[Date]

12ShiroY

AI Computer Vision

OCR Computer Science Project

Table *of* Contents

[1 Analysis 2](#_Toc4084703)

[1.1 The Problem 2](#_Toc4084704)

[1.2 The Stakeholders 2](#_Toc4084705)

[1.3 Existing Solutions 3](#_Toc4084706)

[1.3.1 In-Depth Breakdown of Hawk-Eye 4](#_Toc4084707)

[1.4 The Essential Features 6](#_Toc4084708)

[1.5 Potential Limitations 6](#_Toc4084709)

[1.6 Hardware and Software Requirements 6](#_Toc4084710)

[1.7 Success Criteria 8](#_Toc4084711)

[2 Design 9](#_Toc4084712)

[2.1 Structure 9](#_Toc4084713)

[2.2 Decomposition 10](#_Toc4084714)

[2.2.1 Computer Vision 10](#_Toc4084715)

[2.2.2 Tracking the Ball 10](#_Toc4084716)

[2.2.3 User Interface 10](#_Toc4084717)

[2.3 Algorithms 10](#_Toc4084718)

[2.3.1 Accessing camera and displaying camera feed 11](#_Toc4084719)

[2.3.2 HSV colour mask 11](#_Toc4084720)

[2.3.3 Detecting and tracking the tennis ball 13](#_Toc4084721)

[2.4 Usability 15](#_Toc4084722)

[2.5 Variables, Data Structures and Classes 15](#_Toc4084723)

[2.6 Test Data 16](#_Toc4084724)

[2.7 Further Data 17](#_Toc4084725)

[3 Developing 17](#_Toc4084726)

[3.1 Iterative Developments 17](#_Toc4084727)

[3.2 Prototypes 17](#_Toc4084728)

[3.2.1 Prototype 1 – Displaying a video taken on the picamera (testing the camera works) 17](#_Toc4084729)

[3.2.2 Prototype 2 – Accessing a single image taken on the picamera module 21](#_Toc4084730)

[3.2.3 Prototype 3 – Accessing the video stream of the picamera module 22](#_Toc4084731)

[3.2.4 Prototype 4 – Tracking a tennis ball with a Raspberry Pi and picamera module using Python and OpenCV 24](#_Toc4084732)

[3.3 Structures 30](#_Toc4084733)

[3.4 Examples of Code (Annotated) 30](#_Toc4084734)

[3.5 Validations 30](#_Toc4084735)

[3.6 Review 30](#_Toc4084736)

[4 Evaluation 30](#_Toc4084737)

[4.1 Success Criteria 30](#_Toc4084738)

[4.2 Potential in Future 30](#_Toc4084739)

[4.3 Future Improvements 30](#_Toc4084740)

[4.4 Usability 30](#_Toc4084741)

[5 Bibliography 30](#_Toc4084742)

# 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

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. **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.
3. **Do you find anything confusing about the hardware?**No
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. **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.
9. **Would you like to use a voice operated system for this product or not?**That would be awesome!

## 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.

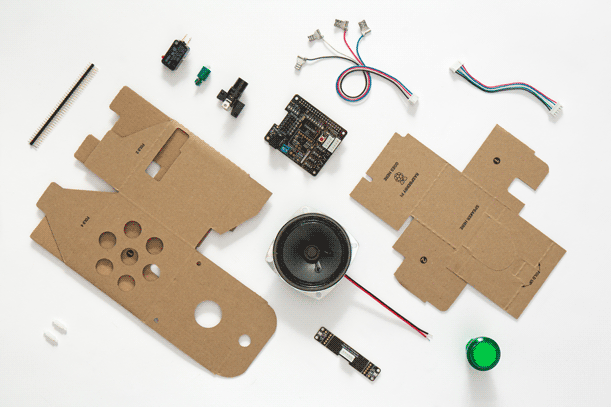
The way in which my program will track the tennis ball, will be mainly focused by tracking the HSV values of the tennis ball, as the colour is very bright and distinct. This will make it much easier as the I will not have to find the path of the tennis ball for the tracking of the ball, but only for detecting the bounce of the tennis ball.



## 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 any surface, which should be detected via an unnatural change of direction of the tennis ball.

## 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; self-made alternate hardware such as a homemade Google AIY kit. 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 was 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.

## 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 | AI – an evolutionary AI, which learns how to move the car around the track, when it crosses the finish line and how to decrease the time taken to complete a lap. This AI will do this using computer vision and measuring the outputs via ammeter and voltmeter. |
|  | (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. | Faulty computer vision – if the computer vision cannot see where the ball is and how it travels through the air. This could cause major issues with my project and could mean that I would have to start again on working with finding true positives and false positives. |
|  | 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 problem to recognise, when the path of the ball changes in a way that is considered not natural (caused by gravity) allowing it to then again predict the path of the ball quickly after the bounce. |
| 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 (general “boxy” and “homemade” aesthetic). |
| 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) (possibly) The program will use voice command | The program needs to use voice command in order for the user to turn the program on | I will do this by using the google AIY kit, which comes with a microphone and basic voice commands already preinstalled. These can be altered to be specified to my specific situation |

# 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.

### 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 understand the rough shape of the ball and the HSV (Hue Saturation Value). This will allow the program to detect the ball candidates in a constantly changing environment (in terms of lighting). This will work hand in hand with the actual path detection and path prediction for tracking the ball.

### 2.2.2 Tracking the Ball

#### 2.2.2.1 Finding Ball Path

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 taking three frames as soon as a ball candidate is detected on the screen. If the three frames contain three ball candidates in very similar position (tracklet), then the program will be able to determine, that this ball candidate is most likely a true positive. This will then be further checked if the ball candidate has actually remained in the same position in the camera’s field of view, if so, then this could possibly be a false positive (however, if not, the ball candidate is almost definitely a true positive). If no true positive is detected, then this will be repeated until one is found. The next part of the program is to use the trajectory of the ball, and the path detected, to determine the most probable predicted path of the ball.

#### 2.2.2.2 Bounce Detection

This is a relatively simple part of the program, as this will require the program to detect an unnatural change in the ball’s path. By using techniques from finding the ball path and predicting it, we can save a significant amount of time here as this can mainly be done by reusing code. This will be done by using the already detected and predicted path of the ball, and cutting this off (and repeating the whole process again) when an unnatural change in the ball’s trajectory occurs.

### 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-plain', 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()

## 2.4 Usability



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 around 60.

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.

## 2.5 Variables, Data Structures and Classes

|  |  |  |
| --- | --- | --- |
| **Variable / Data Structure Name** | **Type** | **What The Variable Does** |
| BallShape | Array | Stores the shape of the ball from a repository of images of it. |
| 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 sizes of the ball and HSV values, to eliminate false positives |
| BallCandidateDetected | Variable - Boolean | Returns whether there are any ball candidates in its field of view |
| Bounce | Function | Responsible for sending information to the program that the path of the ball has changed in an unnatural way, meaning that a new path should be predicted now. However, the ball should still be tracked. |
| BounceDetected | Variable - Boolean | Returns whether there are any bounces detected. Does this by detecting whether the ball’s path changes in an unnatural game |
| Camera | String | Determine that camera = picamera |
| 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. |

## 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

# 3 Developing

## 3.1 Iterative Developments

## 3.2 Prototypes

### 3.2.1 Prototype 1 – Displaying a video taken on the picamera (testing the camera works)

#### 3.2.1.1 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/>)

#### 3.2.1.2 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.

#### 3.2.1.3 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.

##### 3.2.1.3.1 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 |

#### 3.2.1.4 Review and changes made

### 3.2.2 Prototype 2 – Accessing a single image taken on the picamera module

#### 3.2.2.1 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.

#### 3.2.2.2 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.

#### 3.2.2.3 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.

##### 3.2.2.3.1 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 3.2.1) 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 |

#### 3.2.2.4 Review

### 3.2.3 Prototype 3 – Accessing the video stream of the picamera module

#### 3.2.3.1 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.

#### 3.2.3.2 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”.



#### 3.2.3.3 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.

##### 3.2.3.3.1 Test Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Can a photo taken with the picamera module be displayed on the screen |  | 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 |  | 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 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. |

#### 3.2.3.4 Review and changes made

### 3.2.4 Prototype 4 – Tracking a tennis ball with a Raspberry Pi and picamera module using Python and OpenCV

#### 3.2.4.1 Structure

This section is going to be split up into multiple 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.

#### 3.2.4.2 Examples of code (annotated)

##### 3.2.4.2.1 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.

##### 3.2.4.2.2 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

#open a window to define the upper and lower HSV boundaries  
open.namedwindow(‘HSV’)  
#lower range sliders  
cv2.createtrackbar(‘lowHue’, ‘HSV’, icol [0], 255, nothing)  
cv2.createtrackbar(‘lowSaturation’, ‘HSV’, icol [1], 255, nothing)  
cv2.createtrackbar(‘lowValue’, ‘HSV’, icol [2], 255, nothing)   
#upper range sliders  
cv2.createtrackbar(‘highHue’, ‘HSV’, icol [3], 255, nothing)  
cv2.createtrackbar(‘highSaturation’, ‘HSV’, icol [4], 255, nothing)  
cv2.createtrackbar(‘highValue’, ‘HSV’, icol [5], 255, nothing)

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 3.2.4.2.3 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.

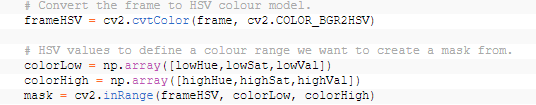
##### 3.2.4.2.3 Loop to get HSV values and create a mask

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.

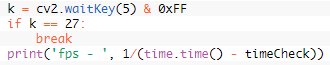
In the next section of code, we display the first mask in a separate frame. Then we find the contours using the “cv2.findContours” function to allow the mask to work. Then, the contour sizes are defined, in order to then allow us to track the biggest contour, which, as you can see is defined in the next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

##### 3.2.4.2.4 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 3.2.5) 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.

##### 3.2.4.2.5 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.”

#### 3.2.4.3 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.

##### 3.2.4.3.1 Test table (for first iteration of prototype 4)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run 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. | No, the program does not run without crashing, it crashes straight away |
| Can the video feed be displayed |  |  |  |
| Does the program end when the “escape” 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 |  |  |  |

#### 3.2.4.4 Review and changes made (for the first iteration of prototype 4)

#### 3.2.4.5 Test table (for the second iteration of prototype 4)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Test Data** | **Expected Outcome** | **Actual Outcome** |
| Does the program run 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 |  | 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 |
| Does the program end when the “escape” 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 |  |  |  |

#### 3.2.4.6 Review and changes made (for the second iteration of prototype 4)

## 3.3 Structures

## 3.4 Examples of Code (Annotated)

## 3.5 Validations

## 3.6 Review

# 4 Evaluation

## 4.1 Success Criteria

## 4.2 Potential in Future

## 4.3 Future Improvements

## 4.4 Usability

# 5 Bibliography