Single-camera motion capture application using a smartphone and Opencv

James Miller

P-NUMBER: P15212247

2019

Contents

[Introduction 3](#_Toc5701591)

[The Purpose of the Project 3](#_Toc5701592)

[Intended audience 3](#_Toc5701593)

[Project description 3](#_Toc5701594)

[Overall functionality descriptions 4](#_Toc5701595)

[camera calibration 4](#_Toc5701596)

[object detection 4](#_Toc5701597)

[3d position calculation 5](#_Toc5701598)

[3d position exporting 6](#_Toc5701599)

[Goals of the Project 7](#_Toc5701600)

[Conclusion 7](#_Toc5701601)

[**Functional requirements** 9](#_Toc5701602)

[Literature Review 9](#_Toc5701603)

[Introduction 10](#_Toc5701604)

[**Context** 10](#_Toc5701605)

[A brief introduction to OpenCV 10](#_Toc5701606)

[Key terms 11](#_Toc5701607)

[Use Case Diagram 12](#_Toc5701608)

[Project lifecycle 15](#_Toc5701609)

[Testing 16](#_Toc5701610)

[Design 20](#_Toc5701611)

[User Interface 21](#_Toc5701612)

[Navigation 23](#_Toc5701613)

[Noteworthy functions 24](#_Toc5701614)

[Camera Calibration 25](#_Toc5701615)

[Processing calibration: 27](#_Toc5701616)

[Distance approximation 28](#_Toc5701617)

[Calibration 28](#_Toc5701618)

[Undistortion of coordinates. 31](#_Toc5701619)

[Project Issues 32](#_Toc5701620)

[Kivy and opencv camera 32](#_Toc5701621)

[Recording of object 33](#_Toc5701622)

[Open Issues 34](#_Toc5701623)

[Android APK 34](#_Toc5701624)

[References / Bibliography 35](#_Toc5701625)

# Introduction

## The Purpose of the Project

The purpose of the application is to enable the use of any camera as a source for motion capture of a given object in order to produce a JSON file containing a series of 3d coordinates for use in a variety of 3d animation platforms. In the case of this document the object is a tennis ball, the camera is the rear-facing camera, of a Samsung galaxy s9 and the outputted JSON file will be processed using a simple script written within the blender 3d environment (blender.org, 2019).

## Intended audience

This application is intended for anyone with an interest in 3d animation of any kind who does not have access to or the resources available to afford the more elaborate/expensive purpose made hardware that would create similar results.

Although bespoke hardware and software would undoubtedly produce more accurate data this application could even serve as a quick prototyping mechanism for 3d animators both with and without huge amounts of resources due to its lightweight design and flexibility of different video device platforms due to its ability to calibrate any camera for its uses.

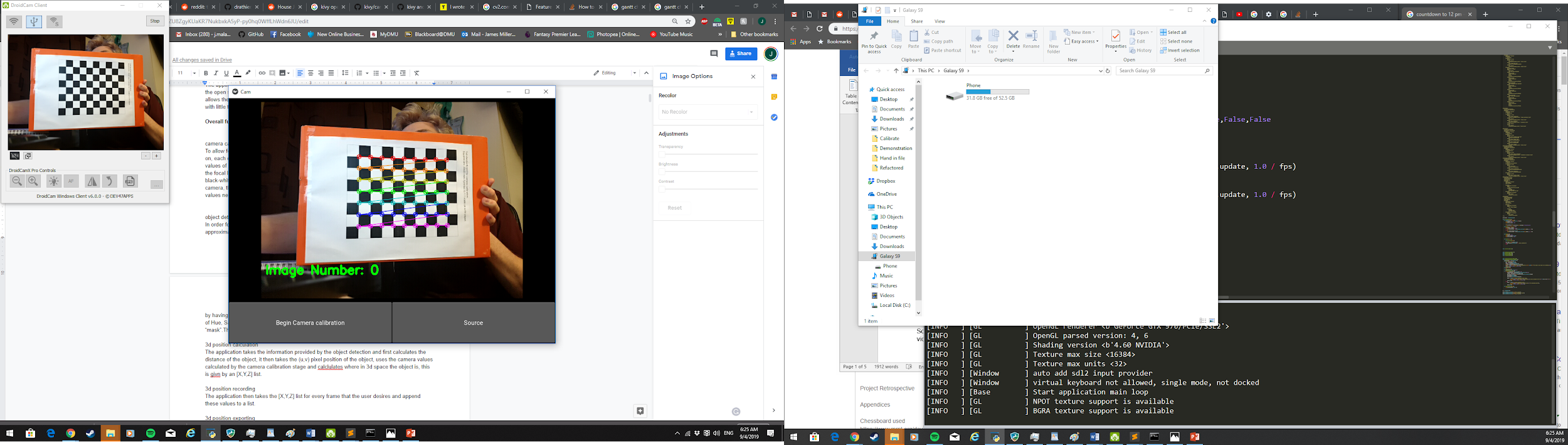
## Project description

The application is written in python. It utilizes many of the computer vision processing aspects of the open source OpenCV module and is written within the open source Kivy UI wrapper which allows the application to run on multiple platforms, such as OSX, Linux, Windows and Android with little to no modification of the underlying code.

## Overall functionality descriptions

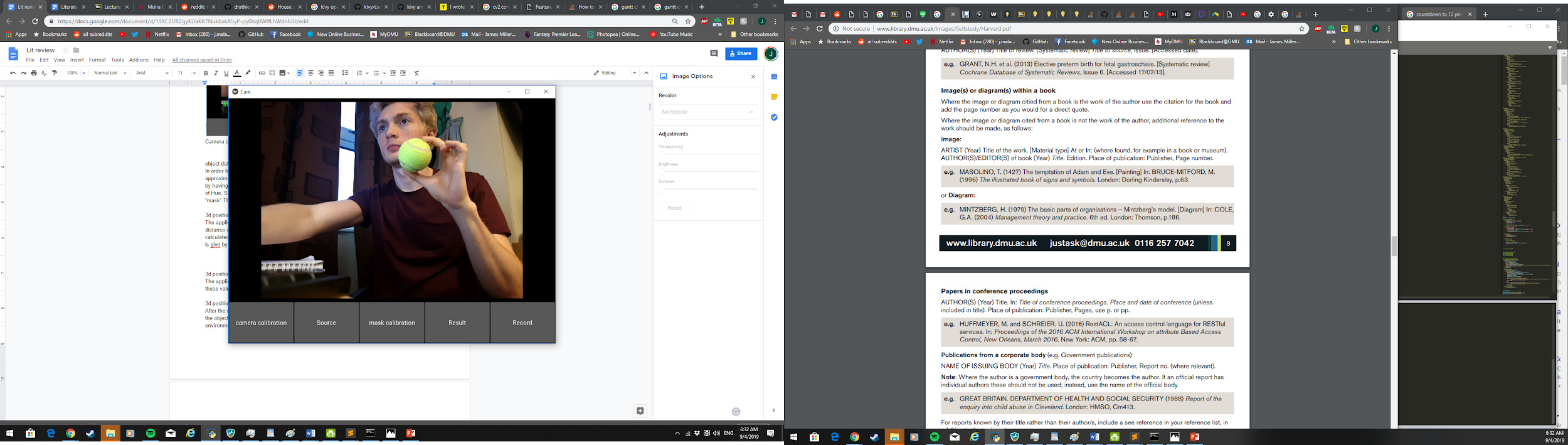
### camera calibration

To allow for many of the calculations of the 3D position of an object to be accurately made, later on, each camera the application runs on must be calibrated in order for certain underlying values of the camera to be found. The values calculated are the camera matrix (which includes the focal length of the camera), and the distortion coefficients. This is done by taking photos of a black-white chessboard pattern printed out by the user at multiple angles and distances from the camera, the application then takes the information collected from these images to calculate the values needed.

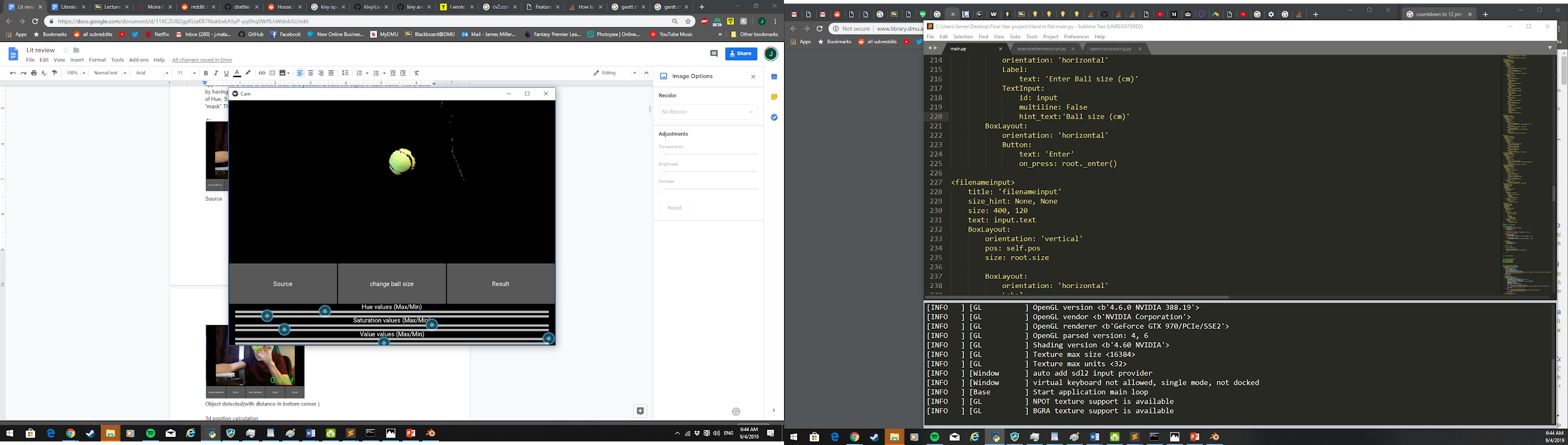
Camera calibration

### object detection

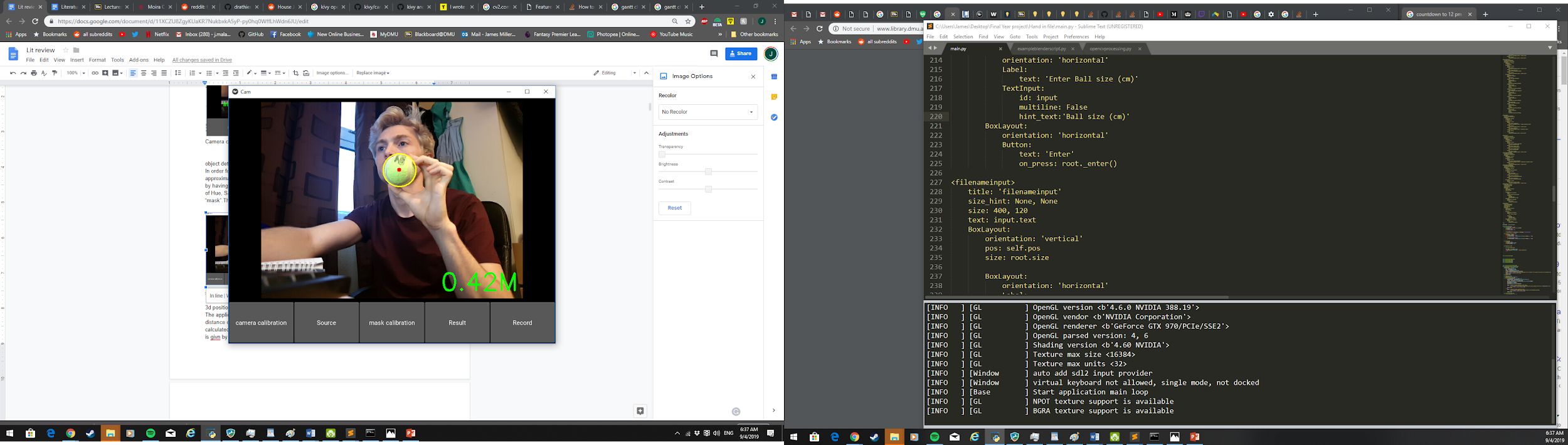
For the 3d coordinates of a specified object to be calculated, the application must first approximate a circle of known width and position around the object in each frame. This is done by having the object a different colour than the rest of the scene and then adjusting thresholds of Hue, Saturation and Value (HSV) so that only the desired object appears in what’s called the “mask”.The application then reads this mask and a circle is then drawn around the object.



Source



Mask



Object detected(with distance in bottom corner )

### 3d position calculation

The application takes the information provided by the object detection and first calculates the distance of the object, it then takes the (u,v) pixel position of the object, uses the camera values calculated by the camera calibration stage and calculates where in 3d space the object is, this is given by an [X,Y,Z] list.



Object 3d position calculated (top left)

3d position recording

The application then takes the [X,Y,Z] list for every frame that the user desires and append these values to a list.

### 3d position exporting

After the recording is completed by the user the list containing all of the X,Y and Z positions of the object are then saved to a JSON file, this file will be able to be interpreted by a script in a 3d environment in order to add animation to an object in that environment.

3d positions exported to 3d environment using script (blender)

### Goals of the Project

The goal of the project is to enable animators with less access to expensive equipment the ability to easily run this application in order to produce reliable positional data for animation purposes. Although the scope of use of the 3d data shouldn’t be limited to only that use, it could for example also be used as a data acquisition platform for some kind of engine to calculate properties of an ball when bounced off a surface (such as it’s “springiness” or elasticity).

## Conclusion

The application does meet the aims presented in the initial project contract, those being:

* Effectively detect and capture the motion of an object in 3d space using a single camera
* use the gathered data to construct a file that can be used by a 3d environment such as blender
* Enable a smartphone camera to be the capture device

It of course does this with a few extra steps, notably the camera calibration, of the camera which was not originally understood to be a prerequisite of the ability to track an object in 3d space using a single camera.

The application does create quite smooth animations when the files are transferred into blender, although there is sometimes a bit of a noticeable “jiggle” at some points if the mask calibration is slightly off or the background of the image is too close to the colour of the target object. But this can easily be counteracted with the built in “snapshot “option during the recording phase, this can allow animators make use of a stop motion type animation technique by taking a series of positions, these positions can be a fair distance apart from one another meaning that there is no notice “jiggle” within the animation.

One aspect I am particularly proud of is actually the structure of the system, the classes are created in such a way that you could theoretically be able to scale up this project to be able to track many multiple object, all with varying HSV thresholds, object sizes and maybe even captured from different cameras as the object variables are kept in their own class this would allow for objects to interact with each other within the animation and give a very “real” feeling to an animations as h animations are from, well, *real* objects.

Another aspect of the project I am particularly proud of is the overall ability of the program to take what is essentially just a 2d image and extrapolate the 3D position of a specified object, because as before this I had next to no experience in image processing or 3D coordinate geometry I feel that it’s quite a large step in terms of understanding of the subject that can now be backed up with an implementation of the theories of image calibration, distance approximation, object detection, object tracking, 3D reprojection and this all turns out in a fairly light weight program that produces fairly accurate portrays of the movement of an object considering it’s a single camera running at a scaled down resolution.

If given more time I would definitely implement multiple objects able to be tracked by the system and I would certainly spend time on the UI because as the bulk of the development time was spent on technical resolutions to a few mathematical problems I feel I may have fallen short a little bit on A user friendly GUI, although I could explain this away by saying that this is designed to be more of a utility product for animators rather than a super user friendly consumer facing product.

I would also spend some time trying to find a way to make the calibration screen less intensive on the system when searching for a chessboard pattern as this sometimes produces a “shuttering” effect on the camera that can be frustrating and get in the way of interacting with the application.

I would also maybe implement a system in which a script for a selected animation platform can be generated by the application by inputting the desired time between key frames, the animation platform and the kind of object the user would like rendered.

After finishing the last testing of the overall application, I attempted to

In terms of project management, the agile approach taken whereby there were periods or code “sprints” followed by a period of validation and functionality testing. And then repeating this process for all of the different functions of the program worked fairly well, although in the future if I attempt another solo project in similar scope I would elect to spend more time refactoring my code much earlier on in the process as this took up a comparably large amount of time at the end of the project, in fact it took more time than one other aspect of the development process.

I were to do the project again I would definitely prioritize cleaner code from an earlier point as before the final refactoring there were a lot of fundamentally messy functions and variables and maybe I would think about a alternative way of structuring a lot of the code in the first place because as much as I like the object variables class I ca also see why it may be seen as a little clunky and over engineered in some aspects.

The languages used were python and kivy’s own styling language for the gui called .kv. The libraries that were used were cv2, numpy, kivy, and imutils. For the task at hand I feel these severed me well but I would admit that some of the issues I ran into on the android APK generation and deployment would’ve been all but dealt with if I just used a language more suited to android device such as java or c++ which both have opencv support, therefore I may slightly regret the use of python inn terms or that one aspect.

Although I did still a phone as a source for the camera by using droidcam it was definitely a bit of a work around making up for the lack of utility of python in this particular case.

Although the flexibility of python would definitely have me coming back for different projects with similar and larger scope.

Much was learned during the development of this project, but perhaps the thing that stands out the most is that when a sensible timeline is given, with a suitable development plan it is possible to create a fairly complex sounding application in an orderly and thoroughly enjoyable manner.

## **Functional requirements**

|  |  |
| --- | --- |
| **Major Requirements** | **Priority** |
| The system should be able to actively track the movement of an object on a screen | **1** |
| The system should be able to translate the values obtained during object tracking into accurate 3D coordinates | **1** |
| The system should be able to output the 3d coordinates of the object during a recording phase to a file | **1** |
| The file produced by the system should be able to be processed by a script within a 3d environment to accurately recreate the movement of the original object | **2** |
| The system should be able to calibrate a camera in order to get inherent camera values such as camera matrix and distortion coefficients | **2** |
| The system should be able to take inputs for the object tracking function from the user (hsv thresholds and object size) | **2** |

## 

## Literature Review

### Introduction

The barrier to entry of 3d animation has never been lower, with the introduction of free open source software such as blender, ZBrush, daz3D and others entering the market anyone can start animating in a 3D environment as long as they have access to a computer.

In the past, an average person wanting to animate the movement of an object in a 3D scene in a way that would look as they imagined in their mind was usually to iteratively keyframe the object in a trial and error basis until the animation looked as they desired. This was due to the fact that if an animator wanted to physically move an object within their scene in a way that they desired they would have to invest in some specialized software and equipment to achieve this goal.Such as after effect’s mocha (Helpx.adobe.com, 2019) Therefore it is crucial to bring down the barrier of entry for motion capture technology in order for more animators to be able to create in a more intuitive and organic way.

This literature review aims to firstly provide an insight into the current state of the 3d animation motion capture landscape and, secondly to explore the potential for more accessible technology to be introduced into the market.

### **Context**

There isn’t a huge amount of options for animators to choose from when it comes to cheap, accessible ways to use motion capture of objects as a source for natural looking movement of objects within their 2d environments. Those without access to reliable motion capture technology are restricted to simulations, archived capture data or some other form of motion capture.

There aren’t that many solutions out there right now for this particular use case, the closest low barrier to entry applications would be something like accelblend (Talk.maemo.org, 2019) or wear notch (Notch, 2019) would be products similar but they use either a wearable accelerometer o the motion of the phone to gather data, whereas if you were to use the information from the camera you would be able to eventually track multiple objects at once, therefore you could argue to use the camera has a grater use in the long run.

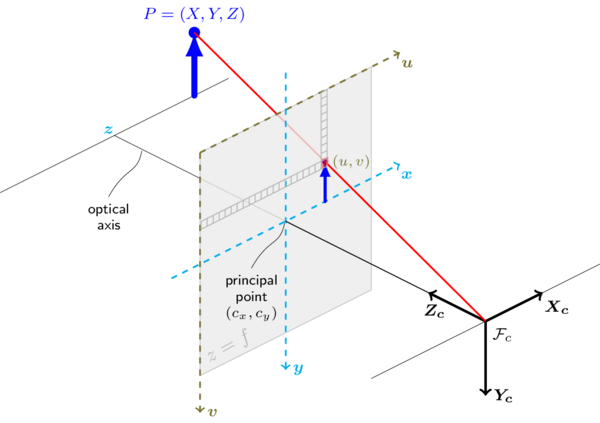
### A brief introduction to OpenCV

Opencv is an open source computer vision library originally written for C++ but has been translated into Python as well, this library contains a large variety of image manipulation and processing algorithms that allow for a wide variety of use cases. In this project for example it is used as image acquisition, processing and 3d point reprojection

### Key terms

#### 3D reconstruction

In order to take a 2d image point and output, 3d coordinates the (u,v) coordinate point must be unprotected. This is described in great depth in the opencv documentation. It describes “the so-called pinhole camera model, in this model a scene view is formed by projecting 3D points into the image plane using perspective transformation” This basically means that using this pinhole camera model to give a pixel point (u,v) within an image and inputting the camera matrix (which includes the focal length of the camera), the image parameters. It provides a line on a 3d axis, similar to the diagram below (provided by the opencv documentation) upon which you can input the Z value (the distance of the point from the camera) and it gives P, which is the [X, Y, Z] coordinates of the point in 3D space (Docs.opencv.org Camera calibration, 2019)

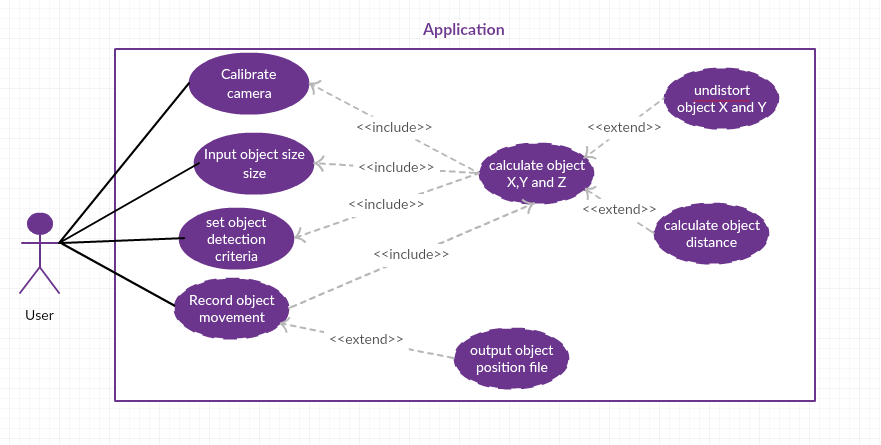


**Contouring and moments**

Image moments help you to calculate some features like the center of mass of the object and area of the object, this is extremely useful for object detection, as after a mask has been applied to an image with thresholds complementary to that of the target object the mask is just a black and white 2 channel image with a white area where the object is, so in order to in the area on the screen counter finding is used to first find all points surrounding the object, after this a circle of minimum radius can be applied round this found contour. (Docs.opencv.org, Contour features, 2019)

After this the moment of the counter can be used to find the centroid of the object, this can be used to approximate the center of the object.

## Use Case Diagram



Use case diagram

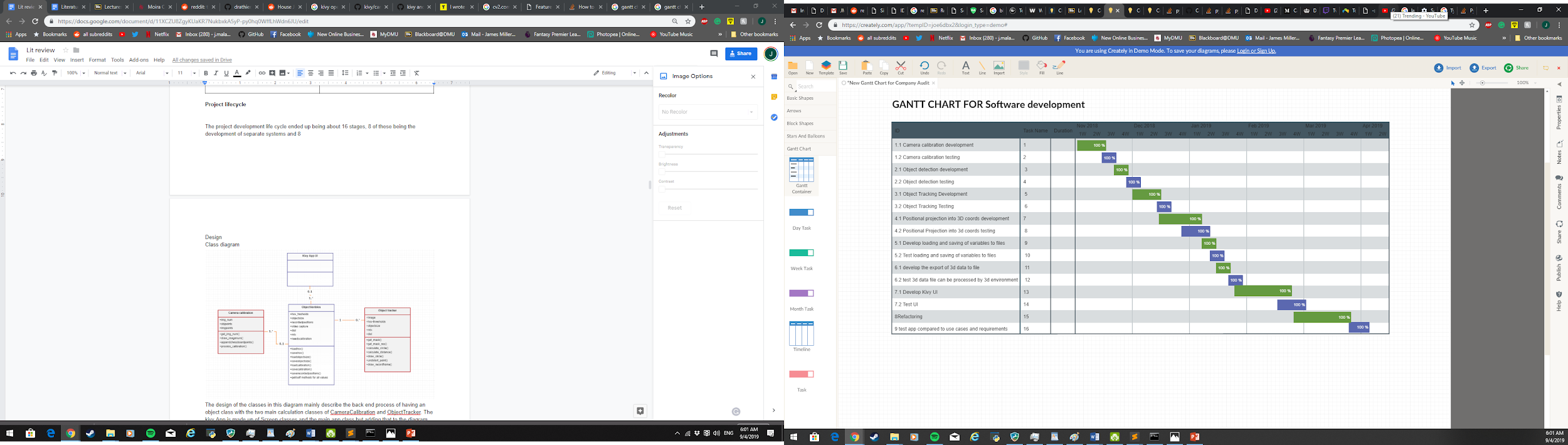
|  |  |
| --- | --- |
| Use case description | |
| System: 3d position recorder Name: Camera calibration | |
| Scenario:  A user calibrates a camera for use in 3d positional tracking | |
| Pre-condition:  The user has the camera calibration window open and there are no stored variables indicating a cameras Values (Camera matrix, Distortion coefficients) | |
| Postcondition:  Camera matrix and distortion coefficients are stored for later use within a JSON file | |
| Typical course of events | |
| User | Application |
| 1. The User selects the “begin camera calibration” button  3. User dismisses explanation popup  4. User presses “take photo” button when calibration pattern is in view of camera  4.4 user repeats taking photos of the chessboard in various positions  5 when the user decides enough photos have been taken they press the “process calibration” button | 2. A pop up appears telling the user how to properly calibrate the camera using the black-white chessboard pattern  4.1application checks image for chessboard pattern  4.2 if the chessboard pattern is detected in the image then the values of the chessboard are appended to two lists (objpoints and imgpoints)  4.3 image number value on the screen increment by 1  4.5 each photo’s chessboard values are appended to lists  5.1 the two lists are parsed through Opencv’s calibratecamera() function and the camera values are output  5.2 camera values are then saved to a JSON file so they can be accessed when the application is next used  5.3 the total error for the values are then calculated and shown to the user, it suggests recalibration if the error is above 2 |

|  |  |
| --- | --- |
| Use case description | |
| System: 3d position recorder Name: Object detection setup | |
| Scenario:  A user sets up the mask for object detection in order for the application to calculate where the object is in 3d space | |
| Pre-condition:  The user has completed camera calibration and has the mask calibration window open.  No HSV thresholds or ball size has been set for the object | |
| Postcondition:  Suitable object width and HSV thresholds have been set by the user and those values have been saved to a JSON file | |
| Typical course of events | |
| User | Application |
| 1. A user selects the “Change object width button” button  3.user inputs object width and presses ‘enter’ button  5. The user uses the hue, saturation and value sliders to adjust the mask until only the desired object is visible then closes  6 user then selects the “result” button to see if a suitable circle is being displayed over the object | 2. A pop up appears instructing the user to Enter object width in cm  4. Object width is saved to JSON file  7. When the user changes to the result screen the HSV thresholds are saved to JSON file, these thresholds can be overwritten anytime the user changes the values within the mask calibration screen |

|  |  |
| --- | --- |
| Use case description | |
| System: 3d position recorder Name: Object position recording | |
| Scenario:  A user records the movement of an object and the 3d coordinates are output to a JSON file that can be used in a 3d animation environment | |
| Pre-condition:  The user has completed camera calibration and has selected suitable object width and HSV thresholds in the mask calibration stage producing a consistent and steady circle around the desired object and a correct distance value is being displayed in the results screen, and the user is now on the “record ” screen | |
| Postcondition:  The user has recorded the movement of an object and the positions of the object in 3d space over time have been put into a list and output to a JSON file | |
| Typical course of events | |
| User | Application |
| 1. User presses” record” button  3.when the user has finished recording they press the “end record ” button  4. When a user has input a filename they press the “enter” button on the popup | 2. Each frame the x,y coordinates of the centre of the circle are fed into the Unproject() function taking into account the camera matrix, the distortion values and the distance of the object to the camera  2.1 the undistorted X,Y,Z values are then stored in a list  3.1 A pop up appears instructing the user to enter a filename for the recording  4.1 a popup appears confirming to the user that the 3d values have been saved to the file of their choice |

## Project lifecycle

The project development life cycle ended up being about 16 stages, 8 of these being the development of separate systems and the other 8 being various testing for each of these development sprints plus the final testing of the full application against original use cases and requirements.



This gannt chart displays the various stages of the development cycle for this project.

As you can see the general structure of the workload was to spend time developing an aspect of the codebase and as the development of each section was coming to a close a testing schedule of that particular aspect was undertaken.

Most code sprints and testing went smoothly with very few issues although to give an example of when an issue did crop up during testing it was during the testing of the 3d data being processed into the 3d environment the movement of the ball did not seem correct i.e they did not match the movements made when inputting the positional data using the camera.

After some time troubleshooting it turned out that the reprojection of the 2d (u,v) points was being handled as if there were two cameras in stereo mode, that is when two cameras are imaging the same object, this caused a lot of confusion but was quickly resolved when having a another look at the documentation and finding that stereo was in fact the default way reprojection of points was processed on cv2.

At the end of the lifecycle much time was spent refactoring the code in order to aid readability for any future programmer to be able to more easily discern what everything is doing, refactoring included changing the names of various method names to more completely describe what they are doing, adding a few comments where the logic for a certain piece of code is not immediately obvious and changing the class structrure of objectvalues as before a few variables that pertained to the object in question were stored within the kivy UI class structure meaning that in the future if this application was further developed it may not be able to scale to multiple object being tracked simultaneously.

## Testing

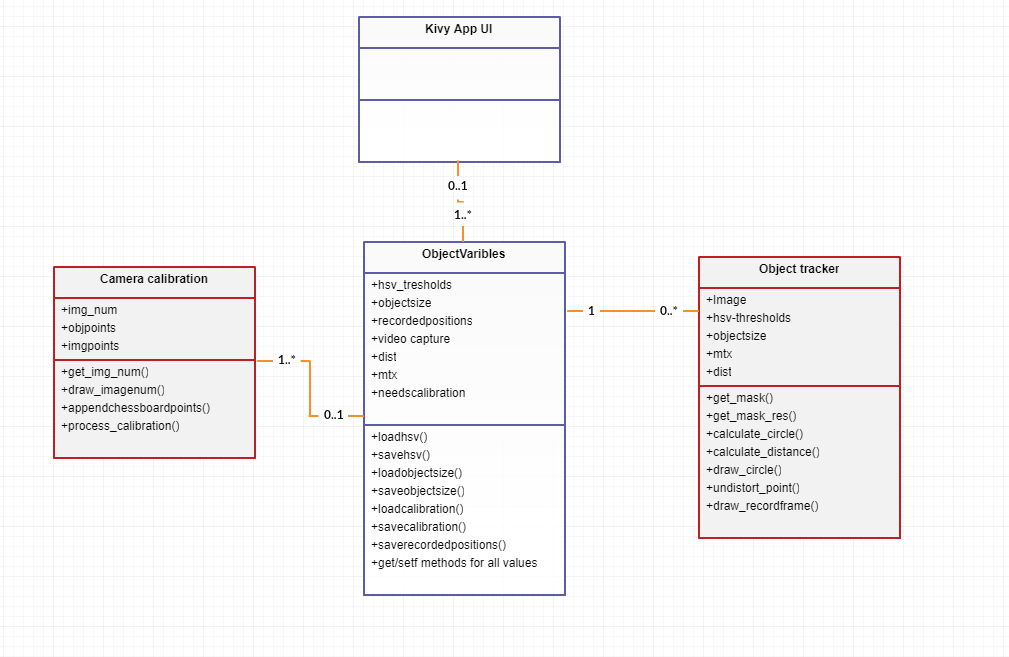
Although most testing during the development of the code was simply checking the output of the system in comparison to the movement of an object and just normal validation testing, the final application testing used the use cases as test cases.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case description | | | | |
| System: 3d position recorder Name: Camera calibration | | | | |
| Scenario:  A user calibrates a camera for use in 3d positional tracking | | | | |
| Pre-condition:  The user has the camera calibration window open and there are no stored variables indicating a cameras Values (Camera matrix, Distortion coefficients) | | | | |
| Postcondition:  Camera matrix and distortion coefficients are stored for later use within a JSON file | | | | |
| S,No | Action | input | Expected output | Actual output |
| 1 | The User selects the “begin camera calibration” button | “begin camera calibration” button selection | A pop up appears telling the user how to properly calibrate the camera using the black-white chessboard pattern | A pop up appears telling the user how to properly calibrate the camera using the black-white chessboard pattern |
| 3. | User dismisses explanation popup | Click outside pop up | Pop up disappears | Pop up disappears |
| 4 | User presses “take photo” button when calibration pattern is in view of camera | User presses “take photo” button when calibration pattern is in view of camera | image number value on the screen increment by 1 | image number value on the screen increment by 1 |
| 5 | when the user decides enough photos have been taken they press the “process calibration” button | User presses the “process calibration | A JSON file is created including calibration data  And a pop up appears displaying the total error, is expected to be less than 2. | A JSON file is created including calibration data  And a pop up apears saying that the total error is 1.56. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case description | | | | |
| System: 3d position recorder Name: Object detection setup | | | | |
| Scenario:  A user sets up the mask for object detection for the application to calculate where the object is in 3d space | | | | |
| Pre-condition:  The user has completed camera calibration and has the mask calibration window open.  No HSV thresholds or ball size has been set for the object | | | | |
| Postcondition:  Suitable object width and HSV thresholds have been set by the user and those values have been saved to a JSON file | | | | |
| S,No | Action | input | Expected output | Actual output |
| 1 | User presses selects the “change ball size ” button | “change ball size ” button is pressed | A pop up appears instructing the user to Enter object width in cm | A pop up appears instructing the user to Enter object width in cm |
| 3. | .user inputs object width and presses ‘enter’ button | “enter” pressed | Pop up disappears | Pop up disappears |
| 5 | The user uses the hue, saturation and value sliders to adjust the mask until only the desired object is visible then closes screen | adjusted the mask until only the desired object is visible then closes open result screen | Circle overlaying object is visible | Circle overlaying object is visible |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use case description | | | | |
| System: 3d position recorder Name: Object position recording | | | | |
| Scenario:  A user records the movement of an object and the 3d coordinates are output to a JSON file that can be used in a 3d animation environment | | | | |
| Pre-condition:  The user has completed camera calibration and has selected suitable object width and HSV thresholds in the mask calibration stage producing a consistent and steady circle around the desired object and a correct distance value is being displayed in the results screen, and the user is now on the “record ” screen | | | | |
| Postcondition:  The user has recorded the movement of an object and the positions of the object in 3d space over time have been put into a list and output to a JSON file | | | | |
| S,No | Action | input | Expected output | Actual output |
| 1 | User presses “record” button | “record” button  pressed | the undistorted X,Y,Z values are shown on screen | the undistorted X,Y,Z values are shown on screen, they changes visible the same distance that the object is being moved |
| 3. | when the user has finished recording they press the “end record ” button | “end record ” button pressed | Pop up appears asking for filename | Pop up appears asking for filename |
| 4 | When a user has input a filename they press the “enter” button on the popup | File name “abc” input | Pop up appears confirming that the 3d values have been saved to the file “abc.JSON” | Pop up appears confirming that the 3d values have been saved to the file “abc.JSON” |

## Design



Class diagram

The design of the classes in this diagram mainly describe the back end process of having an object class with the two main calculation classes of CameraCalibration and ObjectTracker. The kivy App is made up of Screen classes and the main app class but adding that to the diagram may complicate the idea of the application attempting to be put across.

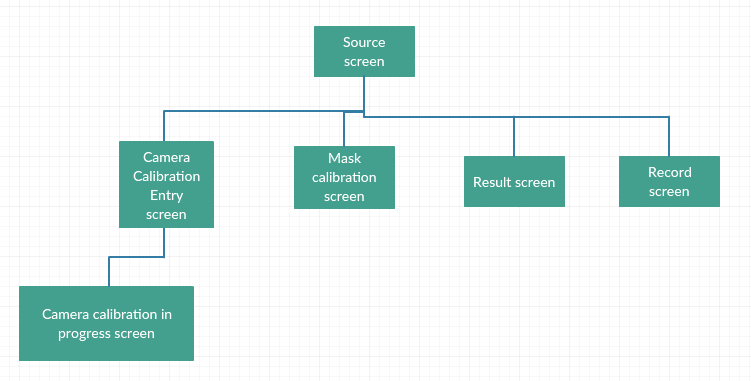
The way the object variables class has been set up allows the ability in the future, if development continues on this project, to perhaps add the ability to track multiple objects, these objects could have their own sizes, recorded positions and even capture camera if for example the application was expanded to a multi camera setup.

In the meantime, though the structure of these classes allows for simple get requests from the main kivy UI for different object variables, image calibration and position calculations the method calls.

### User Interface

The UI for this application runs off kivy, it is a cross-platform UI development tool that runs on python, this UI generator was chosen because of its functionality as a wrapper for the application in order to run on a variety of different platforms as this might allow the application to run on Android devices.

The overall structure of the UI is extremely simple, as shown in the following diagram.



There are five primary screens on the application, and each of these primary screens has direct access to all other primary screens through buttons when the application is first started the initial screen is that of the source screen.

The UI was written in kivy’s purpose built language .kv the code for this UI consisted of 8 classes (<SourceScreen>,<CalibrationScreen>,<ResultScreen>,<RecordScreen>,<CameraCalibrationScreen>,<CalibrationinprogressScreen>,<ballsizeinput> and <filenameinput>)

The first five of these are different screen of the application, the last two being custom pop ups for user input.

The layout of .kv is very similar to other style sheet languages such as CSS, here is the source camera screen's code for reference, most other screens follow this same layout:

<SourceScreen>

on\_enter: root.dostart()

#on\_pre\_leave: root.doexit()

BoxLayout:

orientation: 'vertical'

KivyCamera:

id: sourcecam

BoxLayout:

id: sourcebuttonbox

size\_hint\_y : None

Button:

text: "camera calibration"

on\_press: root.manager.current = 'Camera Calibration Screen'

Button:

text: "Source"

Button:

text: "mask calibration"

on\_press:root.manager.current = 'Calibration Screen'

Button:

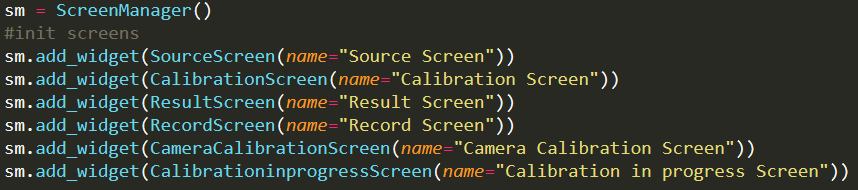
text: "Result"

on\_press: root.manager.current = 'Result Screen'

Button:

text: "Record"

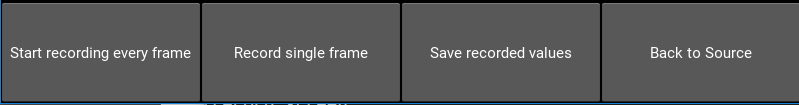
on\_press:root.manager.current = 'Record Screen'

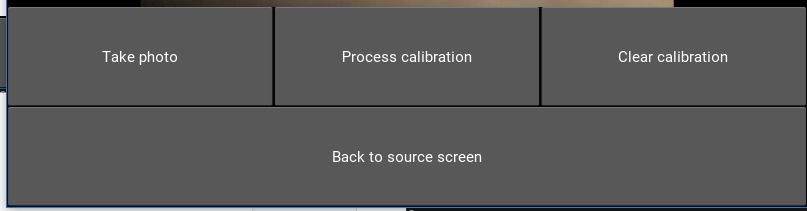
As you can see a box layout was used in a vertical configuration that included the camera and another box layout in the default horizontal configuration that was made of buttons for the different screens. As you can see on the action of on\_press the value of managerger.current on from the root is set to the name of the screen relating to the button, this is done in this way because the application uses a screen manager provided by kivy, the layout of this screen manager is as follows

### Navigation

Within the screen, primary navigation is provided on the bottom of the screen

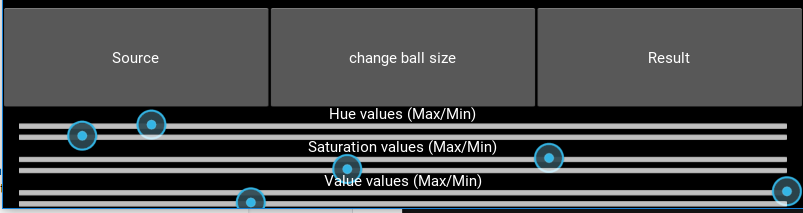
This navigation bar is consistent across most of the application’s screens other than the calibration in progress screen and the record screen in which the buttons are repurposed somewhat:

Record screen bar



Camera Calibration in progress screen buttons

The only screen with slightly different user interface option is the maskcalibation screen in which there are sliders to change the HSV threshold values for the mask in a smooth way:



Mask calibration navigation bar and HSV sliders

Due to the nature of mask calibration using HSV thresholds, it takes a bit of trial and error to find suitable values and therefore if the application were to use some other form of value input such as text input the user might find it very unpractical for the purpose.

## Noteworthy functions

The main task of this whole application is to approximate the position of a given object within an image within 3-dimensional space(X, Y and Z). To do this the application uses 3D

### Camera Calibration

Camera calibration is at the heart of the whole program, it is mainly needed in order to find the camera matrix and distortion coefficient and of the camera being used. The camera matrix contains the focal length of the camera, this is used for distance approximation using triangle similarity as will be described in more detail in the distance approximation section.

The distortion coefficients are used in conjunction with the focal length and the approximated distance the reproject the u,v pixel position of the object into a calculated 3d coordinate using the pinhole model described in the literature review and later run over in the Undistortion of coordinates section.

Camera calibration has its own separate class which upon its initialization creates three important variables:

def \_\_init\_\_(self):

self.img\_num = 0

# Arrays to store object points and image points from all the images.

self.objpoints = [] # 3d point in real world space

self.imgpoints = [] # 2d points in image plane.

Img\_num keeps track of how many calibration images have been taken, this is then fed back into the the display image using the draw\_imagenum() function this utilizes the cv2.puttext() function to draw the image number onto the output image so the user can see how many images have been taken.

The second two variables initialized are the lists objpoints and imgpoints. Imgpoints will keep track of the (u,v) positions of the corners of each chess board square. Object point are then the calculated 3d positions of the same points, this time taking the size of the chessboard and it’s angle to the camera into account.

The main three methods in the camera calibration class. The first being draw\_imagenum which as the name suggests draws the current image num onto an inputted image, this is mainly for ease of reading for the user. There is also a popup on the kivy UI side that notifies the user that it is recommended to take at least 10 photos of the calibration pattern in order to get fairly good calibration values, therefore this displayed number allows users to keep track of this.

The second important method of the camera calibration class is the append chessboardpoits method. This is the method that actually finds the chessboard within the image and appends both the objpoints (the u,v coordinates of the chessboard square corners) and the img points (the calculated X,Y of the corners) to their respective lists:

def appendchessboardpoints(self,image):

gray = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

# Find the chess board corners

ret, corners = cv2.findChessboardCorners(gray, (9,7),flags=cv2.CALIB\_CB\_ADAPTIVE\_THRESH | cv2.CALIB\_CB\_NORMALIZE\_IMAGE)

# If found, add object points, image points (after refining them)

if ret == True:

self.objpoints.append(self.objp)

self.imgpoints.append(corners)

corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),self.criteria)

# Draw and display the corners

chessboardrawn = cv2.drawChessboardCorners(image, (9,7), corners2,ret)

self.img\_num = self.img\_num + 1

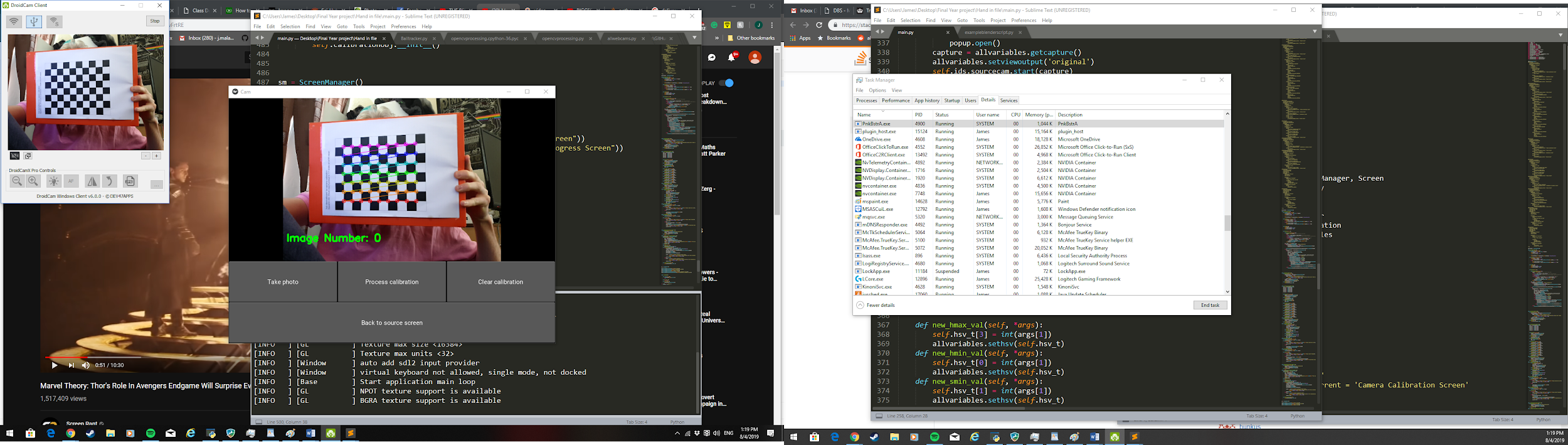
These photos are first processed into a grayscale image in order for the program to more easily find the chessboard.



Calibration greyscale.

Then to check for the chessboard in the image the cv2 method findchessboardcornrs() is called, with both the adaptive threshold and image normalization flags in order to bring up the contrast of the image enabling for easier detection of the pattern.

If found the objpoints are appended to their list. As are the img points appened to it’s list corner points. After this, in order for the user to get feedback as to whether the processed worked or not the chessboard corners are then drawn onto the image.



### Processing calibration:

After a suitable amount of image’s chess board corner positions have been collected in both the imgpoints and objpoints lists the user is able to select the “process calibration button”. This runs the process\_calibration() method.

def process\_calibration(self,image):

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(self.objpoints, self.imgpoints)

tot\_error = 0

for i in range(len(self.objpoints)):

imgpoints2, \_ = cv2.projectPoints(self.objpoints[i], rvecs[i], tvecs[i], mtx, dist)

error = cv2.norm(self.imgpoints[i],imgpoints2, cv2.NORM\_L2)/len(imgpoints2)

tot\_error += error

return mtx, dist, tot\_error

The calculation of the camera matrix and the distortion coefficient is handled by the calibratecamera() method.

In order to check the accuracy of the camera matrix and distortion coefficients, a total error is calculated. This is done using the calculated values and the objpoints (the original u,v positions of the corners) and inputting them into the cv2.projectpoints() method, this method takes the (u,v) values and calculates where there should be in 3d space when accounting for the camera values, ideally this should output values equal to that of the imgpoints (the calculated 3d position of the chessboard pattern corners), the closer these two values are, the lower the calculated error is.

After the total error is calculated a popup is shown to the user displaying the error and advising them that it is recommended to have an error below two in order to get reliable results in their 3D motion capture.

### Distance approximation

In order to calculate the distance of the object from the camera, thereby giving the application the Z coordinate of the object, the application utilizes triangle similarity, the equation to find the distance of a given object from a camera is:

D = (W x F) / P

Where D is the distance to the camera, W is the known width of the object, F is the focal length of the camera and P is the perceived width, in pixels, of the object in the image. Within the code the equation is just a fairly simple function:

def calculate\_distance(width\_of\_object, focalLength, detectedwidth):

return ((width\_of\_object \* focalLength) / detectedwidth)

In order for the application to take advantage of this function, the focal length, perceived width and actual width of the object must be known. The input of the width of the object has been left to the user, in order to find the focal length of the camera calibration must be undertaken for every different camera used by the application.

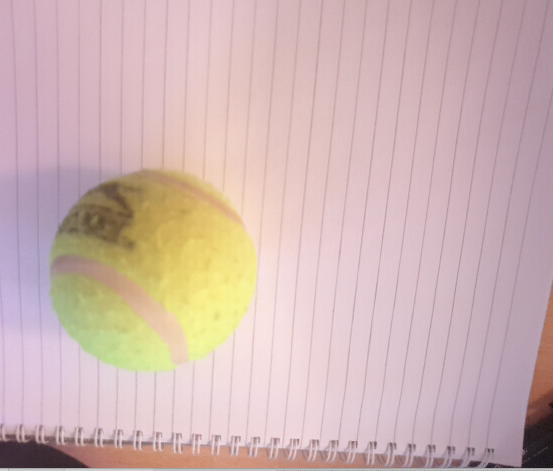
### Calibration

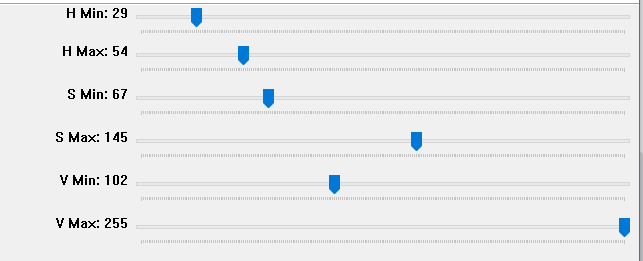
Camera calibration is used to gather the focal length and distortion coefficients of any given camera used by the application. This calibration only has to be undertaken once as these values are saved by the application.

In order to calibrate the camera, the application uses OpenCV's calib3d module, this module calculates a distortion matrix and a camera matrix (the camera matrix includes the focal length), it does this by taking snapshots of a given pattern, in the case of this application it is a black-white chessboard pattern, when many snapshots of this pattern in a variety of positions are processed a distortion matrix and a camera matrix are determined, this values are then used in later calculation in the application such as distance approximation and undistortion of x,y points in an image into usable points for 3d reprojection.

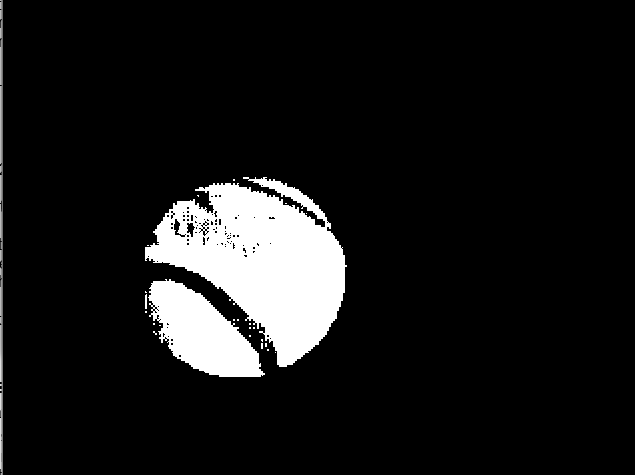
Object detection

Object detection is needed to approximate the perceived width of the object and its distorted x,y values that will later be undistorted into using the distortion matrix collected earlier using the camera calibration.

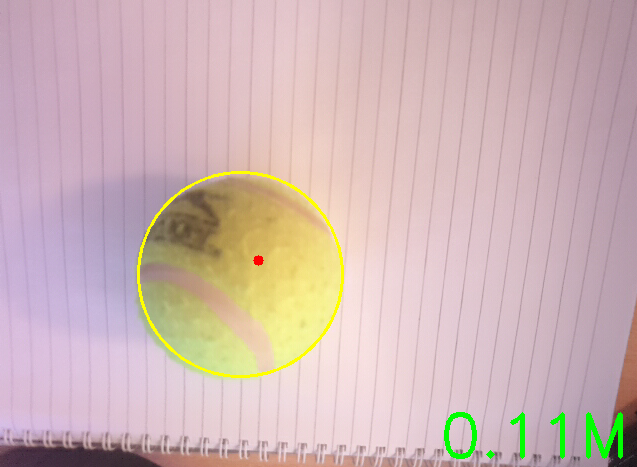
The goal of object detection in this application is to have a circle of known size and position being overlaidexactly where the object is on the image frame, this is done by creating a “mask” that excludes all of the source image other than the given object, this is achievedby having an object of a different colour from the rest of the scene, converting the source image from RBG (red, green, blue) into HSV (Hue, Saturation, Value) and setting thresholds of HSV within which only the object in question matches. This can be seen in the example images given, the first is the source image, the second is the mask, and the third is the HSV settings for that particular object.

 Source image

HSV Thresholds

After a suitable mask image is being produced it is then time to approximate a circle around the area of the mask where the object is, this is done by first using the cv2.findcontours() function which is used to finds curve joining all the continuous points (along the boundary), having same color or intensity, as the mask is a binary mask it easily finds the object. Followingthis, the application finds the contour with the largest area, as sometimes there are false positives within the image and then uses cv2.minEnclosingCircle() to get the x y coordinates and the radius of a circle enclosing the largest contour in the mask.

mask

A circle is then drawn onto the source image matching that of the cv2.EnclosingCircle() in order for the user to see if the process is working correctly. Below shows the resulting image.

Result

### Undistortion of coordinates.

In order to get coordinates of the object in 3d space given the distance from the camera, perceived x and y positions in the image. The application has to “unproject” the coordinates of using the unproject() function, which takes as its input; the perceived x,y coordinates of the object in the image, the camera matrix and the distortion matrix and the Distance of the object from the camera.

def undistort\_point(self):

x,y,radius,center = self.calculate\_circle()

z = self.calculate\_distance()

assert z != None

f\_x = self.mtx[0, 0]

f\_y = self.mtx[1, 1]

c\_x = self.mtx[0, 2]

c\_y = self.mtx[1, 2]

#points = [x,y]

points = np.array([x, y]).reshape(1,2)

points = np.ascontiguousarray(points[:,:2]).reshape((1,1,2))

# Step 1: Undistort points

if len(points) > 0:

points\_undistorted = cv2.undistortPoints(points, self.mtx, self.dist, P = self.mtx)

points\_undistorted = np.squeeze(points\_undistorted, axis=1)

result = []

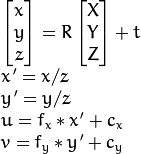
x = (points\_undistorted[0, 0] - c\_x) / f\_x \* z

y = (points\_undistorted[0, 1] - c\_y) / f\_y \* z

result = ([x, y, z])

return result

As you can see, the function first breaks down the camera matrix into the focal length of the camera in both the x and y directions f\_x, f\_y (these are usually identical) and then gets the principal point of the image, that is usually at the image centre. It then un distorts the inputted x,y coordinates using the cv2.undistortPoints() method, giving undistorted points equal to when z = 0, the function then solves x and y in the following equation given by the opencv documentation.



Resulting in:

X = (u - c\_x)/f\_x \* z

Y = (v - c\_y)/f\_y \* y

These points are put into a list called results and returned as such.

## Project Issues

### Kivy and opencv camera

When first implementing the kivy UI into the project there was a lot of difficulties displaying the opencv processed images onto the kivy canvas, after much reading of the documentation and github issues threads it was found that support for the opencv camera has been a point of issue for the kivy development team, and the support for the particular version of opencv being run on this program often has issues.

Due to these issues a new image class had to be created to handle opencv images, the main point of this class was to mimic the camera class of kivy. It first initializes a clock variable that intervals once every specified time frame i.e the fps:

def start(self, capture, fps=30):

self.capture = capture

if self.clock\_variable is None:

self.clock\_variable = Clock.schedule\_interval(self.update, 1.0 / fps)

else:

self.clock\_variable.cancel()

self.clock\_variable = Clock.schedule\_interval(self.update, 1.0 / fps)

When this clock variable is called, it calls the update() method within the class, this update method includes all of the image manipulation of the program, by calling different methods with the objecttTracker class, the objectvariable class and the camera calibration class.

After all of the image manipulations have completed the method displayimge() is called:

def displayimage(image):

texture = self.texture #get kivy texture from class

try:

w, h = image.shape[1], image.shape[0] #get image parameters

except AttributeError: # attribute error catch

w, h=None,None

if w != None: #resize if texture is empty or not matching image size

if not texture or texture.width != w or texture.height != h:

self.texture = texture = Texture.create(size=(w, h))

texture.flip\_vertical()

texture.blit\_buffer(image.tobytes(), colorfmt='bgr')

self.canvas.ask\_update()

The display image method is essentially the solution to the issue, it first shapes the kivy texture to the size of the incoming image, then it takes the image, converts it into bytes and adds it to the texture using the blit\_buffer() method that implements the python buffer interface, e.g. array.array, bytearray, numpy arrays etc. and enables the texture to read it, in this case in the format bgr (blue, green, red).

After this the canvas is then updated with the texture.

### Recording of object

When first using the record feature it was difficult to get a smooth recording of values as initially thought, it was possible but required a near perfect mask calibration stage of setup, this meant that many recordings of an object’s positions had an odd “jiggly” effect to them.

It is thought that this effect is caused by a couple of factors. The first being the lighting of the environment in which the recording is taking place, if for example, the room is too light or too dark it can be hard for the system to consistently accurately approximate the position of the object on every frame, therefore, a slight “jiggly” motion occurs

Another factor could be the resolution of the processing image, although the camera being used in testing was that of a Samsung Galaxy S9 takes photos at 1080, the image was being downsized in the processing of the image in order to aid the speed of the program.

And while this could maybe be counteracted by adding some sort of “smoothing” feature to read the positions list and adjusting values in a way to counteract this effect it was felt that this feature may take up far too much time in the development of the application, perhaps if this application is worked on at a later time such a feature could be added.

Another way to help stop the “jiggling” of the object positions would be to increase the effectiveness of the mask and object detection in some way, maybe adding a similar “smoothing” feature to the initial position approximation results or by adding additional values to the mask calibration such as image smoothing or some form of blur.

The solution that was come to was to add a feature so that a “snapshot” could be taken of certain frames rather than recording every frame, this would allow those who could not get a perfect object tracking for one reason or another the ability to stack up a series of positions in a way similar to stop motion animation and then save the results just as before.

This snapshot technique proved quite effective and in hindsight should've perhaps been a planned feature from the start as now new effects could be achieved compared to before. Because even though this technique loses the “real-time” effect of the original recording method now an animator could plan precisely where an object will be on every frame.

## Open Issues

### Android APK

One of the main reasons for selecting kivy as the UI for this project was for its cross-platform capability, this promised the ability for this application to be ported onto different platforms with little to no change in the code of the program. As the initial project contract implies, this application was intended to run on a smartphone and run the image processing of the camera on the phone and then later have some functionality by which the 3d coordinate files could be exported to other devices.

Due to this desire to have the application run on a mobile device once a simple version of the program was completed within the kivy UI. an android APK was attempted to be made. This was done by creating a virtual machine on Oracle’s VM Virtualbox, this virtual machine was running on xubuntu and had all necessary dependencies such as up to date Android SDKs and NDKs installed.

In order for a python application running on kivy to be made into a deployable android APK kivy documentation suggests using buildozer to streamline the APK building process. Although this seems like a fairly simple process on the surface many difficulties were run into, such as certain SDK builds were not compatible with python3.7 (the python version that was being used) and certain Android builds were not compatible with kivy, so after much reading of the documentation and browsing many bug reports on these exact issues A bulldozer run did eventually succeed in creating a deployable android APK but after the APK was deployed onto the phone it would crash whilst starting up. After much debugging and many hours spent reading the log cast generated from the phone whilst the crash occurred no obvious cause was found.

Sadly, the Android version of this application did not get completed due to this reason if a solution is found the GitHub repository will be updated. To make use of android smartphone cameras during the development of this application “DroidCam” was used in order to use the phone camera in a similar way to a built-in webcam

## References / Bibliography

Notch. (2019). *3D Motion Capture for Smartphone*. [online] Available at: https://wearnotch.com/ [Accessed 9 Apr. 2019].

blender.org. (2019). *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. [online] Available at: https://www.blender.org/ [Accessed 9 Apr. 2019].

Docs.opencv.org. (2019). *Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation*. [online] Available at: https://docs.opencv.org/2.4/modules/calib3d/doc/camera\_calibration\_and\_3d\_reconstruction.html [Accessed 9 Apr. 2019].

Docs.opencv.org. (2019). *Camera calibration With OpenCV — OpenCV 2.4.13.7 documentation*. [online] Available at: https://docs.opencv.org/2.4.13.7/doc/tutorials/calib3d/camera\_calibration/camera\_calibration.html [Accessed 9 Apr. 2019].

Docs.opencv.org. (2019). *Contour Features — OpenCV 3.0.0-dev documentation*. [online] Available at: https://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_imgproc/py\_contours/py\_contour\_features/py\_contour\_features.html [Accessed 9 Apr. 2019].

Daz3d.com. (2019). *DAZ 3D | 3D Models and 3D Software by Daz 3D*. [online] Available at: https://www.daz3d.com/ [Accessed 9 Apr. 2019].

Pixologic.com. (2019). *Pixologic : ZBrush 2019 Features*. [online] Available at: http://pixologic.com/features/about-zbrush.php [Accessed 9 Apr. 2019].

Reporter, C. (2019). *Using an Android Phone as a Virtual Camera - BlenderNation*. [online]

BlenderNation. Available at: https://www.blendernation.com/2012/08/27/using-an-android-phone-as-a-virtual-camera/ [Accessed 9 Apr. 2019].

Helpx.adobe.com. (2019). *Tracking and stabilization motion workflows in After Effects*. [online] Available at: https://helpx.adobe.com/uk/after-effects/using/tracking-stabilizing-motion-cs5.html [Accessed 9 Apr. 2019].

Talk.maemo.org. (2019). *[Alpha] accelBlend - controlling Blender with accelerometer data - maemo.org - Talk*. [online] Available at: https://talk.maemo.org/showthread.php?t=76991 [Accessed 9 Apr. 2019].

Appendix

Github of software: <https://github.com/JamesMMiller/opencvapp>

Chessbaord calibration pattern <https://www.mrpt.org/downloads/camera-calibration-checker-board_9x7.pdf>

Project contract: