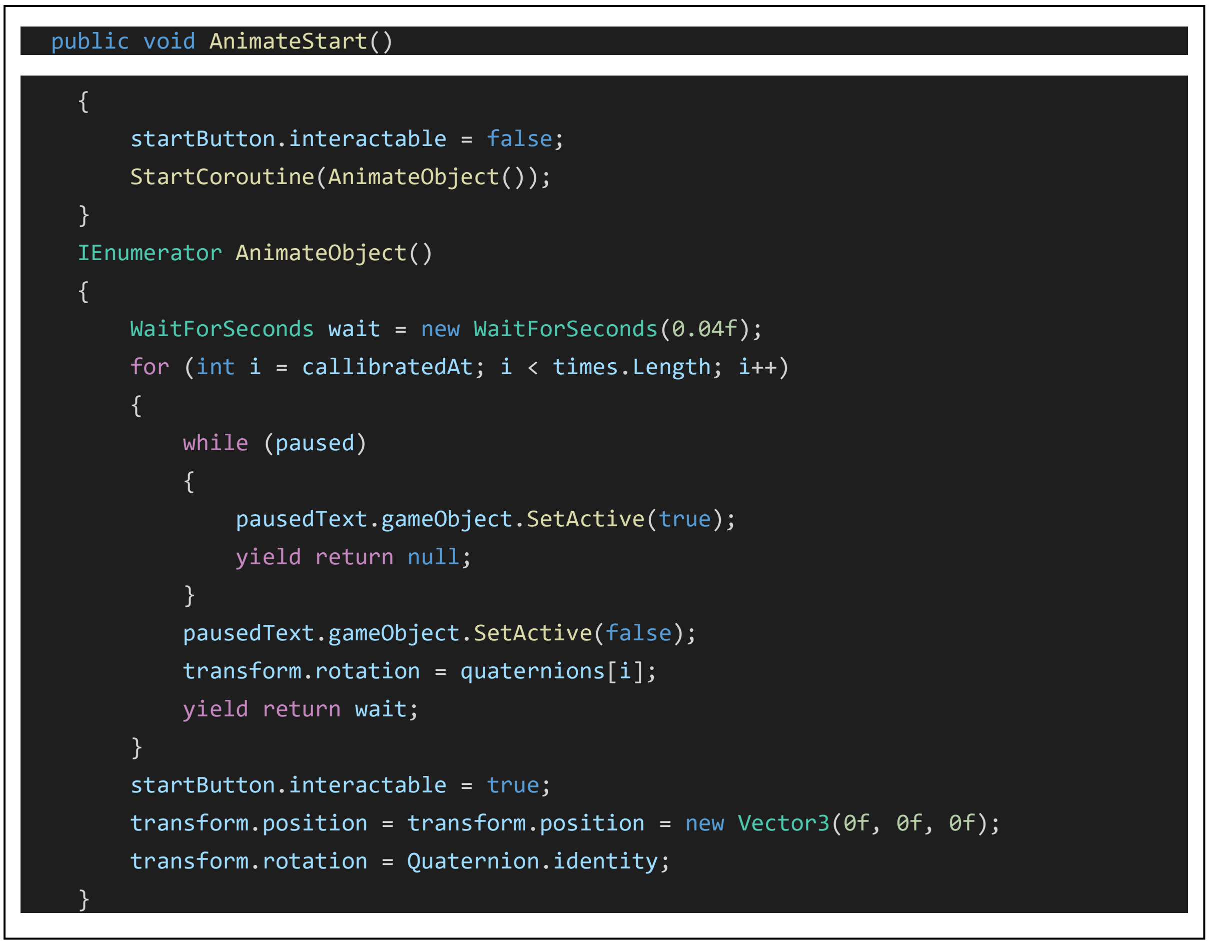
Mesh Renderer

Rigidbody

Mass	1
Drag	0

Figure 4-7 Image showing scene inside Unity

On the righthand side of the image we can see the inspector for the object selected. The inspector is where a user can assign new attributes to an object e.g. scripts, collision detection, material, textures. The top item in the inspector is the transform of an object. This matrix can change the position, scale, and rotation of an object, and this is where the recorded data will used.
For readability, the inspector converts quaternions into Euler angles, but for movement quaternions will be used directly. **Figure 4-4** showed how the recorded quaternion data can be converted from a string to useable data, we will now look to how we can implement that data.

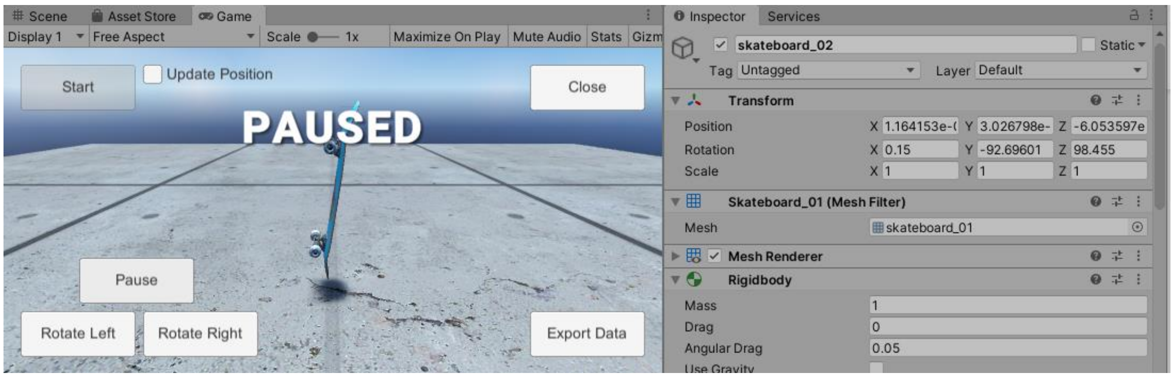


```
public void AnimateStart()
  {
      startButton.interactable = false;
      StartCoroutine(AnimateObject());
  }
  IEnumerator AnimateObject()
  {
      WaitForSeconds wait = new WaitForSeconds(0.04f);
      for (int i = callibratedAt; i < times.Length; i++)
      {
          while (paused)
          {
              pausedText.gameObject.SetActive(true);
              yield return null;
          }
          pausedText.gameObject.SetActive(false);
          transform.rotation = quaternions[i];
          yield return wait;
      }
      startButton.interactable = true;
      transform.position = transform.position = new Vector3(0f, 0f, 0f);
      transform.rotation = Quaternion.identity;
  }
```

Figure 4-8 Image showing the code for rotating an object.

The code sample above shows how the quaternion data is used. When the user presses the start button (part of the UI), it runs a coroutine called AnimateObject. A coroutine is a function which can be paused during execution, this will need to be used to simulate the sample frequency (25hz). If a coroutine was not used, the function would try to apply all the rotations within one frame, and this would mean the rotation will not be displayed in the correctly.
Inside the coroutine the variable wait is declared, in the example above it is 0.040 seconds. This figure is the mode time that is captured when calculating the difference in timestamps recorded. It was later found during testing that the sample frequency causes the playback to be much slower than the real-life recording. At first the mode difference between recordings was used, it was later tweaked by hardcoding an offset to counteract the slower sample rate
The next part of the code shows a for loop, the calibrated at variable is a number which is calculated on start up. This number indicates at what point the IMU was calibrated. Therefore, allowing the calibration movements to be hidden to the user. The next bit of functionality inside the loop handles the pausing of the scene, this allows the user to stop the movement whenever they want. This is done by setting wait duration to null, which means the coroutine will stop until the pause button is pressed again.
The line of code which states “ transform.rotation = quaternions[i];” is where the rotations are performed. This script is attached to the skateboard GameObject, to change the rotation of the object we assign the transform.rotation (as seen in the inspector figure4-12) to the current quaternion defined. The code then waits the previously defined wait time until the next reading can be applied to the mode.

Here are some example screenshots of the rotational changes being applied.



skateboard_02 Static
Tag Untagged Layer Default

Transform

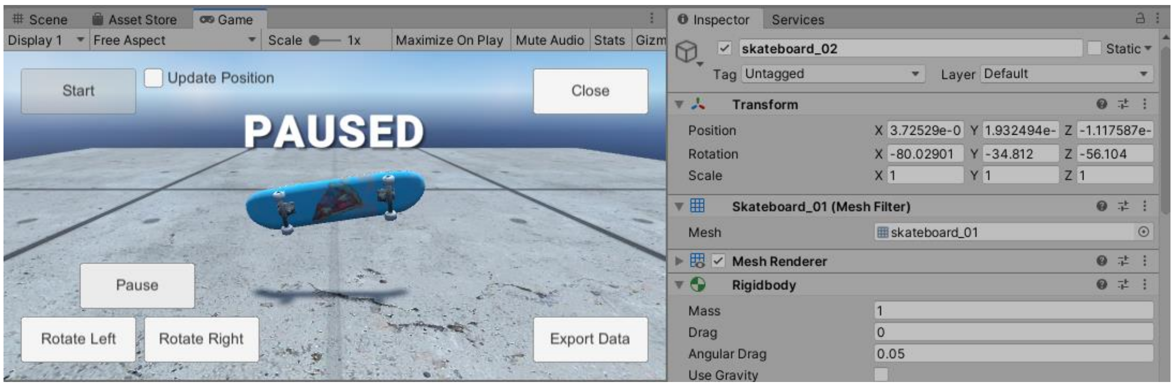
Position	X 1.164153e-(	Y 3.026798e-	Z -6.053597e
Rotation	X 0.15	Y -92.69601	Z 98.455
Scale	X 1	Y 1	Z 1

Skateboard_01 (Mesh Filter)
Mesh skateboard_01

Mesh Renderer

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity

Figure 4-9 Image taken visualisation tool.



skateboard_02 Static
Tag Untagged Layer Default

Transform

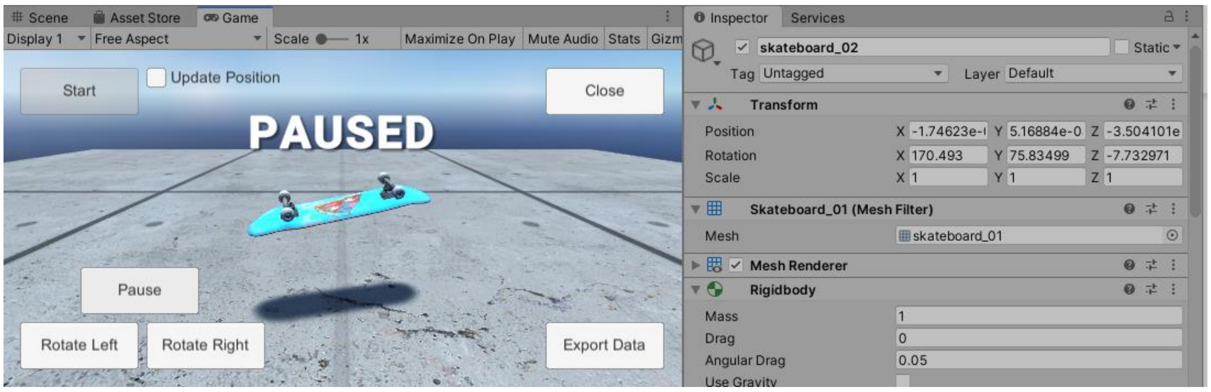
Position	X 3.72529e-0	Y 1.932494e-	Z -1.117587e-
Rotation	X -80.02901	Y -34.812	Z -56.104
Scale	X 1	Y 1	Z 1

Skateboard_01 (Mesh Filter)
Mesh skateboard_01

Mesh Renderer

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity

Figure 4-10 Image taken from visualisation tool



skateboard_02 Static
Tag Untagged Layer Default

Transform

Position	X -1.74623e-I	Y 5.16884e-0	Z -3.504101e
Rotation	X 170.493	Y 75.83499	Z -7.732971
Scale	X 1	Y 1	Z 1

Skateboard_01 (Mesh Filter)
Mesh skateboard_01

Mesh Renderer

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity

Figure 4-11 Image taken from visualisation tool

Rotational data can now be recorded and visualised using Unity. One problem that has been made noticeable is how Unity and the IMU have different arrangements for axis alignment. The IMU registers the Z axis as the axis that comes up out of the board. In terms of Euler angles, this can be called heading. In Unity this is the Y-Axis, therefore the readings have been swapped to accurately align the axis.

4.7 Positional Data

With rotational data now recording and displaying, it was time to investigate positional data. Positional data will be more difficult to implement, and this is due to the lack of reference point for the device. Motion capture and motion controllers use a combination of a camera and a LED or reflective surface to give the device a point of reference in the world around it.

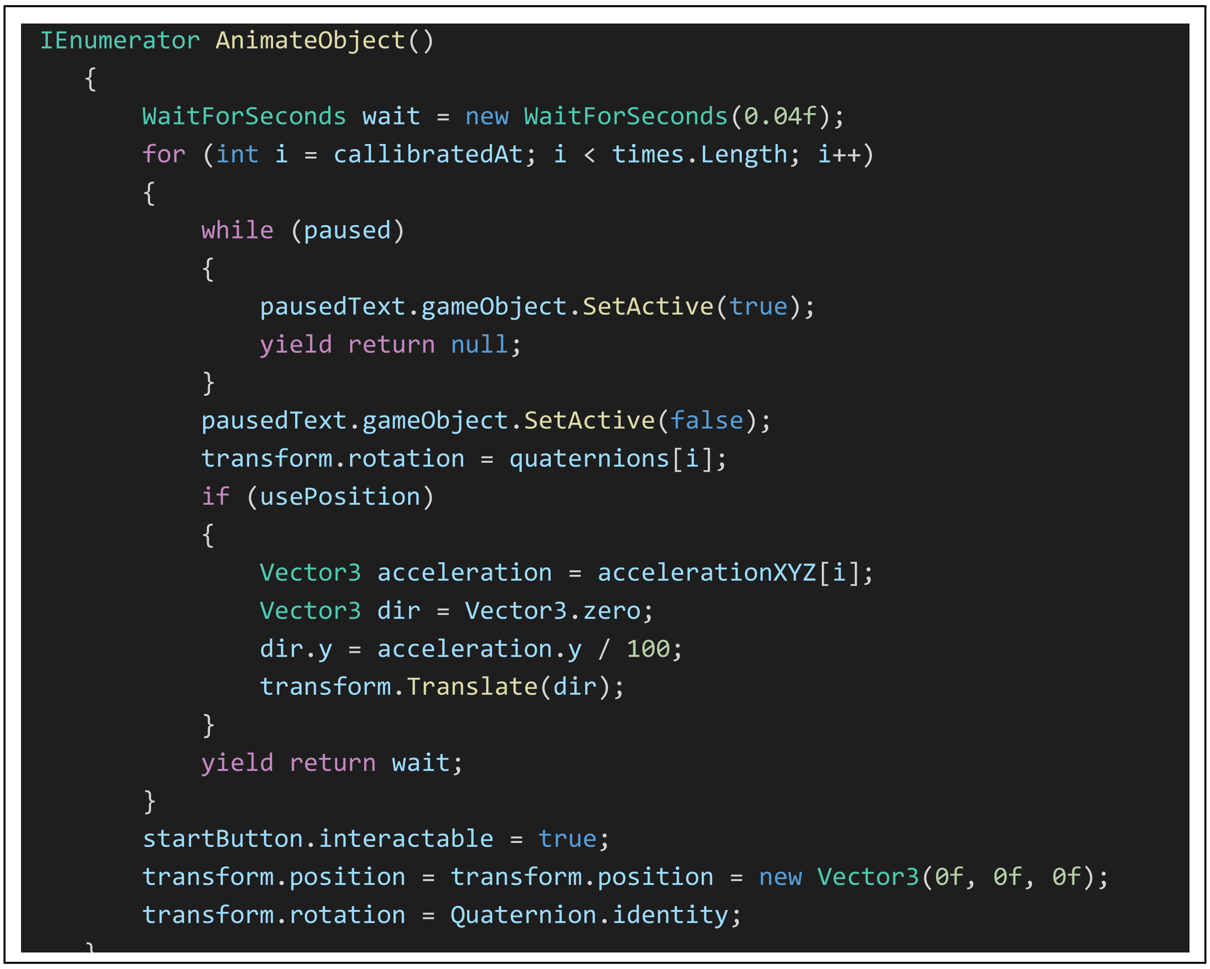
For this project, the IMU will record the linear acceleration in the Z-axis, the other axes will be ignored to help reduce the amount of drift that can occur, and we are only interested in the device moving up.

To record the data we can use the function “double(mySensor.readLinearAccelZ())”. This will then store the acceleration as a double which will then be converted to a vector 3 on the visualisation tool.

In the visualisation tool we convert the string acceleration to a float and then insert these into a vector.

```
  Vector3 vector3 = new Vector3(0, accelZ + 0.28f, 0);
```

The figure added to the acceleration in the Z-axis, is an offset to stop the object from floating upwards. This needs to be done, due to the noise that is generated from the IMU. The device when stable does not read 0.00.

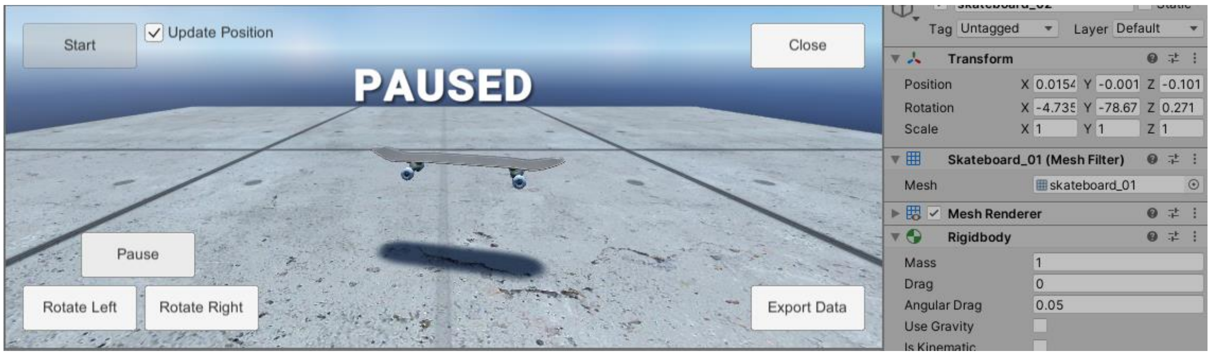


```
IEnumerator AnimateObject()
   {
       WaitForSeconds wait = new WaitForSeconds(0.04f);
       for (int i = callibratedAt; i < times.Length; i++)
       {
           while (paused)
           {
               pausedText.gameObject.SetActive(true);
               yield return null;
           }
           pausedText.gameObject.SetActive(false);
           transform.rotation = quaternions[i];
           if (usePosition)
           {
               Vector3 acceleration = accelerationXYZ[i];
               Vector3 dir = Vector3.zero;
               dir.y = acceleration.y / 100;
               transform.Translate(dir);
           }
           yield return wait;
       }
       startButton.interactable = true;
       transform.position = transform.position = new Vector3(0f, 0f, 0f);
       transform.rotation = Quaternion.identity;
   }
```

Figure 4-12 Code snippet with positional changes added

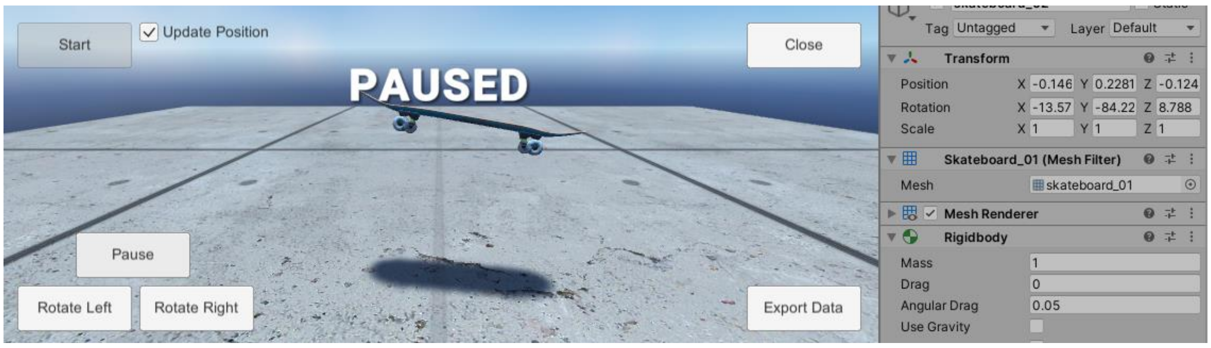
To move the model, a toggle for using movement data is added. Inside the coroutine an IF statement handles the movement with the command transform.Translate. This moves the object in the Y axis by the acceleration supplied. This figure needs to be reduced to create a more realistic movement therefore the amount is divided by 100.

The images below show the model changing position due to the acceleration added to it.



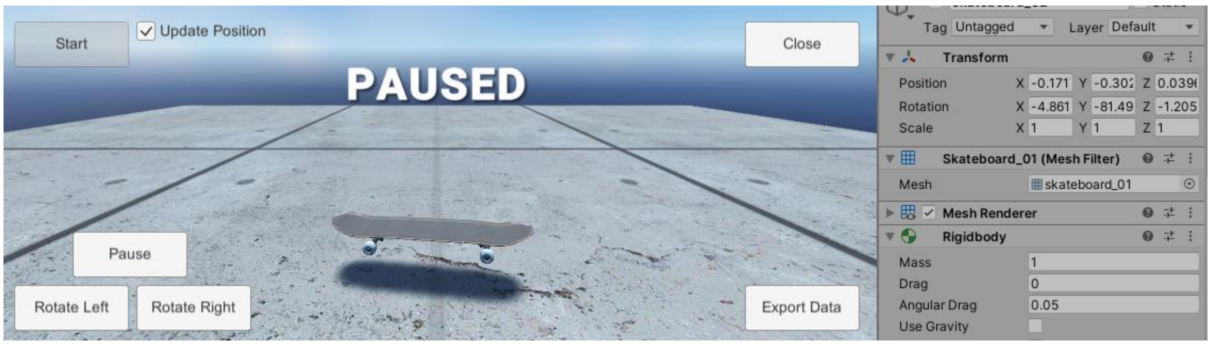
```
Tag Untagged        Layer Default
Transform
Position    X 0.0154  Y -0.001  Z -0.101
Rotation    X -4.735  Y -78.67  Z 0.271
Scale       X 1       Y 1       Z 1
Skateboard_01 (Mesh Filter)
Mesh        skateboard_01
Mesh Renderer
Rigidbody
Mass          1
Drag          0
Angular Drag  0.05
Use Gravity
Is Kinematic
```

Figure 4-13 Image showing model in starting position



```
Tag Untagged        Layer Default
Transform
Position    X -0.146  Y 0.2281  Z -0.124
Rotation    X -13.57  Y -84.22  Z 8.788
Scale       X 1       Y 1       Z 1
Skateboard_01 (Mesh Filter)
Mesh        skateboard_01
Mesh Renderer
Rigidbody
Mass          1
Drag          0
Angular Drag  0.05
Use Gravity
```

Figure 4-14 Image showing model position in Y-Axis increasing



```
Tag Untagged        Layer Default
Transform
Position    X -0.171  Y -0.302  Z 0.0396
Rotation    X -4.861  Y -81.49  Z -1.205
Scale       X 1       Y 1       Z 1
Skateboard_01 (Mesh Filter)
Mesh        skateboard_01
Mesh Renderer
Rigidbody
Mass          1
Drag          0
Angular Drag  0.05
Use Gravity
```

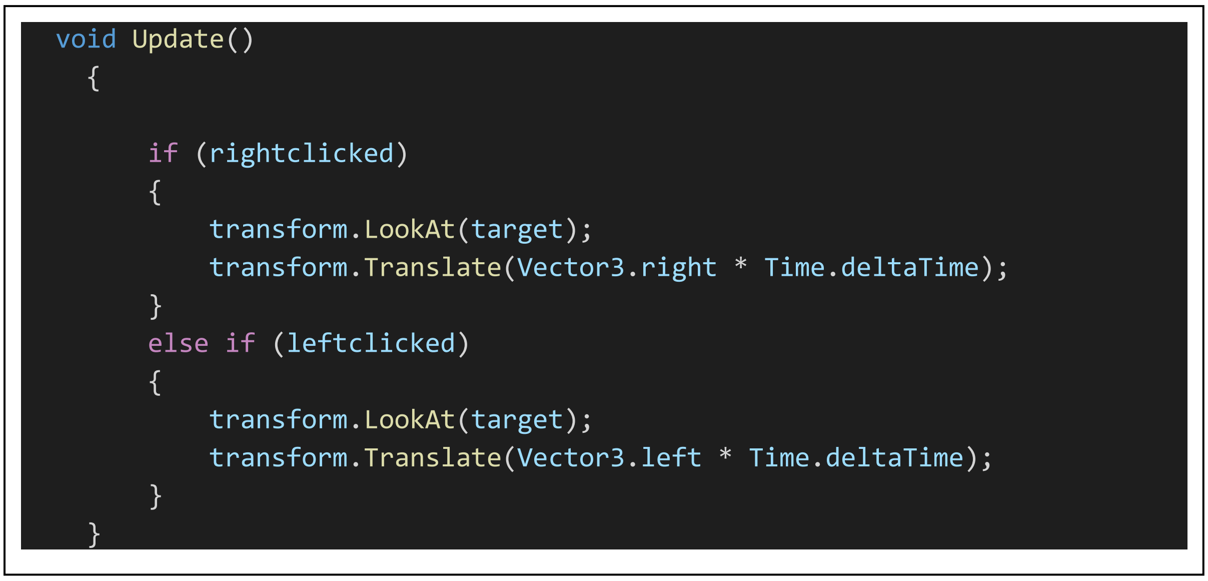
Figure 4-15 Image showing model position in Y-Axis decreasing

The decision was taken to add a toggle for positional changes as they were not accurate enough to simulate the movement. When acceleration was applied to the model during a rotation, the model will traverse the map, making it more difficult to view.

4.8 Supporting Functionality

To increase usability of the application, additional functionality was added. Previously mentioned and seen in screenshots, is the functionality to pause and to toggle positional movement. Extra elements were added to improve functionality, this included the ability to rotate the camera around the moving model, saving data to the device running the app, and general UI.

Rotating the camera around the model is a simple function. The camera is passed a target, in this example the skateboard model is passed. Two buttons have been added to the scene via a canvas (a 2D overlay on the scene where UI elements can be added). When the user presses either left or right the camera forces its view onto the object then rotates around this object until the user stops clicking on the button.

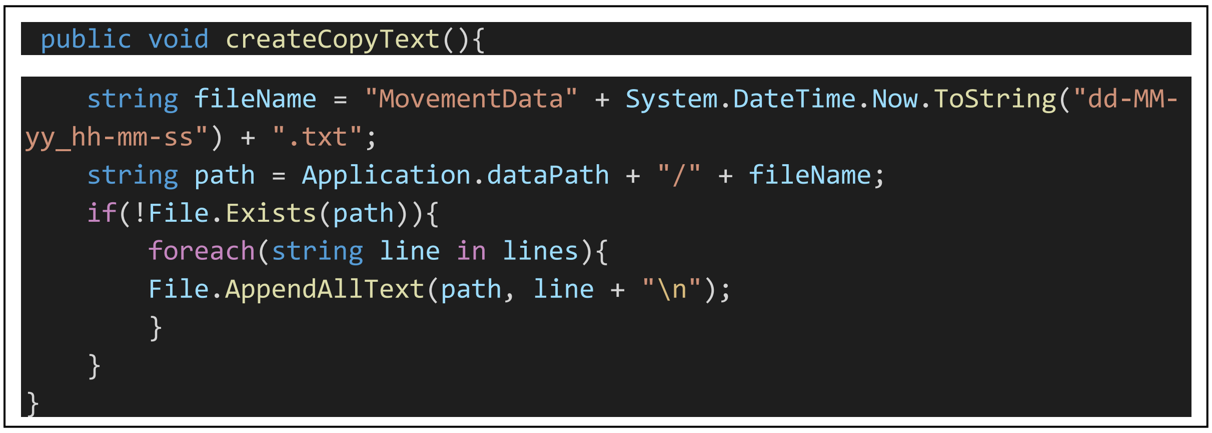


```
  void Update()
    {

        if (rightclicked)
        {
            transform.LookAt(target);
            transform.Translate(Vector3.right * Time.deltaTime);
        }
        else if (leftclicked)
        {
            transform.LookAt(target);
            transform.Translate(Vector3.left * Time.deltaTime);
        }
    }
```

Figure 4-16 Code snippet show camera rotation around skateboard

This code (pictured below) shows how to save the data from the SD card to the application folder. This is so the user can create a record of previous data, that can easily be switched in or out.



```
 public void createCopyText(){

    string fileName = "MovementData" + System.DateTime.Now.ToString("dd-MM-
yy_hh-mm-ss") + ".txt";
    string path = Application.dataPath + "/" + fileName;
    if(!File.Exists(path)){
        foreach(string line in lines){
        File.AppendAllText(path, line + "\n");
        }
    }
}
```

Figure 4-17 Code snippet showing how data is exported.

5 Testing

With the device being able to record movement and positional data (albeit with issues) it was time to start testing the project.
As discussed in previous chapters, this device was created to capture rotational and movement data as a cheap alternative to motion capture. The original plan to test the effectiveness of the device was to compare the data recorded to that of a motion capture studio. Unfortunately, due to the pandemic of Covid-19, this became impossible. Without the use of a motion capture studio, it will become difficult to create quantitative data to evaluate the project's success. Therefore this section on testing will be less formal, but it will try to give a balanced and objective view on the performance of the project.
The testing plan had to change, the next best solution to testing would be to film the device being used and compare to the output from Unity. This tied in nicely with how the project was first hypothesised when skateboarding.

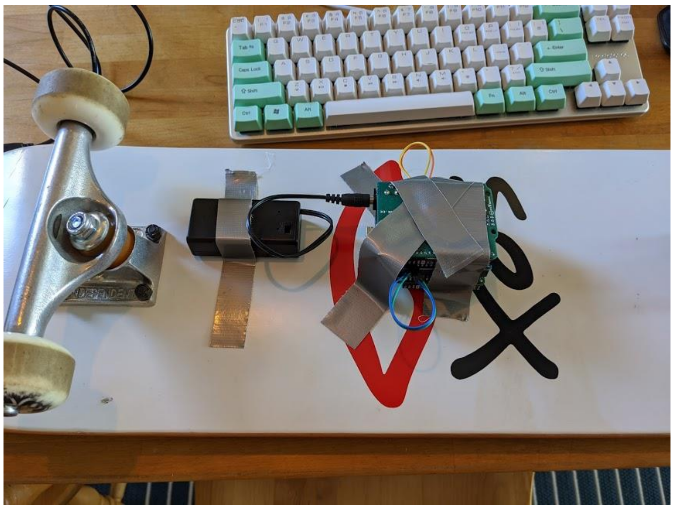


Figure 5-1 Image of device attached to the board.

A lot of the testing was done during development, this meant testing to see if the device was recording correctly and outputting the right data. Testing against movements such as rotating around an axis was also done. What was not clear though was if the recording were in synch with the movements made. By filming the movements when the device was recording it will allow us to see how accurate the device can be.

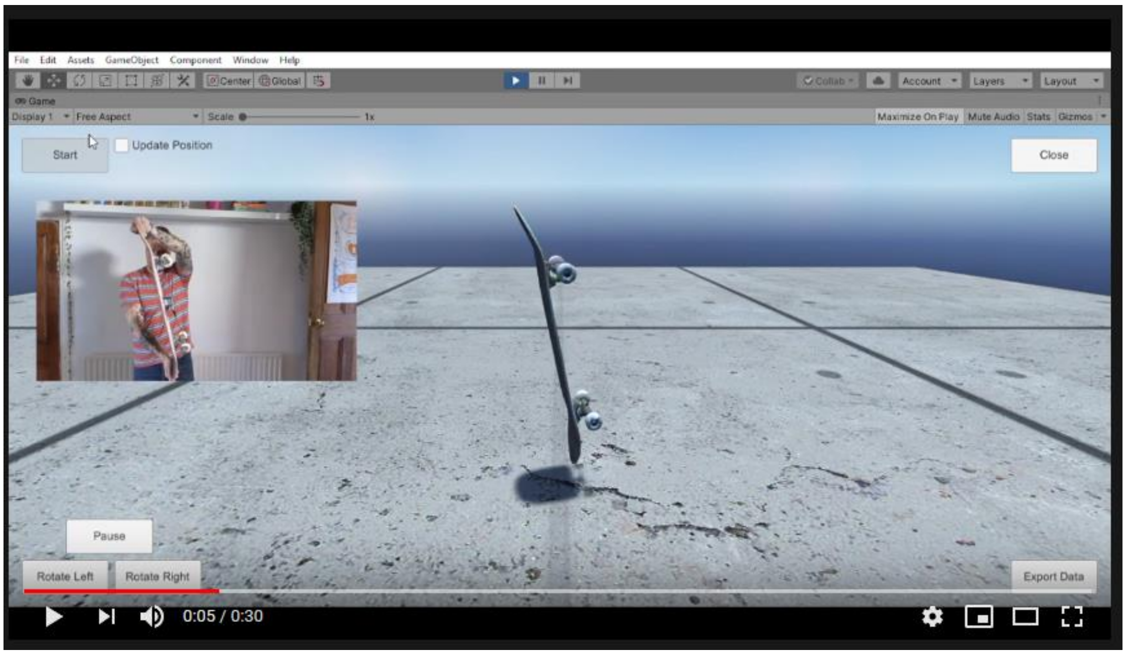

File Edit Assets GameObject Component Window Help
Center Global
Collab Account Layers Layout
Game
Display 1 Free Aspect Scale 1x
Maximize On Play Mute Audio Stats Gizmos
Start
Update Position
Close
Pause
Rotate Left
Rotate Right
Export Data
0:05 / 0:30

Figure 5-2 Image showing movements performed and movement visualised

The videos which can be viewed at https://www.youtube.com/watch?v=Gyud54S6IMw (Rotational example) and https://www.youtube.com/watch?v=mF0-_A0_VM4 (Positional changes).
These videos show the movements performed by the author and how they are viewed in Unity. There is a delay in when videos are synched, this is down to the time it took the screen recorder to start. The side by side video starts whilst the IMU is being calibrated, this is done by rotating the device by 90° in the Y axis. After calibration, we can see that the board follows the real-life movements, but there is a noticeable lag in movements. The reason for this is down to the sampling rate used by the IMU. A higher rate could be used however this would require more testing and may affect the ability to write to the SD Card therefore it was decided that a visible lag was acceptable. The decision to believe this lag was acceptable is the fact that it is only noticeable when doing side by side comparison. When viewing the application separately it is very hard to notice this lag. Understandably this is subjective to the user and without quantitative data to support this claim.

When setting the update position flag to true, the results get more unpredictable. Positional data is gained from using the accelerometer, this means when the device is lifted the speed which it moves is recorded. This force is then applied in Unity to the model and when visualised the board drifts around the scene. This is due to the noise recorded from the IMU, and evidence for this can be seen in the figures saved to the file. Even when stable and resting the device still outputs very small figures. To stop this from happening a filter such as Kalman can reduce noise produced, and this filter can be added to either the recorder or to the visualiser. A filter has not been added to the project due to the amount of research required and time to implement such a complex functionality. Even with Kalman filters to clean the data, the device would need an external reference to make sure the positioning was accurate. For the device this could mean simply adding a LED light. But to track the position of the LED and the position of the object a camera would be required.

The limitation to testing is this project's weakest section, however what has been recorded does indicate the successes and failings of the product. In the next section we will look at these and how we can evaluate on them.

6 Evaluation

This section will focus on how the solution created compares to the initial aims and objectives set out. We will also look at how the solution compares to existing products available today. Finally, we will evaluate the author and what was learnt during the project. The aims and objectives were identified as:

1. To program a device that can capture real-time physics data from a skateboard (or another object).
    a. Record rotational and acceleration data in 3-axis.
    b. Operate with one button. The device will not have a screen, it will need to be turned on then record.
    c. Save and store the data so it can be transferred to an external application.
    d. Be able to run off battery power

2. Create an application which allows the user to see a 3D model of the board (or other object), performing the recorded data (from task 1).

6.1 Product

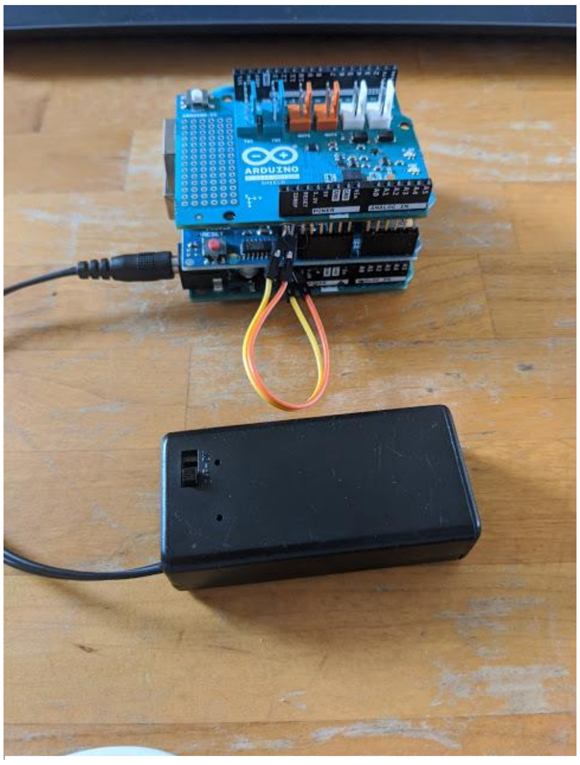


Figure 6-1 image showing final build

The device pictured to the left is the final prototype build. Here we can see the three shields stacked together and the 9v power supply.
The device can record and store data from the IMU, it can be controlled via one button, and it is battery operated. From a physical standpoint the device meets the aims and objectives.
By using stacked shields, the device is stable. The battery is capable of detaching under high force, and the wires that connect the IMU to the UNO would benefit from being soldered. If the device was to be improved beyond a prototype, the device would need casing to keep it protected.

The data capture device can record rotations in three axes, it can also capture acceleration that can give an indication of positional data but not fully capture it. To get better positional data, the device could incorporate a Kalman filter to reduce noise from the accelerometers, but the device would still require a camera to give it a point of reference. This would stop drift in the visualisation tool. Therefore, the device can meet some of the objectives but not all. A user can rotate fully around the axis as the device uses quaternions instead of Euler angles.

In the examples researched (2.2.2 *MPU-6050 & Processing* and 2.2.3 MPU-*6050 & Uduino*) the users used Euler angles to get an object to rotate in a virtual space. These examples were limited in the rotations they could make, meaning they could not rotate 360° in all three axes. These examples did not show any changes to positional data, whereas this device can record movement in the Y-axis. Another improvement that this device makes over the examples above is the how the IMU and Arduino are combined into one device, including storage. In examples 1 and 2 in the literature review, they have wires connecting the IMU to a separate breadboard via wires, which unless soldered would be less stable than the one created.

The application to read the data allows the user to view, export and change the camera angle when viewing. Compared to the method that was used in the introduction, it has more features. It allows a user to pause playback and rotate the camera around the skateboard. Positional data is not as clear as using a camera but, as a proof of concept it does work. With further development this can be improved.

6.2 Self

Self-development is an important process, and the project allowed considerable growth of this. Before beginning, the author had a naive outlook on physical computing. Videos, articles, and other media gave the impression that physical computing offered “plug and play” functionality without the need of skillsets such as electrical engineering, allowing a user to create prototypes that were only limited by the imagination. Soon after receiving the first components it was clear that this project would be more troublesome then originally perceived. At times it felt that the components had taken on a life of their own, working when “they” felt like it. Combined with a lack of documentation, it was easy to become overwhelmed. At times, it felt that this task was impossible. Without the use of debugging software found in more complex IDEs, it felt like a lot of the work was done on a trial and error basis.
Instead of creating the data recorder device in one sweep; it was decided to break down each component into its smallest part. This approach had been used by the author previously when developing larger applications, as it is easy to gain tunnel vision and try to do too much at once. By focusing on one part at a time it allowed the author to gain confidence in development and the nuances that physical computing brings. When each component was able to do its job independently, it was then time to start merging functionality. At this point the author had a good understanding of what each device did and how it worked, making the process of creating the device more manageable.
When receiving a new device, the lack of documentation or code uploaded to it can be daunting. The Arduino community gave great tips and help with suggesting libraries that would allow the components to work. It did not occur to the author that this is very similar to how a programmer uses resources such as stack overflow to solve code-based issues. When this became apparent, the fear of the project subsided moderately (however, plenty of other issues were encountered.).
One clear area in self-development was the confidence that the author had to recognise problems and solutions when encountered. Evidence of this can be found when Gimbal Lock was discovered in the visualisation tool. Previous work with animation in Direct X11 allowed the author to identify the problem and recognise that the solution would be to implement quaternions.

Overall this project has helped the author to become more confident in searching for solutions, as well as becoming vocal in discussions within the community without fear of imposter syndrome.

7 Conclusion

When compared to the aims and objectives, the device has achieved most of the objectives initially set. However, with positional data being inadequately represented, the project is not a complete success. In this section we will look at the limitations and future implementations that would be able to progress the project.

7.1 Limitations

The biggest limitation that can be found has been mentioned. This is the drift that happens when trying to use the positional data. To limit the drift on the visualisation tool, the positional data for X and Z was removed. By doing so, the device can only record data In the Y axis. This means that the data recorded is not truly accurate, but it makes it easier to visualise inside Unity (If all axis were used the model can move out of the scene and not be seen by the user.).
Another limitation that became apparent was the durability of the device. Towards the end of testing the SD card slot on the Ethernet shield broke. This was most likely down to the amount of use it was getting, unfortunately due to lack of funds and delivery times a replacement was unable to be secured before the deadline. To negate this problem an unsophisticated fix was used - duct tape. This allowed further development to continue, but instead of using battery power, a USB cable was required. The reason for using a USB cable was to make sure that the SD card was still in place.

With focus mainly on the physical computing side of the project, the final application could be expanded to make it robust for an end user. At present the user can only access the data file that is hardcoded. Rotations can also get reversed. The IMU axis and Unity's axis are formatted differently. On the IMU Z is up (in line with gravity when placed flat), however in Unity Y has this value. This meant the axis needed to be swapped before formatting for movement. The problem that can happen with this is when the axis is 90° in Y and then rotated around X. The axis will become misaligned. Meaning the rotation performed will be misrepresented. This can be solved by finding the range in which the axis is flipped and flipping them back to the correct axis.

7.2 Future implementations

To further improve this project the following implementations could be made, that could take the device from a proof of concept to a fully-fledged prototype.
Positional recording on the X, Y, and Z axis would allow this device to compete against modern motion capture. Rotational data is already being reproduced at an accurate degree (subjective opinion as unproven with inadequate testing). Accurate positional data without drift would require the use of a constant reference point, this could come in the form of a LED attached to the Arduino board. The trickier side to referencing a LED would be using a camera to track the device and utilising the data.
The device itself can be improved by using a 3D printer to make a case that will cradle the device, keeping it safe from drops and heavy use. Adding other components, such as a buzzer or multiple LEDS to indicate to the user when the device has been calibrated, would help the user identify when the device is ready to use.
The device can only hold one set of data at present. To improve the concept, a system that allows

the user to create multiple recordings could be added. Without buttons or other physical inputs this will be tricky and would require an overhaul on the code currently uploaded. Adding buttons would allow the user to set states for the Arduino such as start recording and end recording, this would allow multiple files to be created and uploaded.

The visualisation tool can be improved by adding the ability to choose where the data is read from, instead of using a hardcoded path. Using a filter such as the Kalman Filter would help reduce the amount of noise being output by the IMU, especially so for the positional data.

7.3 Final Statement

The device does meet many of the initial aims and objectives. At present, the proof of concept design can capture rotational data and a limited amount of movement. But it would require further designs, and functionality to become a true low-cost alternative to simple motion capture. The device in its current state could be utilised to provide motion control to video games in Unity. This is something that the Author will investigate next, being inspired to make a Skateboard game where the player can use a static skateboard (in which the wheels cannot rotate) as a controller. The overexaggerated movement in the Y axis could be used as a feature, to allow the user to jump over higher obstacles in-game.

The project has been frustrating at times, but overall an enjoyable experience. Giving the Author insight into physical computing and new way to implement code.

8 Appendices

A Data

```
Time: 7464ms  H: 359.81deg  R: 0.00deg P: 7.63deg  A: 0 M: 0 G: 3 S: 0

Time: 7584ms  H: 356.00deg  R: -0.31deg P: 7.56deg  A: 0 M: 0 G: 3 S: 0

Time: 7704ms  H: 348.31deg  R: -0.94deg P: 7.44deg  A: 0 M: 0 G: 3 S: 0

Time: 7824ms  H: 342.00deg  R: -1.44deg P: 7.31deg  A: 0 M: 0 G: 3 S: 0

Time: 7944ms  H: 336.38deg  R: -1.88deg P: 7.06deg  A: 0 M: 0 G: 3 S: 0

Time: 8064ms  H: 328.50deg  R: -2.88deg P: 6.50deg  A: 0 M: 0 G: 3 S: 0

Time: 8184ms  H: 322.00deg  R: -3.00deg P: 6.31deg  A: 0 M: 0 G: 3 S: 0

Time: 8304ms  H: 323.63deg  R: -2.81deg P: 6.38deg  A: 0 M: 0 G: 3 S: 0
```

Figure 8-1 readings from IMU showing changes in Heading

```
Time: 20064ms  H: 353.94deg  R: -0.37deg P: 4.50deg  A: 0 M: 0 G: 3 S: 0
Time: 20184ms  H: 355.31deg  R: -6.75deg P: -0.69deg  A: 0 M: 0 G: 3 S: 0
Time: 20304ms  H: 354.19deg  R: -27.87deg P: -2.94deg  A: 0 M: 0 G: 3 S: 0
Time: 20424ms  H: 350.88deg  R: -60.19deg P: -7.75deg  A: 0 M: 1 G: 3 S: 0
Time: 20544ms  H: 350.81deg  R: -82.56deg P: -62.19deg  A: 0 M: 2 G: 3 S: 0
Time: 20664ms  H: 352.81deg  R: -74.81deg P: -152.50deg  A: 0 M: 3 G: 3 S: 0
Time: 20784ms  H: 356.56deg  R: -59.38deg P: -170.38deg  A: 0 M: 3 G: 3 S: 0
Time: 20904ms  H: 7.63deg  R: -43.38deg P: -177.69deg  A: 0 M: 3 G: 3 S: 0
Time: 21024ms  H: 98.06deg  R: -32.06deg P: 179.50deg  A: 0 M: 3 G: 3 S: 0
Time: 21144ms  H: 95.37deg  R: -32.00deg P: -178.88deg  A: 0 M: 3 G: 3 S: 0
Time: 21264ms  H: 88.94deg  R: -55.06deg P: -179.81deg  A: 0 M: 3 G: 3 S: 0
Time: 21384ms  H: 85.69deg  R: -85.19deg P: 4.25deg  A: 0 M: 3 G: 3 S: 0
Time: 21504ms  H: 91.94deg  R: -37.00deg P: 19.31deg  A: 0 M: 3 G: 3 S: 3
Time: 21624ms  H: 95.56deg  R: 7.13deg P: 15.19deg  A: 0 M: 3 G: 3 S: 3
Time: 21744ms  H: 93.56deg  R: 44.19deg P: 17.81deg  A: 0 M: 3 G: 3 S: 3
Time: 21864ms  H: 92.87deg  R: 76.94deg P: 39.13deg  A: 0 M: 3 G: 3 S: 3
Time: 21984ms  H: 93.75deg  R: 77.94deg P: 161.75deg  A: 0 M: 3 G: 3 S: 3
```

Figure 8-2 Reading from IMU showing change in Roll

```
Time: 45264ms  H: 86.44deg  R: -0.06deg P: 4.31deg  A: 0 M: 3 G: 3 S: 3
Time: 45384ms  H: 86.69deg  R: -0.06deg P: 4.38deg  A: 0 M: 3 G: 3 S: 3
Time: 45504ms  H: 86.81deg  R: -0.12deg P: 4.44deg  A: 0 M: 3 G: 3 S: 3
Time: 45624ms  H: 86.87deg  R: -0.12deg P: 4.50deg  A: 0 M: 3 G: 3 S: 3
Time: 45744ms  H: 86.87deg  R: -0.12deg P: 4.50deg  A: 0 M: 3 G: 3 S: 3
Time: 45864ms  H: 87.06deg  R: -0.12deg P: 4.56deg  A: 0 M: 3 G: 3 S: 3
Time: 45984ms  H: 87.06deg  R: -0.19deg P: 4.56deg  A: 0 M: 3 G: 3 S: 3
Time: 46104ms  H: 87.06deg  R: -0.19deg P: 4.56deg  A: 0 M: 3 G: 3 S: 3
Time: 46224ms  H: 87.06deg  R: -0.19deg P: 4.56deg  A: 0 M: 3 G: 3 S: 3
Time: 46344ms  H: 87.06deg  R: -0.19deg P: 4.56deg  A: 0 M: 3 G: 3 S: 3
```

Figure 8-3 Readings from IMU after movements performed.

```
Time: 22707ms  H: 66.25deg  R: -3.00deg P: 99.69deg  A: 1 M: 3 G: 3 S: 3
Time: 22827ms  H: 72.37deg  R: -2.94deg P: 97.81deg  A: 1 M: 3 G: 3 S: 3
Time: 22947ms  H: 79.06deg  R: -2.81deg P: 97.94deg  A: 1 M: 3 G: 3 S: 3
Time: 23067ms  H: 89.94deg  R: -2.13deg P: 98.50deg  A: 1 M: 3 G: 3 S: 3
Time: 23187ms  H: 104.81deg  R: -1.31deg P: 98.44deg  A: 1 M: 3 G: 3 S: 3
Time: 23307ms  H: 127.19deg  R: -1.12deg P: 97.50deg  A: 1 M: 3 G: 3 S: 3
Time: 23427ms  H: 145.63deg  R: -1.56deg P: 97.69deg  A: 1 M: 3 G: 3 S: 3
Time: 23547ms  H: 158.31deg  R: -1.94deg P: 98.69deg  A: 1 M: 3 G: 3 S: 3
Time: 23667ms  H: 163.31deg  R: -2.56deg P: 98.06deg  A: 1 M: 3 G: 3 S: 3
Time: 23787ms  H: 170.69deg  R: -3.50deg P: 96.00deg  A: 1 M: 3 G: 3 S: 3
Time: 23907ms  H: 184.13deg  R: -4.69deg P: 95.37deg  A: 1 M: 3 G: 3 S: 3
Time: 24027ms  H: 210.69deg  R: -5.00deg P: 96.81deg  A: 1 M: 3 G: 3 S: 3
Time: 24147ms  H: 248.69deg  R: -2.13deg P: 98.00deg  A: 1 M: 3 G: 3 S: 3
Time: 24267ms  H: 261.06deg  R: -1.56deg P: 99.69deg  A: 1 M: 3 G: 3 S: 3
```

Figure 8-4 readings from IMU showing rotation around the X-Axis whilst Y-axis = 90°

```
Time: 30987ms  H: 82.19deg  R: 12.88deg P: 1.06deg  A: 1 M: 3 G: 3 S: 3
Time: 31107ms  H: 83.69deg  R: 22.19deg P: 0.06deg  A: 1 M: 3 G: 3 S: 3
Time: 31227ms  H: 84.06deg  R: 25.25deg P: 0.19deg  A: 1 M: 3 G: 3 S: 3
Time: 31347ms  H: 83.56deg  R: 25.12deg P: -0.88deg  A: 1 M: 3 G: 3 S: 3
Time: 31467ms  H: 82.94deg  R: 21.56deg P: -1.50deg  A: 1 M: 3 G: 3 S: 3
Time: 31587ms  H: 80.31deg  R: -2.25deg P: -4.50deg  A: 1 M: 3 G: 3 S: 3
Time: 31707ms  H: 75.50deg  R: -39.50deg P: -9.69deg  A: 1 M: 3 G: 3 S: 3
Time: 31827ms  H: 76.50deg  R: -68.44deg P: -20.19deg  A: 1 M: 3 G: 3 S: 3
Time: 31947ms  H: 81.12deg  R: -86.81deg P: -2.50deg  A: 1 M: 3 G: 3 S: 3
```

Figure 8-5 readings from IMU showing rotations around X-axis whilst Y-axis = 0°

B Project Specification

ProjectSpecification
Mpeake.pdf

9 References

Arduino. (2018, November 14). *Arduino Playground - I2cScanner*. Retrieved March 11, 2020, from playground.arduino.: https://playground.arduino.cc/Main/I2cScanner/

Arduino. (2020, March 05). *Arduino Ethernet Shield 2 | Arduino Official Store*. Retrieved March 05, 2020, from store.arduino: https://store.arduino.cc/arduino-ethernet-shield-2

Arduino. (2020, March). *Arduino Playground - MPU-6050*. Retrieved March 08, 2020, from playground.arduino: https://playground.arduino.cc/

Arduino. (2020, March 04). *Arduino Uno Rev3 | Arduino Official Store*. Retrieved 03 04, 2020, from Arduino Official Store: https://store.arduino.cc/arduino-uno-rev3

Banzi, M. (2011). *Getting Started with Arduino* (2nd ed.). US: O`Reilly. Retrieved March 04, 2020

chauhannaman98. (2020). *Using 9 Axes Motion Shield With Arduino*. Retrieved march 8, 2020, from Instructables: https://www.instructables.com/id/Using-9-Axes-Motion-Shield-With-Arduino/

Electronics Hub. (2017, Decemeber 6). *What are the differences between Raspberry Pi and Arduino?* Retrieved 03 04, 2020, from Electronics Hub: https://www.electronicshub.org/raspberry-pi-vs-arduino/

Fauvel, C. (2012, August 10). *Avoid Gimbal Lock for Rotation/Direction Maya Manipulators*. Retrieved from Around the Corner: https://around-the-corner.typepad.com/adn/2012/08/avoid-gimbal-lock-for-rotationdirection-maya-manipulators.html

Hardach. (2019, July 22). *9 Axis motion Shield : BNO055*. Retrieved March 11, 2020, from forum.arduino: https://forum.arduino.cc/index.php?topic=627645.0

HobbyTransform. (2020). *MPU6050: Arduino 6 Axis Accelerometer + Gyro - GY 521 Test & 3D Simulation*. Retrieved March 08, 2020, from instructables: https://www.instructables.com/id/MPU6050-Arduino-6-Axis-Accelerometer-Gyro-GY-521-B/

Mark, H. (2017, March 22). *Capturing IMU Data with a BNO055 Absolute Orientation Sensor - Projects*. Retrieved March 13, 2020, from allaboutcircuits: https://www.allaboutcircuits.com/projects/bosch-absolute-orientation-sensor-bno055/

mathworks. (2020). *Connect Arduino Ethernet Shield to Arduino Hardware - MATLAB & Simulink - MathWorks United Kingdom*. Retrieved March 10, 2020, from mathworks: https://uk.mathworks.com/help/supportpkg/arduino/ug/connect-arduino-ethernet-shield-to-arduino-hardware.html

Nayyar, A., & Puri, V. (2016). A review of Arduino board's, Lilypad's & Arduino shields. *3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 1485-1492). Bharati Vidyapeeth, New Delhi as the Organizer of INDIACom - 2016. Retrieved March 04, 2020, from https://ieeexplore.ieee.org/abstract/document/7724514

Nield, D. (2017, 07 23). *All the Sensors in Your Smartphone, and How They Work*. Retrieved from Gizmodo: https://gizmodo.com/all-the-sensors-in-your-smartphone-and-how-they-work-1797121002

Processing. (2020). *Processing.org*. Retrieved from Processing.org: Processing.org

Rowberg, J. (2020, February 21). *i2cdevlib/Arduino/MPU6050 at master · jrowberg/i2cdevlib*. Retrieved from GitHub: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

seeedstudio. (2020). *Music Shield V2.0*. Retrieved March 10, 2020, from seeedstudio: https://www.seeedstudio.com/Music-Shield-V2-0.html

Teyssier, M. (2020). *Connect a IMU sensor to Unity (MPU-6050)*. Retrieved March 08, 2020, from https://marcteyssier.com: https://marcteyssier.com/uduino/projects/connect-a-imu-to-unity

Unity Technologies. (2020). *Unity - Manual: Rotation and Orientation in Unity*. Retrieved from docs.unity3d.com: https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html

Zhang, Y., Fei, Y., Xu, L., & Sun, G. (2015). Micro-IMU-based motion tracking system for virtual training. *34th Chinese Control Conference (CCC)* (pp. 7753-7758). Hangzhou: IEEE. doi:10.1109/ChiCC.2015.7260871