**Designing and making a judging aid for canoe slalom**

**CM3203 – One Semester Individual Project – 40 Credits**

**Final report**

A red sign with white text

Description automatically generated with medium confidence

**Cardiff University School of Computer Science and Informatics**

**Author: Harry Batchelor (C18163377)**

**Supervisor: Dr Ian Cooper**

**Moderator: Dr** [**Yipeng Qin**](https://pats.cs.cf.ac.uk/!user_info?u=scmyq)

**11/05/2022**

# Abstract

Many professional sports utilise video footage and replay systems to aid the decision-making or the role of umpires and referees. Cricket was one of the earliest sports to implement this technology with Hawk-eye, introduced in 2001. With bowls travelling at ~90mph, it can be hard for TV spectators or umpires to follow the ball. Hawk-eye allows spectators and umpires to watch replays of these fast-paced instances to allow them to make better judging decisions on close-call moments. Canoe slalom is no different. With athletes trying to reduce their times by fractions of seconds and executing turns within millimetres of the gates, a judge can only see so much from the bank.

The purpose of the project is to look at the feasibility of making an accessible system for canoe slalom judges to allow them to make better split decision calls. The project also aims to evaluate the success of the proof of concept and verify how it could be upscaled and used in competitions across all levels.

# Acknowledgments

I would like to thank my supervisor, Dr Ian Cooper whose friendly advice and guidance throughout the whole project allowed me to work through the hurdles I encountered in the project.

Table of Contents

[Abstract 2](#_Toc103155539)

[Acknowledgments 2](#_Toc103155540)

[1. Introduction 4](#_Toc103155541)

[2. Background 5](#_Toc103155542)

[2.1. Method 5](#_Toc103155543)

[2.2. A beginners guide to canoe slalom 5](#_Toc103155544)

[2.3. Current implementations 6](#_Toc103155545)

[2.4. Tools used 7](#_Toc103155546)

[3. Approach 9](#_Toc103155547)

[3.1. Fundamental system behaviour 9](#_Toc103155548)

[3.2. Live System 10](#_Toc103155549)

[3.3. Test System 11](#_Toc103155550)

[3.4. System Flow 11](#_Toc103155551)

[3.5. Expected limitations of the system 12](#_Toc103155552)

[4. Implementation 14](#_Toc103155553)

[4.1. Live System 14](#_Toc103155554)

[4.1.1. Overview 14](#_Toc103155555)

[4.1.2. Live Camera 14](#_Toc103155556)

[4.1.3. Live Data and charts 14](#_Toc103155557)

[4.2. Test System 15](#_Toc103155558)

[4.2.1. Overview 15](#_Toc103155559)

[4.2.2. Database management 16](#_Toc103155560)

[4.3. Network Overview 17](#_Toc103155561)

[5. Results and evaluation 17](#_Toc103155562)

[5.1. Evaluation of Live system 18](#_Toc103155563)

[5.2. Analysis of results 18](#_Toc103155564)

[5.2.1. Analysis of results test system 18](#_Toc103155565)

[5.3. Analysis of Live system 24](#_Toc103155566)

[6. Future work 25](#_Toc103155567)

[7. Conclusions 25](#_Toc103155568)

[8. Reflection on learning 26](#_Toc103155569)

[9. Appendix 26](#_Toc103155570)

[9.1. Glossary of Canoe Slalom terms 26](#_Toc103155571)

[10. References 27](#_Toc103155572)

# Introduction

As with any top level sport, canoe slalom races often come down to the finest margins, with paddlers trying to find the smallest margins possible to help them win races. This has evolved over time with boat manufacturers using the lightest materials to reduce weight and allow the paddler to move through the water faster. In more recent years, manufacturers such as PeakUK have been developing garments such as the racer ST2020 [8], as seen in figure 1, which take the vital buoyancy aid on the front of the paddler and integrate it into the spray deck, this allows for the paddlers chest to be unimpeded with the traditional buoyancy aid. Although these new designs only eliminate a few millimetres from the chest, it allows the paddlers to get even closer to the poles. This new innovation helped the PeakUK athletes at the 2020 Tokyo Olympics to win a total of 9 out of the 12 possible medals[9].



Figure 1 - PeakUK Racer ST2020, showing now foam on the front, to allow for tighter turns around the poles. Image Source: [8]

With the margins being as small as mentioned above, the pressure that the judges are under to make accurate calls is huge, especially when a penalty means the difference between 3rd and 4th place, as with the Olympic games this summer (ICF 2021)[6]. My project aims to create a product that will aid judges in making these difficult decisions while the race is still going. The product will not solely rely on video evidence which is typically only available at the higher levels of the sport. Replaying and analysing the video footage can be a lengthy process whereas my product will give instant information from a range of cheap cameras and sensors. This project will look to design a solution for one set of poles with a view that it can be scaled up to cover a 25 gate (50 pole) course.

Although the project has been developed with the main use being for the judges at the competitive levels of canoe slalom, it could also be used as a training tool for all the athletes in the sport. Canoe slalom coaches have to perform two jobs simultaneously, one as a judge making sure their paddler makes it through the gates and secondly as a coach to see what the paddler can improve on. If my project was used, the role of the judge can be taken away and the coach can focus on coaching. The coach will also be able to gain more data for quantitative feedback from the various sensors and outputs on the poles, allowing them to gain insight into which poles their paddler can get even closer to.

As of the 2018 season, the ICF (International Canoe Federation) has brought in a new system for video judging, to be used in the World Cup competitions and the World Championships [5]. The ICF are relatively late in introducing this technology, compared to tennis or cricket who have been using video judging since 2001. In canoe slalom there is still a very large team of judges, some watching the live video feed of the run and some reviewing the replay of any contested decision. The introduction of video footage and replays has not reduced the manpower of judging for canoe slalom. Whereas in a sport such as rugby, there is only one video judge who runs it all.

My proposed project would utilise the judges already on the river bank. These judges would first watch the paddler come through the gates and if they then decided there is a close call they can refer to the mobile device which will have data for them to review instantly. All of this will reduce the pressure on the judges, reduce the number of judges needed, and cut down on the time taken to review close calls.

# Background

### Method

After completing my background research and assessing the different possible outcomes of my project, I have decided to make a web application for the judges at a canoe slalom race. The app is developed so that judges need no prior computing or canoe slalom skills to use the product. On this web app they will be shown a live stream of the camera from the pole, as well as 6 different graphs relating to the x, y, z coordinates of each pole. This will form the ‘live’ system. Along with the live system I have also produced a ‘test’ system. This test system takes the acceleration data from the two poles, stores it in a database to allow the user to query the data from the past. The bank side judges would not have access to the test system, rather this would be controlled by the head judge, who would need some technical training. Both these systems, and the technology behind them, will be discussed in more detail in the later sections. This project is a proof of concept, as such there are a few limitations and factors that have been overlooked to allow the project to reach the minimum requirement. These limitations will be discussed later on.

### A beginners guide to canoe slalom

The main focus of this project isn’t to teach the reader about the intricacies of canoe slalom, but an understanding of this sport is useful to fully understand the project. A full list of canoe slalom terminology can be found in the appendix.

In canoe slalom, athletes race down a course approximately 200 metres long consisting of between 18 and 25 gates, of which 6 must be upstream gates. Paddlers can incur time penalties of 2 seconds for having a touch on the poles, this can be from any part of the paddlers body, or any of their equipment. The paddler can also gain a 50 second penalty for not having their complete head, or for not having any part of their boat go in between the two poles, or for navigating the course in the wrong direction or order. [5]

Typically for a canoe slalom race the whole course is split into multiple sections. Within those sections there will be two judges focussing on a set of 4-5 gates between the two of them. They will watch the paddler head down the course and report back to the race control what the results are for their set of gates. This will be in the form of numbers, for example a “zero, zero, fifty, two” means that the paddler has successfully navigated the first two gates, missed the third gate, and touched the fourth gate. At the top level of the sport, if there is a disagreement between the two judges or they aren’t sure on decision what to give they can ask the video judge to review the footage they have.

### Current implementations

Currently at the top level of the sport there is no product used that is similar to this proposed product. However one paper by Ringwood et al, 2015, have looked into this problem before. Their aim was to produce a gate hit detection system for canoe slalom. Ringwood [12], has a slightly different technique, using an inertial measurement unit, with an Arduino microcontroller, which sends data via Bluetooth to a PC. Once the data has been received on the PC they use a classifier to detect if there has been a hit or not. Although both products want to achieve similar goals, we have gone about it in very different ways.

Ringwood’s[12] implementation used multi-layer perceptron (MLP) artificial neural network (ANN) to classify the hits and produce a result. During my first meeting with Dr Cooper, we discussed where our expertise lie and felt that we were better off taking the project in a different direction than [5]. This gave us the creative freedom to produce something new, which to my knowledge hasn’t be done in canoe slalom yet.

On a typical canoe slalom broadcast such as the Olympics or the World Cups, the spectators never get to see any of the judging or video judging taking place. The viewer might get some slow motion replays of the gates, if there is time between each paddler or if the broadcaster chooses to show it, but none of the formal judging is shown. The only aid that the viewer has, is a small icon in the corner of the screen showing if the paddler has hit or missed the gate. A typical broadcast with such icon can be seen in figure 2.



Figure 2 : Typical canoe slalom world cup broadcast, with icon showing penalties. Source [11]

Although my project isn’t aimed towards viewers of canoe slalom, there is scope for the data produced by the system to be displayed alongside the replay of the gate that is under question, this will be discussed more in sections below. This would be something similar to ultra-edge in cricket where they have the sound wave produced, helping the viewers and umpires make their decision. An example would be figure 3 below. It is reasonable to think that if the system is designed towards judges with no prior skills in canoe slalom, then the average viewer would be able to understand the graphs and make a decision for themselves.

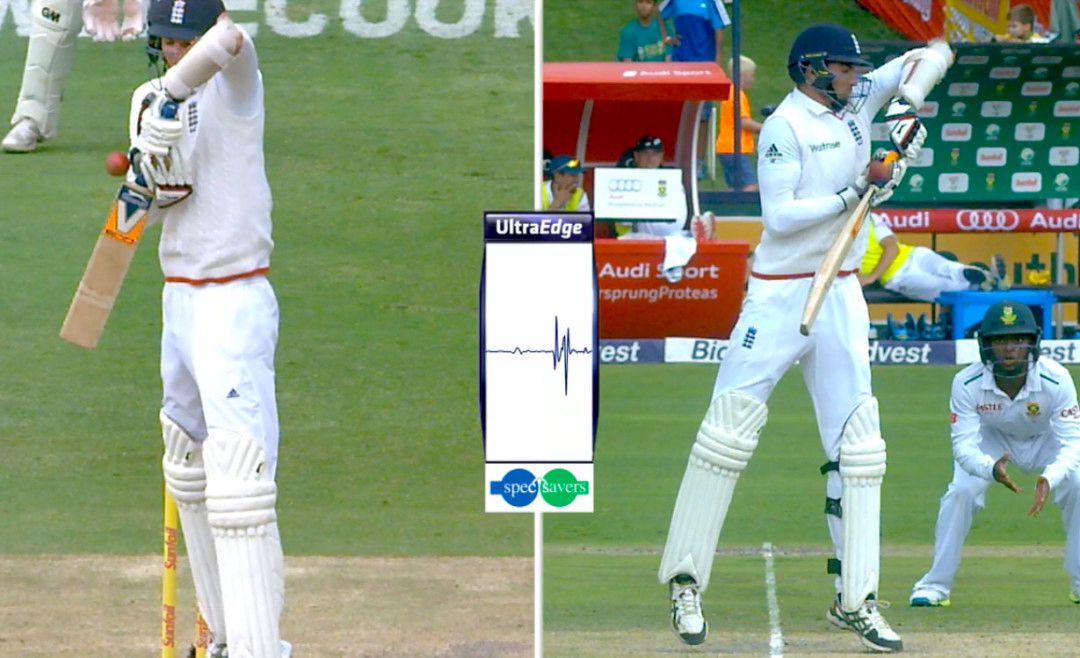


Figure 3: Ultra-edge used in cricket [13]

### Tools used

#### Flask

The first and arguably the most important tool I used was Flask. This is a popular web app framework for Python. I decided to use Flask over other frameworks such as Django, NodeJS or other web frameworks as I already had some experience with using Flask in other projects. Flask is also lighter weight compared to Django, which was important as everything would be running off a Raspberry Pi, which are notorious for struggling to run heavy programmes with only having 1gb of ram[10], more discussion about the Raspberry Pi will come later.

Although Flask might not have been the best framework to use, I believe that the time required to learn a new language, such as JavaScript (for NodeJS) would have significantly set back my timeframe and made the proof of concept project harder to complete.

Flask was able to provide the basic functionality to get all the required data onto a web app. This was a vital criteria for the project, without the use of a web app the judges would need a lot more training on the programme to be able to use it effectively.

Alternatively I could have developed an app for android or IOS, but I felt that a web app gave more flexibility for the judges to decide what device they would like to use. For example, some judges might struggle seeing the detail needed for a close call on a small screen such as a phone and would prefer to use a laptop.

#### SQLite

SQLite has been another backbone to the project. SQLite is a small and fast database engine. The database allowed me to easily link the data from the accelerometers to the Flask server referred to above. Storing the data within the database also allowed for easy retrieval for plotting graphs to show changes in the accelerometer data. There is also scope for being able to collect and plot old data in a competitive sense, for example if a racer has challenged one of the rulings.

I picked SQLite compared to other database engines such as Neo4J or MongoDB as I already had a good understanding of SQL. In a 12 week project, where time is a key factor, I didn’t want to be spending extra time learning a new database format, as such I stuck to SQL. In addition SQLite is a fully self-contained system meaning it is very simple to set up and get running quickly, and it doesn’t take up much storage or processing power. If we compare these benefits to MySQL, another popular database management tool, we can see that they’re both very similar. Although SQLite is designed for more lightweight systems. Nevertheless, MySQL and NoSQL based systems have some major benefits which will be talked about later on in section 6.

#### Charting

Once I had the basic functionality of the programme down I started to work more with Matplotlib. This is a Python library built upon NumPy for Python, it allows the user to create all sorts of static, animated, and interactive graphs. There are a few other alternatives to Matplotlib, but none based around Python, and had the ease of use I wanted. I have also used a charting programme based on JavaScript which I will move onto now.

HighCharts is very similar to the aforementioned Matplotlib, this time based around JavaScript. HighCharts has been used to take the accelerometer data from the Live system to the Flask site. I have used the animation functionality of HighCharts to get a set of graphs which are atomically refreshing at a high rate to allow the judge to effectively see a live update of the accelerometer reading in a user-friendly form.

During development I struggled to get Matplotlib to produce these auto updating, live graphs, which is why I had to use a combination of HighCharts for the live system, and Matplotlib for the test system.

In an ideal world I would have kept using Matplotlib, due to how powerful it is, yet how easy it is to use. In the real world, this would have meant that I would have potentially struggled to finish development of the live system on time.

The reason why I didn’t choose to use another charting package such as ChartJS or any of the other JavaScript based charting software was because I found some good examples online while looking through the documentation of HighCharts and an online tutorial that was similar to what I needed[14] both of these gave me confidence in HighCharts, and that it could do what I actually wanted it to do. I could have done some more comparisons between the hundreds of different JavaScript charting packages, but I was running out of time to get everything finished, and so I picked one which I understood and could see a path to a complete solution.

#### VNC viewer

VNC viewer has been a key productivity booster for this project. VNC viewer enables me to remotely access the Raspberry Pi, removing the need to have a keyboard or mouse plugged in. This would have been a key problem as the slalom pole can often be a couple metres above the water. VNC is automatically installed on all Raspberry Pi making it an obvious choice. Another alternative that we could have used would have been SSH, this allows for command line execution on the Pi. However, there is no graphical user interface but there is with VNC. This was a key downfall of SSH as I knew I would be needing to look at different graphs and different code . Therefore SSH would lower my productivity as I would need to upload everything to Git each time I made a change. The Nano command could have worked as an alternative for this, but when working with larger files, the luxuries of a quality source code editor, such as Visual Studio Code[7] come into account. Some of these luxuries are things like automatically closing brackets, or div tags, having things colour coordinated and showing when variables haven’t been accessed and why. All of these helped make the coding section of the project slightly easier and more productive.

#### Hardware

The main hardware I have used during the project is a Raspberry Pi 3 model B, two Adafruit ADXL345 accelerometers and one Raspberry Pi camera. I decided this was this was going to be the best solution after researching different microcontrollers. The main competition would have been using an Arduino.

I decided to go with the Raspberry Pi due to it being most similar to what I was already using. The graphical interface, and ease of compatibility, and the fact Arduino code is written in C++, where as Raspberry Pi’s have a range of languages you can use. I felt my time was better off spending more time coding the project rather than learning a new language such as C++ .

Another benefit of the Pi was that my supervisor Dr Ian Cooper already had one spare which I could borrow for the project, allowing me to swiftly move onto the coding stage. At the time of my initial research, there was a worldwide shortage of Raspberry Pi, meaning there would have been at least a 4 week wait, which would have caused an unacceptable delay to the project.

The Adafruit ADXL345 were used as they are readily available for a low cost, a pack of 4 costing approximately £8. The accelerometer also has a variable sensitivity level, set by the user. The sensor can either be set to +-2g, +-4g, +-8g or +-16g[1]. This was important for the project, as we had to think about balancing the sensitivity to pick up the smallest of touches, against the sensor being too sensitive to the wind. In [12], Ringwood et al discussed the various different sensors that could have been used, discounting aural, electrical contact, mechanical, optical and motion for various different reasons, all of which are valid. However, Ringwood never gave any disadvantages to accelerometer and gyroscopic systems. Which is why, when comparing the price (£30 vs £8), accessibility, and ease of use I decided with the ADXL345.

#### Ajax

Ajax, or **A**synchronous **J**avaScript **A**nd **X**ML, allows web pages to be updated asynchronously. This has been one of the backbones for the Live system. Ajax has allowed me to retrieve data from the client, the Raspberry Pi, and send it to the Flask server without having to update or refresh the page. This has been implemented in the retrieval of the sensor data, and also the retrieval of the camera live stream. The retrieval of this has been done with the HTML GET protocol, this will be talked about in more detail later on. Without Ajax none of the live data or any form of auto-updating graph would have been feasible.

# Approach

## Fundamental system behaviour

I started off the project by defining the basic requirements for the system, they are:

* The final product must use some form of sensor to track the movement of the pole
* The solution could use a camera to help the judges
* The solution must be presentable and easy to use for the judges
* There should be a web app as the final solution
* On the web app there should be a graph to show the changes of acceleration
* The graph could show rate of change
* The system could produce a judgement as to whether the pole has been hit or not

The first design decision I came to was to use accelerometers as the main sensor in the solution. I found that the Adafruit ADXL\_345 sensor was cheap, small, and compatible with the Raspberry Pi, even coming with its only library. The documentation on the device was also excellent and very detailed, along with other people’s implementation of the accelerometer. All of these things combined made it an easy decision to choose accelerometers and especially the ADXL\_345.

Ease of use was a major underlying system behaviour that I defined. By defining my target audience and users, it made it easier to clarify what the final solution would be. The primary users would be the judges standing at the water’s edge at canoe slalom races. They are often just volunteers, who possibly have not raced canoe slalom before, nor had any deep understanding of computers, Raspberry Pi’s, or the accelerometers I used. I therefore had to focus on making the user experience as simple and easy to use as possible. This was particularly challenging as it was hard to find the right balance between overcomplicating things and not showing enough detail.

A web app is essential for the final product to allow the it to be easy to set up and use for the wide variety of users. 87% of people in UK own a smartphone[2] meaning that there is a high chance that all of the judges would have access to a smartphone. Accessing a web app on a smart phone is a relatively easy step, requiring the user to open a web browser and enter a web address. The system was designed to run as a web app on portable devices, such as phones and tablets. I made sure to take this into account when developing the product, by using bootstrap, which allowed me to make the live system responsive to what size device was used.

I could have implemented a system to work out the rate of change on the accelerometers and put this on the graph, this would have also led nicely onto the judgment call. I decided against doing this in the end as I wanted to focus more on keeping the design simple and straightforward. Having extra data, such as the rate of change would require more training for the judges and would possibly overcomplicate things. I also over-ran on my expected development time so I did not have the time to research how to implement the rate of change or the judgment call, even if I had wanted to.

## Live System

Originally, I hadn’t planned on developing a live system but this is exactly what I ended up developing. After my first full test of the solution I found that I was not satisfied with the product. This is because it was lacking the functionality of the basic programme. There was no way for the judge to see data from the paddler without having to run another programme themselves. Therefore something extra was needed.

From here I developed a live system, which showed a live feed of both the camera and the sensors that are available, via a web app on a mobile device. This now forms the main system that the judges will use to help them in making their decisions. Therefore I continued with the aim of it being minimalistic and a simple design, as talked about in the section above.

## Test System

The test system is mainly aimed towards a large competition, with scope to be used by a canoe slalom coach. It has no live feed to it. Once again the interface for this system is very simple with only two buttons, and a table showing the last data entry. The two buttons allow the user to enter the time of when they believe there was a hit. After the race has finished, or the training session is over, the user can then collate all the recorded hit times into a PDF document. The document shows a graph of the next 500 data entries from the time of hit, this is repeated for data entry there was a hit. This would work well in a race if there was a dispute as all the data entries from the sensors have been logged and can be queried at any time. A combination of both the live and test systems should lead to a better judging experience for everyone.

## System Flow

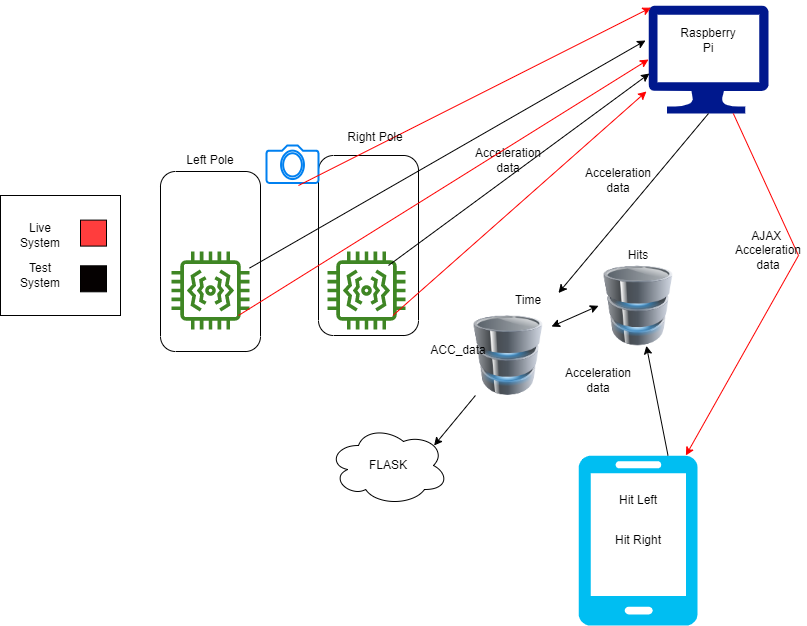
The network diagram for both systems and information flowchart for the test system.

Figure – 6 Network diagram for both systems

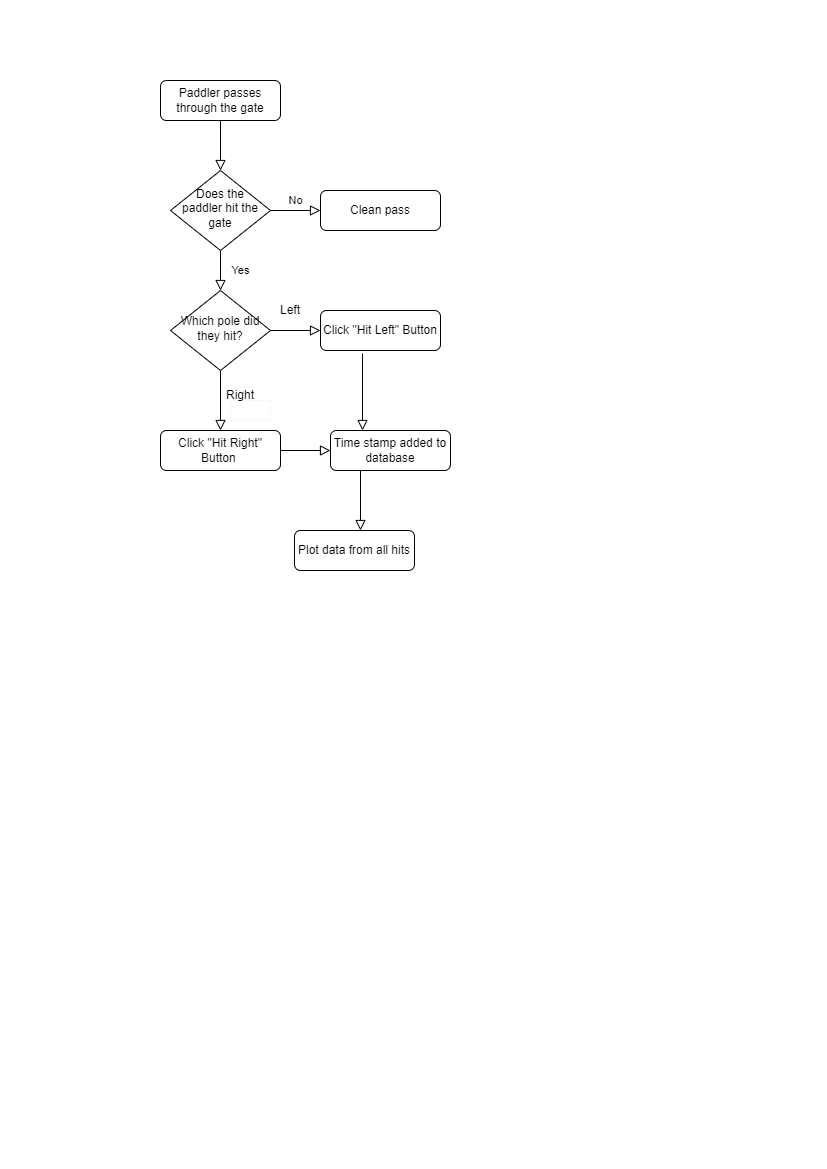


Figure – 7 Information flowchart for the test system

## Expected limitations of the system

As with any project there will always be some limitations, but due to the nature of this project being a proof of concept there are some major limitations we have had to overlook. For example, canoe slalom courses are normally a minimum of 200m[5] with many being longer and often in a straight line. For example, the courses can be very linear and several hundred meters long. Having such a large course produces problems for my solution as a good internet connection is required to run the web application and live stream the sensor data quickly. One possible solution to this would be to set up a mesh network between the different Pi’s to spread the internet between all the different sections of gates.

Another limitation that I have had to overlook in this proof of concept is the effect the variations in weather could cause. I didn’t know if the wind would make a big difference in registering a hit on the slalom pole. I considered a possible solution being to weigh down the pole so the wind wouldn’t move it, but then the little hits, which is the main activity that is being judged would be a lot harder to detect. As I will discuss later on, during the testing there was some wind, but it did not appear to impact detecting big or small hits. However if there was high wind this could interfere with the detection, but further testing in a variety of weather conditions would need to be undertaken to be sure.

The next limitation is scalability. I have developed the system for one pole running off one Raspberry Pi, but to be used in the real world with 18—25 different gates[5] having a Raspberry Pi for each individual pole would be very expensive. Therefore, the challenge is to wire the sensors from each individual pole to the relevant Raspberry Pi in the sector, here is where I would need to find a Bluetooth sensor that the Pi could connect to from about 50m away.

Increasing the scale brings along other challenges as the volume of data that is being processed increases. Currently the Raspberry Pi copes with reading the data from sensors, but if we were to increase this by 3, 4 or 5 times then the Pi is likely to slow down and struggle to output the data required. Volume testing would be required to prove this.

A similar problem occurs with the test system when scaled up, but this time the bottle neck will be the database. Adding just one more sensor would mean that the current SQLite database will not be able to handle the volume of data and will lock itself. If we then reduced the number of times per second that the sensors were writing to the database, the graphs would not be as accurate and potentially hits would be missed. The solution to this would be to move to a heavier database that could handle the amount of data coming in, and the amount of data needing to be stored.

A rather significant limitation of the system is to do with the sensors. There is a pay off with the sensors to get the correct sensitivity setting. You need to balance the sensitivity to be able to pick up the smallest of hits, but then it means the sensors become more susceptible to background disturbances such as wind.

The final limitation of the scaled up system is the design. Although this was not taken into account when developing the system, if the final product is to be produced for a customer the final design and user interface would need to be thoroughly planned and researched. For this proof of concept, I have the Raspberry Pi taped to the wire overhanging the white water with the cables from the sensors coming up through the pole, a mock up is seen in figure 4. Therefore the Raspberry Pi can only be a short distance away from the pole. In an ideal world I would need to find a wireless accelerometer, with a range that would cover 4 or 5 slalom poles, which would be a total of around 50m.

Figure 4: Shows the testing set up in my back garden

# Implementation

## Live System

## Overview

Having a live, up to date, system was possibly the most important part of the whole project, unfortunately I did not see this until the latter half of the project. Having established that I needed the live feed as well as the hit data and graphs I had planned to combine all of this into the one system. After testing and thinking more in-depth about the flow of data through the network I realised it was easier to separate the two.

Realising I needed to separate the two systems I conducted some research into the best way to livestream the data from the sensors to a web app. I decided to use HighCharts and Ajax as discussed above in section 2.3.

## Live Camera

The first functionality to be implemented was the camera. I decided on this first as I had already seen cameras in use on many popular implementations for similar products during my primary research. One such implementation was a camera live stream using Flask by user EbenKouao[3]. This GitHub repository was the foundation upon which I based my camera solution. Using existing Python modules such as OpenCV, an open source computer vision library, and imutils, a set of functions to make image processing more convenient, the repository makes livestreaming the camera to a Flask site very easy and convenient.

Overall the repository saved me a lot of time and effort, allowing me to have the camera set up and working from the first day of development.

## Live Data and charts

The live streaming of the data to the charts was slightly harder and required my own solution. From my earlier research I had already ruled out matplotlib from earlier livestreaming the data, this led me to search for JavaScript based alternatives such as HighCharts. In an ideal world I would have kept using matplotlib due to it being more simplistic and easier for me to understand. But HighCharts would ultimately allow me to do what I had planned. More on this can be found above in section 2.3.

The solution that I came up with was using Ajax. I would read the data coming off the accelerometers onto its own route in the Flask server. This was done using the Adafruit Python library that supports my specific accelerometer.

One major challenge I had to overcome was using two accelerometers at a time. This is because as standard the Raspberry Pi only comes with 1 I2C bus enabled. This meant that wiring one accelerometer to the Pi was simple and very easy, but two was a new challenge. To overcome this, I had to go into the configuration file of the Raspberry Pi and add lines of code to enable the new I2C buses. I had to make sure when I was doing this to never use bus 0 or 2 as they have other uses, as Pi hats. These new I2C buses also meant that I needed to change the standard Adafruit Python script to read the data and tell the new script which buses to read from.

Once I had overcome the problem of using two accelerometers at once I moved onto getting the data to the Flask site. I already had working static graphs using matplotlib, so went straight onto producing the live graphs. Researching ways to do this, I found a couple solutions which were similar to what I was aiming for and as such helped influence my decision making, such as identifying Ajax as a good way to get the data from the sensors, this is done by using the GET method within HTML.

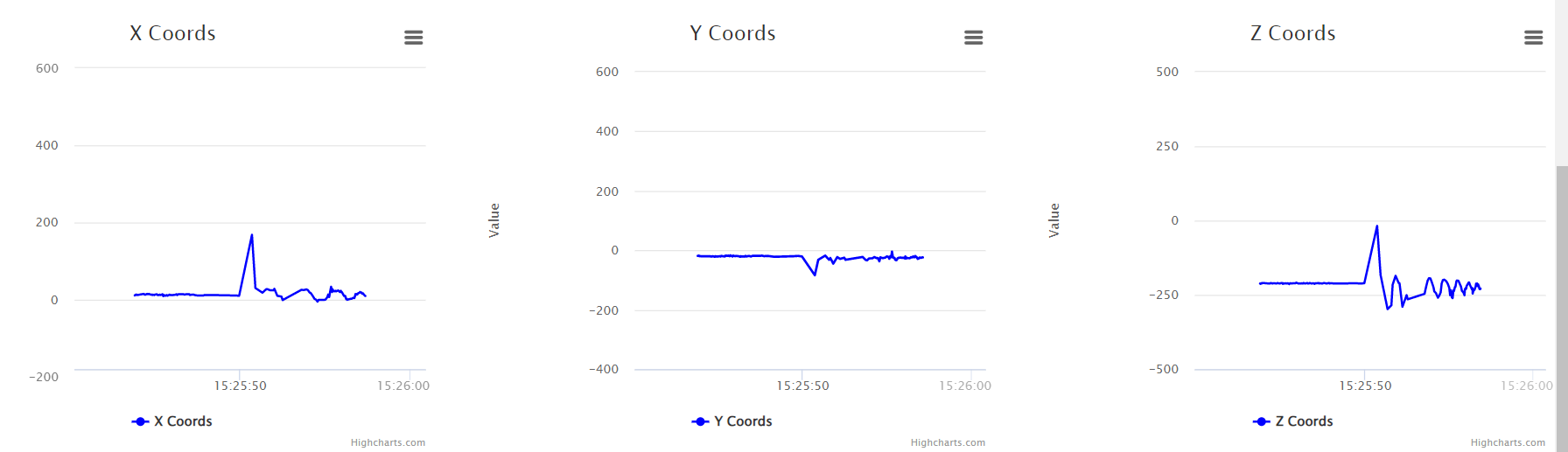
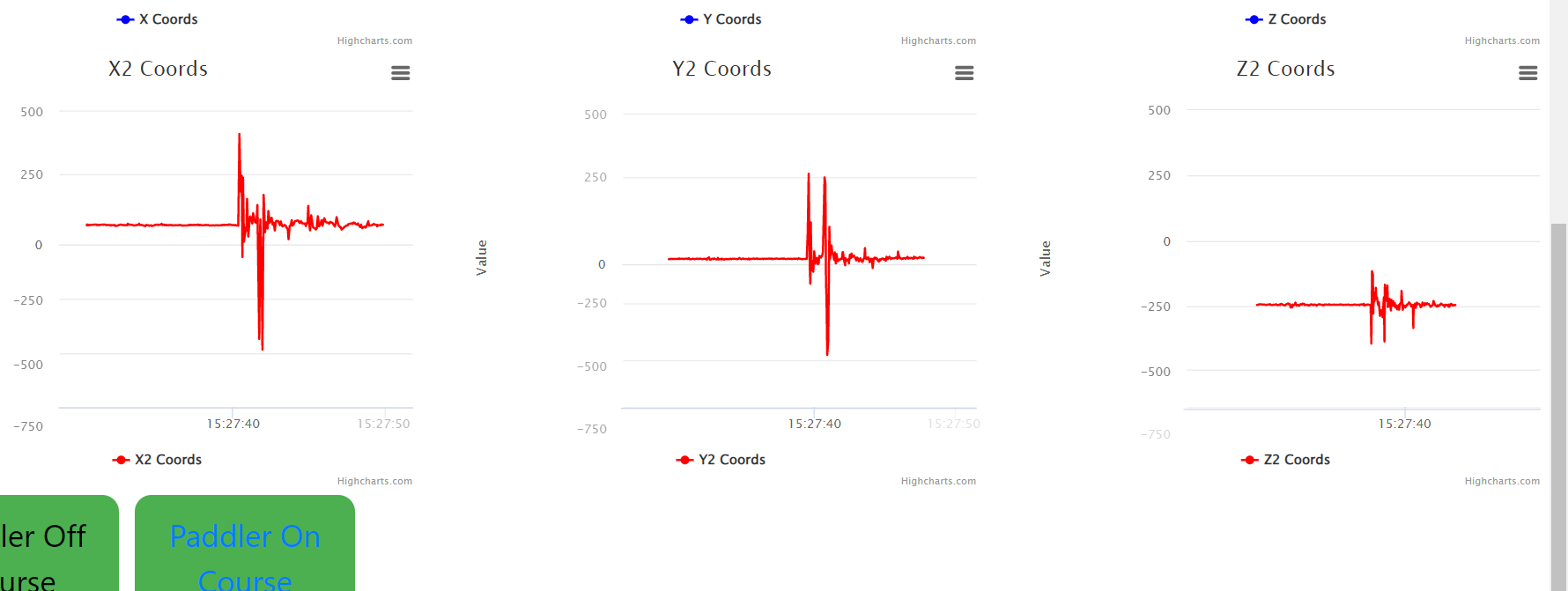
Once data was being livestreamed to the Flask site I was finally able to develop the charts designed. There is a lot of documentation on the HighCharts website as to how to set up the graphs with a variety of examples. To make it even more user friendly, as talked about in sections above, I tried to implement three axis onto one graph but found it too time consuming to complete. I carried on with each axis of the sensor on its own graph. This still produced a satisfactory result with an easy to read graph. I have set the Ajax function to call every 0.2 seconds, resulting in 5 new points being added to the graph every second. This could be increased to get an even more accurate reading of the sensors, but I have balanced the call time with the limitations of the Raspberry Pi’s computing power, as talked about above in section 3.5. The results from testing the product can be seen below in figure 5.

Figure 5 – charts are two different hits on the live system



## Test System

## Overview

The ‘test system’ is the first system I developed, before re-evaluating the network. This system takes the accelerometer data and stores it in an SQLite database. There are many advantages and disadvantages to this system with many already being discussed in the sections above. Such as the ability for the past data to be queried and the drawback of not being able to see any of the readable data, such as a graph, on the web app. Although there are many disadvantages, I still found it more important to keep this section in the final product, as it shows the history of the product and the many decisions I had to make to get here.

During development, there were many hard decisions to make, to do with the functionality of the final solution. One of the main choices I needed to make was whether to base the UI as something the judge would look at all the time, or just if they are unsure of their decision. This would then lead to whether I would implement a way for the judge to note down if the gate was cleanly passed, hit, or missed.

The potential different paths taken can be seen with having the two systems. For example you could see how the test system could be used by a judge as a judging system, and it would produce the graph after the user presses the button. I found this to be too time consuming, compared to having the live system, as well as it being impractical for the judge to use, there would be a 30 second, or more, delay to render the graph. This would make the system unusable if I were to expand the product to include more poles and having racers coming down in quick succession. Although this idea failed, it grew into what I have developed now as a system to test the product and save results, with more scope for the system to be used in a competitor, if the product were to ever expand.

## Database management

The key component of the test system is the database and its connectivity with the Flask server. As discussed above, I have used SQLite as my database, and you can find my reasoning in section 2.3. To start with I created my database with two tables, ACC\_data and Hits. The ‘Hits’ table is populated with a timestamp, which pole has been hit, and the Row\_ID. The Row\_ID is the foreign key linking the two tables together. ACC\_data table has the x, y, z from each sensor, with the right sensor being x2, y2, z2. There is also a timestamp which corresponds to when the reading was taken and the ROWID built-in function of SQLite. This system works well to allow me to store all the data that would be needed to identify a hit.

A picture containing chart

Description automatically generated

Figure 6: shows how the database is setup

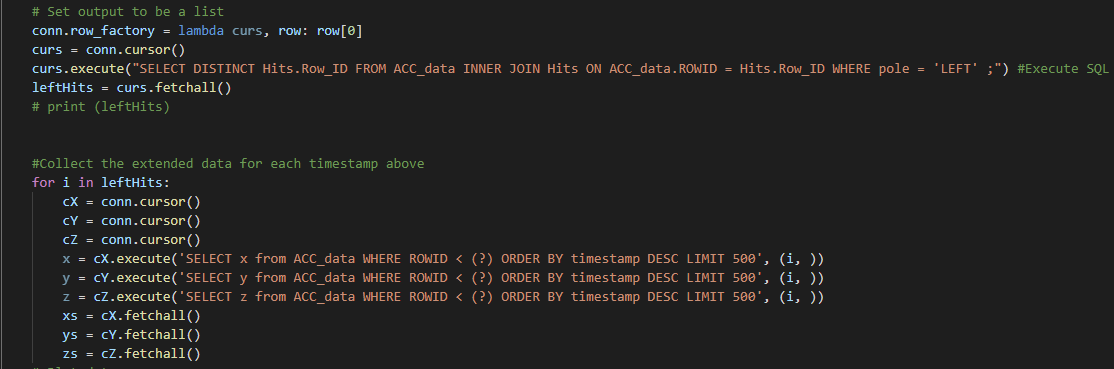
A limitation of this system is there isn’t a way to distinguish between what each hit is from. For example there is no way in the database to see whether the hit was with a paddle or a helmet. I decided that this would not be needed for the way I was using it. This is because I was able to note down what each type of hit was as I was testing it in my back garden.

If I were to use this in a faster paced environment, where the judge wouldn’t have the time to note down the different types of each hit, I could add a field into the Hits table to identify the type. This could then be queried and added onto the header of the plot. An example of this would be figure 6. The judge would be able to select what type of hit it is from a drop down menu, or a set of button. This still isn’t an ideal way as it can be time consuming to looking through a list and select which option is needed.

Once the user has clicked the button that corresponds to what side the paddler hit the pole on, the system enters the last timestamp and last ROWID from the ACC\_data table and saves it in the Hits table. Once the testing session is over the user will run the file ‘PlotfromHits.py’. This is where the user is able to plot all the hits they have just logged and get an output of PDF files for the left and right poles.

To achieve this plotting, the system must first select a wider range of results from the time of the hit. This is done by joining the two tables on Row\_ID and ROWID and selecting the Row\_ID’s where the pole is either left or right. A for loop then selects 500 x, y, z entries where the ROWID is less than the selected Row\_ID from the last query. These coordinates are then plotted on three subplots and saved to a PDF file. This is then repeated for the right sensor, meaning that the two different sides are outputted to their own PDF document.

Figure 7: Snippet of code showing the SQL queries needed to plot the data.



One major problem I found when first starting this section was the format that SQL queries output in. This was a significant problem for me because as you can see from the code, I need to query the output from the first query. SQL outputs all the results from the first query in a tuple usually. This will not work when trying to iterate through for the next for loop. To combat this I found the row\_factory feature in the documentation of SQLite. Row\_factory allows the user to set the output of their query to a list making it much easier to use in the for loop. This was a major hurdle I overcame by using the documentation.

To log the data needed to run all of this the user must run the file “logADXL.py”. This is a very simple file that reads the x, y, z data from the accelerometer using the already mentioned Python module. It then uses the ‘INSERT’ feature of SQL to add all of the data to the ACC\_data table.

## Network Overview

The network has already been briefly discussed above in section 3.4, but here I will go into more detail.

First of all, the network is running off the Raspberry Pi, as I aimed to keep everything as lightweight as possible, due to the lack of computation power of the Raspberry Pi. The next major part of the network is the Flask servers, one for each system (live and test). On boot, the Raspberry Pi automatically runs the Live system Flask server, to reduce the set up team needed when using the system. According to the network diagram, we can see that the information from the sensors flows to the Pi, then to the Flask server, this is done via the GET method in html, along with some Ajax and jQuery in ‘index.html’ template that is loaded by the Flask server.

Currently the test system is running a centralised database, with the two sensors reading to the one database. If I were to expand this system and use it in the real world, I would move towards a distributed database. This is because it allows the network to scale easier depending on the different number of sensors needed for each different course. Each Raspberry Pi would effectively have its own database that is connected to the corresponding sensors for that section of course.

# Results and evaluation

Initial tests were completed in my back garden, but final tests were completed by taking the product to Cardiff International Whitewater Centre (CIWW), during one of their slalom training nights. Here I was able to hang the pole above the water and use the product as if it was in an actual race, with paddlers coming down the course in quick succession.

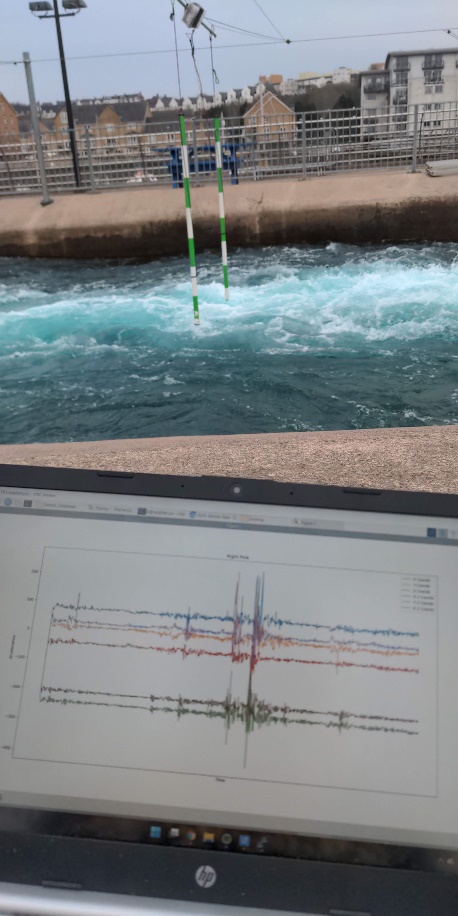


Figure 8:

Showing the testing set up at CIWW, with a graph of the historical data from the session

## Evaluation of Live system

The live system was very easy to test, it was evident straight away if it was working or not. I was able to test the live system simply by hanging the slalom poles on my washing line in the garden. Then monitoring the graphs on my phone, I was able to replicate hitting and brushing the poles in the same way a slalom paddler would do so. I did this 20 times on each pole for each type of hit- paddle, buoyancy aid and light tap. After conducting this test I could see that the graphs had registered each hit and it would show the judges if there was a hit or not

The next stage for the live system would be to go and test it once again at CIWW. Unfortunately at the time of writing this I have not been able to test it again at the centre, due to slalom practice happening only once a week.

One factor of the live system that has been hard to test is the camera. Although the camera is ‘working’ and shows a live view, it is hard to have it actually at the angle that would be required to help the judge. The usefulness of it has been hard to test at home as well because I haven’t been going through the pole as a canoe would. Instead I have just been hitting it from the side. This is more of a limitation of the design than anything else. For example I could build a mount for the camera to sit in and find the perfect angle, but I didn’t have time, and decided to focus more on the graphing. Another downside of the camera is it is currently livestreaming to the Flask site. In a real world situation, it would be better to have a delay, maybe around 5-10 seconds. This is because the judge will be primarily watching the course and only look at the charts and camera if they’re unsure. Therefore to make the camera useful it should have the same delay on it as it would take for the paddler to pass their section, then for the judge to look at the Flask site. This was not implemented as the graphs were more important as they show more information in clearer detail.

## Analysis of results

## Analysis of results test system

The test system has been through rigorous testing twice. The first in late March at CIWW, the second more recently in my back garden. In this section I will mainly talk about the later testing. But to start with I will analyse the first set of results.

A screenshot of a computer

Description automatically generated with low confidenceChart

Description automatically generatedThe two graphs below are the total data from the testing session at CIWW, although this is the combination of over 14,000 data entries it is still easy to see where the paddler has hit the pole. There was also strong supporting evidence from where all three x, y and z graphs had the same spikes, showing there was a hit. It was even easier when zoomed in on each specific time of hit to see the change in acceleration from the pole. Interestingly it was also a windy night and so the pole was moving around a fair bit before the paddler had come through, but this did not affect the final graph as even the smallest of hits were registered on the overall graph.

Figure 9: Left and Right Pole Graphs March 2022

One unforeseen conclusion we were able to draw from the testing was, because of the spreader bar, which keeps the poles a set distance apart, as referenced by the ICF[4], we found that the poles would bounce in unison. This meant that the graphs for both the left and the right look very similar and often will have hits in the exact same places, even though the paddler didn’t hit both poles at the same time. This could mean that theoretically you could get away with only using one accelerometer in the system, but this limits the potential to pick up smaller hits where the spreader bar doesn’t have as much of an effect.

The second set of results, completed in my garden, were not as successful as the first. Although I produced over 80 graphs, not all were useable. I have a few ideas about what could have caused the unsuccessful graphs, such as a weak internet signal and loose wire to the sensor. But ultimately, I repeated the testing and produced meaningful and relevant graphs.

Figure 10:

Graph showing a large hit on Left sensor

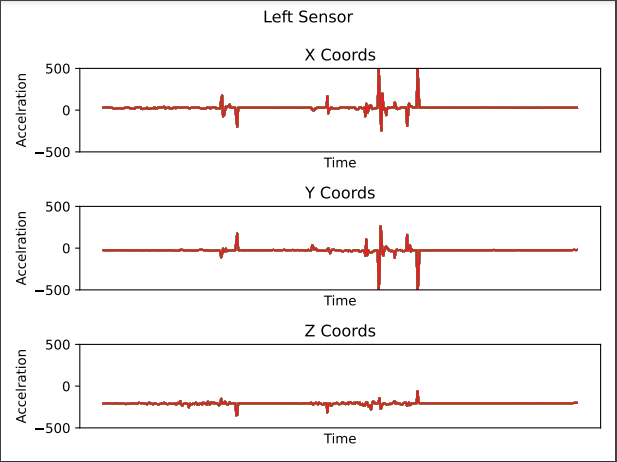


Figure 11:

Graph showing a Light hit on Left sensor

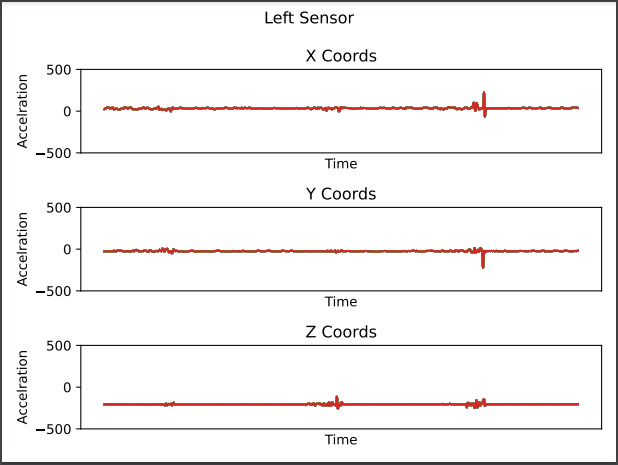


Figure 12:

Two graphs showing the problem from the first set of tests on this day

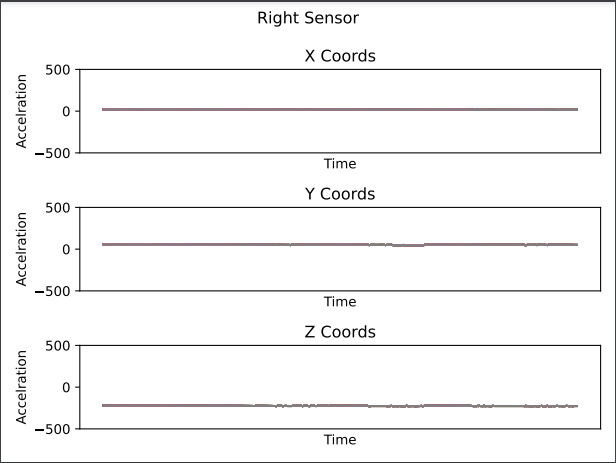
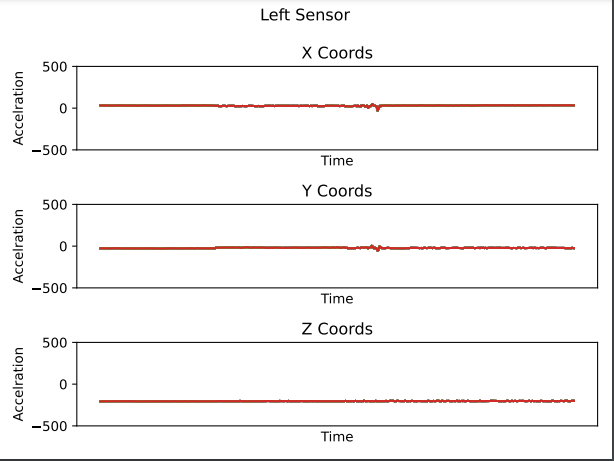


Figure 14:

Graph showing a light hit on right sensor

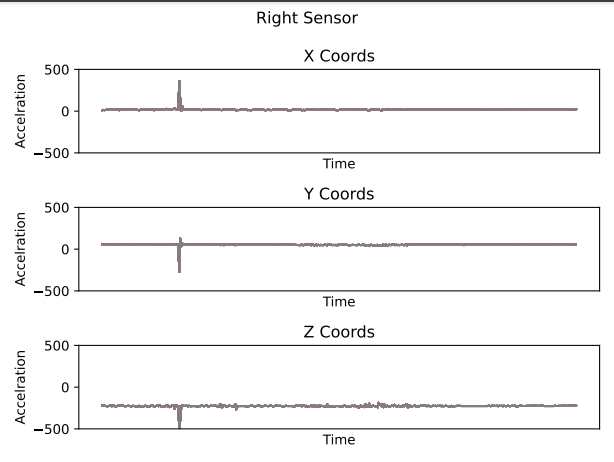
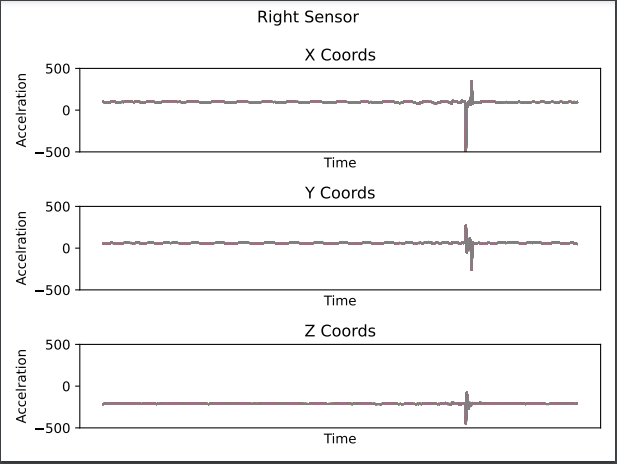


Figure 13:

Graph showing a normal hit on right sensor



The figures above represent each of the specific types of results found from testing the product at home in my garden. I will start with figure 13 and 14. Both the figures show a hit on the right pole. We can see a lot of information from only a small amount of data on the graph. For example, on figure 14 we can see that I logged the hit late on my device, we can see the spike is at the end of the graph not the middle, like the others. We can also see that it was a relatively light hit with only a small spike. Compared to figure 13 we can see more of a swing in the pole with a spike at both the bottom and the top of the average line.

If we compare these graphs to figure 12 we can see why the testing had to be re-done. Although I had hit the pole and logged a hit, the graph shows very little with only very slight tremors. Towards the start of the testing I was having problems with the internet signal not being strong enough, this could have led to the Raspberry Pi not sending data consistently to the Flask server when there was a GET request.

Figures 10 and 11 are another good example of the types of hits you could get. figure 11 was a buoyancy aid hit, and as such very light. It would have been interesting to see if the graph would be any different if the same hit had happened while the pole was hanging up at the CIWW, with the more flexible wire. We can also see from figure 10 that there was a lighter hit before the heavier hit was recorded.

All of these conclusions can be drawn very quickly by looking at the graphs, which helps support the idea for a scrolling graph in the live system. There is also scope for being able to change certain parameters when querying for these graphs, such as changing the limit on the number of entries.

## Analysis of Live system

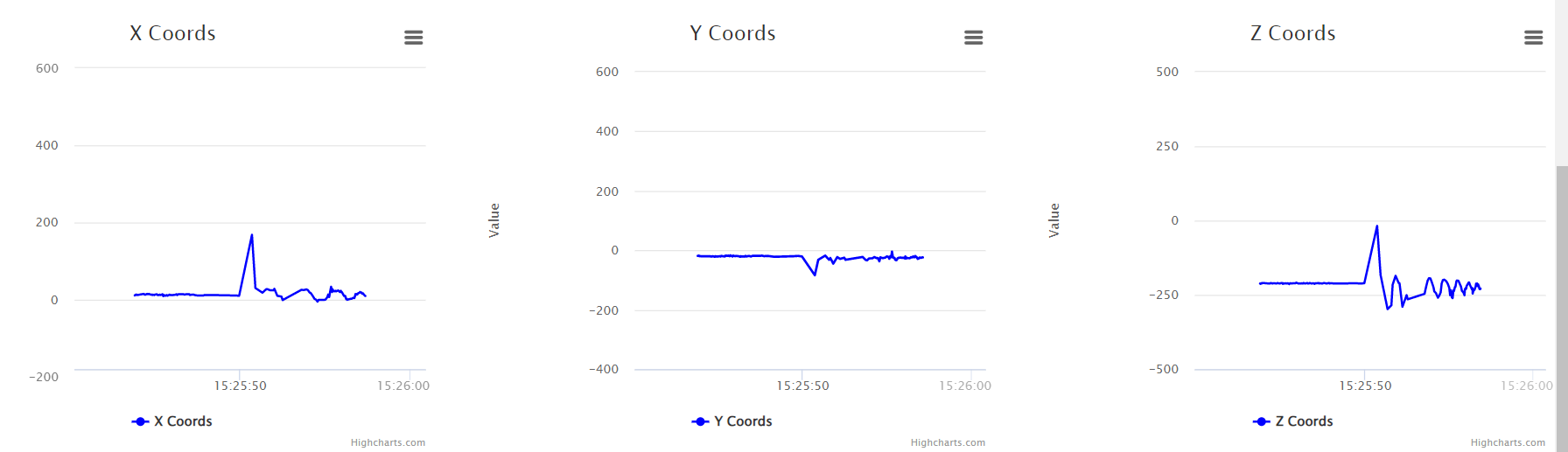
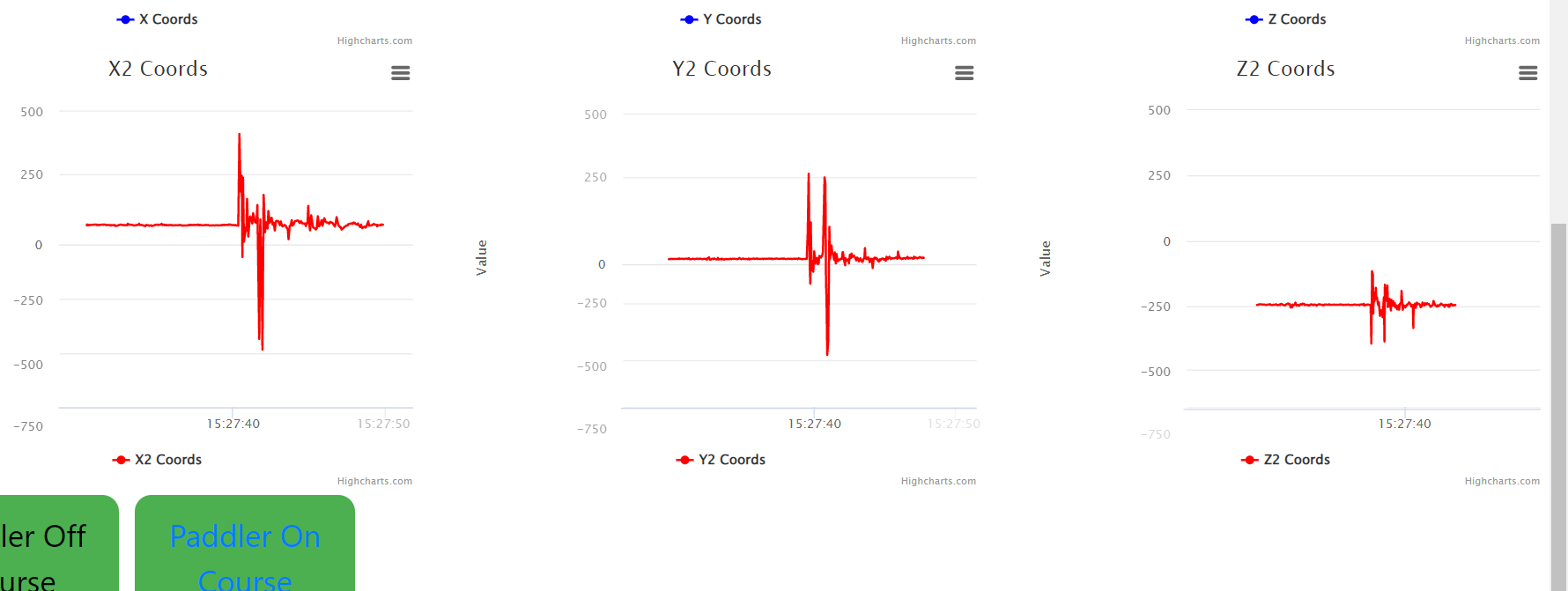
The analysis of results from the live system is a little less quantitative due to the nature of the system. We can see from figure 5 below (which has been copied from section 4.1.3 for ease of reading) that the live system is able to pick up hits very well and could be an excellent help for a judge. Along with the project, I have also made a short video of the system in use as to demonstrate how a judge might use the system. I believe that the video shows a satisfactory result and meets the title of the project.

Figure 5- charts are two different hits on the live system.

Duplicated from above for ease of reading

# Future work

Overall, the product I developed has reached a good level of functionality and has achieved its goal of being a proof of concept. However, there is still a large number of avenues that should be explored, before this product would be at a level that could be used on the world stage.

Further work could be put in to combine the two systems. Taking the functionality of logging the times of each hit and combining that with the live streaming of the graphs. This would let the judges keep the historical data and query the results in case any paddlers disagreed with the decisions that the judges made. This would be vital if it were to be used on a large scale. Something similar to this was in place, albeit a manual process when I attended the British University and Colleges Canoe Slalom Championships. When a penalty was given, as the judge you had the task to write a basic note of why. For example, you could simply put down “head hit, right pole” and this would be used as an argument to withhold the penalty if the paddler disagreed. This system could be expanded, adding a field into the ‘Hits’ table as to what actually caused the penalty. Although the limitation here would be making sure the UI isn’t crowded with so many options for the judge to choose from.

A few of the original requirements set out at the start have not been fully achieved, I believe that with more time these could have been implemented, especially the judgment call as implemented by Ringwood in [12]. This could be particularly useful for the judge and could change the way the sport is judged. I sadly ran out of time, as I believe to implement this it could take a lot of work, especially ‘training’ the system to know what limits a hit are and what are not so as to produce an accurate judgment. If the whole product were to be moved onto a world stage, this is definitely something I would want to see implemented, as there is a large scope for this to also be used on a TV or livestream broadcast for the viewer.

Another area with room for improvement is the camera. The basis of this was discussed in section 5.1. If the project would go further, there would need to be a serious discussion about if the camera is useful, even with the delay I talked about. I believe that with a short delay, it definitely would be more useful than currently, but it would throw the sync of the graph and camera out, making it slightly more confusing to tell which is a hit and what is wind, for example. There would also need to be a discussion about the cost to reward of implementing the camera, as it could get expensive to put a camera on every pole.

Finally there is room for improvement on the tech stack. As discussed above in the earlier sections some of the tools I have used were picked because they got the job done, while also being light weight for the Raspberry Pi to handle. In the future these tools could be changed to tools that are better suited for larger sets of data, or for multiple users. The main one here is the database. But there is also room for improvement on the web server side, as Flask would struggle with the amount of poles needed from a full length slalom course.

# Conclusions

I believe that ultimately the project has been successful. A judge is able to look at the live system and detect if there has been a hit or not, depending on the spike in the graph produced by the accelerometer sensor in the pole. There is a camera, which can be used by the judge to see a top down view from the pole. I believe the UI of the web app is simple and easy to use for the judges. The product has satisfied 5 out of the 7 system behaviours to a reasonable extent, with the final two not being crucial to the product, and the reasoning why they have not been implemented discussed above.

Although all of the main system behaviours have been implemented there are still some limitations. For example, the camera has not been able to achieve the full usefulness I had envisioned at the start. This is because of the lack of delay, which was discussed above in section 5.1. With the current implementation I believe that it is almost entirely useless and is not needed. However if this was to be developed further the camera should be implemented, properly with a delay.

The graphing on the web app has been implemented successfully. However it is not without flaws. There is the obvious sensitivity pay off discussed above in the limitations, but there is also minor styling flaws, which could be easily adjusted with a little bit more time.

The project has been a good proof of concept and an excellent starting point to carry on for. There is still a large scope for the project to be taken further to develop some more complex themes, such as an artificial neural network as discussed in [12] or other ideas discussed above in section 6.

# Reflection on learning

In the end, the project has turned out very different than I had originally anticipated. I underestimated the number of cross roads I would come to, and the number of major decisions I would have to make. I specifically remember having one meeting with Dr Cooper where at the end of it the large number of different decisions I had to make was daunting but these ultimately led me to the final product I have now.

This style of project is very different from the university coursework I have completed before. There is often only a couple of different solutions for a coursework. Almost all of my coursework can be started and know there is an end in sight. But in this project I found 10 different solutions, all with their own different paths leading off to what seemed like an endless amount of options, possibly with no end. For me, this could sometimes be overwhelming and would make it harder for me to make the right choices. However, as the project went on, I was able to get better at managing this and thinking clearly and logically when there were significant choices to be made. Specifically, this came after testing the project at CIWW at the end of March, where I realised that the solution that I had at this stage with would not fully satisfy my requirements. As such I was able to make a clear and conscious decision to make my live system.

From the 12 week project the main personal outcome for me is that I have taught myself to think more like a software developer. Thinking about each choice I make, and where each choice will lead me. I found that in the past working on coursework, I wouldn’t think where each choice would take me, whereas here almost every line of code I thought about what it’s impact will be for the future of the project. Sometimes I had to have blind faith and guess that it will help me reach my goal, other times I had to stop and think where it would take me. This mindset might have been the hardest thing to get to grips with as well. But once I got into the right mindset and was thinking ahead I found development to be fun and challenging (in a good way).

When I began this project, I hadn’t used the majority of the programming skills for over 2 years, some over 3 years. To say I was rusty was a massive understatement, I had to almost re-learn Python from scratch. I enjoyed the challenge of having to re-learn these skills, some of my favourite times were when things would come flooding back to me and I would remember how to use Python, or Flask. Although I have only touched the surface of what could have been done with the project and the power that Python has, it has re-sparked my love for Python and software development. The development has got me thinking about what other projects there are that I could work on outside of this one.

# Appendix

## Glossary of Canoe Slalom terms

**Canoe –** Kneeling in the boat, legs under the paddler with a single bladded paddle. Also the international term for both types of boats.

**Kayak –** Seated in the boat, legs forward with a doubled bladded paddle

**Downstream –** The direction the water is flowing

**Downstream gate –** A gate you pass pointing direction the water is flowing, will have green and white stripes

**Upstream** – the opposite direction that the water is flowing

**Upstream** gate – A gate you pass going against the flow of the water, will have red and white

**River left –** The left hand side of the river, if you are looking at it downstream

**River right –** The right hand side of the river, if you are looking at it downstream

**Spray Deck –** the piece of neoprene that covers the cockpit of the kayak to stop the paddler falling out

**Bank–** River bank, or side of the white water course

# References

1. Adafruit. *ADXL345 Digital Accelerometer*. Bill Earl. Available at: https://learn.adafruit.com/adxl345-digital-accelerometer [Accessed: 02/02/2022].
2. CyberCrew. 2022. *How Many People Own a Smartphone in the UK?* Available at: https://cybercrew.uk/blog/how-many-people-own-a-smartphone-in-the-uk/#:~:text=As%20of%20March%202020%2C%2087,the%20UK%20were%20smartphone%20owners. [Accessed: 27/04/2022].
3. EbenKouao. 2020. *Pi Camera Stream Flask*. Available at: https://github.com/EbenKouao/Pi-camera-stream-flask [Accessed: 08/02/2022].
4. ICF. 2016. *Rio 2016 canoe Slalom Format and Rules*. Available at: https://www.canoeicf.com/olympic-canoe-slalom-format-and-rules [Accessed: 03/02/2022].
5. ICF. 2018. *Behind the scenes - ICF Canoe Slalom judging*. Available at: https://www.youtube.com/watch?v=symeJ4kKsCY&ab\_channel=PlanetCanoe [Accessed: 03/02/2022].
6. ICF. 2021. *Canoe slalom results book 2021*. Available at: https://www.canoeicf.com/sites/default/files/tokyo\_2020\_olympic\_games\_canoe\_slalom\_results\_book\_v1-3\_og2020-\_csl\_b99\_csl.pdf [Accessed 11/03/20222].
7. Microsoft. 2015. *Visual Studio Code*. Available at: https://code.visualstudio.com/
8. PeakUK. 2020. *Racer ST 2020*. Available at: https://peakuk.com/index.php?route=blog/blog&blog\_id=283 [Accessed: 09/03/2022].
9. PeakUK. 2022. *Racer ST2022*. Available at: https://peakuk.com/index.php?route=blog/blog&blog\_id=340 [Accessed: 09/03/2022].
10. 2016. *Raspberry Pi 3 Model B*. Available at: https://www.raspberrypi.com/products/raspberry-Pi-3-model-b/ [Accessed: 01/02/2022].
11. PlanetCanoe. 2021. *2021 ICF Canoe-Kayak Slalom & Wildwater World Championships Bratislava Slovakia / Slalom Kayak Final*. Available at: https://www.youtube.com/watch?v=QuZ1iMEBPj4&ab\_channel=PlanetCanoe [Accessed: 14/03/2022].
12. Ringwood, J. 2015. *A gate hit detection system for canoe slalom*. Available at: https://www.researchgate.net/publication/283655752\_A\_gate\_hit\_detection\_system\_for\_canoe\_slalom [Accessed: 04/05/2022].
13. Sharma, D. 2016. *How does 'Ultra-edge' technology work in cricket*. Available at: https://www.inshorts.com/en/news/how-does-ultraedge-technology-work-in-cricket-1478524315208 [Accessed: 18/03/2022].
14. soumilshah1995. 2019. *Display Live Sensor Data on Live Chart and Gauge using Flask Python*. Available at: https://www.youtube.com/watch?v=gopZN\_yVANQ&ab\_channel=soumilshah1995 [Accessed: 31/03/22].