[Date]

12ShiroY

AI Computer Vision

OCR Computer Science Project

Table of Contents

[1 Analysis 3](#_Toc5987350)

[1.1 The Problem 3](#_Toc5987351)

[1.2 The Stakeholders 3](#_Toc5987352)

[1.2.1 My Stakeholders 3](#_Toc5987353)

[1.2.2 Why this problem is suited to a computational solution 3](#_Toc5987354)

[1.2.3 Stakeholder Interviews 4](#_Toc5987355)

[1.3 Existing Solutions 6](#_Toc5987356)

[1.3.1 In-Depth Breakdown of Hawk-Eye 6](#_Toc5987357)

[1.3.2 Existing solutions of tracking a tennis ball using a Raspberrypi 9](#_Toc5987358)

[1.3.3 Face Recognition and Tracking 9](#_Toc5987359)

[1.4 The Essential Features 11](#_Toc5987360)

[1.5 Potential Limitations 11](#_Toc5987361)

[1.6 Hardware and Software Requirements 11](#_Toc5987362)

[1.7 Success Criteria 13](#_Toc5987363)

[2 Design 14](#_Toc5987364)

[2.1 Structure 15](#_Toc5987365)

[2.2 Decomposition 15](#_Toc5987366)

[2.2.1 Computer Vision 15](#_Toc5987367)

[2.2.2 Tracking the Ball 15](#_Toc5987368)

[2.2.3 User Interface 16](#_Toc5987369)

[2.3 Algorithms 16](#_Toc5987370)

[2.3.1 Accessing camera and displaying camera feed 16](#_Toc5987371)

[2.3.2 HSV colour mask 17](#_Toc5987372)

[2.3.3 Detecting and tracking the tennis ball 18](#_Toc5987373)

[2.3.4 Bounce Detection – String from Tennis Ball Detection and Tracking 20](#_Toc5987374)

[2.4 Usability 21](#_Toc5987375)

[2.4.1 Design of User Interface 21](#_Toc5987376)

[2.4.2 Stakeholder Input 22](#_Toc5987377)

[2.5 Variables, Data Structures and Classes 22](#_Toc5987378)

[2.6 Test Data 23](#_Toc5987379)

[2.7 Further Data 24](#_Toc5987380)

[2.7.1 Bounce Detection 24](#_Toc5987381)

[2.7.2 Flowchart Breaking Down Bounce Detection 25](#_Toc5987382)

[3 Developing 26](#_Toc5987383)

[3.1 Prototypes 26](#_Toc5987384)

[3.1.1 Prototype 1 – Displaying a Video Taken on the Picamera (testing the camera works) 26](#_Toc5987385)

[3.1.2 Prototype 2 – Accessing a Single Image Taken on the Picamera Module 29](#_Toc5987386)

[3.1.3 Prototype 3 – Accessing the Video Stream of The Picamera Module 31](#_Toc5987387)

[3.1.4 Prototype 4 – Tracking a Tennis Ball with a Raspberry Pi and Picamera Module Using Python and OpenCV 33](#_Toc5987388)

[3.1.5 Prototype 5 – Bounce Detection 43](#_Toc5987389)

[4 Evaluation 55](#_Toc5987390)

[4.1 Testing 55](#_Toc5987391)

[4.2 Success Criteria 58](#_Toc5987392)

[4.2.1 Successfully Met Success Criteria 60](#_Toc5987393)

[4.2.2 Partially Met or Unmet Success Criteria 61](#_Toc5987394)

[4.2.3 Stakeholder Interview for Success Criteria 61](#_Toc5987395)

[4.3 Usability Features 63](#_Toc5987396)

[4.3.1 Successful Usability Features 64](#_Toc5987397)

[4.3.2 Partially Successful and Unsuccessful Usability Features 64](#_Toc5987398)

[4.3.3 Final Stakeholder Interview 64](#_Toc5987399)

[4.4 Limitations 65](#_Toc5987400)

[4.4.1 Limitations of the Solution 65](#_Toc5987401)

[4.4.2 Dealing with Limitations in the Future 66](#_Toc5987402)

[4.5 Future Maintenance 66](#_Toc5987403)

[4.6 Finished Code 66](#_Toc5987404)

# 1 Analysis

## 1.1 The Problem

The program I wish to investigate in the program is the potential abilities of computer vision, to complete tasks, which could be applied to the real world. In this case, different types of sports. Above all preference, I would like to create an AI with the ability to find an object in its field of view, and track it, as it travels through its field of view.

In order to do this, I will need to take into consideration the constantly changing light environment, bounce detection and true/false positive ball candidates. I will most likely use a bright luminous yellow tennis ball (as seen on the left), as this has a very clear and bright colour and a very specific spherical shape, making the object unique, allowing me to decrease the room for error in the computer vision by reducing the chances of detecting false positives. I will do this by linking a Pi-Camera to a raspberry pi, and creating a program, using public libraries online, to achieve this.

I believe this can attract a wider range of people to become interested in computer science, and possibly pursuit this as a career. As this product targets people who are interested in computer science and/ or sports, this can allow people who may normally never consider computer science as a subject of study (or possibly even career) to at least think about this.

I will attempt to make this program as accurate and efficient as possible, recording each prototype as I develop this.

## 1.2 The Stakeholders

### 1.2.1 My Stakeholders

Currently, my stakeholders are people who are interested in computer vision, who would like to see this similar idea be used to track any ball within sports. This could make life much easier for camera crews in sports in the future, and much more cost effective as this eliminates a significant amount of skilled human work, which could be replaced by artificial intelligence. My stakeholder age range is between the ages of 14 and 40, as this can range from students interested in sports and/ or film.

My main stakeholder is Tom Smith, who is a 15-year-old male, with an interest in both computer science and Sports. He would like to be able to have a camera, which tracks his tennis ball, while he is playing. Tom would also like to have this possibly implemented with an ability to record himself playing tennis, and possibly create content for a platform such as YouTube. The convenience of having a stakeholder similar to Tom is that he can provide very quick and accurate feedback frequently, allowing me to understand the possibility for improvement. This will allow me to build upon multiple prototypes and have a relevant member of the public to test my product from a separate view to me. This could allow me to make my hardware and/or software more ergonomic to the user.

This was the response from my stakeholder, Tom Smith, when I asked him 10 questions about the project.

### 1.2.2 Why this problem is suited to a computational solution

The problem lends itself to computational methods of finding and implementing the solution due to a variety of methods. The solution will be software, which runs on a Raspberrypi and interacts with a camera of sort to track motion of a tennis ball. This will naturally run on a computer (in this case Raspberrypi) as the computer is necessary for the camera to run (this can be either a webcam or a similar product). This however will limit the products application to an outdoor environment, as this can be easily set up in an indoor tennis court (if there is access to a power supply, monitor, keyboard and mouse), but in an environment where the weather can damage the hardware components, this simply cannot work. This could be further developed in the future if I carried on developing this product in the future however, in this current moment that is not the aim of my project. There is no alternative solution, which would not require a computer.

The main aspects of the solution, which would be controlled by the computer, would be the camera as this is what is used to display a video feed, for the software to analyse and manipulate, in order to detect and track the tennis ball in the field of view.

This may be a major limitation for my stakeholders, as they may want to use this product in an outdoor environment. I will make sure to figure this out in my (1.2.3) stakeholder interview and if necessary, inform them that this product will not work in an outdoor environment.

### 1.2.3 Stakeholder Interviews

#### 1.2.3.1 First Stakeholder Interview

This interview began with me summarising my project, including both its pros and cons. I explained all of the hardware, which I would use, and how it would interact with the software. I heavily emphasised that this product would not work in an outdoor environment, with the software and hardware accessibility, which I possess.

1. **What age range do you think this project is suitable for?**  
   I would like it to be suitable for 15-16 year olds like me. Either gender
2. **Do you find anything confusing about the idea for the project?**No, nothing particularly.
3. **Do you find anything confusing about the hardware?**No, the way you have explained the hardware you are planning to use seems very understandable.
4. **Do you find anything confusing about the software?**I am not sure yet what the software would run on - that is for you to decide. As long as it can track the ball, I do not really mind.
5. **Would you buy this product from a store?**  
   Yeah, if it was relatively cheap and looked high quality. It would be creating YouTube videos etc.
6. **What would you want this product to be able to do?**It must be able to track a ball, hopefully a tennis ball, so I guess it should sense where the ball is in the field of view and track it. If it could learn from its mistakes that would be great.
7. **Do you see any potential application of this in real life?**Yeah, I see great potential for this project in the sports industry and potentially in the film industry, as this could save extreme amounts of money for companies spent on human resources.
8. **Where would you regularly play tennis and are these facilities indoor or outdoor?**  
   I would normally play tennis either at my local gym, which has indoor and outdoor tennis courts however; I only use the indoor ones.
9. **How simple do you think that the user interface needs to be?**The user interface should be simple to use and must include things such as begin tracking; display a message when the ball is located in the field of view.
10. **Would you like to use a voice operated system for this product or not?**That would be awesome!

#### 1.2.3.2 Second Stakeholder Interview

This interview was conducted after I finished both my analysis and research sections. I had a few questions at this stage to ask Tom, in order to find out his opinions on my work so far and whether he had any more input. I also explained to Tom that the software would be running on the Raspberrypi, using Python and either OpenCV or SimpleCV, explaining the differences between them. I was also interested in whether this project could potentially be useful to inspire other students to go ahead and study computer science. If so, I would most likely donate my project to my computer science teacher. Similarly, I was interested in whether or not Tom wanted to go study computer science at a further level himself. This could mean that this project could have many positive implications on people’s education and their choice of study.

1. **After observing and reading the research displayed, which of the existing solutions seems the most impressive?**  
   I believe that Hawk-Eye is by far the most impressive concept of tracking a tennis ball. The level of detailed precision is so high it really just seems impossible.
2. **Would you have any objections (or other suggestions) to me using Adrian’s tennis ball tracking as the basis of my project?**  
   No, I would not have any objections. I understand that your resources are limited and creating a “hawk-eye like” project would probably be impossible.
3. **Do you believe my design ideas seem reasonable for the look of the final product?**I think it is a great design! It looks so cool and simple to understand. If this is what the final product looked like I would be more than satisfied.
4. **Do you believe that this project could potentially be used to inspire younger students to get involved with computer science as a field of study?**  
   I think this would definitely interest younger students in studying computer science. Concepts like this are so impressive and when I show stuff like this to my friends they are always envious that they do not possess the ability to create stuff like this.
5. **Do you find the concept of computer vision interesting?**  
   I find it mesmerising. It is one of the coolest applications of computer science and it is used nearly everywhere now.
6. **Does this project inspire you to study computer science at a higher level than secondary school education?**  
   Definitely! I really want to study computer science in further education at degree level preferably. It would be so cool to make stuff like this one day.

#### 1.2.3.3 Third Stakeholder Interview

This interview was conducted once I finished my fourth prototype. This was done in order to show Tom how the project worked at this stage, what other additions would be made and to understand what alterations Tom would like to make to this project. This would allow me to not have to create unwanted features of this project and remove them later on. Similarly, this would also let me not have to add as many features at the end, in the case that Tom was not satisfied with the final product.

1. **Do you understand how the project works at this moment in time?**  
   Yes, I understand how all the features of the project work and how they link together.
2. **Do you believe that there are any features of this project that are not needed?**  
   Nope. I do not believe that there are any unnecessary features at this moment in time.
3. **If so, would you want me to remove these features?**  
   Not Applicable
4. **Do you understand what sections and features are going to be added to the project?**  
   Yeah, it all makes sense.
5. **Would you like to have any extra features added in the future?**  
   Not really, the voice operated system, which was originally potentially proposed sounded pretty cool, however I understand if that is too ambitious.

## 1.3 Existing Solutions

Hawk-Eye, developed by Paul Hawkins, is a computer system owned by Sony, used in many sports to visually track the trajectory of the ball and display a profile of its statistically most likely path as a moving image. This piece of technology is used for sports such as tennis, football, cricket and more. In summary, this is six very high performance cameras used to create a 3D representation of the ball’s trajectory, frame by frame. This then allows the use of complex mathematics to predict the movement of the ball based on its trail. It is not perfect, but it is pretty close, as Hawk-Eye is accurate to 3.6 millimetres.

### 1.3.1 In-Depth Breakdown of Hawk-Eye

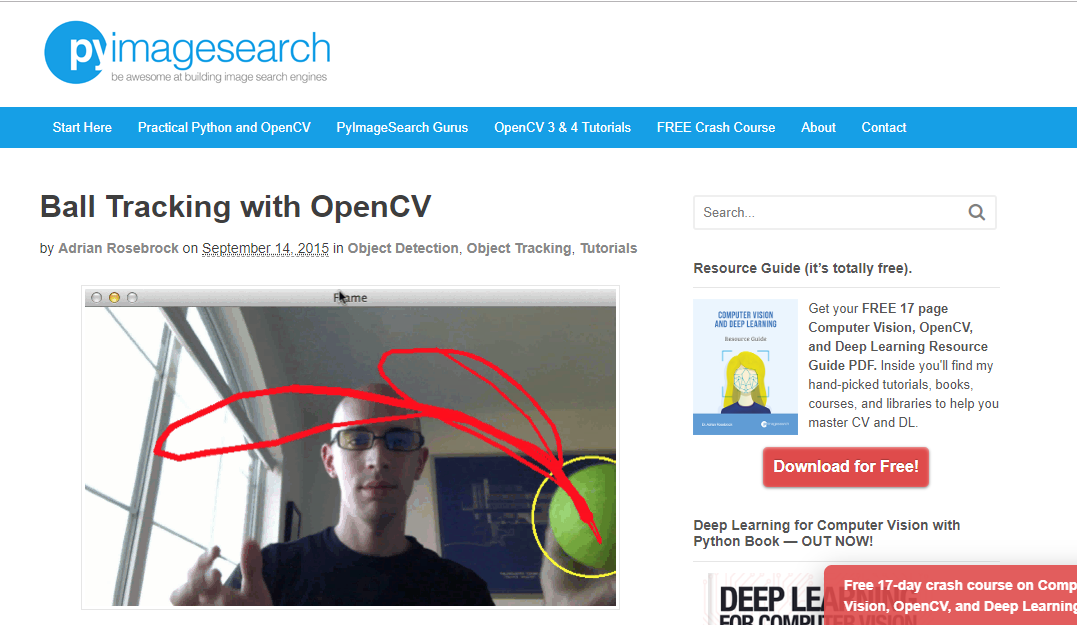
All information learnt, and screen shots used are from this video (“Visual tracking of a tennis ball” - <https://youtu.be/iRlWw8GD0xc>) on the 13/11/2018.

Assuming that ball candidates (what has been detected as a potential tennis ball) in each frame have already been detected, tracking the tennis ball is then broken down into which candidates, are object-originated (true positives), and which are clutter-originated (false positives).   
The candidates can be plotted in a row-column-time 3D space. The objective of this is to recover the class labels of the candidates. The approach used here is “a layered data association scheme.” A candidate triplet (three frames containing the potential tennis ball in each of them, very close to one another) is selected from the beginning of the sequence, which has an extremely high probability of continuing with only true positives. A dynamic model is then fitted, and then optimised recursively until convergence (the other candidates (near the triplet) seem to line up). This optimised model is called a “tracklet”. As a sliding window moves, a sequence of tracklets are generated. Simultaneously, a graph is constructed, where each node is a tracklet, and the edge difference between two nodes is defined according to the “compatibility” of the two tracklets. The graph is then sectioned into sub-graphs, and the optimal path in each graph is found. The desired data association result is then contained in the optimal paths. Next, interpolation and hit/bounce detection is used to create the final product, the tennis ball tracking computer vision.

The way in which my program will track the tennis ball, will be mainly focused by tracking the HSV values of the tennis ball, as the colour is very bright and distinct. This will make it much easier as I will not have to find the path of the tennis ball for the tracking of the ball. I could use path prediction and detection for detecting the bounce of the tennis ball. However, this is a very complex solution to the task, which isn’t exactly necessary for my situation. I can use hit detection instead, by tracking the HSV values of the tennis ball and the floor (this could be a tennis court in reality however, I will use a piece of card which will most likely be either red or blue as these colours are easier to track, due to how they stand out in a room) with a box around the contours. If the box tracking the tennis ball overlaps with the box, which tracks the floor, this can allow us to assume that a bounce was detected. I will speak further on this topic in the design section (2.7.1 Bounce Detection)



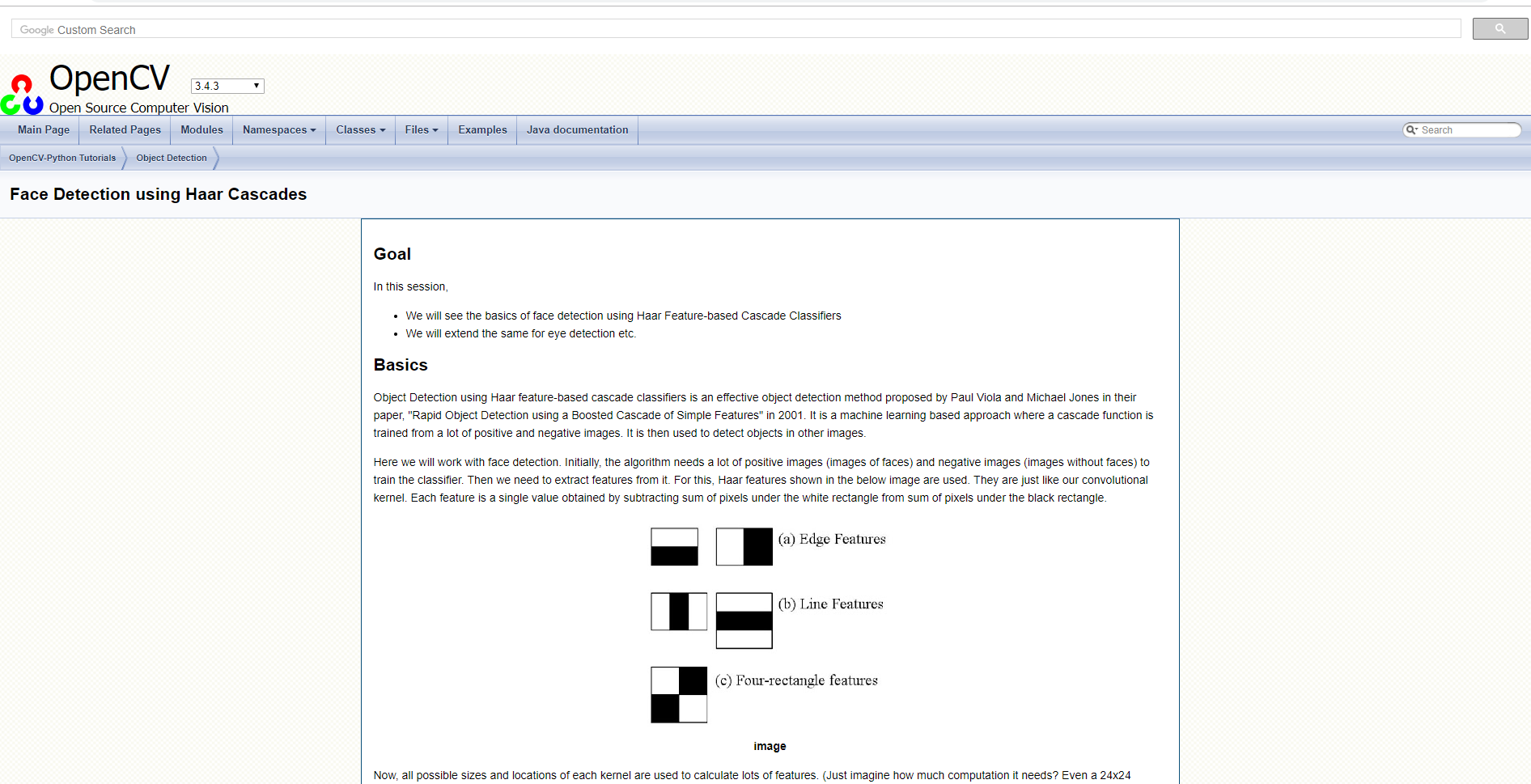
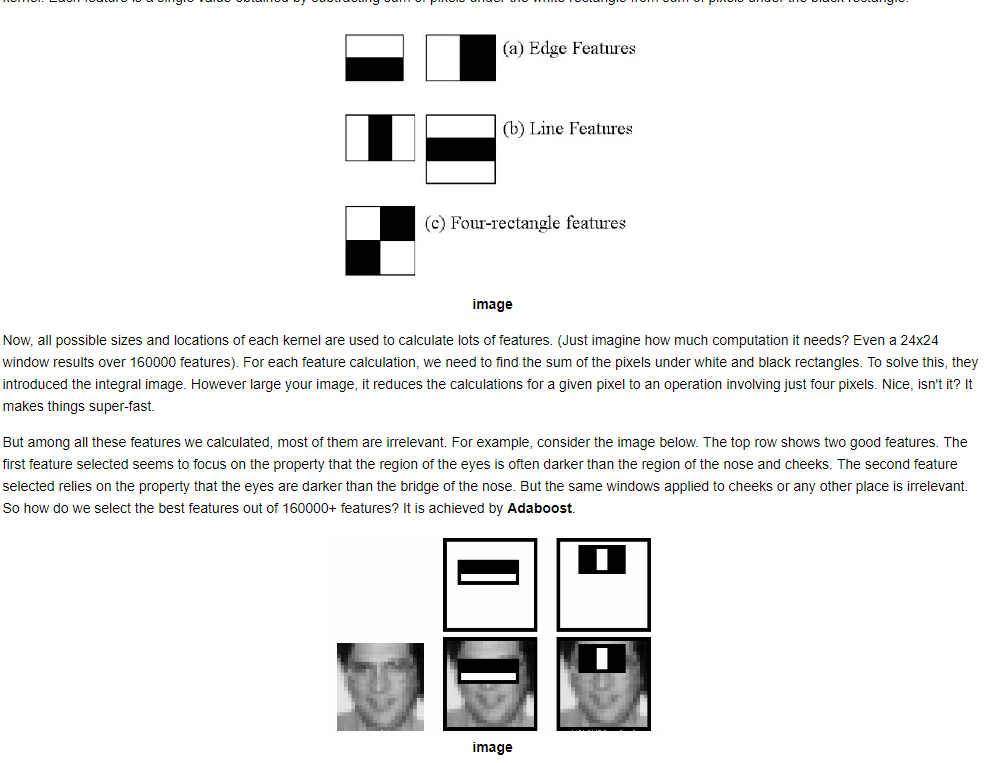
### 1.3.2 Existing solutions of tracking a tennis ball using a Raspberrypi

One of the more popular and easier solutions to tracking a tennis ball using a Raspberrypi is by Adrian Rosebrock, published on the 14th of September 2015, on his blog, “PyImage Search”. (link to referenced website – “<https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/>”) In this post, Adrian tracks a tennis ball using a Raspberrypi and a webcam. He tracks a tennis ball by drawing a circle around the tennis ball and drawing the tennis ball’s path in the camera’s field of view with a red trail.

I came across Adrian’s blog after I ran into a problem with using SimpleCV in my development section (3.2.1 – prototype 1). Adrian simply finds the HSV upper and lower boundaries of the tennis ball using a separate mask and stores these. He then tracks these by drawing the contours and tracking the largest contours with a circle around them. Adrian uses many different techniques to make tracking the tennis ball much easier such as blurring the image and removing the static, as this removes many of the false positives. Finally, Adrian finds the midpoint of the circle and draws a red trail behind the ball as it travels through the webcam’s field of view.

Adrian’s solution was very clear and helpful for me in many ways, as it allowed me to get the basic understanding of the computational solution to my problem, as well as the necessary help in understanding the code.

### 1.3.3 Face Recognition and Tracking

This is another example of object recognition and tracking using computer vision. This problem is solved using the Haar feature-based cascade classifiers. In summary, a Haar Cascade is a classifier which is used to detect the object for which it has been trained for, from the source. This is posted on the OpenCV website (the website referred to in example – “<https://docs.opencv.org/3.4.3/d7/d8b/tutorial_py_face_detection.html>”). The Haar Cascade is trained by superimposing the positive image over a set of negative images. The training is generally done on a server and on various stages. This is similarly used for eye detection here. This is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. This is a machine learning based approach where a cascade function is trained from many positive and negative images. It is then used to detect objects in other images. “Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.”

## 1.4 The Essential Features

The essential features of this project consist of a program with the ability to detect a tennis ball in its field of view, and then track this tennis ball as it moves across the room. Another key feature is for the program to be able to detect when the tennis ball bounces off a surface. Some of the stakeholder requested features consist of ergonomic usability and a simple to understand user interface.

## 1.5 Potential Limitations

The price of this project is a major limitation as in the worst-case scenario; I may have to spend over £100 on this project. However, I will try to avoid this by using few expensive components such as the raspberrypi and the picamera; possibly components which I already possess such as a webcam. Another limitation is the complexity of the coding necessary to complete this task. This project requires an extremely high knowledge of python, which I do not yet possess, and this will mean that I will have to put in extra time at home to build my knowledge of this. Another possible limitation for the project would be if there were a flaw with the hardware (e.g. response time or frame rate of camera, faulty pins on the raspberry pi etc.). This could potentially limit the processing speed and overall, limit the ability of the camera to track the ball in the air. I will try to avoid this by keeping track of the computer usage throughout the evolution of the project, making sure nothing gets overloaded or potentially break, which could spawn other limitations such as price instantly.

Other limitations of my project will be the actual application of my solution. An example of this is that my solution can only be used indoors. This is because it would have to be connected to a monitor as well as being plugged into a power supply. Another reason for this is that none of my components are waterproof, which would mean that this product could not be used in all weather environments, even if it was possible to make it portable. Another possible limitation is the ability to accurately make sure, than no false positives occur in a constantly (light) changing environment.

## 1.6 Hardware and Software Requirements

|  |  |  |
| --- | --- | --- |
| **Number of Requirement** | **Hardware Requirements** | **Software Requirements** |
|  | A raspberry pi – most likely a raspberry Pi zero as I am working on a limited budget but ideally, a Raspberry Pi 3 B+, as it has a variety of essential upgrades in comparison to the cheaper alternative. I would also need to buy essential parts to use the raspberry pi such as their power block, a micro SD card and potentially, a case (to prevent any damage being done to the raspberry pi). The features of the Pi Zero: BCM 2835 SOC @ 1GHz, 512MB of RAM, micro-SD, mini-HDMI, micro-B USB for data, micro-B USB for power, CSI camera connector (needs adaptor cable for an extra £5), Unpopulated 40-pin GPIO connector, Compatible with existing HAT add-ons, Dimensions: 65mm x 30mm x 5mm  The Features of the Pi B+: 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU (BCM2837), 1GB RAM (LPDDR2 SDRAM), On-board wireless LAN - dual-band 802.11 b/g/n/ac (CYW43455), On-board Bluetooth 4.2 HS low-energy (BLE) (CYW43455), 4 x USB 2.0 ports, 300Mbit/s Ethernet, 40 GPIO pins, Full size HDMI 1.3a port, Combined 3.5mm analog audio and composite video jack, Camera interface (CSI), Display interface (DSI), microSD slot, VideoCore IV multimedia/3D graphics core @ 400MHz/300MHz | Computer Vision – the program needs to be able to understand that a tennis ball is in its field of view and be able to track it. Similarly, the program needs to understand where the floor (surface) is located in its field of view. |
|  | (possibly) Voice Recognition kit for Google Cloud Speech – this consists of 170 tie point mini breadboard, 10 male to male 20cm breadboard cables, MAX9812 amplified microphone, NPN transistor, red LED and two resistors (1k and 220ohm). | A UI – easily understandable and useable user interface, suitable for all the essential criteria of the stakeholder |
|  | The Raspberry Pi Camera -  “Raspberry Pi v2.1 8 MP 1080p Camera Module” is the perfect camera for me, however it is on the expensive side of the discussion, however the benefits of its features may outweigh the con of the price. It records in 1080p with 60 fps, meaning that the potential limitation of lack of frames is most likely eliminated here; making prediction of the path (of the ball travelling through the air) will be more accurate. It also has a wide-angle lens, meaning that it can track a tennis ball significantly further in the environment. It is worth around £24 on Amazon.  “Raspberry Pi 3 2 model B B+ A+ Mini Camera Video Module 5MP 1080p OV5647 Sensor with 15 Pin FPC Cable + Pi Zero Ribbon Cable 15cm” is significantly cheaper than the v2.1 (only £11) however, it is limited to 30 fps and a regular flat lens. This could mean that it would be a waste of money for me to buy this product, as in my circumstance, I will definitely need more than 30 fps.  The features of an available webcam (Logitech 960-000582 C270 USB HD Webcam – Black) – The camera records in 720p. Logitech HD Webcam C270 is an easy-to-use webcam for widescreen video calling and recording. You can take 3 MP photos (software enhanced). Dim or poorly backlit settings and background noise will not ruin videos thanks to automatic light correction and a noise-filtering microphone. This webcam offers high video quality and is easy to set up. Fluid Crystal is a Logitech proprietary algorithm that automatically adjusts the frame rate, sharpness, colour saturation and audio to provide picture and motion. | Artificial Intelligence – a fully functional AI with the ability to detect when a tennis ball bounces off the floor (surface). This AI needs to be able to have the flawless bounce detection function, as well as tracking the tennis ball in the camera field of view. |
|  | Currently no other hardware requirements in the original stage | Currently no other software requirements in the original stage |

## 1.7 Success Criteria

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Explanation of Criteria** | **How Criteria is Met** |
| 1) The program needs to understand where the ball is in the field of view | The program must be able to recognise the ball if it is in its field of view, and if it is where bouts it is in each frame. | I will implement this by allowing the program to understand the HSV colour of the ball, and how it changes in differently lit environments. |
| 2) The program needs to understand when the ball bounces | The program must be able to understand whether the ball bounces and recognise when this occurs. | I will implement this by allowing the program to recognise, when the tennis ball connects with the surface. This will be done via hit detection. I will track the HSV values of the tennis ball with a box and the surface HSV values with another box. When the boxes overlap, a bounce will be detected. |
| 3) The program needs to track the ball | The program needs to be able to track the ball in its field of view. | I will implement this by using probability, after eliminating the false positives. This will be done by observing the location of the ball in each frame via tracking the HSV values of the tennis ball. |
| 4) The program needs to eliminate false positives | The program needs to be able to distinguish between false positives and true positives via probability, and eliminate the false positives. | I will implement this by using probability of potential ball candidates in each frame (size of the largest HSV contours detected) to allow me to understand which candidates are true positives. |
| 5) The whole project needs to be suitable for both genders | The project needs to be simple, easy to use and suitable for both genders | I will implement this by keeping by project as easy to understand as possible for someone without any knowledge of computing. I shall also keep the colours and style based around brown, grey and black. |
| 6) The project needs to be suitable for ages between 15-16 | The project needs to be ergonomic for users and suitable for a wide variety of children/ teenagers of different ages | I will do this by keeping my product slick, easy to use and exciting. This will be achieved by not undermining the potential, which this technology can achieve. |
| 7) The project needs to be relatively simple to use | The project needs to be simple to use and understand by my users, as my stakeholders will be children aged 15-16. Simultaneously, this shouldn’t patronise them | I will do this by keeping my user interface simple and easy to understand. In order for the user to fully understand the program they will be using, I will write a short instruction list, explaining how the project works and how to use it. |

# 2 Design

## 2.1 Structure

## 2.2 Decomposition

My project breaks down into two main sections, software and hardware. These then break down further as shown in the hierarchy diagram above. This shows the basic “barebones” concept of my project, which shows how I could potentially dedicate time to each section, building it up to form the combined final product. Finally linking this into an understandable user interface. This is not one of the main sections as this is not as important in the program working effectively.

### 2.2.1 Computer Vision

This is the most important part of the program itself, as without the ability for my program to detect a tennis ball candidate in its field of view, none of the other parts of the program can work as intended.

First, I will need to create a repository for the program to store the HSV (Hue Saturation Value) values of the tennis ball. This will allow the program to detect the ball candidates in a constantly changing environment (in terms of lighting) much more accurately.

### 2.2.2 Tracking the Ball

#### 2.2.2.1 Finding Ball

This is a key stage in the object tracking, as it will allow the program to determine whether the ball candidate is a true positive or a false positive. This will be done by finding the HSV values of a tennis ball, using a HSV colour mask. Then the program will find the biggest contours in the frame with those HSV values and track them with a box around them.

#### 2.2.2.2 Bounce Detection

This is a relatively simple part of the program, as this will require the program to detect a “hit”, hence when the ball collides with the surface. We can save a significant amount of time here as this can mainly be done by reusing code, as we will be tracking the HSV values of a specific surface (most likely a piece of card of either blue or red colour) with a box. This box will not be visible in the final version of the solution. When the box, which tracks the tennis ball, overlaps with the box, which tracks the surface, we can assume that a “hit” was detected. Therefore, a bounce can be detected. This will be done by using the already existing code to track the HSV values of the surface.

### 2.2.3 User Interface

For this part of my project, I would like to keep as close to my stakeholder wishes as possible. Tom Smith requested that the program was gender neutral, suitable for a wide age range and simple yet easy to use. I would like my UI to have a simple black and white colour scheme, with green and red boxes around the tennis ball, which would be tracked. This part is very simple and the project does not heavily rely upon the design of the UI.

## 2.3 Algorithms

For this section, I will have three separate sections of pseudocode. The first section of my pseudocode will be to access the raspberry pi camera and display the video feed. The second section will be for the HSV mask, which will take the input (video feed) from the webcam and separate it into Hue Saturation and Value, helping me find the specific HSV of my tennis ball. The third section of pseudocode is to both detect and track the tennis ball, which would use both the shape of the tennis ball and the HSV. For my situation, the HSV is the main variable, which I will use to track the tennis ball by, as it is very easy to isolate the HSV colour values of a tennis ball.

### 2.3.1 Accessing camera and displaying camera feed

#import packages  
import system.packages

#define camera resolution + fps and reference to raw camera capture  
camera = picamera()  
camera.resolution = (x,y)  
camera.framerate = 32  
rawcaprute = PiArray(camera, size =(x,y))

#camera warmup time  
time.to.warmup(1)

#capture frames from camera and loops until we stop running the program  
for frame in camera.capture\_continous(rawcapture, format = ‘bgr’, use\_video\_port= true):

#grab raw image  
image = frame.array

#show frame  
 img.show(‘frame’, image)

#clear stream and prepare for next frame  
 rawCapture.truncate(0)

#end loop  
break

#destroy all windows and release the video capture  
destroy.all.windows()  
camera.capture.release()

### 2.3.2 HSV colour mask

#import packages  
import system.packages

#define camera resolution + fps and reference to raw camera capture  
camera = picamera()  
camera.resolution = (x,y)  
camera.framerate = 32  
rawcaprute = PiArray(camera, size =(x,y))

#open a window to define the upper and lower HSV boundaries  
open.namedwindow(‘HSV’)  
#lower range sliders  
createtrackbar(‘lowHue’, ‘HSV’, trackbar[0], 255, nothing)  
createtrackbar(‘lowSaturation’, ‘HSV’, trackbar[1], 255, nothing)  
createtrackbar(‘lowValue’, ‘HSV’, trackbar[2], 255, nothing)   
#upper range sliders  
createtrackbar(‘highHue’, ‘HSV’, trackbar[3], 255, nothing)  
createtrackbar(‘highSaturation’, ‘HSV’, trackbar[4], 255, nothing)  
createtrackbar(‘highValue’, ‘HSV’, trackbar[5], 255, nothing)

#create loop to get HSV values from UI sliders  
while true:   
#return the fps of camera  
 timeCheck = time.time()  
 #now get HSV values  
 lowHue = getTrackbarPos(‘lowHue’, ‘HSV’)  
 lowSaturation = getTrackbarPos(‘lowSaturation’, ‘HSV’)  
 lowValue = getTrackbarPos(‘lowValue’, ‘HSV’)  
 highHue = getTrackbarPos(‘highHue’, ‘HSV’)  
 highSaturation = getTrackbarPos(‘highSaturation’, ‘HSV’)  
 highValue = getTrackbarPos(‘highValue’, ‘HSV’)

#camera warmup time  
time.to.warmup(1)

#capture frames from camera and loops until we stop running the program  
for frame in camera.capture\_continous(rawcapture, format = ‘bgr’, use\_video\_port= true):

#grab raw image  
image = frame.array

#show frame  
 img.show(‘frame’, image)

# Convert the frame to HSV colour model.  
 frameHSV = BGR.cvtColor(frame, COLOR\_BGR2HSV)

# HSV values to define a colour range to create a mask from  
colorLow = array([lowHue,lowSat,lowVal])  
colorHigh = array([highHue,highSat,highVal])  
mask = inRange(frameHSV, colorLow, colorHigh)  
# Show the first mask  
imshow('mask-tennis-ball, mask)

im2, contours, hierarchy = findContours(mask, RETR\_TREE, CHAIN\_APPROX\_SIMPLE)

contour\_sizes = [(contourArea(contour), contour) for contour in contours]  
biggest\_contour = max(contour\_sizes, key=lambda x: x[0])[1]

# Show final output image  
imshow(‘HSV’, frame)

#clear stream and prepare for next frame  
 rawCapture.truncate(0)

k = waitKey(5) & 0xFF  
if k == 27:

#end first loop  
break  
#print fps  
print('fps - ', 1/(time.time() - timeCheck))

#end second loop  
break

#destroy all windows and release the video capture  
destroyAllWindows()  
vidCapture.release()

### 2.3.3 Detecting and tracking the tennis ball

#import packages  
import system.packages

#define camera resolution + fps and reference to raw camera capture  
camera = picamera()  
camera.resolution = (x,y)  
camera.framerate = 32  
rawcaprute = PiArray(camera, size =(x,y))

# define the lower and upper boundaries of the colour of the tennis ball in the HSV colour space, then initialize the list of tracked points  
greenLower = (best lower HSV boundaries found for tennis ball)  
greenUpper = (best upper HSV boundaries found for tennis ball)  
pts = deque(maxlen=args["buffer"])

#camera warmup time  
time.to.warmup(1)

# keep looping  
while True:

# grab the current frame  
frame = grab.frame()

# resize the frame, blur it, and convert it to the HSV colour space  
 frame = resize (frame, width=320)  
 blurred = camera.blurred (frame, (11, 11), 0)  
 HSV = BGR.cvtColor(blurred, COLOR\_BGR2HSV)

#construct a mask for the colour of the tennis ball “green”, then preform dilations and erosions in order to remove tiny blobs left in the mask.  
mask = inRange(hsv, greenLower, greenUpper)  
mask = erode(mask, None, iterations=2)  
mask = dilate(mask, None, iterations=2)

# find contours in the mask and initialize the current (x, y) center of the ball

cnts = findContours(mask.copy(), RETR\_EXTERNAL,  
CHAIN\_APPROX\_SIMPLE)  
cnts = grab\_contours(cnts)  
center = None

# only proceed if at least one contour was found  
if len(cnts) > 0:  
# find the largest contour in the mask, then use it to compute the minimum enclosing box around the tennis ball  
contour\_sizes = [(contourArea(contour), contour) for contour in contours]  
biggest\_contour = max(contour\_sizes, key=lambda x: x[0])[1]  
x,y,w,h = boundingRect(biggest\_contour)  
rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

#make sure that if the tracked points are lost, then they are ignored  
for i in range(1, len(pts)):  
 if pts[i – 1] = 0 or pts[i] = 0 then  
 continue  
#clear stream and prepare for next frame

rawCapture.truncate(0)

k = waitKey(5) & 0xFF

if k == 27:

#show the frame to my screen and detect any keypresses   
imgshow(‘Frame’, frame):  
key = waitkey(1):  
#if q key is pressed then stop the loop  
if key == ord(‘q’):  
 break

#print fps  
print('fps - ', 1/(time.time() - timeCheck))

#end second loop  
break

#destroy all windows and release the video capture  
destroyAllWindows()  
vidCapture.release()

### 2.3.4 Bounce Detection – String from Tennis Ball Detection and Tracking

#tracking the surface  
#set values already found for HSV of surface  
hsvLower = x1,y1,z1  
hsvUpper = x2,y2,z2

# resize the frame, blur it, and convert it to the HSV colour space for a separate frame. The frame for the surface  
frame2 = resize (frame, width=320)  
blurred2 = camera.blurred (frame2, (11, 11), 0)  
HSV2 = BGR.cvtColor(blurred2, COLOR\_BGR2HSV)

#construct a mask for the colour of the tennis ball “green”, then preform dilations and erosions in order to remove tiny blobs left in the mask.  
mask2 = inRange(hsv, hsvLower, hsvUpper)  
mask2 = erode(mask, None, iterations=2)  
mask2 = dilate(mask, None, iterations=2)

# only proceed if at least one contour was found  
if len(cnts) > 0:  
# find the largest contour in the mask, then use it to compute the minimum enclosing box around the surface  
contour\_sizes = [(contourArea(contour), contour) for contour in contours]  
biggest\_contour = max(contour\_sizes, key=lambda x: x[0])[1]  
x,y,w,h = boundingRect(biggest\_contour)  
rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

#make sure that if the tracked points are lost, then they are ignored  
for i in range(1, len(pts)):  
 if pts[i – 1] = 0 or pts[i] = 0 then  
 continue  
#clear stream and prepare for next frame

rawCapture.truncate(0)

k = waitKey(5) & 0xFF

if k == 27:

#show the frame to my screen and detect any keypresses   
imgshow(‘Frame Surface’, frame2):  
key = waitkey(1):  
#if q key is pressed then stop the loop  
if key == ord(‘q’):  
 break

#if coordinates of surface overlap with coordinates of tennis ball, a bounce is detected via hit detection  
if bottomOfBox(tennisball).ycoordinates < topOfBox(surface).ycoordinates

## 2.4 Usability

### 2.4.1 Design of User Interface

This is currently the way that I want my user interface to look like. As you can see, the tennis ball is tracked with a bright green box around it, which changes to red when the tennis ball bounces. The use of the colours green and red are used to allow for the user to easily understand when a bounce is detected, and to allow for simple testing, as it is easy for me to determine the change in the colour from green to red. In the top right corner, there is an fps counter, which will mainly help me to see whether the raspberry pi is doing too much work, or is failing in some sort to achieve the necessary frame count. I want my fps count to be above 10 ideally.

I have tried to make my UI as simple as possible, to make it obvious for the user to understand what is going on. In addition, I have tried to make it suitable for both genders and for a wide age range, by keeping the UI and the colours very basic, using black, green and red. I will also limit the program to only being able to detect one tennis ball at a time, limiting the chance of false positives being detected simultaneously as true positives. This will be done by tracking the largest HSV contours.

I have also displayed that the surface, which I will use, will be blue. I chose this as it clearly contrasts the green of the tennis ball, meaning that tracking the surface would be much easier. This can allow for hit detection to work much more effectively, resulting in accurate bounce detection.

### 2.4.2 Stakeholder Input

My stakeholder, Tom Smith, stated that he would like this project to be suitable for 15-16 year olds of either gender. Therefore, the best thing for my user interface would be of a generic colour scheme. I decided that I would use black, white, green, red and blue as the colours in my project. Green and blue to easily distinguish when bounce detection occurs, and black and white as a basic colour scheme, which is suitable for either gender. The red and green colours are not very ergonomic for colour-blind users; however, this could be altered after publishing the solution.

## 2.5 Variables, Data Structures and Classes

|  |  |  |
| --- | --- | --- |
| **Variable / Data Structure Name** | **Type** | **What The Variable Does** |
| GreenUpper | Array | Stores the HSV upper boundary of the “green” ball (Hue (dominant wavelength), Saturation (purity/ shades of the colour) and Value (intensity)) |
| GreenLower | Array | Stores the HSV lower boundary of the “green” ball (Hue (dominant wavelength), Saturation (purity/ shades of the colour) and Value (intensity)) |
| BallCancidate | Record | Stores a record of different combinations of HSV values and the size of the ball HSV contours, to eliminate false positives |
| BallCandidateDetected | Variable - Boolean | Returns whether there are any ball candidates in its field of view |
| Surface | Function | Tracks the HSV values of the surface |
| Bounce | Function | Responsible for sending information to the program that the ball candidate, which is being tracked, has been exposed to the tracked surface |
| BounceDetected | Variable - Boolean | Returns whether there are any bounces detected. Does this by changing colour of the box, which tracks the tennis ball from green to red |
| Camera | String | Initialise the camera and declare it |
| CameraResolution | Integer | Determine the resolution of the camera, keeping it reasonably low, in order to maximise the framerate |
| CameraFramerate | Integer | Show the fps of the camera |
| Frame | Procedure | Display the frame to my screen |
| Mask | Function | Convert BGR to HSV |
| Track | Function | Draw a green box around the largest contour, and track it as it moves through the camera’s field of view. |
| CloseKey | Function | Stops the program from running and closes all windows when the “escape” key is pressed |

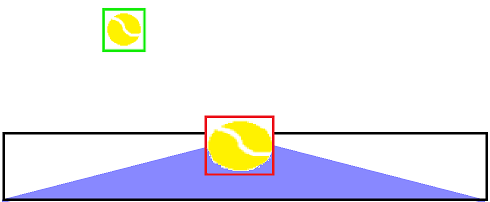
## 2.6 Test Data

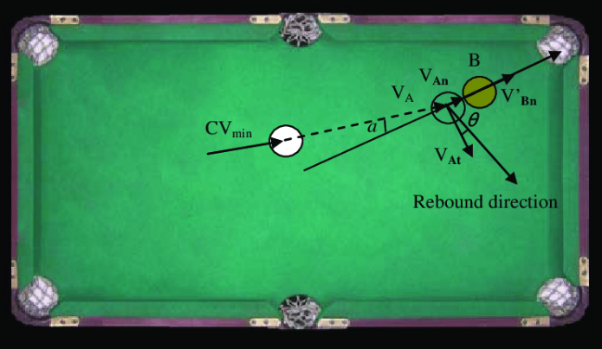
For this section of my project, I intend to create some ideas for testing my first prototype. Once the first prototype is completed, I will fill in this test table with findings and evidence via screenshots.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Is the ball detected | *General description of the data gathered from the test* | Yes the ball is detected |  |
| Is a message given when the ball is detected | *General description of the data gathered from the test* | Yes the box in the top left of the form states “DETECTED” in green when the ball is detected |  |
| Is a message given when the ball is not detected | *General description of the data gathered from the test* | Yes the box in the top left of the form states “NOT DETECTED” in red when the ball is not detected |  |
| Is the ball tracked with a green box | *General description of the data gathered from the test* | Yes the ball is tracked with a green box |  |
| Does the box change to red when the ball bounces | *General description of the data gathered from the test* | Yes the box which tracks the ball changes to red when the ball bounces |  |
| Does the box change the colour back to green after the bounce is detected | *General description of the data gathered from the test* | Yes the box which tracks the ball changes back to green when after the bounce is detected |  |
| Does the fps counter display the fps | *General description of the data gathered from the test* | Yes the fps counter works effectively and accurately |  |
| Is there screen tearing while the ball is being tracked | *General description of the data gathered from the test* | No there is no screen tearing while the ball is being tracked, the camera works perfectly fine and the image doesn’t seem “jagged” at any point during the tracking |  |
| Is there screen tearing while the ball is not being tracked | *General description of the data gathered from the test* | No there is no screen tearing while the ball is not being tracked, the camera works perfectly fine and the image doesn’t seem “jagged” at any point |  |

## 2.7 Further Data

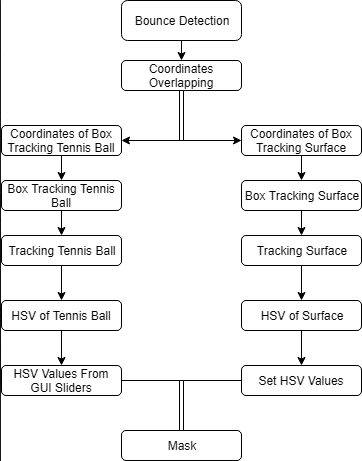
### 2.7.1 Bounce Detection



My bounce detection will work as a relatively simple concept. If the coordinates of the box, which tracks the tennis ball, overlap with the coordinates of the box, which tracks the surface, then we can assume that a bounce was detected. This is a simplified version of collision detection. Collision detection in real life is used to understand concepts such as billiard balls hitting each other, and how their direction changes. The physics of bouncing billiard balls are well understood, under the umbrella of rigid body motion and elastic collisions. An initial description of the situation would be given, with a very precise physical description of the billiard table and balls, as well as initial positions of all the balls.

For my solution, collision detection is significantly over simplified, as I do not need to consider as many factors, as my surface will not change in positioning, hence only one object, the tennis ball, moves towards the other, the surface. The surface does not change its location after it collides with the tennis ball.

### 2.7.2 Flowchart Breaking Down Bounce Detection

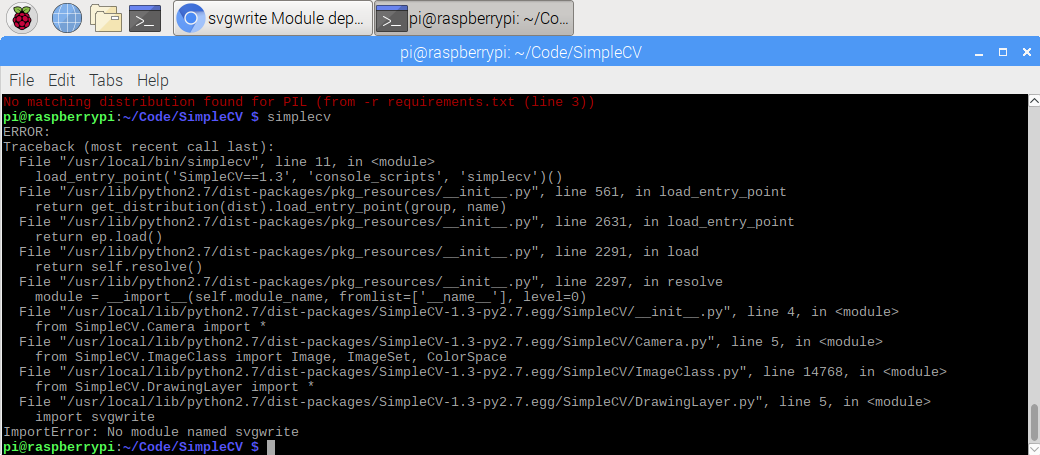
This flowchart summarises the key features necessary in order for the bounce detection to work effectively. Creating this visualisation of the problem decomposition allows us to understand which areas are most important for this part of the solution to work, as well as making testing much easier further down the line. This can allow us to find mistakes in the code much quicker and more efficiently, as we are taking one difficult problem and breaking it down into multiple much simpler and smaller problems. Clearly, in order for me to begin even working on bounce detection, I need to make sure I have a fully functioning HSV mask. From here, I can begin forming a solution piece-by-piece, starting with taking the HSV values in order to track the mask and the ball; as well as ending by taking coordinates of the boxes tracking the objects and detecting when an overlap occurs.

# 3 Developing

## 3.1 Prototypes

### 3.1.1 Prototype 1 – Displaying a Video Taken on the Picamera (testing the camera works)

#### 3.1.1.1 Structure

The aim of the first prototype is to display a photo taken, using the picamera onto my screen. In order for me to achieve this, I will need to set up both the raspberry pi and the picamera. As I will be programing in python, I am going to need to install either SimpleCV or OpenCV. I originally chose to install SimpleCV, and created a new environment on my raspberry pi, to work on my project. I installed many of the necessary required packages, however I quickly ran into an issue, as SimpleCV was written in Python 2.7, not python 3. This should not have been a major issue, as 2.7 and 3 share most of the similar aspects, but as most of the variables in the coding of SimpleCV were technically invalid if I wanted to work in python 3, I decided it would be easier to just switch to OpenCV and install the necessary packages for that. This would be a minor setback, as I would no longer be able to use a very good repository which I found for SimpleCV as shown here however, as OpenCV is much more popular as a library anyway, I didn’t believe this would be much of an issue. 

To no surprise, I found the perfect blog to use for inspiration. *(Adrian’s blog post on tracking a tennis ball using a raspberrypi using python and OpenCV –* [*https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/*](https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/)*)* Adrian Rosebrock runs one of the most famous online blogs for coding using python and raspberrypi. Some of his most popular project include handwriting recognition with HOG and Face detection from video and so on. A summary of this blog is that Adrian wants to create a program, which detects a tennis ball using a raspberrypi and a webcam, track the tennis ball, forming a red circle around it and draw the path of the tennis ball with a red line as it travels across the screen. This is the result.

However, as I am not using a webcam, but a picamera, I am first going to need to find a different guide to accessing and using the picamera. This website proved to be the best place to look, as I found one in no time. (Adrian’s guide to accessing a picamera with OpenCV and Python – <https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>)

#### 3.1.1.2 Examples of Code (annotated)

The first two steps were not necessary for me, as I already knew how to setup a picamera and how to enable the module. This was the only knowledge, which I possessed. When I ran this line of code, to test whether my camera was set up properly and was working, this was my result (photo on the right). I slightly adapted the line of code in order to test whether I could display a short video feed. I repeated the same line of code but instead of “.jpg” on the end, I entered “.mp4”, which resulted in an output of a short video feed, lasting approximately 5-8 seconds, before it stopped running and closed the window (as can be seen in the photo on the left). At this point in time, I knew that everything was working, as it should be. Then, in order to begin integrating my camera module with OpenCV, the guide stated I needed to install the picamera module in my “CV” (virtual environment). In order to access my already existing virtual environment, all I need to do is type these two lines of code. Then I began installing the “picamera[array]”. Adrian states that it is important to state the array specifically, as I need the “optional array sub-module to utilize OpenCV,” as this allows me to access the NumPy arrays from the picamera module.

#### 3.1.1.3 Validation

As can be seen above, I can display a photo and a short video feed using the raspberry pi and the picamera module. This was the aim of my first prototype, and I did not run into any errors along the way, once I chose to switch from using SimpleCV to OpenCV. This was unexpected as I was not sure if my knowledge of raspbian was good enough to download all the necessary packages, and install OpenCV correctly, allowing me to create a virtual environment. This doubt was mainly caused due to my initial failure to correctly research SimpleCV and understand that it does not work conjointly with Python 3.2, but rather with python 2.7 instead. As you can see in this article (<https://www.rs-online.com/designspark/object-tracking-using-computer-vision-and-raspberry-pi>), it is recommended to use SimpleCV rather than OpenCV, however it states that it shares similar concepts which shouldn’t make much of a difference. In contrast, I found that this was not a correct statement.

##### 3.1.1.3.1 Test Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Can a photo taken with the picamera module be displayed on the screen | The photo takes up the majority of the screen and allows approximately 2 seconds before the photo is taken, after which it is displayed for around 5-8 seconds in JPEG format | Yes the photo can be displayed on the screen | Yes, a photo taken on the picamera can be displayed on the screen. This photo is displayed for a period of approximately 5-8 seconds and is in JPEG format |
| Can a video taken with the picamera module be displayed on the screen | The video takes up the majority of the screen, it is reasonably good quality and is displayed on the screen for approximately 5-8 seconds in MP4 format | Yes the photo can be displayed on the screen | Yes, a video taken on the picamera can be displayed on the screen. This video is displayed for a period of approximately 5-8 seconds and is in MP4 format |
| Does the program crash when it is run | The program runs exactly as intended, however it does end abruptly after 5-8 seconds | No the program does not crash when it is run | No the program does not crash when it is run, it runs as expected |
| Does the program crash when any keys are pressed | When you try press any keys on the keyboard, the program does not crash and runs regularly | No the program does not crash when any keys are pressed | No the program does not crash when any keys are pressed, the program just runs normally |
| Does the program crash when the camera field of view is obstructed | When the camera field of view is obstructed with either a piece of paper, a wall or any other object, it runs exactly as expected and does not crash | Not the program does not crash when the field of view is obstructed | Not the program does not crash when the field of view is obstructed, as it is not tracking anything; just a black screen is displayed |

#### 3.1.1.4 Review and Changes Made

At this stage, the only change made was the library, which I was using. I decided to switch from SimpleCV to OpenCV, as SimpleCV was not compatible with Python 3.2, as well as OpenCV being the most widespread library for computer vision. This meant that in future prototypes, when I ran into errors, I could find solutions much quicker and easier than I would with SimpleCV. Other than that, I had no errors at this stage and I understood how everything was working.

### 3.1.2 Prototype 2 – Accessing a Single Image Taken on the Picamera Module

#### 3.1.2.1 Structure

The next stage is to access a single image of my picamera using python and OpenCV, as this will allow me to build the foundation of accessing a video feed of the picamera using python and OpenCV. This will require me to import some necessary libraries such as cv2 and picamera. Libraries contain built in modules, which provide access to system functionality. An example of this is file input and output, which would otherwise be inaccessible to me programing in python. Another key feature of libraries is access to important modules, written in python that provide standardised solutions for problems, which can often occur in everyday programing.

#### 3.1.2.2 Examples of Code (annotated)

These are the necessary packages I will need to import in order for me to access key features of system functionality. The first line shows import of the “PiRGBArray” from the “picamera” array, which produces a three dimensional RGB array from an RGB capture. This class is used to easily obtain a three dimensional numpy array, organised from the RGB capture. The next line shows import of the “PiCamera” package from the “picamera” library, which provides a python interface for the pi camera module, for python 2.7 and onwards. The penultimate module is “time”, which provides a variety of time related functions, such as calendar, dateandtime, etc. This is also commonly used to display fps, which I will later be doing in my program. The final package is access to the OpenCV library via “cv2”, which is necessary for using computer vision and python. This is an open source computer vision library, which is makes tracking an object via computer vision so much simpler.

Next, I need to initialise the PiCamera object (as can be seen on line 8) and grab a reference to the rawCapture component (as can be seen on line 9). The “rawCapture” object is essential as it allows direct access to the camera stream, as well as it avoids excessive compression to the JPEG format “which we would have to take and decode to OpenCV format anyway”(as stated by Adrian who highly recommends the use of the PiRGBArray whenever I should need to access the picamera).

The next line is a very common command, used to allow the camera sensor to warm up (in seconds). Adrian chose to use a tenth of a second, however I would prefer to take a whole second, as I do not want to strain the picamera too much in the long run.

This is where the photo is actually taken from the “rawCapture” object (as can be seen on line 15) and change the format into BGR, rather than RGB, as this is necessary in order to convert the image to HSV later on. OpenCV represents these images as numpy arrays in BGR order rather than RGB. This can be a subtle cog in the works, as if you make this mistake early on, it will not be detected and displayed via syntax highlighting, and can often be very difficult to spot. Because of this, we avoid some very confusing bugs towards the end.

Finally, these two lines display the code to our screen. These are very common commands, and are used in pretty much any OpenCV project you can find.

#### 3.1.2.3 Validation

As you can see, I can access a photo, taken on the picamera module. This was the aim of my second prototype, and I did not run into any errors while trying to achieve this. Everything went exactly as expected, as the tutorial created by Adrian (<https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>) explains everything perfectly. However, as this tutorial does not take into account that some people who use this may have almost no knowledge of Python and OpenCV, meaning that I had to go to other sources on the internet to completely understand every single piece of code used in the tutorial. This was very easy as Adrian did describe and comment on his code in a lot of detail.

##### 3.1.2.3.1 Test Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Can a photo taken with the picamera module be displayed on the screen | The camera allows 0.1 seconds before a photo is taken, the photo is clear and (similarly to 3.1.1) takes up the majority of the screen | Yes the photo can be displayed on the screen | Yes, a photo taken on the picamera can be displayed on the screen. This photo is displayed on the screen until the program is stopped from running |
| Can a video taken with the picamera module be displayed on the screen | When the program is run, it takes a photo within a 0.1 second time period, a video cannot be captured as this is not programmed yet | No the video cannot be displayed on the screen | No, a video taken on the picamera cannot be displayed on the screen as the code is only written to access a photo |
| Does the program crash when it is run | When the program is run, it runs without crashing if no other keys are pressed | No the program does not crash when it is run | No the program does not crash when it is run, it runs as expected |
| Does the program crash when any keys are pressed | The program does not crash when any keys are pressed | No the program does not crash when any keys are pressed | No the program does not crash when any keys are pressed, the program just runs normally |
| Does the program crash when the camera field of view is obstructed | When the camera field of view is obstructed with either a piece of paper, a wall or any other object, it runs exactly as expected and does not crash | Not the program does not crash when the field of view is obstructed | Not the program does not crash when the field of view is obstructed, as it is not tracking anything; just a black screen is displayed |

#### 3.1.2.4 Review and Changes Made

At this stage, I had no errors or failed tests. This meant that no changes were required. This was an important stage for building my understanding of the Picamera module, and how the Picamera library linked in with Python and OpenCV. This was also an important foundation for beginning to understand how the OpenCV library could be implemented in my code. In conclusion, I feel very satisfied with what I accomplished in this prototype.

### 3.1.3 Prototype 3 – Accessing the Video Stream of The Picamera Module

#### 3.1.3.1 Structure

The next stage is to access the video stream, captured on the picamera module using Python and OpenCV. This is simple addition of a few lines of code, on top of the already created code prior in prototype 2. The imports do not change at this stage, as all the necessary modules and libraries were already predefined, in prototype 2.

#### 3.1.3.2 Examples of Code (annotated)

As you can see, the only difference between prototype 2 and 3 in the initialisation of the camera, is defining the camera resolution, and the camera framerate. The resolution is also specified on line 11 (same resolution as specified in line 9) as this is necessary in order to avoid syntax errors when the program is run.

On line 17, the video stream is actually accessed, by calling the “capture\_continuous” method of the “camera” object. This method returns a “frame” from the video stream. The frame has an “array” property, which almost matches the “frame” in the NumPy array format. On line 18, the raw NumPy array is accessed, representing the image, then initialising the timestamps and occupied/unoccupied text.



Lines 21 and 22 are only edited very slightly from prototype 2, as I am no longer trying to access a single image, but now the camera feed, captured on the picamera module. The cv2.waitKey() returns a 32 Bit integer value. The key input is in ASCII, which is an 8-Bit integer value. The input in the brackets is the delay of n milliseconds before the image is displayed. So you only care about these eight bits and want all other bits to be 0. 0xFF is a hexadecimal constant, which is 11111111 in binary. Using waitkey(0) returns the video feed onto the screen with no delay, whereas using waitkey(1) returns a constant video feed with 1 millisecond of delay.



Line 26 is an important frame in order to avoid the program from crashing, as forces the program to clear the current frame, before you load the next one.



The two last lines of code allows the user to break the loop and exit the program when they press “q”.



#### 3.1.3.3 Validation

As you can clearly see, I can now access the video feed captured on the picamera module. This was the aim of my third prototype and heavily relied upon my second prototype working flawlessly. I did not run into any errors along the way, as Adrian’s tutorial (<https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>) contains so much information and detailed explanation on how every piece of code works. In this prototype, I did not have to do much research on the code from the tutorial, as the majority of it was either the same as in prototype 2, or similar developments of this code, which were mainly explained very well by Adrian in his tutorial.

##### 3.1.3.3.1 Test Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Can a photo taken with the picamera module be displayed on the screen | Only a video can be displayed | No, the photo cannot be displayed on the screen | No, a photo taken on the picamera cannot be displayed on the screen, as the code is not written to display a photo but to display the camera feed |
| Can a video taken with the picamera module be displayed on the screen | C:\Users\yaro.shiroki\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_6716.jpg | Yes the video cannot be displayed on the screen | Yes, a video taken on the picamera can be displayed on the screen. The video feed is displayed on the screen until either the program is stopped from running, or the “q” key is pressed. |
| Does the program crash if the Picamera is not connected | Input: program is run without picamera module plugged in  Output: “Error: no camera module detected” | Yes, the program crashes if the picamera is not connected and I attempt to run the program | Yes, the program cannot run if the picamera is not connected and a message is displayed that no camera is connected |
| Does the program crash when it is run | When the program is run, it runs without crashing if no other keys are pressed | No the program does not crash when it is run | No the program does not crash when it is run, it runs as expected |
| Does the program crash when any keys are pressed | The program only crashes when the “q” key is pressed as this is programmed to do so. Otherwise, the program does not crash when any other keys are pressed | Partially. The program does not crash when any keys are pressed apart from the “q” key | Partially. No the program does not crash when any keys are pressed, the program just runs normally, unless the “q” key is pressed. When the “q” key is pressed, the program stops running |
| Does the program crash when the camera field of view is obstructed | When the camera field of view is obstructed with either a piece of paper, a wall or any other object, it runs exactly as expected and does not crash | Not the program does not crash when the field of view is obstructed | Not the program does not crash when the field of view is obstructed, as it is not tracking anything; just a black screen is displayed |

#### 3.1.3.4 Review and Changes Made

At this stage, there were no flaws with the code. This meant that no changes were required at this stage. This stage was important for my understanding of accessing the video feed from the Picamera module. This prototype linked clearly with the previous one. This allows me to understand the difference between accessing an image and a video from the Picamera module. This is key for implementing the Picamera in the next prototype.

### 3.1.4 Prototype 4 – Tracking a Tennis Ball with a Raspberry Pi and Picamera Module Using Python and OpenCV

#### 3.1.4.1 Structure

This section is going to be split up into five separate stages. During this stage of the project, I am going to be analysing Adrian’s tutorial on how to track a tennis ball using a raspberry pi and a webcam (<https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/>) and changing the necessary pieces to work with a picamera module. This will require me to use my previous prototype, and implementing it to work with the code shown by Adrian. In order to do this, I must gain knowledge of the necessary packages to import, construction of an argument parse, understanding of HSV upper and lower boundaries; how they can be defined, converting an image from BGR to HSV (after blurring and resizing the frame), constructing a HSV mask, finding the contours in the mask (as well as the position of the contours), drawing a box around the largest contours (rather than a circle which Adrian does) and stop drawing the box when no contours are detected on screen.

#### 3.1.4.2 Examples of Code (annotated)

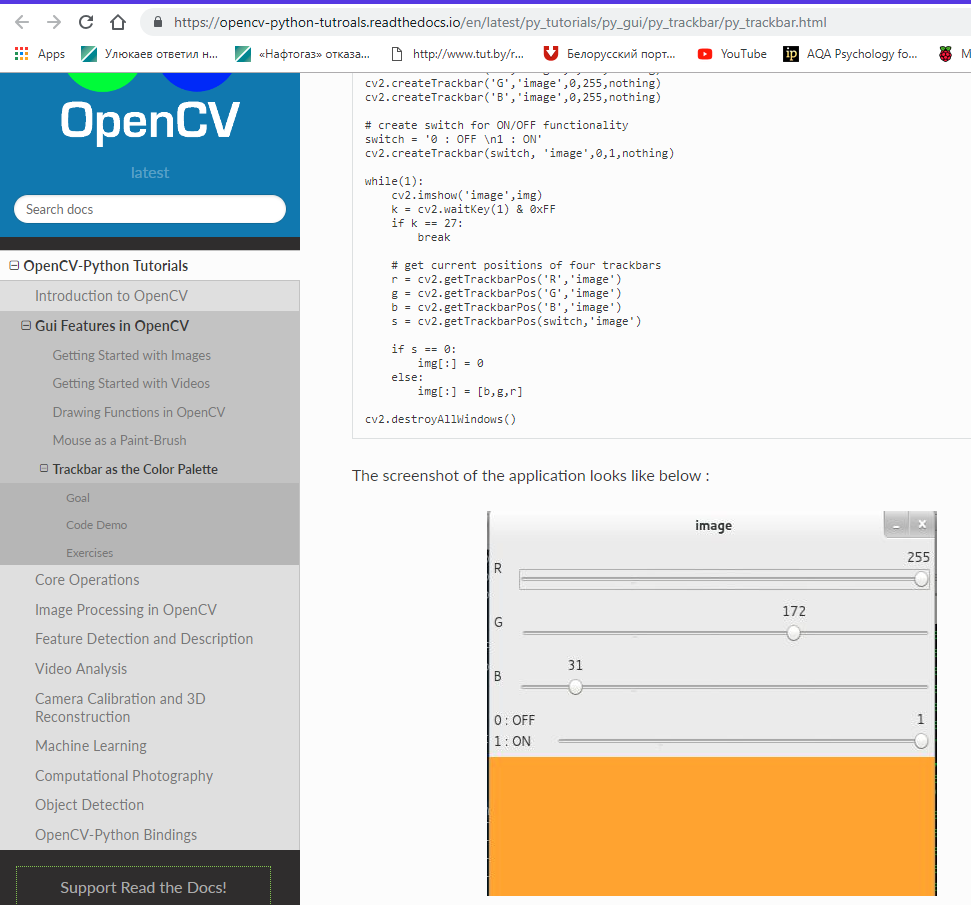
##### 3.1.4.2.1 Importing Necessary Packages and constructing the Argument Parse



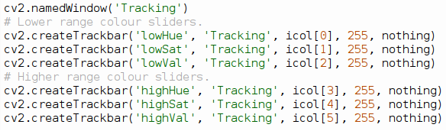
Here we handle the necessary packages for using a webcam to track the tennis ball, as well as importing the previous packages, necessary for using the picamera module. The packages, which we did not see in prototype 2 or 3, are “deque”, “imutils”, “imutils.video import VideoStream”, “numpy” and “argparse”. The “deque” data structure acts like a list, with extremely fast appends and pops, to maintain of the past (x,y) coordinates of the tennis ball. The “imutils” package is a collection of “convenience functions” such as resizing, making life a lot easier. This is a common package used and most people recommend installing it on the raspberry pi. The “argparse” package is a command line parsing module in the Python standard library. The “argparse” module makes it easy to write ergonomic command line interfaces. The program defines what arguments it requires, and “argparse” will figure out how to parse those out of “sys.argv”. The “argparse” module also automatically generates help and usage messages and issues errors when users give the program invalid arguments. The “imutils.video” “VideoStream” package is part of the imutils library, allowing me to display the video stream onto my screen.

The next section of code is constructing the argument parse and parsing the command line arguments. The first switch “—video” is the path to my video file. If the switch is supplied, then OpenCV will grab a pointer to the video file and read frames from it. Otherwise, OpenCV will attempt to access the webcam. This can allow me to have a potential second camera (a cheap webcam) to act as a backup camera, as if the picamera module fails to supply a video file, OpenCV will simply take the feed of the webcam. The second argument is “—buffer”, which maximises the size of my “deque”. This maintains a list of the (x,y) coordinates of the tennis ball which I am tracking, as mentioned previously.

##### 3.1.4.2.2 Declaring Upper and Lower Boundaries of HSV Values and Setting Up the Camera

In the next three lines, you can see the HSV upper and lower boundaries being defined to the “green” tennis ball. The values which Adrian chose will not match the HSV values of my tennis ball. However, this isn’t a major issue as I will set the HSV upper and lower boundaries using track-bars for each of these, allowing me to also display my HSV mask in a separate frame on my screen. Line 23 initialises the “deque” of “pts” using the supplied maximum buffer size (which defaults to 64).When I researched how to create GUI track bars online, I found a website which explained exactly how I would do this (website in reference – <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_trackbar/py_trackbar.html>). This contained information on how to add track bars to OpenCV windows, by using functions such as “cv2.getTrackbarPos()” and “cv2.createTrackbar()”. The first example they use is for BGR colours, as can be seen in the next image.

This allowed me to alter the code presented by Adrian to fit my situation more effectively. These were the new lines of code



This then led me a position where I had to initialise the camera, in order to later obtain the HSV values from the GUI sliders created here.

As you can see here, from “deque” we can obtain access to the video stream pointer “vs”. If a “—video” switch is not supplied, then by default, the program attempts to grab reference to the picamera (or in Adrian’s case, the webcam). This is done by accessing the first available video input. I will need to experiment with this later on however, if the “src=0” needs to be changed to “src=1”, depending on what the picamera is identified as within the raspberry pi module itself. On lines 27 and 28, the “imutils.video”, “ VideoStream” threaded class is used for efficiency. Otherwise, if a video file is supplied, then it is opened for reading and the program grabs a reference pointer on lines 31 and 32, using the built in “cv2.videocapture” command.

Then, we actually grab the HSV values from the GUI sliders, as shown in the next few lines of code,

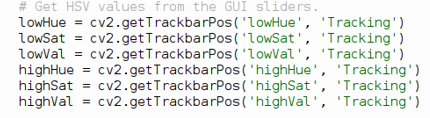
lowHue = cv2.getTrackbarPos('lowHue', ‘HSV’)  
 lowSat = cv2.getTrackbarPos(‘lowSaturation’, ' HSV’)  
 lowVal = cv2.getTrackbarPos('lowValue', ' HSV’)  
 highHue = cv2.getTrackbarPos('highHue', ' HSV’)  
 highSat = cv2.getTrackbarPos('highSaturation’, ' HSV’)  
 highVal = cv2.getTrackbarPos('highValue', ' HSV’)

For the “cv2.getTrackbarPos” function, the first argument is the trackbar name and the second argument, is the window name to which the function is attached. A third argument is a default value, fourth is the maximum value and fifth one is the callback function which is executed every time trackbar value changes. The callback function always has a default argument, which is the trackbar position. This applies in the exact same way to the “cv2.createTrackbar” function, seen prior. These lines of code will be implemented in the loop of 3.1.4.2.3 as I want these values to be constantly updated as the HSV values change on the trackbar.

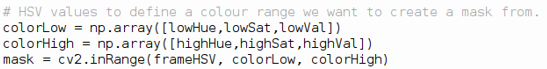
Finally, we allow the program 2 seconds to warmup and get ready, as seen in previous prototypes.

##### 3.1.4.2.3 Loop to Get HSV Values and Create a Mask

For the first part of this section, I am going to go back and declare the frame width and frame height, after the parsing of arguments, as this will be the same size for each window showing the normal video stream, the HSV stream and the stream tracking the tennis ball.

In this section, I created a loop, used the “timeCheck” function (to later display the fps count) and used the previous “cv2.getTrackbarPos” function to keep getting the HSV values from the trackbars.

I found out how to display the fps of the camera, using the “timeCheck” function on stack overflow (page used in reference – <https://stackoverflow.com/questions/43761004/fps-how-to-divide-count-by-time-function-to-determine-fps>), which showed a very simple method of displaying fps. This was a very useful response, as it was easy to follow and to understand how to implement this in my code.

After the fps counter was set up, I began handling of the frame. The current frame is grabbed by making a call to the read method of the picamera pointer, which returns a “2-tuple”. A tuple is a finite ordered list of elements. This means that a 2-tuple is an ordered list (often referred to as sequence) of two elements. A 0-tuple is an empty sequence. The first entry in the tuple, “grabbed” is a Boolean, which indicates whether a frame was successfully read, or not. The “frame” is the video frame itself. The next line of code, handles “VideoSream” vs “VideoCapture”. After this, we define that if a frame was not successfully read, then the program can comprehend that the end of the video is reached and the while loop break.

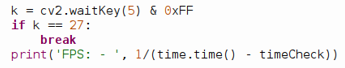
After this, the frame is converted to HSV colour model. Then, the HSV values, which are set with the sliders, actually define the range, which we want in order to create a mask.

In the next section of code, we display the first mask in a separate frame. Then we find the contours using the “cv2.findContours” function to allow the mask to work. Then, the contour sizes are defined, in order to then allow us to track the biggest contour, which, as you can see is defined in the next line of code.

Later, I added this line of code when creating the mask, as this blurs the frame, which reduces high frequency noise and allows the program to focus on the structural objects inside the frame (in this case the tennis ball).

##### 3.1.4.2.4 Tracking the Largest Contour with a Box around It and Breaking the Loop

In this section, we are using the “biggest\_contour” previously created to draw a box around our largest contour. This box grows and shrinks based on the size of the size of the “biggest\_contour” as it creates a square/rectangle based on the biggest and smallest x and y coordinates of the contours. The “cv2.drawContours” function works by drawing all of the contours, which found using the “cv2.findContours” function. The “cv2.drawContours” function can be manipulated to draw all contours, draw the largest contours or to draw a shape around the biggest contours. The rectangle is drawn in green as the BGR values are set to “(0, 255, 0)” however; this can easily be changed to red by changing these values to “(0, 0, 255)”. I am planning on changing these values if a bounce is detected (bounce detection 3.1.5) and reverting to green after the ball has left the ground again.

This line of code is then used to display the final image onto a separate frame. You may have noticed that three separate frames are displayed in this project. This is done so that we can have our “normal” tab (the tab which just displays out camera feed and nothing else), our HSV mask (our video feed with the HSV colour mask applied to it, so that we can isolate the colour of the tennis ball, which we require) and our tracking tab (the tab which displays the object which we are tracking, isolated with a green box around it).

This is the penultimate part of this prototype, as here we break the loop. “cv2.waitKey(5) is used to delay the program by 5 milliseconds before the camera feed is displayed. I used “k== 27” in order to set escape to break the loop. Hence, when escape is pressed, the program stops running, rather than “q” as was used in previous prototypes. The final line is used as previously found, in order to display the fps count.

##### 3.1.4.2.5 Closing All Windows and Releasing Camera Resources

Finally, theses 2 lines are used at the end of the prototype. “cv2.destroyAllWindows()” is used to close all the windows which were created when the program is run. The “vidCapture.release()” is calling “cv2.videoCapture()” which we set it to be. This is releases the hardware and software resources of the camera module. The “cv2.videoCapture()” is set to “vidCapture()” as this is done in order to avoid common errors such as “Device or resource busy.”

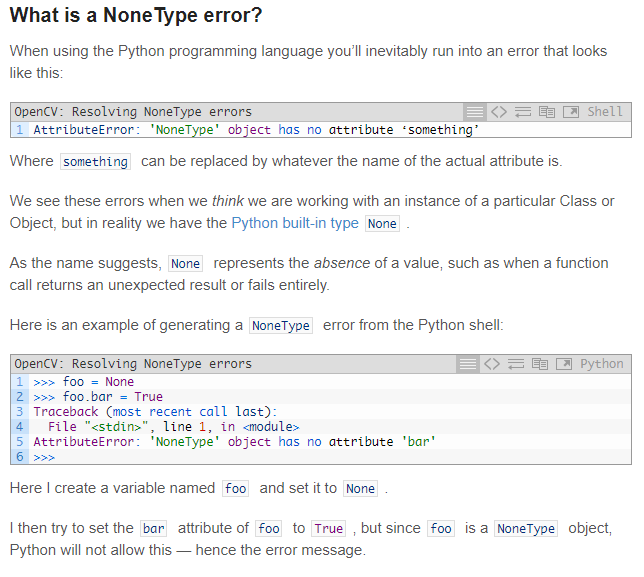
#### 3.1.4.3 Validation

In this section, I have successfully shown that my program can track an object with a green box around it, based on the HSV values and the largest contours. In this prototype, I have shown the ability for the user to set the HSV colours using GUI sliders and be able to see the objects HSV values via the values on the sliders and the mask displayed in a separate window. This was the aim of my fourth prototype and I believe I have accomplished it successfully. I ran into some errors along the way, and after I did my first series of tests, I went and updated some of my code, made alterations and then tested my program once more. I repeated this process until I no longer received any errors, which were not intended in this prototype.

##### 3.1.4.3.1 Test Table (for first iteration of prototype 4)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run without crashing | Input: program is run  Output: crash | Yes, the program runs without crashing as intended. The program should only crash if HSV (lower) values are higher than the HSV upper values | No, the program does not run without crashing, it crashes straight away  (END OF TESTING HERE) |
| Can the video feed be displayed |  |  |  |
| Does the program end when the “escape” key is pressed |  |  |  |
| Does the program end when the “q” key is pressed |  |  |  |
| Does the program crash when any other keys are pressed |  |  |  |
| Does the program crash when the camera’s field of view is obstructed |  |  |  |
| Do 3 separate windows open |  |  |  |
| Do the GUI sliders work effectively |  |  |  |
| Does the HSV mask work as intended |  |  |  |
| Is there a window which displays tracking of the tennis ball with a green box around it |  |  |  |

##### 3.1.4.3.2 Review and Changes Made (for the first iteration of prototype 4)

This is the error “AttributeError: 'NoneType' object has no attribute ‘picamera’ ”, which came up when I first ran my program. In order to find a solution, I googled what this error meant. This was my result. (website in reference – “<https://www.pyimagesearch.com/2016/12/26/opencv-resolving-nonetype-errors/>”)

This meant that I had sections of my code, which were not linked and did not run as intended. I began reworking my solution and attempted to create a solution for my errors. This was caused by me not declaring the picamera as “vs” (or video stream). This is what I corrected on line 28. “vs = VideoStream(usePicamera=True).start()”

This allowed me to link the Picamera to my code, rather than using a webcam.

##### 3.1.4.3.3 Test Table (for the second iteration of prototype 4)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run without crashing | Input: program is run  Output: “Error: No picamera module detected” | Yes, the program runs without crashing as intended. The program should only crash if HSV (lower) values are higher than the HSV upper values | No, the program does not run without crashing as the picamera module is not linked properly |
| Can the video feed be displayed | Input: program is run  Output: “Error: No picamera module detected” | Yes, the video feed can be displayed. For this prototype it does not have to run smoothly as no optimisation has been made yet | No, the video feed cannot be displayed as the pi camera module does not work with the code we have provided (END OF TESTING HERE) |
| Does the program end when the “escape” key is pressed |  |  |  |
| Does the program end when the “q” key is pressed |  |  |  |
| Does the program crash when any other keys are pressed |  |  |  |
| Does the program crash when the camera’s field of view is obstructed |  |  |  |
| Do 3 separate windows open |  |  |  |
| Do the GUI sliders work effectively |  |  |  |
| Does the HSV mask work as intended |  |  |  |
| Is there a window which displays tracking of the tennis ball with a green box around it |  |  |  |

##### 3.1.4.3.4 Review and Changes Made (for the second iteration of prototype 4)

As you can see this is the error, which was displayed. I first attempted to find a solution to this error online. This was one of the most reliable results, which I came across (website in reference – “<https://www.raspberrypi.org/forums/viewtopic.php?t=174375>”). I ran this line of code “vcgencmd get\_camera” and my result was exactly the same as shown above. I tried installing a fresh copy of Raspbian but this did not help me at all. I unattached the ribbon cable, checked whether something looked broken, attached it back to the Raspberrypi and again nothing changed. I tried setting “gpu\_mem” to 128, however it was already set to this. However, none of my attempts to find a solution were effective as I kept receiving this reoccurring error. I decided to go back to my prototype 3, which I saved as a separate document. I found that this error had occurred again, even though this was a fully functioning program when I last left it. From this, I deduced that this might not just be a software issue, but possibly a hardware one. The guide then suggested that I have either a faulty ribbon cable or a fault on the camera board. This could mean either that the Raspberrypi is faulty, or that the Picamera is faulty. I figured that as the Raspberrypi was functioning normally in every other regard, that this is most likely an issue with the Picamera (if this is actually a hardware error). I tried disconnecting the Picamera from the Raspberrypi, and reconnecting it. This did not solve my issue. I decided to take my Picamera and Raspberrypi to a local computer hardware store, which I knew had people employed who would be able to find whether there is a hardware issue as a favour for me. The result was that one of the wires had a faulty ribbon cable, which had unfortunately stopped functioning over time due to an already existing bend. This was a hard blow for my moral and finances, as I knew I would either have to buy a new Picamera, or rewrite a significant part of my code to work with the Logitech webcam, which I had at home. I had to finance the Picamera myself previously, and buying another one was simply not an option for me, as I simply did not possess the necessary funds for this. I cut my losses and decided to begin rewriting my code to work with a webcam.



First, I took out the unnecessary packages, which were used for the Picamera module. This would mean that I would get a large amount of syntax errors, allowing me to utilise syntax highlighting to my advantage. After removing all of the sections, which previously used the picamera, I began replacing them with code, which would work for a webcam instead. This was not very difficult as I used a significant amount of code already presented by Adrian. Once this was complete, I decided to test the program once more.

##### 3.1.4.3.5 Test Table (for the third iteration of prototype 4)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run without crashing | Input: program is run  Output: program runs | Yes, the program runs without crashing as intended. The program should only crash if HSV (lower) values are higher than the HSV upper values | Yes, the program runs without crashing as intended |
| Can the video feed be displayed |  | Yes, the video feed can be displayed. For this prototype it does not have to run smoothly as no optimisation has been made yet | Yes, a video feed can be displayed |
| Does the program end when the “escape” key is pressed | Input: “escape” key is pressed  Output: program stops running and all of the windows are closed | Yes, the program stops running when the “escape” key is pressed as it is supposed to be a way to stop the program from running with a single key press | Yes, the program stops running when the “escape” key is pressed |
| Does the program end when the “q” key is pressed | Input: “q” key is pressed  Output: no changes in program | No, the program does not stop running when the “q” key is pressed as it is not intended to do so | No, the program does not stop running when the “q” key is pressed |
| Does the program crash when any other keys are pressed | Input: a selection of random keys are pressed (other than “q” and “escape”) in a random order  Output: no changes in program | No the program does not crash when any other keys are pressed as it is not intended to do so | No, the program does not crash when any other keys are pressed |
| Does the program crash when the camera’s field of view is obstructed |  | No the program does not crash when the camera’s field of view is obstructed | Partly. The program does not crash if the camera’s field of view is obstructed. However, if the program is already tracking an object and then the camera’s field of view is obstructed, then the program may crash |
| Do 3 separate windows open |  | Yes 3 separate windows open as intended | Yes, 3 separate windows open as intended |
| Do the GUI sliders work effectively |  | Yes the GUI sliders work effectively and set the HSV values for the mask as intended | Partly. The GUI sliders work effectively at setting the HSV upper and lower bounds for the mask however, if the lower HSV values are set to be greater than the higher, then the program crashes |
| Does the HSV mask work as intended |  | Yes, the HSV mask works as intended and can allow us to find the HSV upper and lower values for a specific object, in this case the tennis ball | Partly. The HSV mask works as intended and can allow us to find the HSV values of a tennis ball. However, if the HSV values are set too high or too low, not detecting anything in the mask, the program will crash as it is trying to track something but it cannot track anything |
| Does the program track the tennis ball effectively without crashing |  | Yes, the program tracks the tennis ball effectively without crashing as it is intended to do so | Yes, the program can track an object effectively as intended |
| Is there a window which displays tracking of the tennis ball with a green box around it |  | Yes, there is a window, which displays tracking of the tennis ball with a green box around it. This box does not change colour in this prototype, as it is not intended on doing so | Yes, there is a separate window, which displays tracking of the tennis ball with a green box around it. The box does not change colour |

##### 3.1.4.3.6 Final Review and Changes Made in Prototype 4

This was an important prototype both in terms of my understanding of Python, while also being a key section for my solution. As the tracking of the tennis ball is the key function of the program, this section had to work. It took a lot of effort and testing in order to find which parts required changing. This was also necessary, in order for prototype 5 to work, as I needed to understand how to track an object with the Raspberrypi in order to begin working on bounce detection (2.7.1 and 2.7.2).

### 3.1.5 Prototype 5 – Bounce Detection

#### 3.1.5.1 Structure

This section will be split into four separate stages:   
Finding the HSV values of the surface and storing them. Creating a separate HSV mask for the surface. Tracking the surface with a box around it and finding the coordinates of the box (displaying this in the tracking window). Creating a function for when the box tracking the tennis ball overlaps with the box tracking the surface and finally, implementing this function to change the colour of the box tracking the tennis ball from green to red (and reverting back to green when the boxes no longer overlap). For this section, I will be using the code, which I previously used in prototype 4 (3.1.4) in order to create the mask for the surface. I will be using this previous prototype in order to find the HSV upper and lower bounds of my surface and then storing them separately from the HSV values collected from the GUI sliders for the tennis ball. I will then use the code from prototype 4 in order to track the surface as well as finding the coordinates of the box tracking the surface. Then, I will find a way in order to create a function, which will be responsible for changing the colour of the box tracking the tennis ball to red when the coordinates of the box tracking the tennis ball overlaps with the box tracking the surface. I will first keep the box tracking the surface of a blue colour, changing this to invisible once I finalise all of the logic for the bounce detection function.

Testing and validation for this section should be relatively simple, as I have broken this section down very well in my design stage (2.7.2 Flowchart Breaking Down Bounce Detection). The flowchart I used will allow me to find exactly where any errors occur relatively quickly. I believe this flowchart has helped me get to the position, which I am in now, as I have all of the foundations already existing in order to create a successful collision detection function.

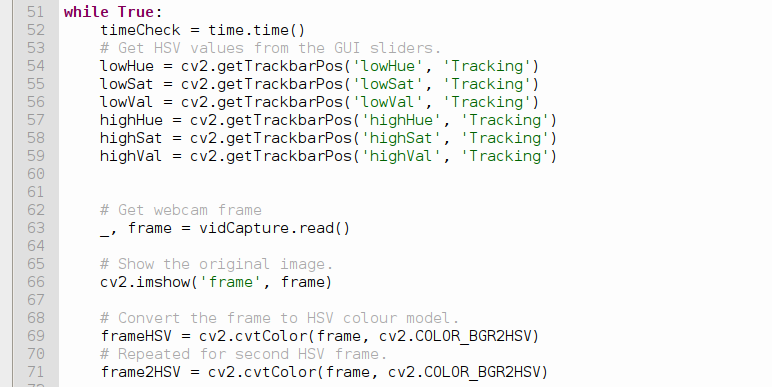
#### 3.1.5.2 Examples of Code (Annotated)

##### 3.1.5.2.1 HSV Values of Surface

First, I had to find the HSV values of the surface, which I was using. I decided to use a blue folder, which was available to me. This was convenient as the folder is of a light blue colour, so it will not clash with the HSV values of the tennis ball.

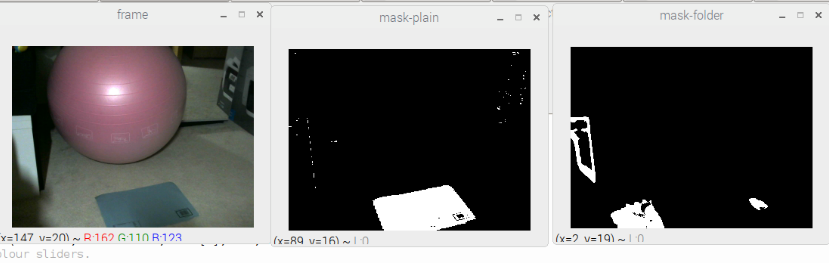
I found the ideal HSV upper and lower bounds of the surface. However, when the environment changed, I had to find new HSV values for the folder. This is a relatively simple task and takes little to no time.

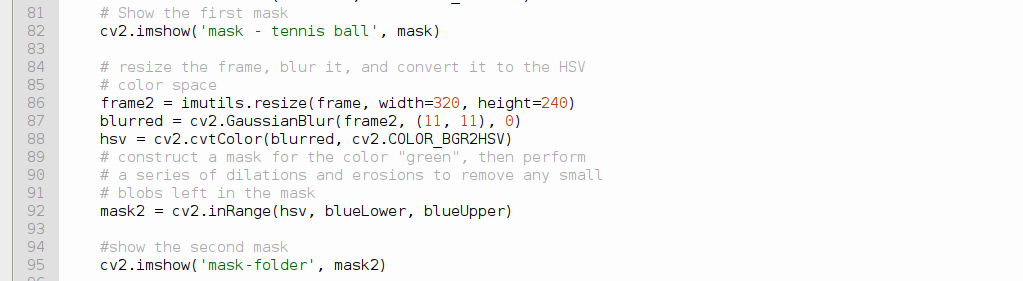
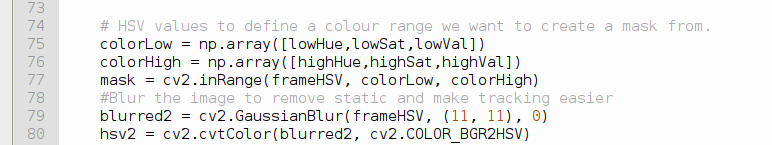
I then stored these values in separate arrays called “blueLower” and “blueUpper”. These values can be changed any time in the code quickly and easily. This allows me to begin building the second mask in the next section of bounce detection. The difference with using an array rather than GUI sliders to store HSV values is the implementation when I build the second mask.

Next, we have to convert our frame into HSV as we have done previously in prototype 4. This has to be a separate function to the prior one, as we want to have two separate masks, one for tracking the contours of the surface and the other for tracking contours of the tennis ball. This will allow us to track these objects separately. In order to test that everything is working correctly, I will have an extra window for the second HSV mask and an extra window for tracking the surface. These windows will be removed when the prototype is complete. Here, you can see I convert the normal frame into HSV in the “frame2HSV”.

Both conversions for the first mask and the second mask are done side by side in order to keep the project well structured. This section is relatively short as everything we use here is using already existing code. This shows a clear understanding of every section in prior prototypes.

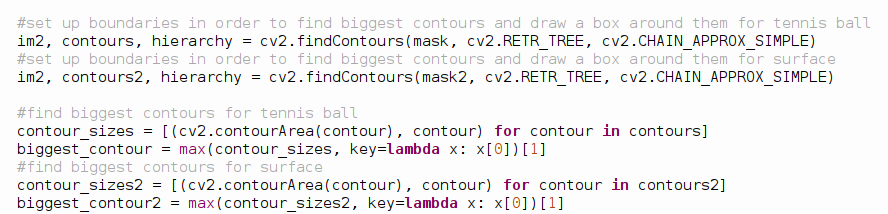
##### 3.1.5.2.2 HSV Mask for Surface

For this section, I used the previous code for creating a HSV mask and reused this. You can see the “blurred2” function and “hsv2” function are used for blurring the frame and converting the frame from BGR to HSV. Then I resized the frame and created the second mask “mask2”. Then this mask is displayed in a separate window “mask-folder”, which will be removed once the prototype is complete. As you can see, as we change the HSV values in the array, the second mask changes. This mask is more blurred and there is less static, meaning that it is easier to track the surface then.

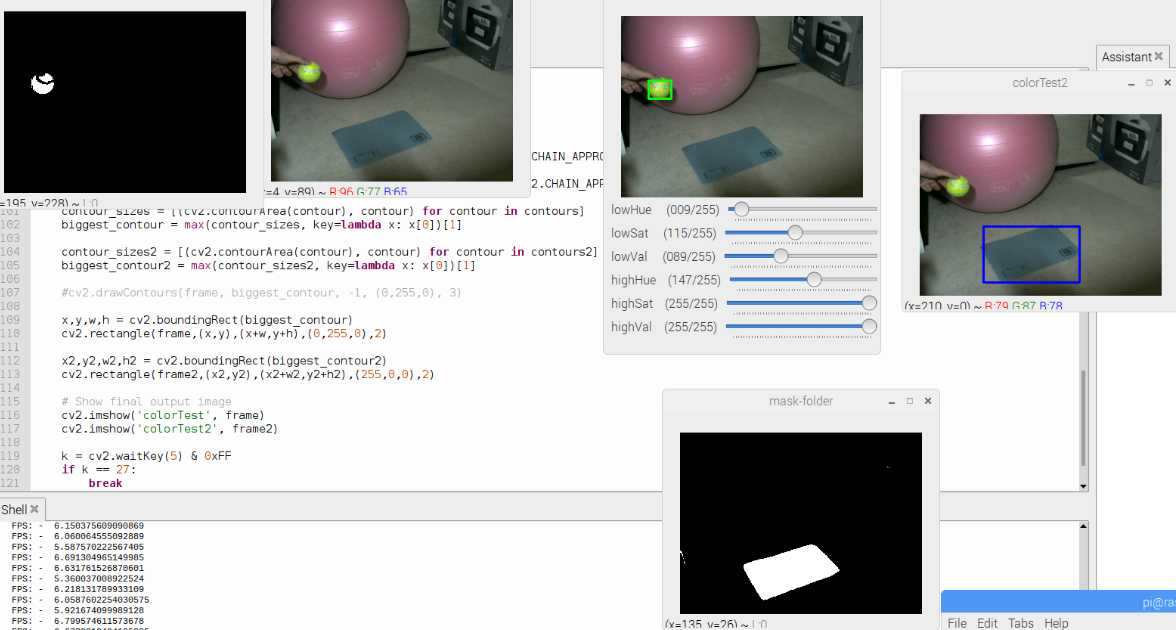


As you can see, the “mask-folder” displays the mask of the folder. As the HSV upper and lower bounds become more accurate, the tracking will become more accurate in the later stages of the prototype.

##### 3.1.5.2.3 Tracking the Surface

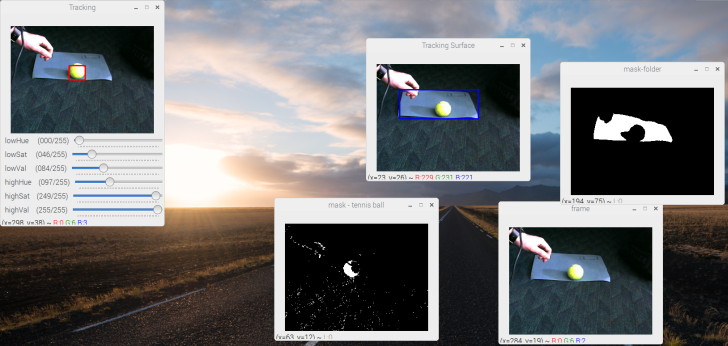
Similarly, we begin this section by reusing previous code, which we used. As you can see via my comments, the tracking for the surface and the tracking for the tennis ball is very similar. The bounding rectangle for the surface is blue (BGR 255,0,0), which contrasts the green (BGR 0,255,0) of the bounding rectangle around the tennis ball. This will allow us to clearly distinguish which sections are working efficiently. We then display both the tracking for the tennis ball and for the surface in separate windows. Finally, we print the fps at the end of the loop and when the escape key is pressed, the program stops running and all windows are destroyed.

This is what the masks look like when the HSV upper and lower bounds are input correctly. Here we can also see the two bounding rectangles, effectively tracking the surface (with a blue rectangle) and the tennis ball (with a green rectangle). As you can see, the fps is very low, as we have multiple windows open, two of which are responsible for HSV masks and another two responsible for tracking objects using said HSV masks. This puts a lot of strain on the Raspberrypi and for this reason, at the end of this prototype, only two windows will open when the software is run. One of them will be the “tracking” window and the other one will be of the HSV mask for the tennis ball. This will take a lot of the strain away from the Raspberrypi and significantly increasing the fps.

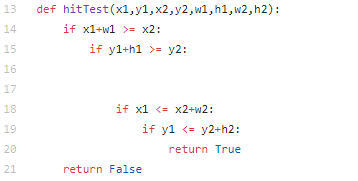


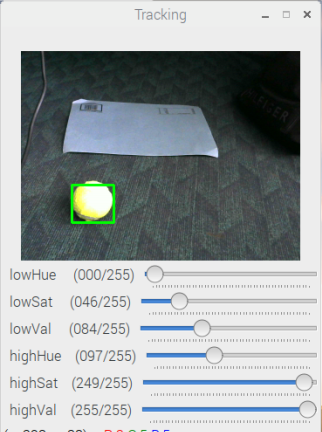
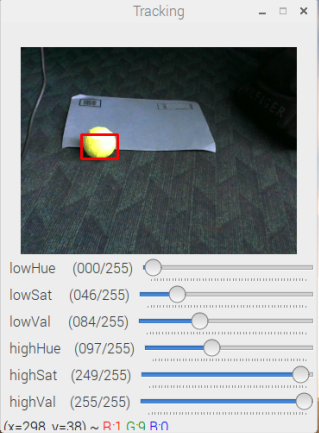
##### 3.1.5.2.4 Overlapping Coordinates between Surface and Tennis Ball and Changing Colour of Box When Bounce Detected

This is where the difficulty in this prototype lies. Here we need to create a function, which is responsible for returning a message, when the two bounding rectangles overlap. First, I began by using my “x,y,w,h,x2,y2,w2,h2” variables, in order to decide the coordinates of the top left corner and bottom right corner of each bounding rectangle. I began using simple maths to deduct a solution for this problem. This was relatively simple, however when implemented, I realised that this would not be a very effective method of displaying collision detection, as a “hit” would only be detected, if the bounding rectangle for the tennis ball was fully inside the bounding rectangle for the surface. I decided to first implement this function and test it.

 This is what the “if statement” looks like. This statement will not need to change, even if the “hitTest” function itself is changed after the testing. This is what this is what this prototype looked like in the first iteration. As you can see, the box tracking the tennis ball does change colour to red when inside the blue bounding rectangle however, only when it was completely inside it. After completing testing, I decided to go back and rewrite this function again.

##### 3.1.5.2.5 Making the Bounce Detection More Accurate

In order to make this function more effective, I had to come up with another mathematical solution to the problem. I had to come up with a solution, which only required one side of either bounding rectangle to overlap. I came up with a solution, which took into account every single side of each rectangle. I rewrote the function to look like this. The new solution was much more effective in theory so I decided to test this again, in order to see whether I made any logical mistakes.

As you can see, this solution was much more accurate. The bounding rectangle now changed colour from green to blue, even if the tennis ball was only slightly within the bounding rectangle of the surface. This was a huge success, as my product was now almost complete. All what was required now was some optimisation and the solution would be complete.

#### 3.1.5.3 Validation

##### 3.1.5.3.1 Test Table (for first iteration of bounce detection)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run without crashing | Program runs successfully without crashing | Yes, the program runs without crashing as intended. The program should only crash if HSV (lower) values are higher than the HSV upper values. | Yes, the program runs without crashing as intended. |
| Can the video feed be displayed | Same as in previous prototypes | Yes, the video feed can be displayed. For this prototype it does not have to run smoothly as no optimisation has been made yet | Yes, a video feed can be displayed |
| Does the program end when the “escape” key is pressed | Input: “escape” key pressed  Output: program ends and all windows are closed | Yes, the program stops running when the “escape” key is pressed as it is supposed to be a way to stop the program from running with a single key press | Yes, the program stops running when the “escape” key is pressed |
| Does the program end when the “q” key is pressed | Input: “q” key is pressed  Output: no change in program | No, the program does not stop running when the “q” key is pressed as it is not intended to do so | No, the program does not stop running when the “q” key is pressed |
| Does the program crash when any other keys are pressed | Input: selection of random keys (other than “escape” and “q”) are pressed in a random order  Output: no change in program | No the program does not crash when any other keys are pressed as it is not intended to do so | No, the program does not crash when any other keys are pressed |
| Does the program crash when the camera’s field of view is obstructed |  | No the program does not crash when the camera’s field of view is obstructed | Partly. The program does not crash if the camera’s field of view is obstructed. However, if the program is already tracking an object and then the camera’s field of view is obstructed, then the program may crash |
| Do 5 separate windows open |  | Yes 5 separate windows open as intended | Yes, 5 separate windows open as intended |
| Do the GUI sliders work effectively |  | Yes the GUI sliders work effectively and set the HSV values for the mask as intended | Partly. The GUI sliders work effectively at setting the HSV upper and lower bounds for the mask however, if the lower HSV values are set to be greater than the higher, then the program crashes |
| Does the HSV mask work as intended (for tennis ball and surface) |  | Yes, the HSV mask works as intended and can allow us to find the HSV upper and lower values for a specific object, in this case the tennis ball | Partly. The HSV mask works as intended and can allow us to find the HSV values of a tennis ball. However, if the HSV values are set too high or too low, not detecting anything in the mask, the program will crash as it is trying to track something but it cannot track anything |
| Does the program track the tennis ball effectively without crashing |  | Yes, the program tracks the tennis ball effectively without crashing as it is intended to do so | Yes, the program can track an object effectively as intended |
| Is there a window which displays tracking of the tennis ball with a green box around it |  | Yes, there is a window, which displays tracking of the tennis ball with a green box around it. This box changes colour when a function returns a value | Yes, there is a separate window, which displays tracking of the tennis ball with a green box around it. The box changes colour to red when the “hitTest” function returns True |
| Is there a window which displays tracking of the surface with a blue box around it |  | Yes, there is a window, which displays tracking of the surface with a blue box around it. This box is not visible in the tracking window | Yes, there is a window, which displays tracking of the surface with a blue box around it. This box is not visible in the tracking window |
| Does the bounce detection work accurately |  | Yes, the bounce detection works accurately and effectively as intended | Party, the bounce detection works however, only if the bounding rectangle which tracks the tennis ball is fully inside of the bounding rectangle which tracks the surface |
| Does the fps remain drop lower than 10 | https://raw.githubusercontent.com/FulfordComputing/AlevelYaros/master/2019-04-10-221408_1824x984_scrot.png?token=AmuAcB1Dueu_ykYIwPM4BveSfLwdIqWfks5cuOW2wA%3D%3D | Yes, the fps drops lower than 10, as there are more than 2 windows open in this iteration | Yes, the fps drops lower than 10, as there are 5 windows open, two of which are responsible for being HSV mask and two more responsible for tracking an object. The fps remains around 6(+/-)2 |

##### 3.1.5.3.2 Review and Changes Made (for the first iteration)

For the first iteration, the only change made, was the function responsible for returning a Boolean value when a bounce is detected. This was done with relative ease, as both of these functions did not rely upon any new understanding of code but only mathematical logic. This solution was created by taking the edges of the bounding rectangles, which tracked the tennis ball and the surface, and creating a function, which returned a false value when these edges did not overlap, but returned a true value when one or more of the edges overlapped.

This was an important iteration, as it helped me understand the importance of testing, as at first glance, I would have believed that my original function was satisfactory. However, after a short series of tests, I concluded that this function was far from accurate enough to publish as my final product.

##### 3.1.5.3.3 Test Table (for the second iteration)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run without crashing | Program runs successfully without crashing | Yes, the program runs without crashing as intended. The program should only crash if HSV (lower) values are higher than the HSV upper values. | Yes, the program runs without crashing as intended. |
| Can the video feed be displayed | Same as in previous prototypes | Yes, the video feed can be displayed. For this prototype it does not have to run smoothly as no optimisation has been made yet | Yes, a video feed can be displayed |
| Does the program end when the “escape” key is pressed | Input: “escape” key pressed  Output: program ends and all windows are closed | Yes, the program stops running when the “escape” key is pressed as it is supposed to be a way to stop the program from running with a single key press | Yes, the program stops running when the “escape” key is pressed |
| Does the program end when the “q” key is pressed | Input: “q” key is pressed  Output: no change in program | No, the program does not stop running when the “q” key is pressed as it is not intended to do so | No, the program does not stop running when the “q” key is pressed |
| Does the program crash when any other keys are pressed | Input: selection of random keys (other than “escape” and “q”) are pressed in a random order  Output: no change in program | No the program does not crash when any other keys are pressed as it is not intended to do so | No, the program does not crash when any other keys are pressed |
| Does the program crash when the camera’s field of view is obstructed |  | No the program does not crash when the camera’s field of view is obstructed | Partly. The program does not crash if the camera’s field of view is obstructed. However, if the program is already tracking an object and then the camera’s field of view is obstructed, then the program may crash |
| Do 5 separate windows open |  | Yes 5 separate windows open as intended | Yes, 5 separate windows open as intended |
| Do the GUI sliders work effectively |  | Yes the GUI sliders work effectively and set the HSV values for the mask as intended | Partly. The GUI sliders work effectively at setting the HSV upper and lower bounds for the mask however, if the lower HSV values are set to be greater than the higher, then the program crashes |
| Does the HSV mask work as intended (for tennis ball and surface) |  | Yes, the HSV mask works as intended and can allow us to find the HSV upper and lower values for a specific object, in this case the tennis ball | Partly. The HSV mask works as intended and can allow us to find the HSV values of a tennis ball. However, if the HSV values are set too high or too low, not detecting anything in the mask, the program will crash as it is trying to track something but it cannot track anything |
| Does the program track the tennis ball effectively without crashing |  | Yes, the program tracks the tennis ball effectively without crashing as it is intended to do so | Yes, the program can track an object effectively as intended |
| Is there a window which displays tracking of the tennis ball with a green box around it |  | Yes, there is a window, which displays tracking of the tennis ball with a green box around it. This box changes colour when a function returns a value | Yes, there is a separate window, which displays tracking of the tennis ball with a green box around it. The box changes colour to red when the “hitTest” function returns True |
| Is there a window which displays tracking of the surface with a blue box around it |  | Yes, there is a window, which displays tracking of the surface with a blue box around it. This box is not visible in the tracking window | Yes, there is a window, which displays tracking of the surface with a blue box around it. This box is not visible in the tracking window |
| Does the bounce detection work accurately |  | Yes, the bounce detection works accurately and effectively as intended | Yes, the bounce detection works accurately and exactly as intended. |
| Does the fps remain drop lower than 10 | https://raw.githubusercontent.com/FulfordComputing/AlevelYaros/master/2019-04-10-221408_1824x984_scrot.png?token=AmuAcB1Dueu_ykYIwPM4BveSfLwdIqWfks5cuOW2wA%3D%3D | Yes, the fps drops lower than 10, as there are more than 2 windows open in this iteration | Yes, the fps drops lower than 10, as there are 5 windows open, two of which are responsible for being HSV mask and two more responsible for tracking an object. The fps remains around 6(+/-)2 |

##### 3.1.5.3.4 Final Review and Changes Made

This prototype was a very effective way of solidifying my newfound level of understanding Python. This prototype did not require any new knowledge of code, which made it so much easier and quicker, when in reality it was one of the more difficult prototypes. This displayed how much my understanding of Python had improved since I started this project.

The final changes for my solution consisted of removing a large amount of white space from my code, while also removing the lines of code necessary for displaying my second HSV mask, my original frame and the tracking of my surface. The lines of code necessary for displaying the HSV mask for my surface I did not delete however, I commented them out, as these lines may be useful in the future, if I decide to use this product in a different light space, and I wasn’t to make sure I can easily find the HSV values for the surface. This resulted in a very significant improvement in terms of optimisation, as my fps shot up to a constant rate over 15.

# 4 Evaluation

## 4.1 Testing

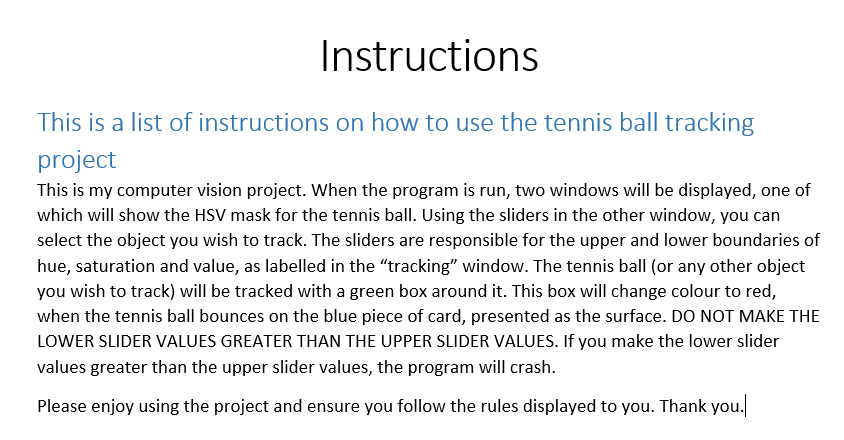
|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run without crashing | Program runs successfully without crashing | Yes, the program runs without crashing as intended. The program should only crash if HSV (lower) values are higher than the HSV upper values. | Yes, the program runs without crashing as intended. |
| Can the video feed be displayed | Same as in previous prototypes | Yes, the video feed can be displayed. For this prototype it does not have to run smoothly as no optimisation has been made yet | Yes, a video feed can be displayed |
| Does the program end when the “escape” key is pressed | Input: “escape” key pressed  Output: program ends and all windows are closed | Yes, the program stops running when the “escape” key is pressed as it is supposed to be a way to stop the program from running with a single key press | Yes, the program stops running when the “escape” key is pressed |
| Does the program end when the “q” key is pressed | Input: “q” key is pressed  Output: no change in program | No, the program does not stop running when the “q” key is pressed as it is not intended to do so | No, the program does not stop running when the “q” key is pressed |
| Does the program crash when any other keys are pressed | Input: selection of random keys (other than “escape” and “q”) are pressed in a random order  Output: no change in program | No the program does not crash when any other keys are pressed as it is not intended to do so | No, the program does not crash when any other keys are pressed |
| Does the program crash when the camera’s field of view is obstructed |  | Partly | Partly. The program does not crash if the camera’s field of view is obstructed. However, if the program is already tracking an object and then the camera’s field of view is obstructed, then the program may crash |
| Do 5 separate windows open |  | No, only 2 separate windows open | No, only 2 separate windows open. One window is the tracking window and the other is the HSV mask for the tennis ball |
| Do the GUI sliders work effectively |  | Yes the GUI sliders work effectively and set the HSV values for the mask as intended | Partly. The GUI sliders work effectively at setting the HSV upper and lower bounds for the mask however, if the lower HSV values are set to be greater than the higher, then the program crashes |
| Does the HSV mask work as intended (for tennis ball and surface) |  | Yes, the HSV mask works as intended and can allow us to find the HSV upper and lower values for a specific object, in this case the tennis ball | Partly. The HSV mask works as intended and can allow us to find the HSV values of a tennis ball. However, if the HSV values are set too high or too low, not detecting anything in the mask, the program will crash as it is trying to track something but it cannot track anything |
| Does the program track the tennis ball effectively without crashing |  | Yes, the program tracks the tennis ball effectively without crashing as it is intended to do so | Yes, the program can track an object effectively as intended |
| Is there a window which displays tracking of the tennis ball with a green box around it |  | Yes, there is a window, which displays tracking of the tennis ball with a green box around it. This box changes colour when a function returns a value | Yes, there is a separate window, which displays tracking of the tennis ball with a green box around it. The box changes colour to red when the “hitTest” function returns True |
| Is there a window which displays tracking of the surface with a blue box around it | No window is displayed | No, there is no window displaying the tracking of the surface, the surface is still tracked but this is not displayed | No, there is no window displaying the tracking of the surface. Only the tracking of the tennis ball and the HSV mask of the tennis ball is displayed. |
| Does the bounce detection work accurately |  | Yes, the bounce detection works accurately and effectively as intended | Yes, the bounce detection works accurately and exactly as intended. |
| Does the fps remain drop lower than 10 |  | No, the fps does not drop lower than 10 | No, the fps does not drop lower than 10. The fps remains at a constant level of around 15 |

## 4.2 Success Criteria

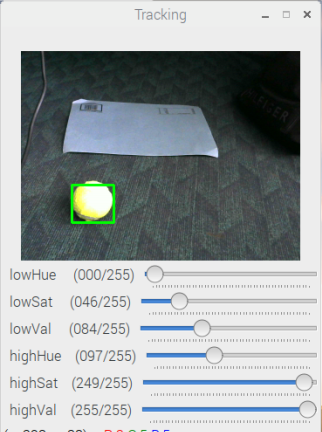
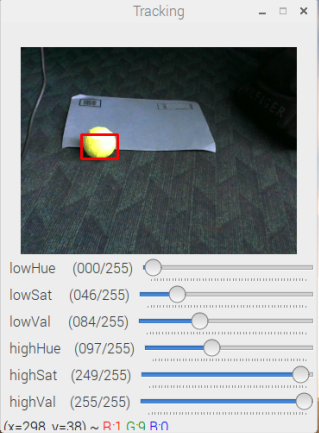
|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Explanation of Criteria** | **How Criteria is Met** | **Is The Criteria Met** |
| 1) The program needs to understand where the ball is in the field of view | The program must be able to recognise the ball if it is in its field of view, and if it is where bouts it is in each frame. | I will implement this by allowing the program to understand the HSV colour of the ball, and how it changes in differently lit environments. | Yes |
| 2) The program needs to understand when the ball bounces | The program must be able to understand whether the ball bounces and recognise when this occurs. | I will implement this by allowing the program to recognise, when the tennis ball connects with the surface. This will be done via hit detection. I will track the HSV values of the tennis ball with a box and the surface HSV values with another box. When the boxes overlap, a bounce will be detected. | Yes |
| 3) The program needs to track the ball | The program needs to be able to track the ball in its field of view. | I will implement this by using probability, after eliminating the false positives. This will be done by observing the location of the ball in each frame via tracking the HSV values of the tennis ball. | Yes |
| 4) The program needs to eliminate false positives | The program needs to be able to distinguish between false positives and true positives via probability, and eliminate the false positives. | I will implement this by using probability of potential ball candidates in each frame (size of the largest HSV contours detected) to allow me to understand which candidates are true positives. | Yes |
| 5) The whole project needs to be suitable for both genders | The project needs to be simple, easy to use and suitable for both genders | I will implement this by keeping by project as easy to understand as possible for someone without any knowledge of computing. I shall also keep the colours and style based around white, grey and black. | Yes |
| 6) The project needs to be suitable for ages between 15-16 | The project needs to be ergonomic for users and suitable for a wide variety of children/ teenagers of different ages | I will do this by keeping my product slick, easy to use and exciting. This will be achieved by not undermining the potential, which this technology can achieve. | Yes |
| 7) The project needs to be relatively simple to use | The project needs to be simple to use and understand by my users, as my stakeholders will be children aged 15-16. Simultaneously, this shouldn’t patronise them | I will do this by keeping my user interface simple and easy to understand. In order for the user to fully understand the program they will be using, I will write a short instruction list, explaining how the project works and how to use it. | Yes |

### 4.2.1 Successfully Met Success Criteria

I believe that all of my success criteria is met successfully with my final product. This can be broken down into seven stages.

The first stage is for the program to understand where the tennis ball is in its field of view. This is successfully met within my fourth prototype and throughout, continuing to the final solution. This is achieved by creating a HSV mask, for which the upper and lower bounds can be adjusted by my GUI sliders. These sliders are clearly labelled also display the numerical values (next to the labels). The instructions on how to use these GUI sliders are explained clearly in my “Instructions” document.

Then, the program needs to understand and display when a bounce is detected. This is done clearly by changing the colour of the bounding rectangle, which tracks the tennis ball from green to red, when the tennis ball is exposed to the designated surface. This is again explained clearly in the “Instructions” document (displayed above).



The next stage of the success criteria is for the program to be able to accurately tack the tennis ball. As you can see from the images above, this is achieved successfully. The tracking of the tennis ball is very self-explanatory however, I still explain how this works in the “Instructions” document, in order to ensure that my user understands exactly how this program works.

Linking to the previous stage of the success criteria, the program needs to eliminate false positives, in order to accurately track the tennis ball. This is met by only tracking the largest HSV contours in my program. By using this and the GUI sliders, to set the HSV upper and lower bounds to be as accurate as possible, we can track the tennis ball effectively, as all other false positives are eliminated. This is assisted by blurring the frame and removing the static (3.1.4).

The following criteria, is for the project to be easy to use and suitable for both genders. I achieved this by keeping the colour scheme non-biased to both genders. My colour scheme was black, white, grey, green and red. This keeps the product looking professional, while simultaneously remaining exciting. This effectively represents how the software and hardware work in a professional and effective way, while still being extremely exciting in the accomplishments of the computer vision. This directly meets the next success criteria, which is for the project to be suitable for an age range of 15-16 years old. I believe that this is successfully met by the complexity of my project, as well as the “Instructions” document, as this document clearly explains how to use the project, while not patronising the user.

My final success criteria, is for my project to remain relatively simple to use. I believe that this is achieved by keeping the user interface self-explanatory however, the “Instructions” document makes this even easier to understand. This also allows the user to fully utilise all of the functions of the program.

### 4.2.2 Partially Met or Unmet Success Criteria

None of my success criteria was partially met or unmet. This is a great relief, as I believe this will leave my stakeholders satisfied. I will conduct an interview with a selection of 15-16 year olds from my computer science teachers year 11 class, in order to understand whether there are any other criteria, which they would like to be met in the future.

### 4.2.3 Stakeholder Interview for Success Criteria

This interview was conducted after I explained to a selection of three students from Mr Dring’s class, how the project works and allowed them to read the “Instructions” document. I then allowed them all to test out my project and see how it worked.

1. **What age range do you think this project is suitable for?**  
   Student 1 – I think this project tends to a more student-based audience. So age ranges of 14-25 I guess.  
   Student 2 – Any age range after the age of 14 probably. The software is very easy to use.  
   Student 3 – Yeah this is probably best suited for students.
2. **Do you find anything confusing about the project?**Student 1 – The only thing I found a bit confusing is how the hue lower and upper slider worked. After you explained that this represents the blue, green and red (BGR) colour range, it was obvious.  
   Student 2 – The instructions were very clear and well explained so I didn’t struggle understanding anything.  
   Student 3 – Same as what “Student 1” said to be honest.
3. **Do you find anything confusing about the hardware?**Student 1,2 and 3 - No
4. **Do you believe the instructions require any more additions?**Student 1 – You could probably explain how the hue sliders works a bit more. Other than that no.  
   Student 2 – No, I think the instructions are very clear.  
   Student 3 – Yeah, you could explain the hue sliders a bit more, but that is it.
5. **Does this project inspire you to study computer science post GCSE?**  
   Student 1 – Definitely, I think it is really cool that A-level students get to create something like this for their course work.  
   Student 2 – I was already planning on studying computer science at A-level and studying a degree in it. However, I think this is inspirational for students like me to find out the level of understanding (of computer science) which we can gain during our A-level.  
   Student 3 – Yeah, it does tempt me a bit. I just do not ever see myself making anything near this level.
6. **Are there any things, which you would like to be added in the future of this project?**Student 1 – It would be cool if you could make this portable and use it outside. Then it could be used for real tennis games.  
   Student 2 – “Student 1”’s suggestion is great. I would have to agree.  
   Student 3 – Yeah, same.
7. **Do you see any potential application of this in real life in the future?**Student 1 – Tennis clubs could use this to detect whether a ball bounced in or not could probably use this.   
   Student 2 – This could definitely be used to make cool YouTube videos or something like that.  
   Student 3 – I think this could be a really good demonstration for GCSE students, as an example of A-level projects, which they could make for their course work.
8. **Have you heard of Hawk-Eye?**  
   Student 1 – Yeah it is the tracking used in sports, but more commonly known for tennis.  
   Student 2 – Yeah the tennis one.  
   Student 3 – Never heard of it really.
9. **How easy was it to tell that a bounce was detected?**Student 1, 2 and 3 – Yeah, it was clearly explained in the Instructions and already self-explanatory.
10. **Would you like to use a voice-operated system for this product in the future?**Student 1 – It would be cool however, I do not think it would work with this style of software very well.  
    Student 2 – I do not think it would be very beneficial in this product. However, this would still be cool.  
    Student 3 – Yeah that would be cool.

This interview gave me an insight of future implementation, as well as how effective I met both my success criteria and how successful my usability was.

## 4.3 Usability Features

|  |  |  |
| --- | --- | --- |
| **Usability Features** | **Were These Features Met** | **How Were They Met** |
| Tracking a tennis ball with a green bounding rectangle | Yes | This success criterion was met as the tennis ball can be tracked if the HSV upper and lower bounds are input accurately. This allows the tennis ball to be tracked with a green bounding rectangle |
| Tracking tennis ball with a red bounding rectangle when a bounce is detected | Yes | The green bounding rectangle changes colour to red, when a bounce is detected. This occurs when the tennis ball is exposed to the designated surface |
| Is an fps counter displayed | Yes | An fps counter is displayed in the message box, when the program starts running. The fps is no longer displayed when the program stops running |
| Does the fps remain constantly above 10 | Yes | The fps remains at a constant rate around 15. |
| Is the UI simple to understand | Yes | The GUI sliders are labelled appropriately, displaying the values, which are input. The instructions, which I present to the user, allow them to easily understand the user interface accordingly. |
| Black, green, grey, white and red colour scheme | Yes | The user interface follows the designated colour scheme, designed to adapt to fit towards my stakeholders |
| Only one object can be tracked and displayed at a time | Yes | Only one object can be tracked at a time (other than the designated surface). This is met by tracking the largest contours of the HSV values of the tennis ball. |
| One designated surface | Yes | Only one designated surface can be used for bounce detection. In my project, I used a blue piece of card however, the HSV values of any surface can be entered into the program, if the code is adjusted, based on the light environment. |
| Program doesn’t crash when it runs | Partly | The program does not crash when it runs, unless the light environment changes abruptly, or the lower HSV bounds are set to be greater than the upper HSV bounds. This should not occur however, as in the instructions, it is clearly stated not to do this. |

### 4.3.1 Successful Usability Features

The tennis ball can be successfully tracked with a green bounding rectangle. This bounding rectangle changes colour from green to red when a bounce is detected. An fps counter is displayed in the message box, which remains at a constant rate of around 15 fps. Only one object can be tracked at a time and there can only be one designated surface at a time. The user interface is relatively simple to understand, paired together with the instructions, which allows the user to understand the program clearly. The colour scheme matches how it was designed to be, as this was adapted to fit my stakeholders.

### 4.3.2 Partially Successful and Unsuccessful Usability Features

There is only one criterion, which is partly met. This is that the program is not supposed to crash when it runs. This is successfully met, if the user follows all of the instructions displayed to them, unless the light environment changes extremely and abruptly. The program will also crash if the HSV lower bounds are set to be greater than the upper bounds. This can be easily adapted in the future, by creating a separate function, which is responsible for returning nothing if there are no HSV contours and if the HSV lower bounds are set to be greater than the upper bounds.

There are no success criteria, which are not met at all. This is significant, as it shows that the project was extremely successful, suggesting that my stakeholders would be satisfied with the final product, which they are presented with. In this case, the reaction my main stakeholder, Tom Smith, was displayed in an interview (4.3.3).

### 4.3.3 Final Stakeholder Interview

In this interview, I presented Tom with my final product, the instructions, the usability features and the success criteria. This interview allowed me to understand a clear response to my finished product, as well as presenting myself with the limitations and ways to deal with these limitations in the future.

1. **What are your first impressions of the finished product?**I think it is absolutely amazing! This is exactly what I wanted the final product to be. It matches very closely to the design you presented in our previous interviews.
2. **Are there any features, with which you are not satisfied?**Not at all. This product is very satisfactory.
3. **Do you believe that the instructions are detailed enough?**Yes. Personally, after reading the instructions and observing the user interface for a few seconds, I knew exactly how to use the program.
4. **Do you believe that the user interface is easy to understand?**  
   Yes, most definitely.
5. **What are the limitations of this project currently?**  
   I think the only limitations are the fact that this product is not portable, and is not suitable for an outdoor environment. However, considering the financial resources available to you, I do not believe you could have accomplished a project more successful than the one you have.
6. **What features do you think would be useful to implement in the future?**I think it would be cool if you could make this project portable and suitable for an outdoor environment.
7. **Do you see any potential application of this in real life?**Definitely. I believe that tennis clubs, like mine, would extremely benefit if they had a product like this, observing the tennis court, detecting whether the ball bounces inside or outside the court.
8. **Do you think you can make interesting videos with the finished product?**  
   Definitely. I would love to have a product like this, to make videos of me playing tennis, or even just playing around with the product itself.
9. **Does the finished product inspire you to study A-level computer science?**Most definitely. I am already planning to study computer science at A-level however; this project is inspiring, as it shows that when I am doing my computer science course work, I will try to create something as complexed as this.
10. **Do you believe that this project was a success, based on the success criteria?**Yes. This is an extremely successful final product.

## 4.4 Limitations

### 4.4.1 Limitations of the Solution

The limitations of my final product can be decomposed into two separate sections. The limitations of the usability and the limiting features. There are only two limitations of my usability, which the fact that the program can crash if the lighting of the environment is changed abruptly, as well as the fact that the program can crash if the lower bounds for the HSV values are set to be greater than the upper bounds.

The limiting features consist of the fact that my final product is not portable and is not suitable for an outdoor environment. These limitations are based more on my current available financial resources, than my ability to create a solution, which is suitable to fit these requirements. All of these limitations can be potentially addressed in the future if I can collect enough funding for this new solution.

### 4.4.2 Dealing with Limitations in the Future

The limitations for the usability of the program can be easily addressed in the future, with relatively simple validation. I will address these limitations by creating a function, which is responsible for returning nothing, if no HSV contours are detected, or if the lower HSV bounds are set to be greater than the upper. I can also display a message box, if the user sets the HSV lower bounds to be greater than the upper bounds.

The limiting features can be addressed in the future, if I can collect the necessary funds, in order to get a waterproof camera, which can be connected to the Raspberrypi via Bluetooth. This is the only way to address these limiting features, as there is no other way I can make my current hardware combination suited to an outdoor environment.

## 4.5 Future Maintenance

In the future, I could improve my object tracking by creating a function, which tracks the path of the tennis ball and detects if a change in the path was unnatural. This could then be used to detect when a tennis ball is hit with a tennis racket. This can be done by using an evolutionary artificial intelligence, which learns a regular, natural path of a tennis ball by observing hundreds of tennis games. Similarly, it can learn the when the path of the tennis ball changes in an unnatural way. This could then be linked with bounce detection, in order to create a more accurate bounce detection function. The only other future maintenance, which is required, is regularly updating all of my libraries, in order to keep my program working effectively.

All of the other future maintenance, I have addressed in the rest of my evaluation section.

## 4.6 Finished Code

|  |
| --- |
| from \_\_future\_\_ import division |
|  | from picamera.array import PiRGBArray |
|  | from picamera import PiCamera |
|  | import imutils |
|  | from imutils.video import FPS |
|  | from imutils.video import VideoStream |
|  | #threading camera fies |
|  | from threading import Thread |
|  | import cv2 |
|  | import numpy as np |
|  | import time |
|  |  |
|  | def hitTest(x1,y1,x2,y2,w1,h1,w2,h2): |
|  | if x1+w1 >= x2: |
|  | if y1+h1 >= y2: |
|  |  |
|  |  |
|  | if x1 <= x2+w2: |
|  | if y1 <= y2+h2: |
|  | return True |
|  | return False |
|  |  |
|  | class WebcamVideoStream: |
|  | def \_\_init\_\_(self, src=0): |
|  | # initialize the video camera stream and read the first frame |
|  | # from the stream |
|  | self.stream = cv2.VideoCapture(src) |
|  | (self.grabbed, self.frame) = self.stream.read() |
|  |  |
|  | # initialize the variable used to indicate if the thread should |
|  | # be stopped |
|  | self.stopped = False |
|  |  |
|  | def nothing(\*arg): |
|  | pass |
|  |  |
|  | FRAME\_WIDTH = 320 |
|  | FRAME\_HEIGHT = 240 |
|  |  |
|  | # Initial HSV GUI slider values to load on program start. |
|  | icol = (0, 0, 0, 255, 255, 255) |
|  | # Set HSV values for blue folder. |
|  | blueLower = (20, 0, 143) |
|  | blueUpper = (170, 114, 255) |
|  |  |
|  | cv2.namedWindow('Tracking') |
|  | # Lower range colour sliders. |
|  | cv2.createTrackbar('lowHue', 'Tracking', icol[0], 255, nothing) |
|  | cv2.createTrackbar('lowSat', 'Tracking', icol[1], 255, nothing) |
|  | cv2.createTrackbar('lowVal', 'Tracking', icol[2], 255, nothing) |
|  | # Higher range colour sliders. |
|  | cv2.createTrackbar('highHue', 'Tracking', icol[3], 255, nothing) |
|  | cv2.createTrackbar('highSat', 'Tracking', icol[4], 255, nothing) |
|  | cv2.createTrackbar('highVal', 'Tracking', icol[5], 255, nothing) |
|  |  |
|  | # Initialize webcam. Webcam 0 |
|  | vidCapture = cv2.VideoCapture(0) |
|  | vidCapture.set(cv2.CAP\_PROP\_FRAME\_WIDTH,FRAME\_WIDTH) |
|  | vidCapture.set(cv2.CAP\_PROP\_FRAME\_HEIGHT,FRAME\_HEIGHT) |
|  |  |
|  | while True: |
|  | timeCheck = time.time() |
|  | # Get HSV values from the GUI sliders. |
|  | lowHue = cv2.getTrackbarPos('lowHue', 'Tracking') |
|  | lowSat = cv2.getTrackbarPos('lowSat', 'Tracking') |
|  | lowVal = cv2.getTrackbarPos('lowVal', 'Tracking') |
|  | highHue = cv2.getTrackbarPos('highHue', 'Tracking') |
|  | highSat = cv2.getTrackbarPos('highSat', 'Tracking') |
|  | highVal = cv2.getTrackbarPos('highVal', 'Tracking') |
|  |  |
|  |  |
|  | # Get webcam frame |
|  | \_, frame = vidCapture.read() |
|  |  |
|  | # Convert the frame to HSV colour model. |
|  | frameHSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV) |
|  | # Repeated for second HSV frame. |
|  | frame2HSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV) |
|  |  |
|  |  |
|  | # HSV values to define a colour range we want to create a mask from. |
|  | colorLow = np.array([lowHue,lowSat,lowVal]) |
|  | colorHigh = np.array([highHue,highSat,highVal]) |
|  | mask = cv2.inRange(frameHSV, colorLow, colorHigh) |
|  | #Blur the image to remove static and make tracking easier |
|  | blurred2 = cv2.GaussianBlur(frameHSV, (11, 11), 0) |
|  | hsv2 = cv2.cvtColor(blurred2, cv2.COLOR\_BGR2HSV) |
|  | # Show the first mask |
|  | cv2.imshow('mask - tennis ball', mask) |
|  |  |
|  | # resize the frame, blur it, and convert it to the HSV |
|  | # color space |
|  | frame2 = imutils.resize(frame, width=320, height=240) |
|  | blurred = cv2.GaussianBlur(frame2, (11, 11), 0) |
|  | hsv = cv2.cvtColor(blurred, cv2.COLOR\_BGR2HSV) |
|  | # construct a mask for the color "green", then perform |
|  | # a series of dilations and erosions to remove any small |
|  | # blobs left in the mask |
|  | mask2 = cv2.inRange(hsv, blueLower, blueUpper) |
|  |  |
|  | #show the second mask |
|  | #cv2.imshow('mask-folder', mask2) |
|  |  |
|  |  |
|  | #set up boundaries in order to find biggest contours and draw a box around them for tennis ball |
|  | im2, contours, hierarchy = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) |
|  | #set up boundaries in order to find biggest contours and draw a box around them for surface |
|  | im2, contours2, hierarchy = cv2.findContours(mask2, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) |
|  |  |
|  | #find biggest contours for tennis ball |
|  | contour\_sizes = [(cv2.contourArea(contour), contour) for contour in contours] |
|  | biggest\_contour = max(contour\_sizes, key=lambda x: x[0])[1] |
|  | #find biggest contours for surface |
|  | contour\_sizes2 = [(cv2.contourArea(contour), contour) for contour in contours2] |
|  | biggest\_contour2 = max(contour\_sizes2, key=lambda x: x[0])[1] |
|  |  |
|  | #draw green box around tennis ball |
|  | x,y,w,h = cv2.boundingRect(biggest\_contour) |
|  | tennisBall = cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2) |
|  | #draw blue box arround surface |
|  | x2,y2,w2,h2 = cv2.boundingRect(biggest\_contour2) |
|  | surface = cv2.rectangle(frame2,(x2,y2),(x2+w2,y2+h2),(255,0,0),2) |
|  |  |
|  | if hitTest(x,y,x2,y2,w,h,w2,h2) == True: |
|  | tennisBall = cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2) |
|  |  |
|  | # Show final output image |
|  | cv2.imshow('Tracking', frame) |
|  |  |
|  | k = cv2.waitKey(5) & 0xFF |
|  | if k == 27: |
|  | break |
|  | print('FPS: - ', 1/(time.time() - timeCheck)) |
|  |  |
|  | cv2.destroyAllWindows() |
|  | vidCapture.release() |