

Department of Design Engineering and Mathematics

Middlesex University



Tennis Training System

Matheus Abrao

Supervisor: Aleksandar Zivanovic

PDE3400 Major project

BEng Design Engineering

May 2022

Abstract

The main idea of this project is to explore how technology can be used to improve sports and specifically how computer vision and sensors can be used to improve a tennis player's performance. The goal of the project is to have a mobile application that can be used with only a mobile phone that has a camera and potentially receiving data from a sensor mounted on the racket. The two aspects of the players game which are to be tackled with this analysis are technique and game plan. Technique will be corrected and adjusted using an open source pose estimation algorithm, then the players pose can be corrected using the drawing tools that exist in the application. Furthermore, the game plan can be analysed and adjusted using the same drawing tools that are in the application. The last step of the project would be to read and analyse data coming from a sensor mounted on the racket.

Table of contents

1.	Introduction	4
2.	Background Research	5
2.1.	Mobile Technologies	5
2.2.	Computer Vision	6
2.3.	Hardware	7
3.	Development	9
3.1.	Computer Vision	9
3.2.	Mobile Application	12
3.3.	Hardware	16
4.	Results and Discussions	21
5.	Conclusion	27
6.	Bibliography	28
7.	Appendix	29

1. Introduction

It is important to note that this project is not by any means an attempt to make a final commercial product. However, this project is an exploration of what is out there in the market regarding the fusion of technology and sports and then the implementation of such technologies in order to become more familiarized with them. Most specifically the fusion between computer vision, machine learning, electronic sensors, hardware and one of the most played sports in the world, Tennis.

In the early stages of the project, there were four main objectives in order from easiest to hardest – the first objective was for the system to have a sketching tool that you could use to draw and add notes to a video of the player to correct footwork and technique, this objective would be related to user interface and front end design of the application. The second objective was for the system to have an algorithm that would detect the pose of the player and then create a “stickman figure” out of the pose so it makes it easier to analyse groundstroke and serve motions, this objective would be related to computer vision. The third objective was for the system to have a machine learning model that would use computer vision to analyse who won the point and keep track of score during match play, this objective would be related to machine learning. Finally, the fourth objective was for the system to create statistics based on gameplay, examples of such statistics would be: Aces, double faults, 1st serve, 1st serve points won, 2nd serve points won, break points faced, service games played, service games won, total service points won and this objective would be related to computer vision and machine learning.

However, after receiving feedback and doing research, I realized these objectives, specially 3 and 4 were very ambitious and might not be possible in the timeframe of this project. The reason being that live video recording being fed into a machine learning model will be extremely complex data to manipulate. Thus, after further review, I decided to change the latter stages of the project and add a hardware component. The idea was to include a collection of hardware components that would be mounted into the tennis racket, this collection would include sensors such as an accelerometer or a gyroscope – possibly a 6 degrees of freedom IMU. Therefore, it was decided that the project would have three objectives – the first objective was for the system to have an algorithm that would create a “stickman figure” of the player so you can analyse groundstroke and serve motions without external interference, this objective would be related to computer vision. The second objective was for the application to be able to go back and forth between stick figure and regular video and to have a tool that could draw lines, shapes and add notes to the video, this objective would be related to user interface and front end. Lastly, the third objective was for the application to have a functionality to receive live data from a sensor mounted on the racket (accelerometer, gyroscope) – IMU (6 – DOF). The app would then display this data in a manner that could be analysed by the user (player or coach) and this objective would be related to embedded systems.

2. Background Research

In order to start the project, it was first necessary to research projects and products that already existed in the fields of mobile technologies, computer vision and hardware related to sports and tennis technologies. This research will later be useful when making important decisions that could lead to a better project.

2.1. Mobile Technologies

In the present day mobile technologies and mobile applications are everywhere, the goal for this part of the research is to find the best possible mobile technology for the context, which is building a tennis application that can be used on the court with only a mobile phone and it would aid the user in visualising how to improve their technique and/or gameplan.

The first considerations to be made when choosing the technology were what the app would need to do. The first and most important task of the app is to be able to use computer vision and most specifically a pose estimation algorithm to create a stick figure of the player so the technique can be improved. Thus, I had to keep in mind that I need a technology that is easily integrated with Open CV (one of the biggest computer vision libraries in the open source world), I chose to use Open CV with python because I had previous experience with this technology. In addition, I had to choose a technology which could potentially be integrated with the sensor, so the app would need to have WIFI or Bluetooth capabilities.

There are three main types of mobile app frameworks that could be chosen for this project. The first framework is called a native framework and the advantage is that native apps are high-quality in both aesthetics and functionality because they contain native widgets dedicated to their specific operating system (e.g. android or IOS). However, the problem with native apps are that they would be built specifically for one operating system being necessary to build two different applications if there is a necessity to have an app in two different operating systems. The second framework is called a Web App, and these are apps that are distributed using a browser interface, built just like regular websites but have a similar interface to mobile apps and are designed to run on your mobile phone. Finally, the third framework is a hybrid app and what sets them apart is that the software can be launched in different operating systems with little to no changes [1] .

After careful consideration of the types of frameworks listed above, I decided to pursue hybrid apps which seemed like the best option for this project. Then, the two hybrid frameworks that I thought were best suited for this application were Microsoft Xamarin and Python Kivy, because these frameworks are both easily integrated with Open CV and have Bluetooth and WIFI capabilities. In order to decide the best one, I had to weigh in the pros and cons of them both. A Microsoft Xamarin app is built using the C# programming language and the .NET framework and I would have to learn these technologies from scratch. The advantages of Xamarin are that it contains native tool kits so the app would look better and it would be more customizable. However, it is important to note that the main goal of this project is not

the aesthetics and user interface. The advantages of Python Kivy would be having the entire mobile project written in python which is one of the languages that I have most familiarity with, thus, the project would be easier to manage. The disadvantages of using Python Kivy would be losing the native look of the app and missing the opportunity of learning a new programming language which could be beneficial for me in the future. After weighing the pros and cons, I decided to use Python Kivy because of the time limitations of the project [2][3].

2.2. Computer Vision

The purpose of this study is to find applications of computer vision and image processing in tennis that could be useful for decision making in the development part of this project. As stated before, the computer vision library of choice is Open CV as I have previously worked with this library. In more detail, this study will be an investigation on the types of algorithms used for the tennis and what purpose they serve.

Computer vision is an area that consists of obtaining an understanding of an image or a set of images and then processing this image in order to create conclusions (e.g. detect objects, tracking changes in the position of objects and more). In order to better process the image, computer vision uses a series of different techniques such as convolution, filtering, pooling and many more [4]. One possible lane the project could follow would be creating statistics and analysing real time game play, in this case one of the most important computer vision tasks would be real time ball detection. In a related work [4], ball detection is thoroughly studied using different popular object detection algorithms (e.g. SSD, Haar classifiers, YOLOv3-tiny, R-CNN, fast R-CNN and faster R-CNN). The conclusion taken from using these algorithms was that none of them provided real time ball detection with the accuracy needed, the reason for that is that they struggle with detecting small objects. However, the authors ended up customizing the YOLOv3-tiny algorithm and using it together with other techniques for indexing tennis videos and commenting on them so their system came closest to the requirements. After analysing the possibility of creating statistics and possibly determining the winner of points using computer vision, I decided it was a very complex task as it was required to create/customize and train your own algorithm only for the ball detection part. Furthermore, in order to run this algorithm a powerful GPU (Graphical Processing Unit) is necessary, making it harder to perform this task.

Tennis is a sport that consists of six basic strokes (the serve, forehand, backhand, forehand volley, backhand volley and overhead smash). Out of all those strokes, the serve or service can be determined as one of the most important of them to win matches [5]. The serve is also a stroke with high variability in power, ball speed, accuracy, ball impact location and angular velocities [6]. In [6], the study proposes to use a computer vision pose estimation algorithm to compare professional players' serves to amateur players' serves in order to help them improve their serve, it can be assumed that by using computer vision one can efficiently derive compelling results. On the other hand analysing serve motion without computer vision aid would be harder and less effective to visualise the difference between biomechanics of amateurs

and pros. After the data was separated into frames and normalized so it could be compared, it could be concluded that not only the patterns were different but that specific body part positioning was clearly divergent. The study's findings can be concluded to be very informative as it can help players to focus on key points to improve on their serve rather than watching professional players and not knowing where to pay attention. Furthermore, the data from the study can be useful to tennis coaches as they could have a focus point for their lessons.

2.3. Hardware

In order to find the best possible application of a hardware mount to detect and get valuable information coming from a tennis player, it is first necessary to analyse what is currently available in the market.

There are 4 main types of sensors that are or were available for collecting live data from tennis. These sensors are: mounted on handles, embedded in handles, embedded in dampener and wearable. When I say were available it means they were discontinued. Most of these sensors utilize Inertial Measurement Units (IMUs) – “An IMU sensor is a specific type of sensor that measures angular velocity, force, and sometimes magnetic field. The components of the IMU sensors are a 3-axis accelerometer, a 3-axis gyroscope and in this case, we are talking about a 6-axis IMU. An additional 3-axis magnetometer can also be included, so that the IMU sensor configuration is considered to be 9-axis” [7].

TYPES	PRODUCTS	No. Of Shots	Ball Impact Location	Swing Type	Swing Speed	Ball Spin	Power	Other unique tracking Feature/s	ITF PAT?
Sensors mounted on handles	Sony	✓	✓	✓	✓	✓	✗	✗	✓
	Zepp Labs	✓	✓	✓	✓	✓	✓	Serve Practise	✓
Embedded in handle	Babolat Play	✓	✓	✓	✗	✓	✓	Longest Rally	✓
	Head Tennis Sensor	✓	✓	✓	✓	✓	✗	3D Animation	✓
Embedded in Dampener	Qlipp	✓	✓	✓	✓	✓	✗	Service Time, Angle of Impact, Video Synch	✗
	Courtmatrics	✓	✓	✓	✓	✓	✓	Wrist Snaps & Split Step	✗
Wearable	Babolat Pop	✓	✓	✓	✗	✓	✓	Longest Rally	✓
	Pivot	Motion Caption for 3D replay and technique analysis							✗

Figure 1 - Commercial Sensors Comparison [7]

After analysing commercial tennis sensors that are currently available in the market (see table above), I decided that the best option for the hardware was to have an IMU embedded in handle, this would be a better fit than a wearable because the IMU could track movements of the racket instead of the body or wrist. Furthermore, this sensor embedded on the handle would be better than the sensor embedded in dampener because for the entire hardware to be embedded in a small dampener it had to have a custom

built PCB which would not be possible with the timing of the project and would require extra resources that are out of reach. The challenges of using these sensor would now be related to the processing of the data received by the sensor and sending the data (networking) from the sensor to client (computer or mobile app).

In order to make the best possible processing of the data received by the sensor, I decided to review a paper [8] that explores the feasibility of applying machine learning in an IMU dataset from cyclists which almost get into an accident or were involved in an accident. These machine learning model is then trained to realize when there is chance for an accident to happen. Differently from my project, these paper utilizes a dataset produced by a mobile phone which includes more datapoints such as latitude, longitude, phone location and more. However, the dataset contains all the datapoints that would be used in my project (Acceleration X,Y,Z and Gyroscope X,Y,Z). Important takeaways from reviewing this project were useful python libraries to perform data analysis (Matplotlib, Pandas and Numpy), also that in this particular case oversampling increases the accuracy of the dataset and that all readings from the sensor should be in the same data rate.

3. Development

3.1. Computer Vision

The first development process which I went through was implementing a pose estimation algorithm in example videos of professional tennis players serving. The algorithm which I am using comes from the pose module inside the CVZONE library ¹ which uses the MEDIAPIPE algorithm inside its library ². The MEDIAPIPE pose algorithm returns 33 landmarks for each frame and each landmark has x, y and z coordinates, resulting in 99 data points for each frame. This is then used by the CVZONE library to draw the “stick figure” on top of the player. See figure 2 below for the first use of the algorithm. One important remark is that the video which I am using in this example is filmed behind the baseline and behind the player, the pose estimation would yield better results if used in a video which is filmed from the side of the player where you can see the player’s entire body.



Figure 2 - Pose Estimation Algorithm First Trial

The next step on the computer vision part of the project was to figure out how to create a customized stick figure. I opened the GITHUB link of the CVZONE library¹ and downloaded the file which I was using (PoseEstimation.py). Inside this file, I added a description specifying that I was modifying this open source module, the only modification being adding a method inside the Pose Detector class. The method which I created is drawing only specific landmarks of the player determined by the user and creating a

¹ <https://github.com/cvzone/cvzone>

² <https://github.com/google/mediapipe>

personalized stick figure, see the project's GITHUB for more information³. See figure 3 for an example of the stick figure being applied to the original video as well as to a blank image.

The next step of the pose estimation algorithm would be to trace the racket on the screen. My idea to do that was to simply trace the racket from the wrist joint, however it was not as simple as I previously thought because the racket can be at various different angles and it would be impossible to guess the correct angle to match the racket position. Thus, the solution for detecting the racket would be using an object detection algorithm, learning from previous research [4] training an algorithm from scratch such as YOLO would require extra work and time and possibly not yield good results. For that reason, I started looking for alternatives, the best alternative found was to detect an ellipse (racket head) instead of detecting the entire racket, in order to do that I would use an algorithm to search for ellipses in the image, the best algorithm found for my application was the simple blob detector⁴. This algorithm uses a series of parameters to filter the image and search for the "blob" or ellipse, examples of those parameters are by color, by area, by circularity, by ratio of the minimum inertia to maximum inertia etc. In order to apply the algorithm, I decided to create a bounding box around the stick figure with extra room so it searches for the ellipse in only that section of the image, see figure 1 in the appendix for the cropping of the image where the algorithm was searching for the ellipses. However, after exhaustive tuning of the parameters of the algorithm I discovered that it wasn't working because a Blob is a group of connected pixels in an image that share some common property (e.g. grayscale value), therefore after finding this information I concluded the blob detector wouldn't yield good results for detecting the racket head as the pixels in the frame of the racket do not share the same properties with the middle pixels (strings). Trying any sort of racket detection technique would be very difficult and unstable for this application as removing the noise would be hard for different types of backgrounds and courts. Therefore, I decided to set this task aside and possibly work on it if I have finished everything else.

Even though I did not achieve the desired result with racket detection, I had a working pose estimation algorithm that could be applied to different video inputs, including live camera stream. Consequently, I tested the pose estimation algorithm with a video of myself serving, see figure 3 below. Then, I decided the pose estimation algorithm was ready to be implemented on the application.

³ <https://github.com/mabrao/Tennis-Training-System>

⁴ https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html

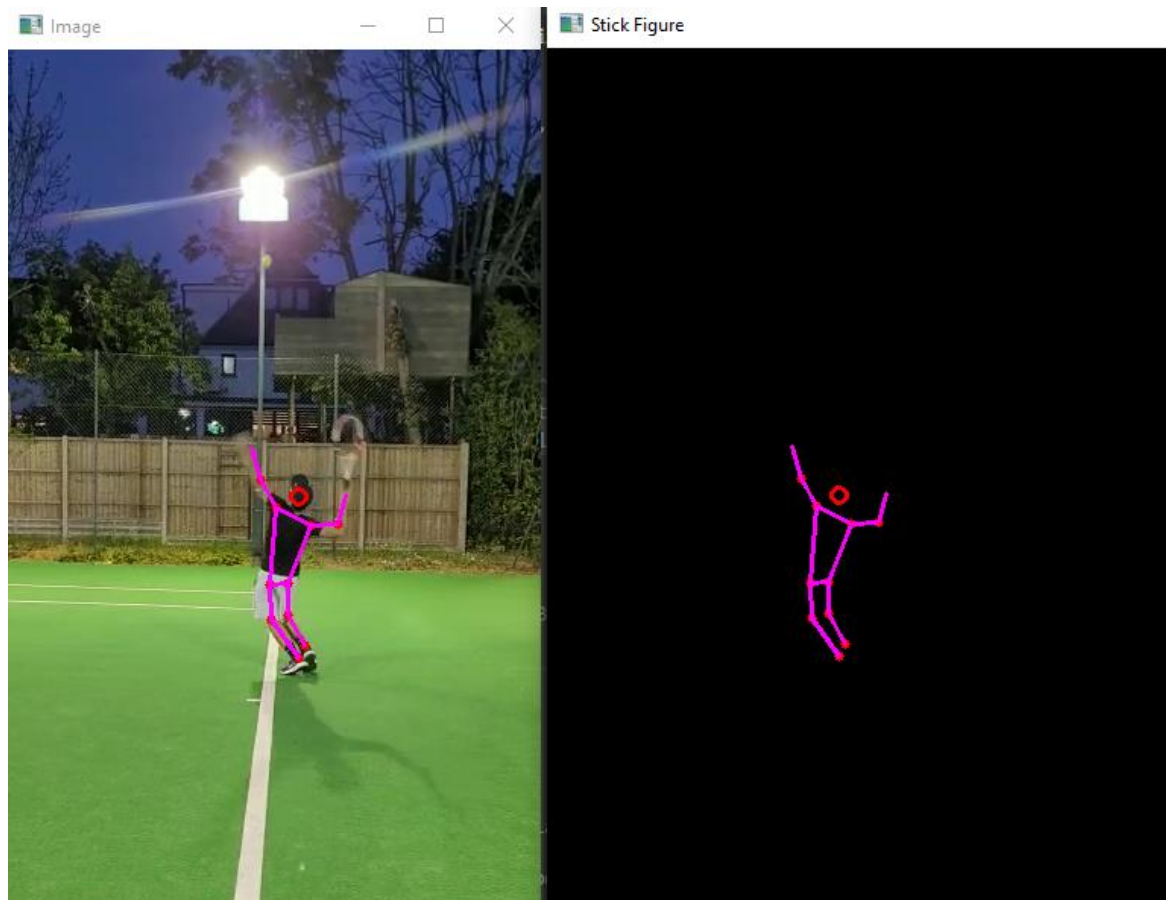


Figure 3 - Pose Estimation Algorithm On Blank Image

3.2. Mobile Application

After conducting research on what the best mobile technology for my project is, I decided I was going to use the Python Kivy framework to build the application. This choice meant that I was not prioritizing making a final commercial application that had the best user interface and I was just building a functional prototype. Before I started learning how to develop the application using this framework, the best thing to do was to sketch how the interface would look in order to have a basis to start developing from. However, these sketches were subject to change in order to adapt for the best possible application of my pose estimation algorithm.

Then, I started learning Kivy by reading the documentation of the library⁵. The important thing is I was only reading the documentation for the parts I was going to use in my project. For example, in Kivy is important to know widgets and layouts in order to place them and use the most appropriate of those when necessary. Widgets are UI (User Interface) elements that receive some sort of user motion event and layouts are used to arrange widgets. I started using a Box Layout, however it is not flexible and the app could have some problems across different devices (i.e. mobile phone). Because I needed it to work on my phone to bring it to court and possibly in other phones, I decided to use Float Layout which places the widgets according to the size of the screen you are using.

When it came to start coding it, I created a simple main page for the application with a button to record and a button to stop recording. Then, I also created a check box that if true creates a pose of the live video frame. Furthermore, the next step was to adapt the main page to work with multiple screens so it can transition smoothly from one screen to another. At first, I thought that in order to have multiple screens on my application I would have to create a .kv file using the Kivy design language⁶ (which would contain all the styling for the app), because most of the tutorials and sources did it this way. However, I would have to reconfigure all my widgets and the layout of my code. Then, after further research I realized I could have multiple screens and use the Kivy screen manager⁷ all in python. After analysing how I designed the code for my application, I realized it might have been easier to learn the Kivy design language from the beginning in order to create a more maintainable and easier to adapt code for the application, but at the time it was too late to start rewriting the front end of the application using the Kivy design language so I decided to stick with python. Going forward, I created a small animation for the record/stop button, improved the positioning of the widgets and added a button that would take you to a different screen using the screen manager, refer to the figure below for the finalized main page.

⁵ <https://kivy.org/doc/stable/gettingstarted/intro.html>

⁶ <https://kivy.org/doc/stable/gettingstarted/rules.html>

⁷ <https://kivy.org/doc/stable/api-kivy.uix.screenmanager.html>

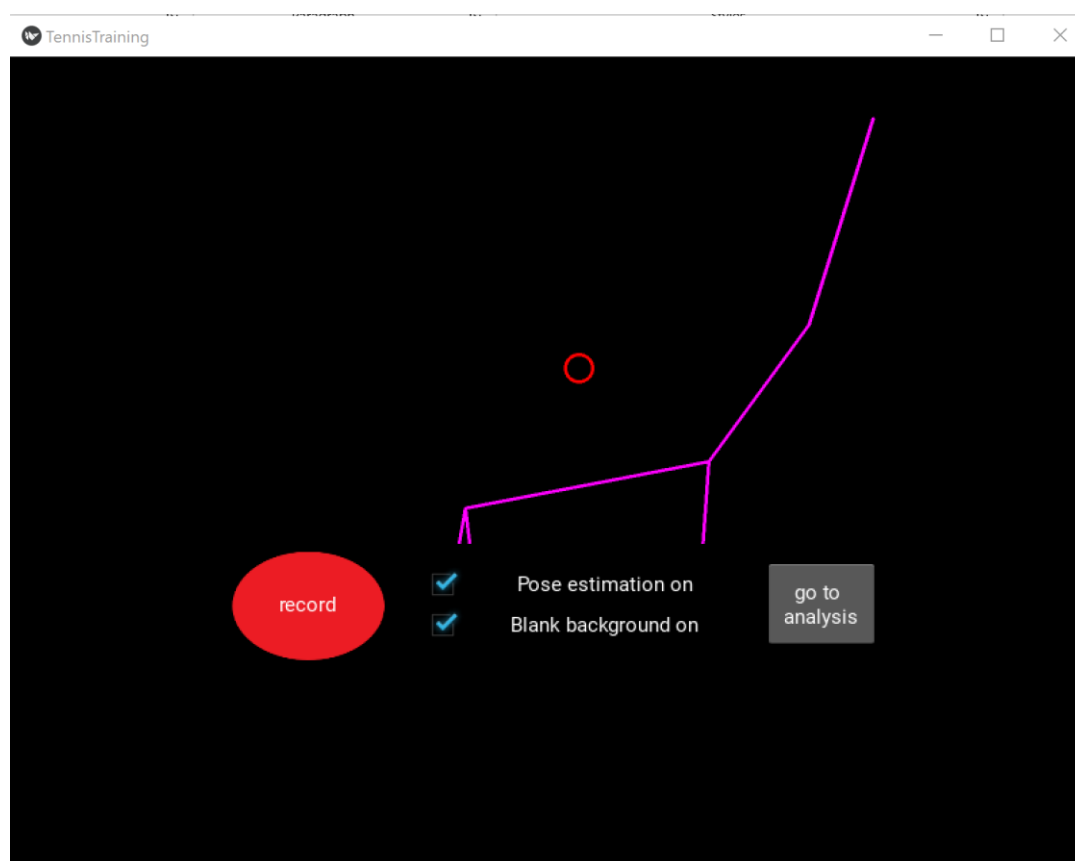


Figure 4 - Finalized Main Page

To investigate for errors, I tested the application in different scenarios to make sure the pose estimation was working properly. This testing showed me that there was an issue when there was no pose detected (when a person wasn't detected on the camera), when this happened the application would crash. Then, I created a github issue specifying what was the problem and fixed it by not doing displaying the pose when it wasn't detected. See GITHUB issue ⁸ for further details.

The next development for the app was building the analysis page. Initially, I had thought about inserting the stick figure of a professional player that was saved on the device on top of the player which was being analysed. However, this would not work as expected because of the frame rate difference between the videos and it would be difficult to match the exact location of the joints from one video to the other in order for the footage to yield good analysis results. The analysis page then became a place where the user plays a video that was already recorded and is stored on the phone, while playing this video the user can switch to pose estimation mode (stick figure being drawn on top of the original image) and also blank background mode (stick figure being drawn on top of a blank image). The first thing I did was creating

⁸ <https://github.com/mabrao/Tennis-Training-System/issues/5>

two buttons, one to go back to the previous page and the other to select a video file. In order to select the file, I am using this module from kivy.uix called FileChooserIconView (refer to Kivy documentation to learn more). I could have also used FileChooserListView, the only difference between them is that one displays a list with the files and the other icons with the files. I decided it was best to use the icons because the more images the UI has, the better.

Now, the video player had to be created in order to receive the files and play the video. I chose to create it first using the Video Player module from kivy.uix. This was not hard to implement. However, it gave me some problems as it was not using Open CV images and converting to a texture in Kivy just like I am doing in the main page to record the live camera. Thus, I couldn't create the buttons for pose estimation and blank background (which are being done with Open CV). Thus, I needed to change the Video Player module that I am using to an Image texture similar to what I am doing in the main page and then create buttons to play, pause and stop the video (video player module has this buttons built-in). In addition, I encountered another problem when passing a variable from one screen to another. I was not being able to pass the video file selected in the File Chooser Page to the Analysis Page. After extensive research on how to solve this issue, I realized you can change pages and access a variable from the new page inside a different class. Refer to the figure below for the snippet of code that changes from one page to the other after selecting a file and therefore can use the page's class variables inside another class, which solved my issue.

```
def selectFile(self, *args):
    try:
        self.videoPath = str(self.files.selection[0]) #grab the filename and directory
        self.manager.current = 'analysis' #change page
        self.manager.current_screen.videoplayer.source=self.videoPath #change video player's source to selected file
        self.manager.current_screen.videoplayer.state = 'play'
    except:
        pass
```

Figure 5 - Use Variable From Other Class

The next step was to fix the analysis page so I could implement the pose estimation algorithm on it. In order to do that I had to get rid of the Kivy native Video Player widget and use an Open CV image, then make that image a Kivy texture just like I did on the main screen, this screen turned out to be similar to the first screen, the only difference being it is playing an already existing video rather than reading from the camera. After creating these two screens, I have realized that it might have been best to design a function that could be reusable and that it could work for both screens depending on the parameters that were input into the function, refer to this GITHUB issue⁹ for further understanding. Nevertheless, I had

⁹ <https://github.com/mabrao/Tennis-Training-System/issues/7>

a working analysis page that was ready for testing. The only thing left was to create a drawing tool in the analysis window that could be used for correcting technique, analysing game plan and annotating the videos. I created a painter class that would draw lines every time there is a user input anywhere on the screen, then I created different buttons so the user can choose different colors, I had some complications with changing the color of the drawings but this was solved by creating buttons with different colors and retrieving the color of the background of the button. Refer to this commit¹⁰ for the addition of the painter class. See figure below for the finalized analysis page.

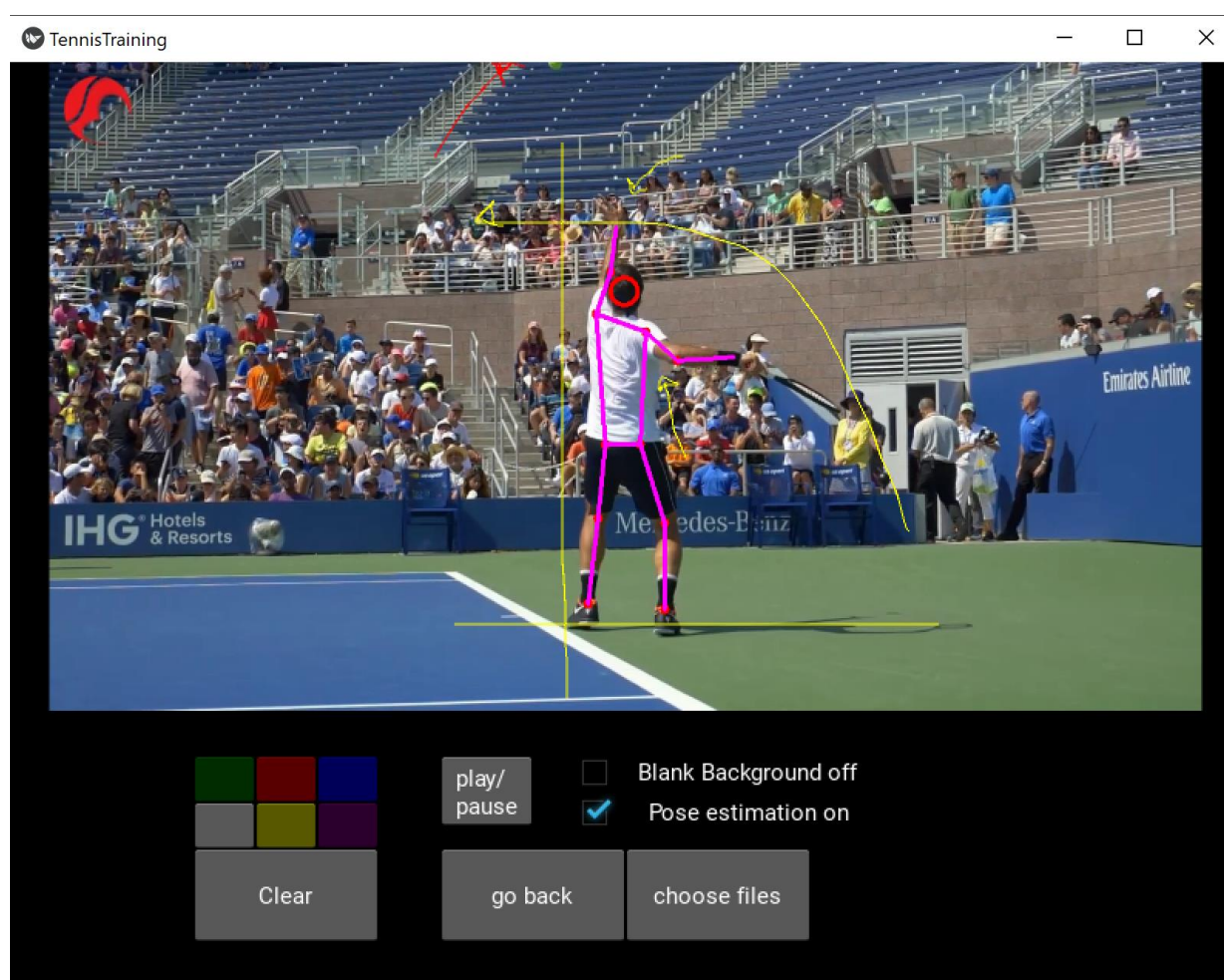


Figure 6 – Finalized Analysis Page

Having finished the main page, I had the pose estimation algorithm in the UI with live camera for recording and saving videos. Also, in the analysis page I had the same algorithm working in the UI with saved videos which would serve the purpose of improving a tennis player's performance.

¹⁰ <https://github.com/mabrao/Tennis-Training-System/commit/f0b6cea04f7cbe335ea36be618ddde81173fe40b>

3.3. Hardware

The third and final part of the project was using a hardware device that sends IMU data to the computer or mobile phone wirelessly. After receiving the data, the goal is to explore the insights that an IMU can give to a tennis player. As seen in previous research, other sensors give insights such as: number of shots hit, ball impact location, swing type, swing speed, ball spin, power and more (see figure 1 in background research). Thus, the first idea I had was to receive the accelerometer and gyroscope data, then use it to train a machine learning algorithm that can detect what kind of shot or swing type that was hit by the player. This feature helps in creating statistics that can be useful for analysing ones game.

In order to start the hardware development, I had to look for a microcontroller that contains a built-in IMU module and a WIFI module. I chose a microcontroller that seemed perfect for the task, the Arduino Nano RP2040¹¹. This was a great choice for my project since it contains a U-blox Nina W102 Bluetooth and WIFI module and a ST LSM6DSOX 6-axis IMU, it also supports micropython as it runs on a raspberry pi processor (RP2040) in case I want to develop the microcontroller firmware using this language to have the entire project python based. Another reason for choosing this microcontroller was that it contains a machine learning core for the IMU which might have been useful.

The first step in the hardware development was to start reading data from the Arduino and trying available example sketches to understand how to read data from the IMU, how to create a WIFI access point with the Arduino and receive and send data from it. I started out trying to use micropython as I thought it would be good to have the entire project written in python. On the other hand, the two IDEs (integrated development environments) I tried for micropython (Thonny and Open MV) were not optimal as they had outdated user interfaces and were hard to use. Therefore, I realized using the Arduino programming language would be a better choice as there is more documentation available for the RP2040 and I could have the entire project inside Visual Studio Code using the PlatformIO extension¹². Furthermore, I was successfully able to read data from the IMU through the serial port using Visual Studio Code (Refer to appendix 2).

After being able to read data from the sensor, I decided it was time to start the design of the sensor case which would be attached to the racket. In order to start the design, first I needed to know which battery I would use for the project, so all components could fit in the case. To choose the best battery, I had to calculate how long the batter would last in different scenarios. In order to know the current that the device would draw, I decided to take into account what sensors and modules would be used by the microcontroller and look up how much current they draw. The modules being used were:

¹¹ <https://docs.arduino.cc/hardware/nano-rp2040-connect>

¹² <https://docs.platformio.org/en/latest/integration/ide/pioide.html>

- LSM6DSOXTR (IMU) which draws 0.55 mA in high performance mode according to its own datasheet (page 13)¹³.
- NiNa W102 UBLOX WIFI module which draws 190 mA with 50% duty cycle on the Tx (when transmitting data) and draws 95 mA on the Rx (receiving data) all according to its own datasheet (page 22)¹⁴. Therefore it can be estimated that this module draws approximately 285 mA.

$$\text{Total Current Consumption} \cong 285.55 \text{ mA}$$

How much would the battery last using a 400mAh battery?

$$\frac{400 \text{ mAh}}{285.55 \text{ mA}} \cong 1.4 \text{ hours} \cong 1 \text{ hour and 24 minutes}$$

On the other hand, according to the Arduino datasheet (page 7)¹⁵, the max current drawn operating at 3.3V is 800 mA (worst case scenario).

$$\frac{400 \text{ mAh}}{800 \text{ mA}} = 0.5 \text{ hours} = 30 \text{ minutes}$$

Thus, it can be concluded that using a 400 mAh battery would be enough for powering the Arduino Nano RP2040 board as I wouldn't need the device turned on for a long period of time in order to record the data. Furthermore, it was also necessary to choose a charging module so the battery could be recharged and I chose the Lipo Charger Plus¹⁶. After making those decisions, it was time to design the case that would be snap fit into the bottom handle of the tennis racket. I decided to use 3D printing as the fabrication method for the case because I have a 3D printer at home and easy access to the printers at university. I was able to print the bottom cup of the part many times and adjust the tolerances to make sure it would be a good fit in the racket. Then, after the part fit the racket, I designed the rest of it to fit the electronic components inside. After I had a working prototype with the 3D printer I have at home, I printed the parts at university with better printers (refer to appendix 3 and 4 for this process). See figure below for the final case design.

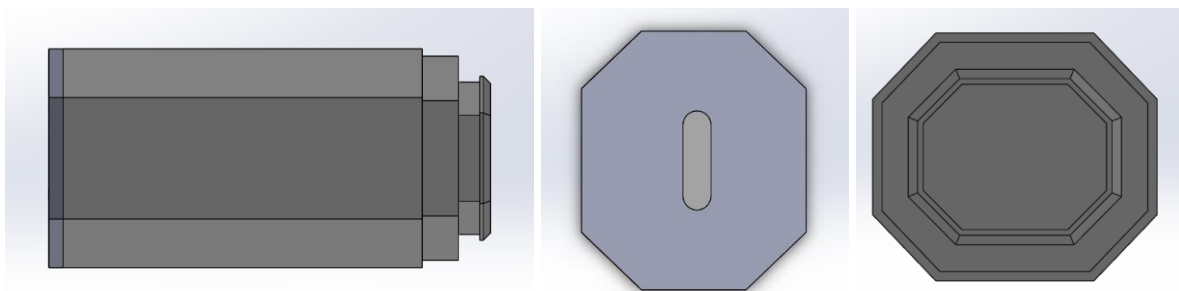


Figure 7 – Final Case Design

¹³ <https://www.st.com/resource/en/datasheet/lsm6dsox.pdf>

¹⁴ https://content.u-blox.com/sites/default/files/NINA-W10_DataSheet_UBX-17065507.pdf

¹⁵ <https://docs.arduino.cc/hardware/nano-rp2040-connect>

¹⁶ <https://www.sparkfun.com/products/15217>

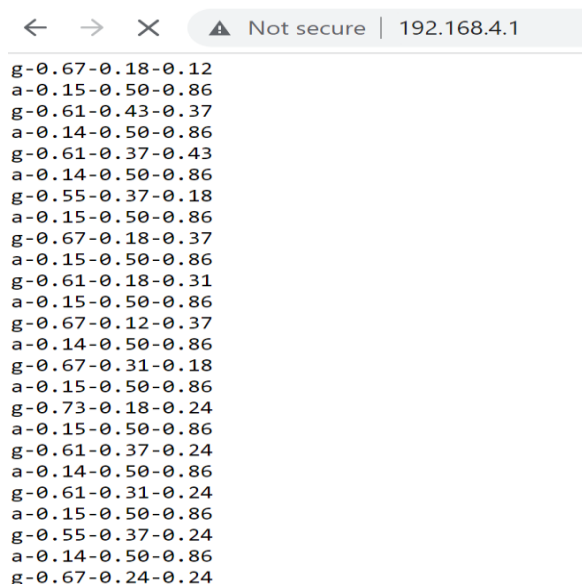
The subsequent challenge of the project was to correctly stream the IMU data wirelessly, then read the data with a python script. I decided to use WIFI over Bluetooth or BLE (Bluetooth low energy) because it has a higher range (400 m compared to 200 m – max Bluetooth range). Also, there was more documentation available on how to send data over WIFI and if I were to connect multiple devices to the sensor (computer and phone), WIFI would make this process simpler. In order to connect the Arduino to the computer or phone (client), I had to create an access point with the Arduino then connect the client to that access point. Once in the same network, I could receive the data being streamed by the IMU. Therefore, I created a server in the local network and had to post the data to the server so it could be read by the client. After looking into different protocols that would post the data into the server (UDP, HTTP, TCP), I realized there was an easy way to just print the IMU data into the server and web browser using the Arduino WIFI NINA library¹⁷, see figure 7 below for the snippet of the code that is printing data to the server. In due course, I was able to connect to the access point on the computer or phone, then open a browser window and the IMU data was being displayed, see figure 8 below for this data.

```
WiFiClient client = server.available(); // listen for incoming clients

if (client) {
  while (client.connected()) {
    if (client.available()) {
      // read bytes from the incoming client and write them back
      // to any clients connected to the server:
      if (IMU.accelerationAvailable()) {
        IMU.readAcceleration(Ax, Ay, Az);
        client.print('a');
        client.print(Ax);
        client.print(Ay);
        client.println(Az);
      }
      if (IMU.gyroscopeAvailable()) {
        IMU.readGyroscope(Gx, Gy, Gz);
        client.print('g');
        client.print(Gx);
        client.print(Gy);
        client.println(Gz);
      }
    }
  }
}
```

Figure 8 - Printing data to the server

¹⁷ <https://www.arduino.cc/reference/en/libraries/wifinina/>



```
g-0.67-0.18-0.12
a-0.15-0.50-0.86
g-0.61-0.43-0.37
a-0.14-0.50-0.86
g-0.61-0.37-0.43
a-0.14-0.50-0.86
g-0.55-0.37-0.18
a-0.15-0.50-0.86
g-0.67-0.18-0.37
a-0.15-0.50-0.86
g-0.61-0.18-0.31
a-0.15-0.50-0.86
g-0.67-0.12-0.37
a-0.14-0.50-0.86
g-0.67-0.31-0.18
a-0.15-0.50-0.86
g-0.73-0.18-0.24
a-0.15-0.50-0.86
g-0.61-0.37-0.24
a-0.14-0.50-0.86
g-0.61-0.31-0.24
a-0.15-0.50-0.86
g-0.55-0.37-0.24
a-0.14-0.50-0.86
g-0.67-0.24-0.24
```

Figure 9 - IMU Data Received Wirelessly

After successfully reading the data on the browser, it was now time to read this data with a python client script and figure out how to use it to train a machine learning model for shot recognition. The first method I thought on how to go about this challenge was to record videos playing tennis with timestamps on it and record data from the IMU simultaneously with timestamps as well. Then, manually classify the types of shots in the video and then feed this training data into a classifier so it can later make decisions on the type of shot being hit. One complication with this problem would be to detect the start and end of the hit. A big process of this step would be data sampling which would aid in equalizing the data. Another complication of this process would be to add the timestamps to the Arduino sketch, while I can add the milliseconds timestamps which tells time in relation to when the program started, the timestamps on the Arduino would be useless since I need to match the timing of the sensor to the timing of the video. In order to print real time of the sensor readings, I would have to fetch this time from a different server, which would have to be done in a different thread (different process running simultaneously) and this would take while to implement. Therefore, I decided not to go through with this idea.

An alternative for the timestamp problem would be to write data to a text file every time there is a hit. In order to do this, I have a python client that receives the data being posted to the server with a python socket¹⁸ and then whenever there is a keyboard press from the user (e.g. space bar) it writes the data to a new text file. After collecting the data, I would sample it and try and feed to a machine learning model. However, because of time limitations, the final idea is to correctly collect data wirelessly from the IMU and create graphs to elaborate on how it would be feasible for a machine learning classifier to generate predictions on the type of shot being hit by the player.

¹⁸ <https://docs.python.org/3/library/socket.html>

When trying to create the python client socket to receive the data, the Arduino server host was disconnecting the client every time after around 20 seconds to 1 minute. I was receiving the following error message: “[WinError 10054] An existing connection was forcibly closed by the remote host”. Then, I decided to study the basics of networking, sockets and transmitting data with Arduino in order to fix this issue. I discovered that there was a problem in the Arduino server side. In order for the python socket to receive the data it had to send a message with any content in binary, and I was sending the binary (b’1’). Moreover, I realized that the Arduino side was not reading the message sent by the python socket, so these messages were getting stuck in the Arduino buffer and it was running out of memory. All it had to be done (see figure 9 below) was adding one line of code to read the data sent by the python socket on the Arduino side so the data doesn’t get stuck in the buffer and the Arduino doesn’t run out of memory.

```
client.read(); //reads data being sent by the client so it doesn't add up in the buffer
```

Figure 10 - Networking Bug Fix

In addition, I added to the python client script a functionality to start writing data to a text file when a key was pressed (R key) and stop when another key was pressed (esc key) so specific data could be recorded for each swing type. Moving forward, it was time to record the data. I went to the tennis court and recorded around 20 files for each simple ground stroke (forehand, backhand and serve). Consequently, I had data that was wirelessly recorded from the IMU sensor and it was time to process this data in order to analyse it and discuss the results.

4. Results and Discussions

In order to analyse and discuss the data collected by the sensor, it was first necessary to create a python script that would parse the IMU logs and place them into a data structure that could then be used to graph those logs. I chose to store my data in a python dictionary with each key being a coordinate (x, y or z) and containing a list of values. See figure below for a better representation. Then, I did a series of string manipulation operations to store each value in its correct key.

```
#dictionaries for acceleration and gyroscope:  
acc = {'x':[], 'y':[], 'z':[]}  
gyr = {'x':[], 'y':[], 'z':[]}
```

Figure 11 - IMU Data Storage

After the data was correctly stored, it was time to calculate the resultant vector ($R = \sqrt{x^2 + y^2 + z^2}$) of each data point and graph all four components (x, y, z and resultant) with the help of the matplotlib python library. In order to get a better observable dataset, I decided to apply a sliding window smoothing filter in order to remove noise. A sliding window smoothing filter takes a number of data points in a window. This window averages the data points, and then slides one spot, and again takes an average. This sliding effect is repeated until all the data has been traversed. This technique is simple but has a downside of causing the heights of peaks to change¹⁹. However, since the goal of this analysis is to find similarities between the graphs, the height of the peaks are irrelevant, therefore it is feasible to use this sliding window smoothing technique. After this filter was applied, I could analyse the results.

The purpose of this data analysis is to find similarities between the same swing types (e.g. backhand) and differences between them (e.g. backhand and forehand). First, for the backhand swing, it can be observed that in the acceleration graph the z component has always a negative peak. Furthermore, in the gyroscope or angular velocity graph it can be observed that: there is always a positive peak in the z component, there is always a negative peak in the y component and there is always less change in the x component curve (it stays more stable). It is important to note that for all swing types, the axes could be flipped if the racket is rotated. However, for the purposes of this study, I tried to use data in which the racket was oriented in the same way during the movement. In case this was to be done using a machine learning algorithm reading live data from the sensor, it would be necessary to have a feature that detects whenever the racket is rotated and so the axes are rotated as well. Please refer to the figures below (11 to 13) for a confirmation of this analysis.

¹⁹https://www.gamry.com/Framework%20Help/HTML5%20-%20Tripane%20-%20Audience%20A/Content/SE/Data_Analysis/Common%20Tools/Smooth%20Data.htm#:~:text=The%20Smooth%20Data%20command%20offers,the%20data%20have%20been%20traversed.

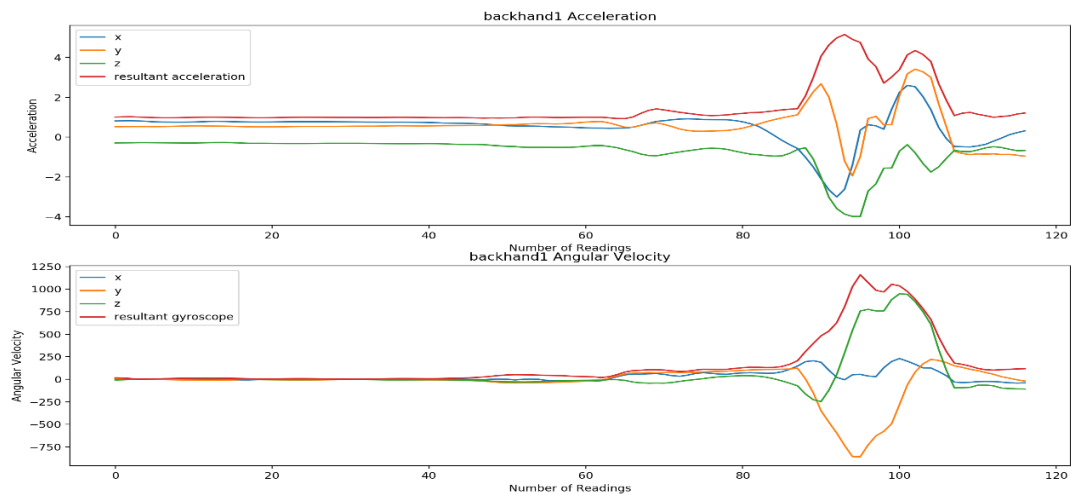


Figure 12 - Backhand Graph 1

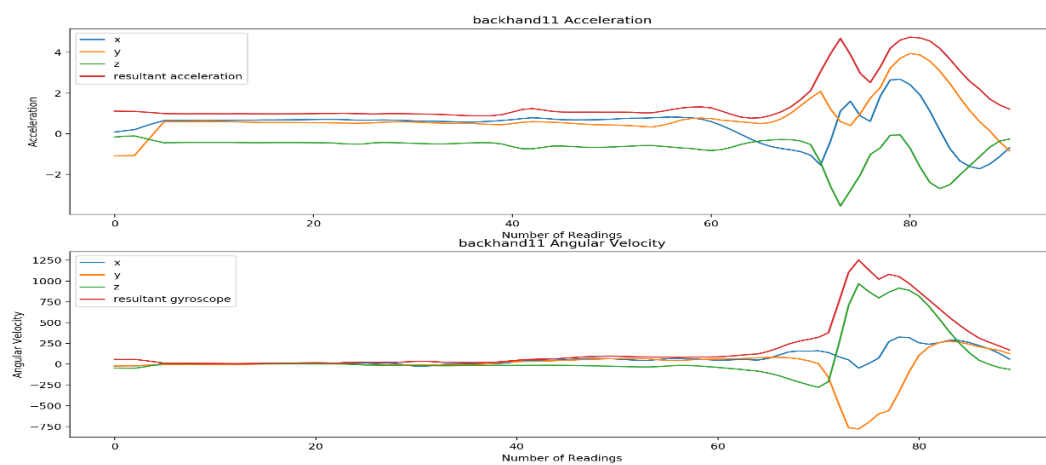


Figure 13 - Backhand Graph 2

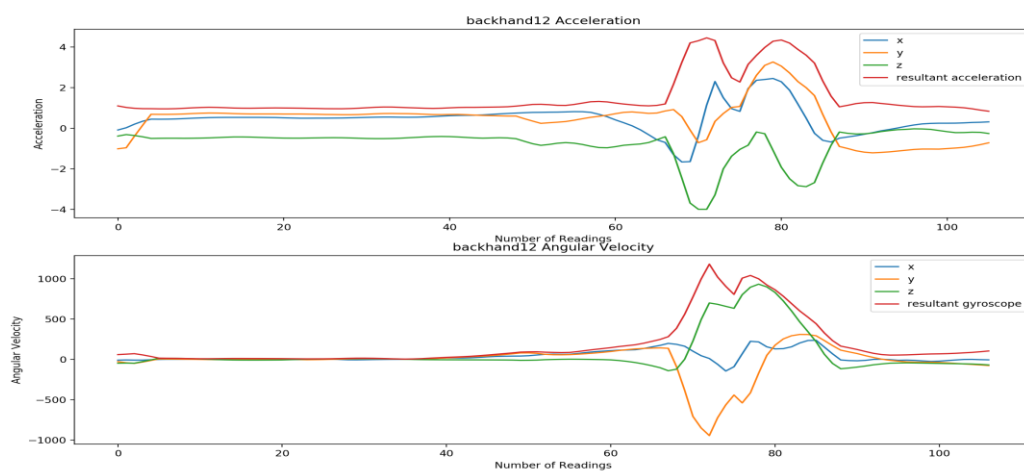


Figure 14 - Backhand Graph 3

Furthermore, the second swing type that can be analysed is the forehand. According to the acceleration graphs, it can be assumed that there is always a double or bimodal peak on the x and z components. On the angular velocity graph, there is a positive peak on the y component, the x component is more stable and has less oscillations and there is a positive peak on the z component. Refer to the figures below to confirm these remarks.

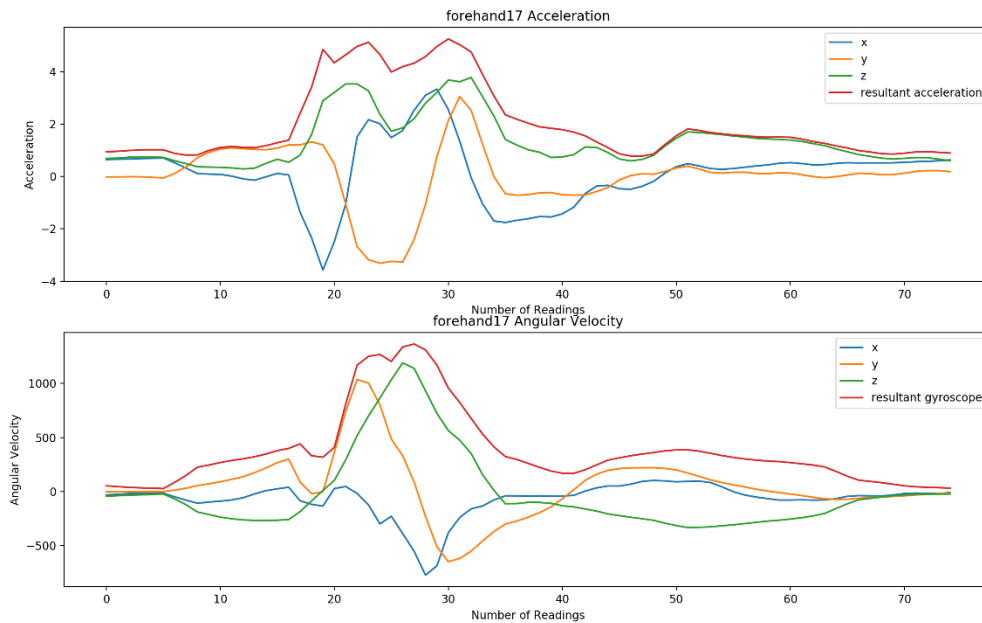


Figure 15 – Forehand Graph 1

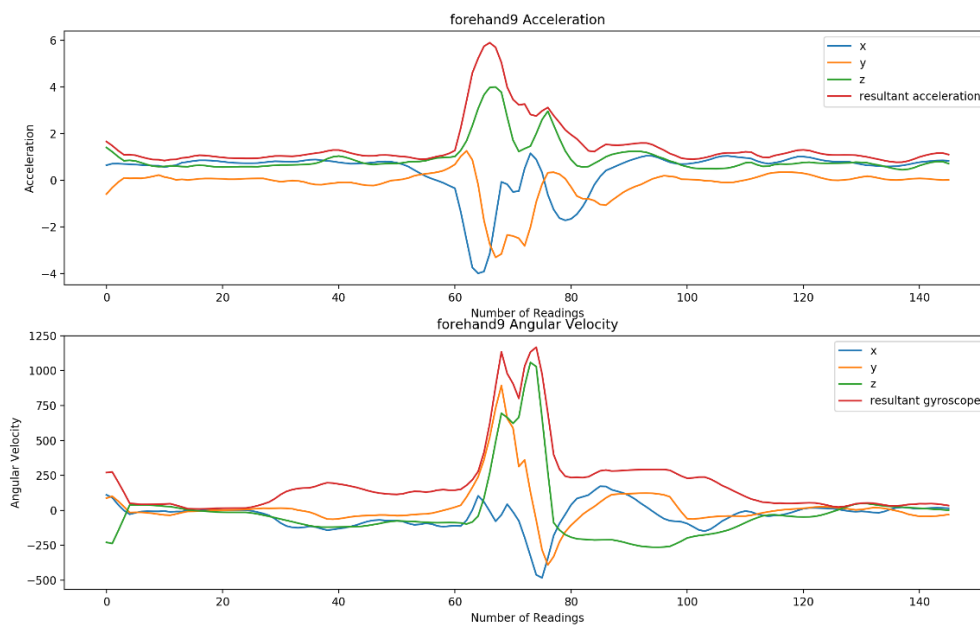


Figure 16 - Forehand Graph 2

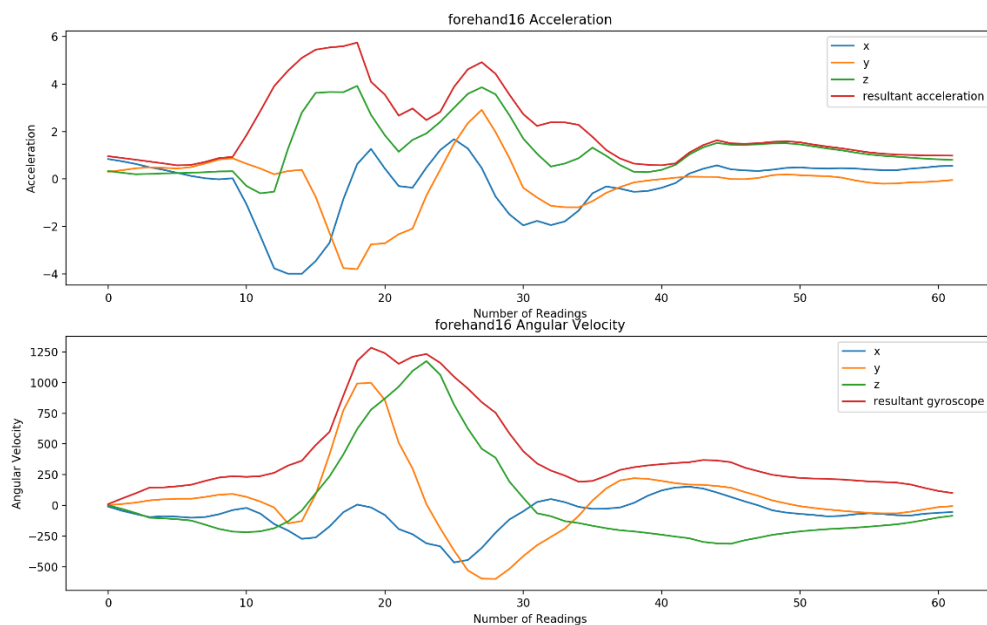


Figure 17 - Forehand Graph 3

The third and final swing type to be analysed is the serve. According to the acceleration graphs, it can be assumed that there is a peak in the z component. On the angular velocity graph, it can be said that there is a positive peak on the y component and a negative peak on the z and x components (the x negative peak being more subtle). Refer to the figures below to confirm these assumptions.

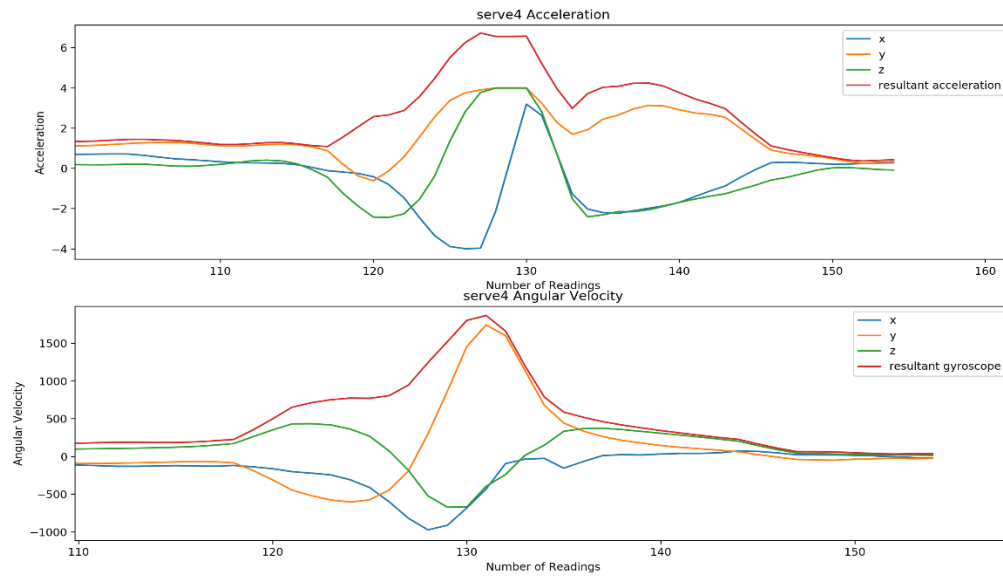


Figure 18 - Serve Graph 1

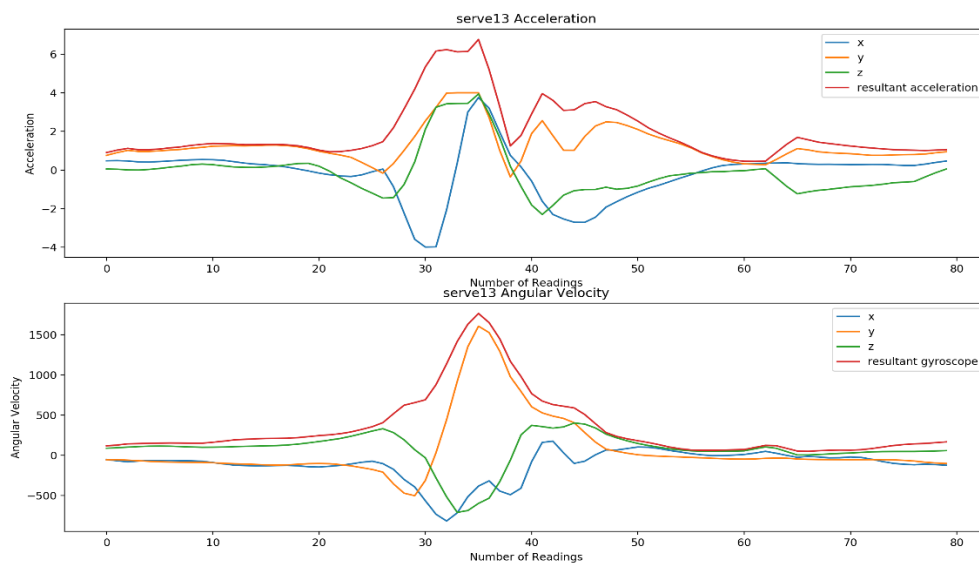


Figure 19 - Serve Graph 2

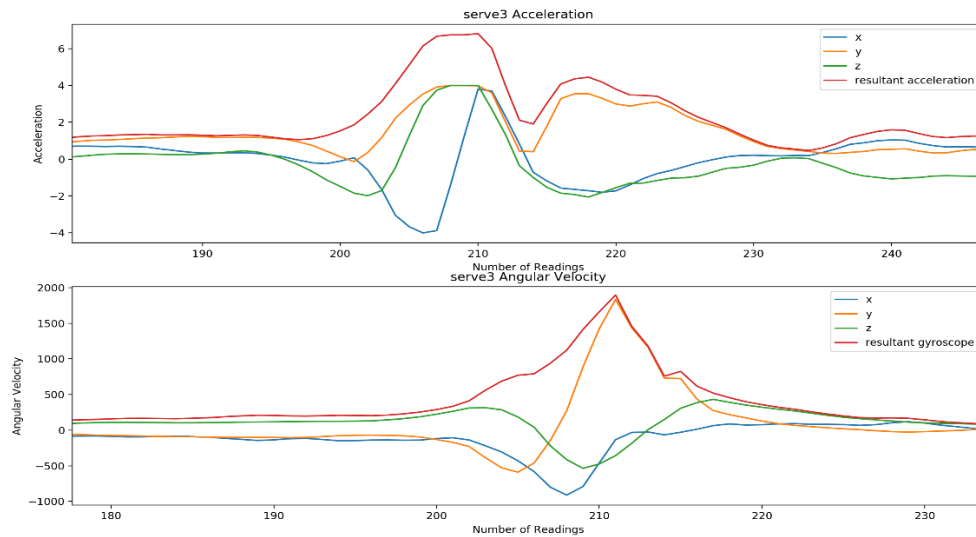


Figure 20 - Serve Graph 3

After reviewing different swing types IMU data through graphs, it can be assured that there are differences between one swing type to the other. Therefore, it can be assumed that these data can be fed into a machine learning classifier that would be able to predict what type of shot is being hit. Furthermore, in order for the classifier to predict what swing type it is, it first needs to know when a hit is starting, according to the graphs above, it can be said that a hit starts when there is a change above a certain threshold in the resultant acceleration. This threshold would then work as a parameter to the classifier that would be able to tell when a hit has started or finished, thus, this threshold parameter for detecting the start of the hit would be tuned for better results. Moreover, the end of the hit could be detected by the same principle, with the algorithm detecting a change in the resultant acceleration below the threshold. In addition, these graphs also tell us the average time of the hit in number of readings which for all of them fall in the range of 20 – 30 readings. It is also known that the frequency in which the sensor is sending the IMU data is 104 Hz, which can give us the average time for these swings to be completed (20 readings = 0.192 seconds approximately and 30 readings = 0.288 seconds approximately) and this time of the hit could also be used to detect the end of the hit. Then, it can be concluded that a classifier would be able to make reliable predictions in relation to what swing is being executed by a player.

5. Conclusion

It is important to note one more time that even though the title of this project is Tennis Training System, this work was not an attempt to make a finalized commercial product. The idea was to explore the major areas which would aid a tennis player in improving his/her game, these areas being computer vision, visual analysis through a user interface and hardware data.

The first objective of this project was to utilize a pose estimation algorithm to correct serve technique, which was accomplished. The second objective was to create a mobile and/or desktop application that would aid the user in visualizing where to improve, which was also successfully completed. The third and final objective was to use hardware to collect live data from a tennis player, which was completed successfully as well. Even though all these objectives were completed, I can conclude that the results that came from them could've been much better if the project had one single focus instead of three different areas of focus. For example, if I had decided to focus solely on computer vision, I could have implemented techniques to detect ball speed, ball rotation, number of shots hit, winner of the point and more. Furthermore, if I had decided to focus solely on the hardware aspect of the project, I might have been able to use a classifier to predict many characteristics from the IMU data. On the other hand, I would have given up the opportunity of learning about different technologies and different areas. Every choice made in this project had advantages and disadvantages, but if I were to start over, I might have focused solely on hardware or solely on computer vision.

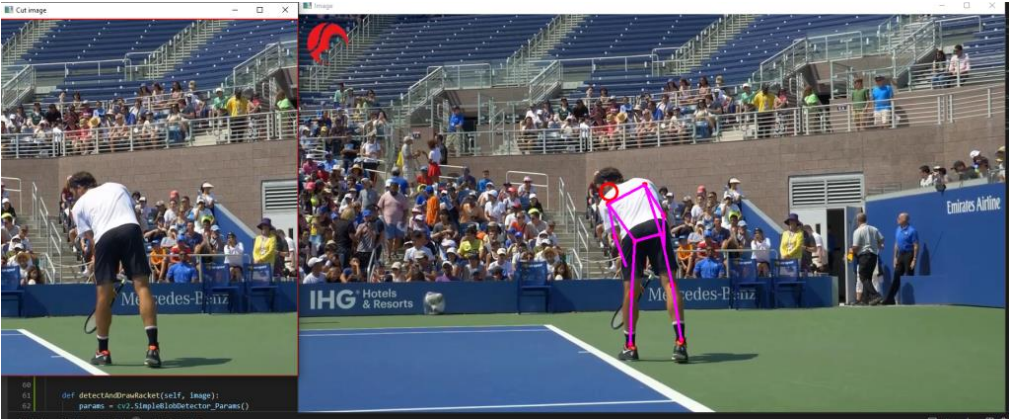
In the computer vision part of the project, one big setback that I had was detecting the racket. If I had previous experience with object detection, I would have known where to start and try the most advanced and complex algorithm from the beginning which had a bigger chance of being successful. Thus, this was a lesson for future object detection endeavours. Moreover, in the app development, one big setback I had was not using the Kivy design language from the beginning, which would have made a following documentation a lot easier. Again, this was my first time creating an app using python Kivy and a lesson that I take for the next times using this technology is to use their own design language. Finally, my application file ended up with some repetitive code that could have been avoided if a plan for the software design of the project was drafted before starting the app, which would be another valuable lesson for when coding an app.

If this project was to be further developed, I would focus on extracting features from the IMU data to train a machine learning model and predict the type of shot being hit by the player and also use an object detection algorithm to detect the racket. In conclusion, this project has taught me new technologies, design strategies, project management and problem solving skills.

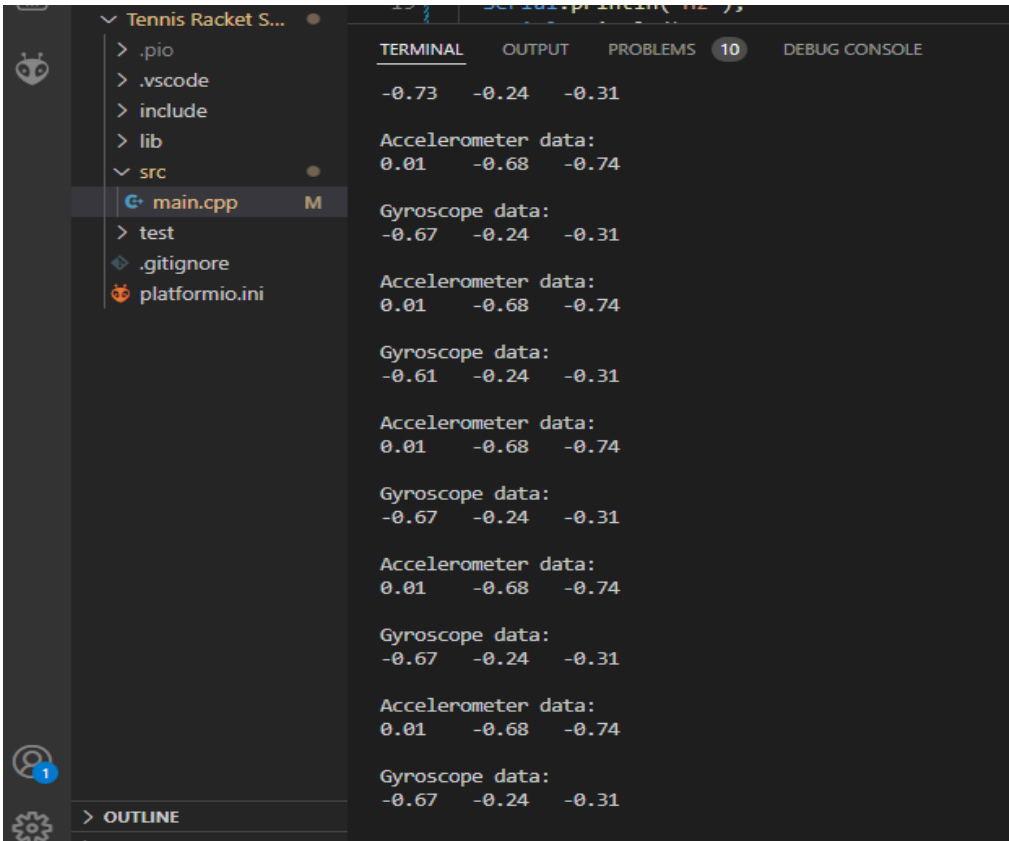
6. Bibliography

- [1] Technostacks (2021). *Top 7 Mobile App Development Frameworks in 2022* / *Technostacks*. [online] Technostacks Infotech. Available at: <https://technostacks.com/blog/mobile-app-development-frameworks>.
- [2] Driscoll, M. (2020). *Build a Mobile Application With the Kivy Python Framework – Real Python*. [online] realpython.com. Available at: <https://realpython.com/mobile-app-kivy-python/>.
- [3] SAKOVICH, N. (2019). *Why Use Xamarin for Cross-Platform Development* / *SaM Solutions*. [online] www.sam-solutions.com. Available at: <https://www.sam-solutions.com/blog/xamarin-cross-platform-development/>.
- [4] Shiralagi, P., Bhandary, R. and Rajeshwari, B. (2020). *A Novel Approach to Tennis Game Analysis and Emulation using Computer Vision*. [online] IEEE Xplore. Available at: <https://ieeexplore.ieee.org/abstract/document/9182453>.
- [5] O'Donoghue, P. and Brown, E. (2008). The importance of service in Grand Slam singles tennis. *repository.cardiffmet.ac.uk*. [online] Available at: <https://repository.cardiffmet.ac.uk/handle/10369/4299>.
- [6] Baily, L., Truong, N., Lai, J. and Nguyen, P. (2020). Stroke Comparison between Professional Tennis Players and Amateur Players using Advanced Computer Vision. *Proceedings of the 8th International Conference on Sport Sciences Research and Technology Support*. [online] Available at: <https://www.scitepress.org/Papers/2020/101458/101458.pdf>.
- [7] Andrea Canella (2021). *Tennis In The Future: An Overview Of Wearables And Sensors*. [online] UBITENNIS. Available at: <https://www.ubitennis.net/2021/01/tennis-in-the-future-an-overview-of-wearables-and-sensors/>
- [8] Mossey, C. (2021). *Cycling Incident Type Detection Using Multivariate Time-Series Classification*. Final Major Project.

7. Appendix



Appendix 1



Appendix 2



Appendix 3



Appendix 4