MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CAM-STYLUS

Minor Project Report
Semester VI

Submitted by:

Shashank Singh	181112060
Ankit Kumar	181112086
Vikas Sao	181112051
Kanishk Singh Pujari	181112076

Under the Guidance of
Dr. Mansi Gyanchandani

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Session: 2020-2021

MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project report carried out on “Cam-Stylus ” by the 3'd year students:

Shashank Singh	181112060
Ankit Kumar	181112086
Vikas Sao	181112051
Kanishk Singh Pujari	181112076

Have successfully completed their project in partial fulfilment of their Degree in Bachelor of Technology in Computer Science and Engineering.

Dr . Mansi Gyanchandani
(Minor Project mentor)

DECLARATION

We, hereby declare that the following report which is being presented in the Minor Project Documentation Entitled as “ Cam-Stylus ” is an authentic documentation of our own original work under the guidance of Dr . Mansi Gyandchandani , Associate Professor, Department Of Computer Science and Engineering, MANIT Bhopal. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization. Any contribution made to the research by others, with whom we have worked at Maulana Azad National Institute of Technology, Bhopal or elsewhere, is explicitly acknowledged in the report.

Shashank Singh	181112060
Ankit Kumar	181112086
Vikas Sao	181112051
Kanishk Singh Pujari	181112076

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide and coordinator Dr . Mansi Gyandchandani, for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully.

It is imperative for us to mention the fact that the report of minor project could not have been accomplished without the periodic suggestions and advice of our project guide Dr . Mansi Gyandchandani and project coordinators Dr. Dhirendra Pratap Singh and Dr. Jaytrilok Choudhary.

We are also grateful to our respected director Dr. N. S. Raghuwanshi for permitting us to utilize all the necessary facilities of the college.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

ABSTRACT

Exploring the possibility of computer vision is both interesting from a scientific point of view and something that could be beneficial to virtual graphics industry .
We will be using the computer vision techniques of **OpenCV** to build this project. The preferred language is Python due to its exhaustive libraries and easy to use syntax but understanding the basics it can be implemented in any OpenCV supported language.

Here Color Detection and tracking are used in order to achieve the objective. The color marker is detected and a mask is produced. It includes the further steps of morphological operations on the mask produced which are Erosion and Dilation. Erosion reduces the impurities present in the mask and dilation further restores the eroded main mask.

To perform video tracking, an algorithm analyzes sequential video frames and outputs the movement of targets between the frames. There are a variety of algorithms, each having strengths and weaknesses. Considering the intended use is important when choosing which algorithm to use. There are two major components of a visual tracking system: target representation and localization, as well as filtering and data association.

Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing

Keywords: **OpenCV , DIP , Color Detection , Contours etc .**

TABLE OF CONTENTS

1. Introduction

Computer vision is a huge part of the data science/AI domain. Sometimes, computer vision engineers have to deal with videos. Here, we aim to shed light on video processing — using **Python**, of course.

This might be obvious for some, but nevertheless, video streaming is not a continuous process, but a discrete one.

That means, each time we deal with videos, we are actually dealing with the sequence of frames themselves. Each frame is just an image, which might be represented as an *m x n* array of pixels, where (*m,n*) is picture size. Each pixel might be represented as color intensity, depending on which color model we are using (gray-scale, RGB, or even multispectrum).

Often, we have to capture live stream with a camera. OpenCV provides a very simple interface to do this. Let's capture a video from the camera (I am using the built-in webcam on my laptop), convert it into grayscale video and display it. Just a simple task to get started.

To capture a video, you need to create a **VideoCapture** object. Its argument can be either the device index or the name of a video file. A device index is just the number to specify which camera. Normally one camera will be connected (as in my case). So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on. After that, you can capture frame-by-frame. But at the end, don't forget to release the capture.

2. Literature review and Survey

Due to object detection's close relationship with video analysis and image understanding, it has attracted much research attention in recent years. Traditional object detection methods are built on handcrafted features and shallow trainable architectures. Their performance easily stagnates by constructing complex ensembles which combine multiple low-level image features with high-level context from object detectors and scene classifiers. With the rapid development in deep learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures. As distinct specific detection tasks exhibit different characteristics, we also briefly survey several specific tasks, including salient object detection, face detection and pedestrian detection. Experimental analyses are also provided to compare various methods and draw some meaningful conclusions. Finally, several promising directions and tasks are provided to serve as guidelines for future work in both object detection and relevant neural network based learning systems.

Bhumika Gupta (2017) et a;. proposed object detection is a well-known computer technology connected with computer vision and image processing that focuses on detecting objects or its instances of a certain class (such as humans, flowers, animals) in digital images and videos. There are various applications of object detection that have been well researched including face detection, character recognition, and vehicle calculator. Object detection can be used for various purposes including retrieval and surveillance. In this study, various basic concepts used in object detection while making use of OpenCV library of python 2.7, improving the efficiency and accuracy of object detection are presented.

Kartik Umesh Sharma (2017) et al, proposed an object detection system finds objects of the real world present either in a digital image or a video, where the object can belong to any class of objects namely humans, cars, etc. In order to detect an object in an image or a video the system needs to have a few components in order to complete the task of detecting an object, they are a model database, a feature detector, a hypothesizer and a hypothesizer verifier. This paper presents a review of the various techniques that are used to detect an object, localize an object, categorise an object, extract features, appearance information, and many more, in images and videos. The comments are drawn based on the studied literature and key issues are also identified relevant to the object detection. Information about the source codes and online datasets is provided to facilitate the new researcher in object detection area. An idea about the possible solution for the multi class object detection is also presented. This paper is suitable for the researchers who are the beginners in this domain.

3. Proposed Work

Dataset:

Colors are made up of 3 primary colors; red, green, and blue. In computers, we define each color value within a range of 0 to 255. So, in how many ways we can define a color? The answer is 256*256*256 = 16,581,375. There are approximately 16.5 million different ways to represent a color. In our dataset, we need to map each color's values with their corresponding names. But don't worry, we don't need to map all the values. We will be using a dataset that contains RGB values with their corresponding names. The CSV file for our dataset has been taken from this link:

Extracting data from CSV files:

The pandas library is very useful when we need to perform various operations on data files like CSV. **pd.read_csv()** reads the CSV file and loads it into the pandas DataFrame. We have assigned each column with a name for easy accessing.

Image Features:

Image feature is a simple image pattern, based on which we can describe what we see on the image. For example cat eye will be a feature on a image of a cat. The main role of features in computer vision(and not only) is to transform visual information into the vector space. This give us possibility to perform mathematical operations on them, for example finding similar vector(which lead us to similar image or object on the image)

```python
class Matcher(object):

    def __init__(self, pickled_db_path="features.pck"):
        with open(pickled_db_path) as fp:
            self.data = pickle.load(fp)
        self.names = []
        self.matrix = []
        for k, v in self.data.iteritems():
            self.names.append(k)
            self.matrix.append(v)
        self.matrix = np.array(self.matrix)
        self.names = np.array(self.names)

    def cos_cdist(self, vector):
        # getting cosine distance between search image and images database
        v = vector.reshape(1, -1)
        return scipy.spatial.distance.cdist(self.matrix, v, 'cosine').reshape(-1)

    def match(self, image_path, topn=5):
        features = extract_features(image_path)
        img_distances = self.cos_cdist(features)
        # getting top 5 records
        nearest_ids = np.argsort(img_distances)[:topn].tolist()
        nearest_img_paths = self.names[nearest_ids].tolist()

        return nearest_img_paths, img_distances[nearest_ids].tolist()
```

Drawing running images from video OpenCV:

- Use cv2.VideoCapture() to get a video capture object for the camera.
- Set up an infinite while loop and use the read() method to read the frames using the above created object.
- Use cv2.imshow() method to show the frames in the video.
- Breaks the loop when the user clicks a specific key.

```python
# import the opencv library
import cv2


# define a video capture object
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    # by frame
    ret, frame = vid.read()

    # Display the resulting frame
    cv2.imshow('frame', frame)

    # the 'q' button is set as the
    # quitting button you may use any
    # desired button of your choice
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# After the loop release the cap object
vid.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

Contour Matching:

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.
- See, there are three arguments in **cv2.findContours()** function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the image, contours and hierarchy. `contours` is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

```
import numpy as np
import cv2

im = cv2.imread('test.jpg')
imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(imgray,127,255,0)
image, contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

4. Methodology

How Air Canvas type of Computer Vision Projects Work ?

Here, We will see the working of this computer vision project in four major points :

1. Understanding the HSV (Hue, Saturation, Value) color space for **Color Tracking.** And tracking the small colored object at finger tip.
2. Detecting the Position of Colored object at finger top and forming a circle over it. That is **Contour Detection.**
3. Tracking the fingertip and drawing points at each position for air canvas effect. That is **Frame Processing.**
4. Fixing the Minor Details of the code to function the program smoothly. **Algorithmic Optimization.**

Colour Tracking of Object at fingertip.

Firts of all, The incoming image from the webcam is to be converted to the HSV colour space for **detecting the colored object at the tip of finger.** The below code snippet converts the incoming image to the HSV space, which is **very suitable and perfect color space for Color tracking.**

Now, We will make the Trackbars to arrange the HSV values to the required range of color of the colored object that we have placed at our finger. The various HUE and other ranges of different colors can be seen here.

Now, When the trackbars are setup, we will get the realtime value from the trackbars and create range. This range is a numpy structure which is used to be passed in the function cv2.inrange( ). This function returns the Mask on the colored object. This Mask is a black and white image with white pixels at the position of the desired color.

Contour Detection of the Mask of Color Object

Now, After detecting the Mask in Air Canvas, Now is the time to locate its center position for drawing the Line. Here, In the below Snippet of Code, We are performing some morphological operations on the Mask, to make it free of impurities and to detect contour easily.

Drawing the Line using the position of Contour

Now Comes the real logic behind this Computer Vision project, We will form a python deque (A data Structure). The deque will store the position of the contour on each successive frame and we will use these stored points to make a line using OpenCV drawing functions. Firstly Make Four deques, for four distinct colors of the project :

Now, we will use the position of the contour to make decision, if we want to click on a button or we want to draw on the sheet. We have arranged some of the buttons on the top of Canvas, if the pointer comes into their area, we will trigger their method. We have four buttons on the canvas, drawn using OpenCV.

- Clear : Which clears the screen by emptying the deques.
- Red : Changes the marker to red color using color array.
- Green : Changes the marker to Green color using color array.
- Yellow : Changes the marker to Yellow color using color array.
- Blue : Changes the marker to Blue color using color array.

Also, to avoid drawing when contour is not present, We will Put a else condition which will capture that instant.

Drawing the points

Now we will draw all the points on the positions stored in the deques, with respective colour.

Features of the Air canvas

1. Can track any specific colored pointer .
2. User can draw in four different colors and even change them without any hussle.
3. Able to rub the board with a single location at the top of the screen.
4. No need to touch the computer once the program is run.

5. Tools and technologies used

Software requirement

The various tools and technologies to be used are as follows:

I) Python Libraries to implement Machine Learning Models -

• **Pandas** - pandas is a software library written for the Python programming language for data manipulation and analysis.

• **NumPy** - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

• **Scikit learn** - Scikit is an open source Python library that implements a range of machine learning, pre-processing, cross-validation and visualization algorithms using a unified interface.

• **OpenCV -** OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

• To build the machine learning models we used The **Jupyter Notebook**. **Jupyter Notebook** is an open-source web application that can be used to implement statistical modelling, data visualization, machine learning etc.

• **Collections -** The collection Module in Python provides different types of containers. A Container is an object that is used to store different objects and provide a way to access the contained objects and iterate over them. Some of the built-in containers are Tuple, List, Dictionary, etc. In this article, we will discuss the different containers provided by the collections module.

• **Matplotlib - Matplotlib** is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Hardware requirement

Working Computer system

Working WebCam System

Processor Requirement: Intel I3 core and above.

Memory Requirement: 4GB and above.

6. Implementation & Code

Code Snippets:

```
In [57]: import time
         import numpy as np
         import pandas as pd
         import seaborn as sns
         import random
         import math
         import sys
         import copy
         import os
```

```
In [58]: import matplotlib.pyplot as plt
         %matplotlib inline
```

```
In [59]: from sklearn.preprocessing import LabelEncoder
         from sklearn.model_selection import train_test_split
```

```
In [60]: import warnings
         warnings.filterwarnings('ignore')
```

```
In [61]: from zipfile import ZipFile
         from collections import Counter
```

Setting the Video Capture through webcam

```
In [64]: cap = cv2.VideoCapture(0)
         # cap.set(3,500)
         # cap.set(4,500)
         IMAGE_SHAPE = (480,640)
```

```
In [65]:
         color_index = 0
         colors=[(255,0,0) ,(0,255,0) ,(0,0,255) ,(0,255,255)]
         kernel = np.ones((5,5),np.uint8)
         print(kernel)

         [[1 1 1 1 1]
          [1 1 1 1 1]
          [1 1 1 1 1]
          [1 1 1 1 1]
          [1 1 1 1 1]]
```

```
In [66]: bpoints = [deque(maxlen=1024)]
         gpoints = [deque(maxlen=1024)]
         rpoints = [deque(maxlen=1024)]
         ypoints = [deque(maxlen=1024)]
         r_index = 0
         b_index  =0
         g_index = 0
         y_index  =0
```

Initializing the canvas

In [67]:

```
# Here is code for Canvas setup
paintWindow = np.zeros((480,640,3)) + 255
paintWindow = cv2.rectangle(paintWindow, (590,0), (639,50), (0,0,0), 2)
paintWindow = cv2.rectangle(paintWindow, (590,100), (639,150), colors[0], -1)
paintWindow = cv2.rectangle(paintWindow, (590,200), (639,250), colors[1], -1)
paintWindow = cv2.rectangle(paintWindow, (590,300), (639,350), colors[2], -1)
paintWindow = cv2.rectangle(paintWindow, (590,400), (639,450), colors[3], -1)
cv2.putText(paintWindow, "X", (595, 45), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255),7, cv2.LINE_AA)
```

Out[67]:

```
array([[[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
        [  0.,   0.,   0.],
        [  0.,   0.,   0.],
        [  0.,   0.,   0.]],

       [[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
        [  0.,   0.,   0.],
        [  0.,   0.,   0.],
        [  0.,   0.,   0.]],
```

```
        ...,
        [255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.]],

       [[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
        [255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.]],

       [[255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.],
        ...,
        [255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.]]])
```

```python
cv2.imshow('paintWindow' ,paintWindow)


store=[]

while True:
    ret , frame= cap.read()
    frame =  cv2.flip(frame ,1)
    hsv = cv2.cvtColor(frame , cv2.COLOR_BGR2HSV)


    frame = cv2.rectangle(frame, (590,0), (639,50), (0,0,0), 2)
    frame = cv2.rectangle(frame, (590,100), (639,150), colors[0], -1)
    frame = cv2.rectangle(frame, (590,200), (639,250), colors[1], -1)
    frame = cv2.rectangle(frame, (590,300), (639,350), colors[2], -1)
    frame = cv2.rectangle(frame, (590,400), (639,450), colors[3], -1)
    cv2.putText(frame, "X", (595, 45), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255),7, cv2.LINE_AA)


    lower =np.array([0, 120, 70])
    upper = np.array([10, 255, 255])
    mask  = cv2.inRange(hsv , lower, upper)
    mask = cv2.erode(mask ,kernel , iterations=1)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask =cv2.dilate(mask , kernel ,iterations=1)
```


```python
    if len(cnts)>0:
        cnt = sorted(cnts, key = cv2.contourArea, reverse = True)[0]
        ((x, y), radius) = cv2.minEnclosingCircle(cnt)
        cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
#         print(x,y)
        M = cv2.moments(cnt)
        center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00']))
        store.append(center)

        if center[0]>590:
            if 0<=center[1]<=50:  #clear button
                bpoints =[deque(maxlen=512)]
                gpoints =[deque(maxlen=512)]
                rpoints =[deque(maxlen=512)]
                ypoints =[deque(maxlen=512)]
                r_index = 0
                b_index  =0
                g_index = 0
                y_index  =0
                paintWindow[50:,:,:]= 255
            elif 100<=center[1]<=150:
                color_index=0
            elif 200<=center[1]<=250:
                color_index=1
            elif 300<=center[1]<=350:
                color_index=2
            elif 400<=center[1]<=450:
                color_index=3
        else:
            if color_index==0:
                bpoints[b_index].appendleft(center)
            if color_index==1:
                gpoints[g_index].appendleft(center)
```

```
Run    Code
            if color_index==3:
                ypoints[y_index].appendleft(center)
    else:
        bpoints.append(deque(maxlen =512))
        b_index+=1
        gpoints.append(deque(maxlen =512))
        g_index+=1
        rpoints.append(deque(maxlen =512))
        r_index+=1
        ypoints.append(deque(maxlen =512))
        y_index+=1

    points=[bpoints ,gpoints,rpoints,ypoints]
    for i in range(0,len(points)):
        for j in range(0,len(points[i])):
            for k in range(1,len(points[i][j])):
                if points[i][j][k-1] is None or points[i][j][k] is None:
                    continue
                cv2.line(frame , points[i][j][k-1] ,points[i][j][k] ,colors[i] ,2)
                cv2.line(paintWindow , points[i][j][k-1]  ,points[i][j][k] ,colors[i] ,2)

    cv2.imshow('frame', frame)
    cv2.imshow('paintWindow', paintWindow)
    cv2.imshow('mask', mask)


    k = cv2.waitKey(5)
    if k==27:
        break
cap.release()
cv2.destroyAllWindows()
```

```
cv2.destroyAllWindows()
```

In [69]:

```
os.system("pause")
```

Out[69]: 0

In [71]:

```
import cv2
import numpy as np
import time
from datetime import datetime
import sys
import math
import pygame
from pygame.locals import *
import RPi.GPIO as GPIO
import os

# Set environment variables
os.putenv('SDL_VIDEODRIVER','fbcon')
os.putenv('SDL_FBDEV', '/dev/fb1')
os.putenv('SDL_MOUSEDRV', 'TSLIB')       # track mouse clicks
os.putenv('SDL_MOUSEDEV', '/dev/input/touchscreen')


#Set GPIO mode
GPIO.setmode(GPIO.BCM)
```

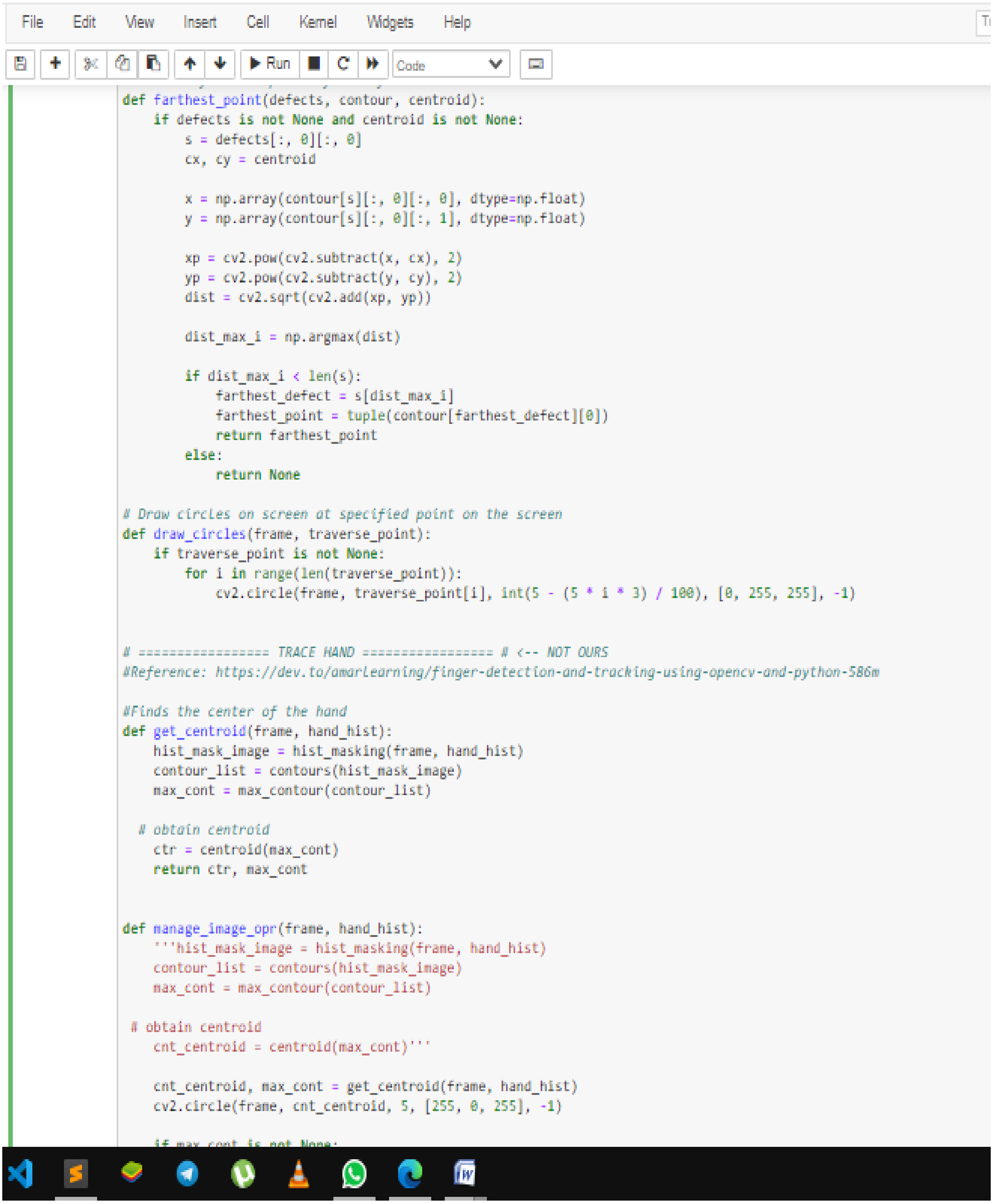
Out[69]: 0

In [71]:

```python
import cv2
import numpy as np
import time
from datetime import datetime
import sys
import math
import pygame
from pygame.locals import *
import RPi.GPIO as GPIO
import os

# Set environment variables
os.putenv('SDL_VIDEODRIVER','fbcon')
os.putenv('SDL_FBDEV', '/dev/fb1')
os.putenv('SDL_MOUSEDRV', 'TSLIB')        # track mouse clicks
os.putenv('SDL_MOUSEDEV', '/dev/input/touchscreen')


#Set GPIO mode
GPIO.setmode(GPIO.BCM)


GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP) #Change color
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP) #Size up
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP) #Size down
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP) #Quit


# Initialize game
pygame.init()


# Screen settings
size = width, height = 320, 240
black = 0,0,0
screen = pygame.display.set_mode(size)

pygame.mouse.set_visible(False)

# Brush settings
radius = 2

#Colors
RED = 255,0,0
GREEN = 0,255,0
BLUE = 0,0,255
WHITE = 255,255,255
```

```python
BTN_SIZE = 50
CENTER_POS = 160,120

#Fill first screen with black
screen.fill(black)


#Create pygame font
font = pygame.font.Font(None, 20)


# ======= SAMPLE_CAM3.PY CODE ============ # <--- NOT OURS
#Reference: https://dev.to/amarlearning/finger-detection-and-tracking-using-opencv-and-python-58
hand_hist = None
traverse_point = []
total_rectangle = 9
hand_rect_one_x = None
hand_rect_one_y = None

hand_rect_two_x = None
hand_rect_two_y = None



#Rescales the output frame to 320 x 240 screen
def rescale_frame(frame, wpercent=130, hpercent=130):
    width = int(frame.shape[1] * wpercent / 100)
  #  print("width: " + str(width) "\n height"
    height = int(frame.shape[0] * hpercent / 100)
    return cv2.resize(frame, (320, 240), interpolation=cv2.INTER_AREA)

#Finds the contours of the hand
def contours(hist_mask_image):
    gray_hist_mask_image = cv2.cvtColor(hist_mask_image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray_hist_mask_image, 0, 255, 0)
    cont, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    return cont


def max_contour(contour_list):
    max_i = 0
    max_area = 0

    for i in range(len(contour_list)):
        cnt = contour_list[i]

        area_cnt = cv2.contourArea(cnt)

        if area_cnt > max_area:
            max_area = area_cnt
            max_i = i
```

```python
    for i in range(total_rectangle):
        cv2.rectangle(frame, (hand_rect_one_y[i], hand_rect_one_x[i]),
                      (hand_rect_two_y[i], hand_rect_two_x[i]),
                      (0, 255, 0), 1)

    return frame

#Attains a histogram of the colors that encapsulate the above retangles
def hand_histogram(frame):
    global hand_rect_one_x, hand_rect_one_y

    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    roi = np.zeros([90, 10, 3], dtype=hsv_frame.dtype)

    for i in range(total_rectangle):
        roi[i * 10: i * 10 + 10, 0: 10] = hsv_frame[hand_rect_one_x[i]:hand_rect_one_x[i] + 10,
                                          hand_rect_one_y[i]:hand_rect_one_y[i] + 10]

    hand_hist = cv2.calcHist([roi], [0, 1], None, [180, 256], [0, 180, 0, 256])
    return cv2.normalize(hand_hist, hand_hist, 0, 255, cv2.NORM_MINMAX)


def hist_masking(frame, hist):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv], [0, 1], hist, [0, 180, 0, 256], 1)

    disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (31, 31))
    cv2.filter2D(dst, -1, disc, dst)

    ret, thresh = cv2.threshold(dst, 150, 255, cv2.THRESH_BINARY)

    # thresh = cv2.dilate(thresh, None, iterations=5)

    thresh = cv2.merge((thresh, thresh, thresh))

    return cv2.bitwise_and(frame, thresh)


def centroid(max_contour):
    moment = cv2.moments(max_contour)
    if moment['m00'] != 0:
        cx = int(moment['m10'] / moment['m00'])
        cy = int(moment['m01'] / moment['m00'])
        return cx, cy
    else:
        return None

#Find the farthest point of hand from the centroid
def farthest_point(defects, contour, centroid):
    if defects is not None and centroid is not None:
        s = defects[:, 0][:, 0]
        cx, cy = centroid

        x = np.array(contour[s][:, 0][:, 0], dtype=np.float)
```



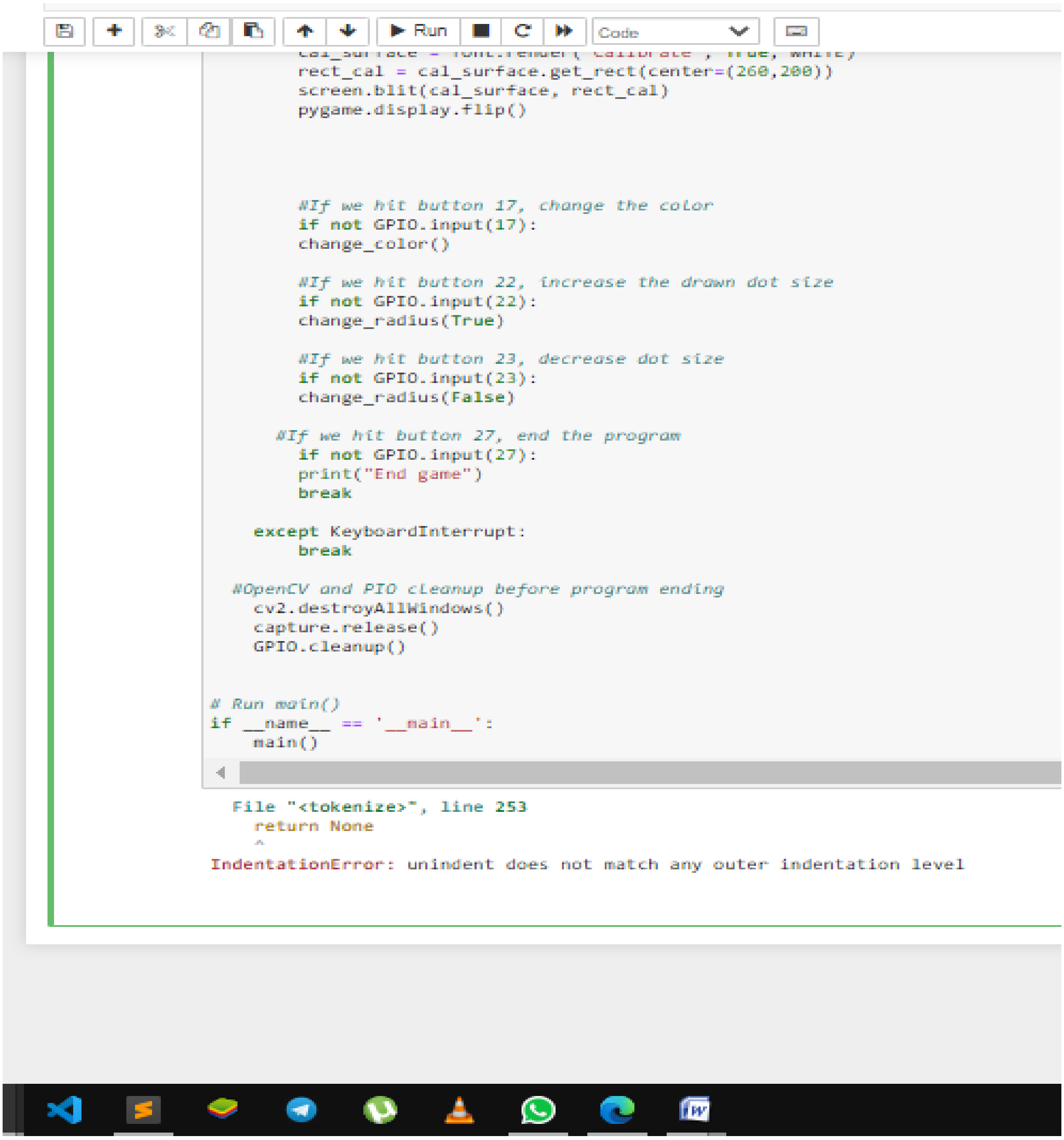

```python
def farthest_point(defects, contour, centroid):
    if defects is not None and centroid is not None:
        s = defects[:, 0][:, 0]
        cx, cy = centroid

        x = np.array(contour[s][:, 0][:, 0], dtype=np.float)
        y = np.array(contour[s][:, 0][:, 1], dtype=np.float)

        xp = cv2.pow(cv2.subtract(x, cx), 2)
        yp = cv2.pow(cv2.subtract(y, cy), 2)
        dist = cv2.sqrt(cv2.add(xp, yp))

        dist_max_i = np.argmax(dist)

        if dist_max_i < len(s):
            farthest_defect = s[dist_max_i]
            farthest_point = tuple(contour[farthest_defect][0])
            return farthest_point
        else:
            return None

# Draw circles on screen at specified point on the screen
def draw_circles(frame, traverse_point):
    if traverse_point is not None:
        for i in range(len(traverse_point)):
            cv2.circle(frame, traverse_point[i], int(5 - (5 * i * 3) / 100), [0, 255, 255], -1)


# ================= TRACE HAND ================= # <-- NOT OURS
#Reference: https://dev.to/amarlearning/finger-detection-and-tracking-using-opencv-and-python-586m

#Finds the center of the hand
def get_centroid(frame, hand_hist):
    hist_mask_image = hist_masking(frame, hand_hist)
    contour_list = contours(hist_mask_image)
    max_cont = max_contour(contour_list)

  # obtain centroid
    ctr = centroid(max_cont)
    return ctr, max_cont


def manage_image_opr(frame, hand_hist):
    '''hist_mask_image = hist_masking(frame, hand_hist)
    contour_list = contours(hist_mask_image)
    max_cont = max_contour(contour_list)

 # obtain centroid
    cnt_centroid = centroid(max_cont)'''

    cnt_centroid, max_cont = get_centroid(frame, hand_hist)
    cv2.circle(frame, cnt_centroid, 5, [255, 0, 255], -1)

    if max_cont is not None:
```

```python
#Checks if a coordinate is within the margins we define
def in_bounds(coord):
    return (coord[0] >= L_MARGIN) and (coord[0] <= R_MARGIN) and (coord[1] >= T_MARGIN) and (coord[1] <= B_MARGIN)


# Draw a dot
# Screen res is 640x480

#Measures the Euclidean distance between two points
def l2_distance(prev_coord, curr_coord):
    return math.sqrt((curr_coord[0]-prev_coord[0])**2 + (curr_coord[1]-prev_coord[1])**2)

#Draws a line between two drawn dots
def interpolate(prev_coord, curr_coord):

    if (prev_coord is not None) and (curr_coord is not None):
  prev_scaled = 320 - int(prev_coord[0]/2), int(prev_coord[1]/2)
  curr_scaled = 320 - int(curr_coord[0]/2), int(curr_coord[1]/2)

  pygame.draw.line(screen, curr_color, prev_scaled, curr_scaled, radius*2)
  pygame.display.flip()

#Draws a dot at a given point in the Pygame display
def draw_dot(coord):
    if (coord != None):
  coord_scaled = 320 - int(coord[0]/2), int(coord[1]/2)
  #prev_scaled = 320 - int(prev_coord[0]/2), int(prev_coord[1]/2)
  print("Dot drawn at: " + str(coord_scaled) )
  #time.sleep(.02)

  if in_bounds(coord_scaled):
      pygame.draw.circle(screen, curr_color, coord_scaled, radius)
     #pygame.draw.line(screen, BLUE, prev_scaled, coord_scaled, radius*2)
      pygame.display.flip()

#Changes the color by iterating through the color list defined earlier
def change_color():
    global curr_color, color_index
    color_index +=1
    if color_index >= len(colors):
  color_index = 0
    curr_color = colors[color_index]
    print(curr_color)

#Increases or decreases the drawn dot and line sizes
def change_radius(up_or_down):
    global radius
    if up_or_down:
  radius+=1
    else:
  radius-=1
```

```
def main():
    global hand_hist

    #Do not draw on init
    draw = False
    is_hand_hist_created = False

    #Create a capture variable
    capture = cv2.VideoCapture(0)

    screen.fill(black)
    videoWidth = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
    videoHeight = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)



  #Intialize the current and previous drawn points
    prev = None
    curr = None
    prev_dot = None
    curr_dot = None
    draw_thresh = 10


    pygame.display.flip()

    #Calibrate histogram on input
    calibrate = True

    while capture.isOpened():
    try:

        # wait for keypress
        pressed_key = cv2.waitKey(1)

        #Read a frame from video capture
        _, frame = capture.read()




        if is_hand_hist_created:
        far_point = manage_image_opr(frame, hand_hist)

        # Draw dot located at farthest point
        ctr, mc = get_centroid(frame, hand_hist)


        if far_point is not None:
            curr = far_point

            #If we're drawing, make sure that we only draw dots if two subsequent
            #Interpolate between two drawn dots
            if draw:
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python

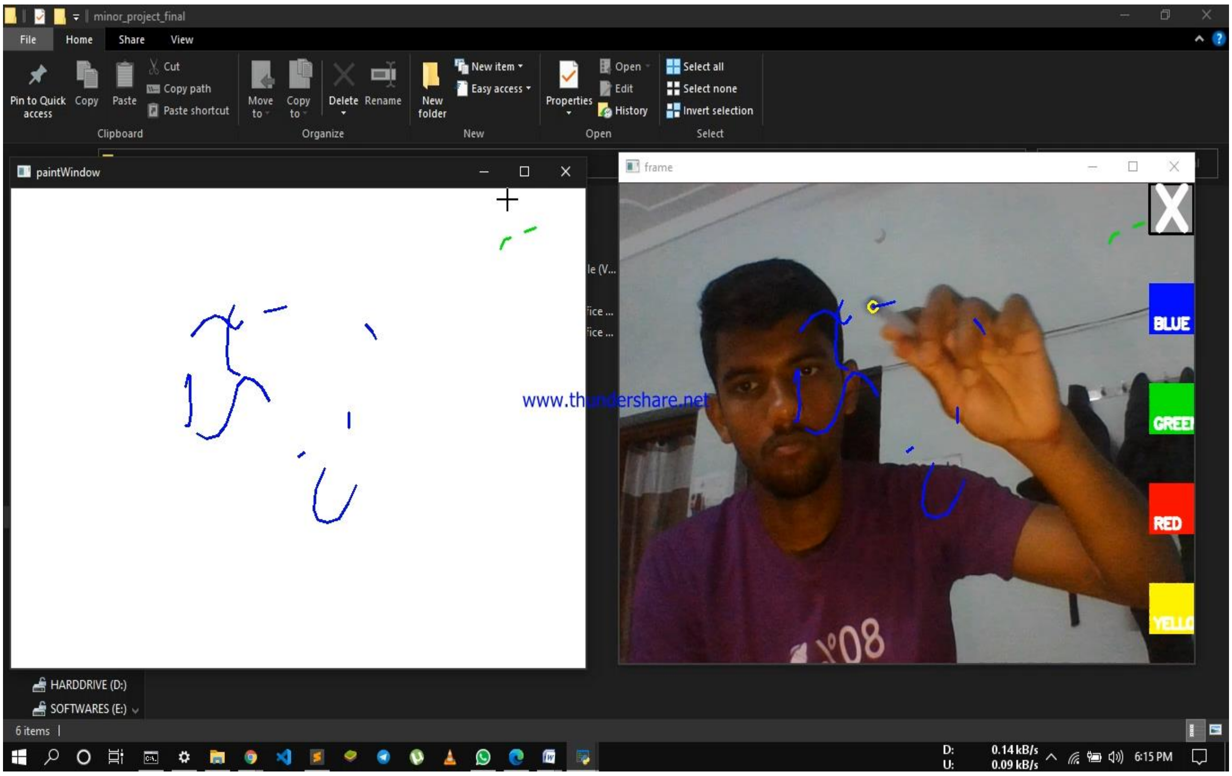
```
      if calibrate:
      if y >= 180 and y <=220 and x>=120 and x<=200:
          is_hand_hist_created = True
          hand_hist = hand_histogram(frame)
          calibrate = False
          screen.fill(black)
          pygame.display.flip()

    #If we're drawing,
    #if we hit the draw button, trigger drawing on and off
    #if we hit the calibrate button, disable drawing, reintialize dot variables, and go back to calibrate screen
    #If we hit anywhere on the screen that is not a button, rotate through the color list
      else:
      if y >= 120 and x <160:
          print("Draw/Not Draw")
          draw = not draw

      elif x >= 160 and y >120:
          print("Calibrate")
          draw = False
          is_hand_hist_created = False
          calibrate = True
          prev = None
          curr = None
          prev_dot = None
          curr_dot = None
      else:
          change_color()


  #Rescale the display frame to 320 x 240 pixels
  rescaled_frame = rescale_frame(frame)

  #Draw the calibrate button on the live cam screen if we're calibrating
  if calibrate:
  #print(rescaled_frame.shape)
  surface = pygame.surfarray.make_surface(rescaled_frame.transpose(1,0,2)[...,::-1])
  surface.convert()

  cal_surface = font.render('Calibrate', True, WHITE)

  rect_cal = cal_surface.get_rect(center=(160,200))

  screen.blit(surface, (0,0))
  pygame.draw.rect(screen, BLUE, pygame.Rect(120, 190, 80, 20))
  screen.blit(cal_surface, rect_cal)


  pygame.display.flip()

#Render the draw and quit buttons on the drawing page
  else:

  pause_surface = font.render('Draw', True, WHITE)
```

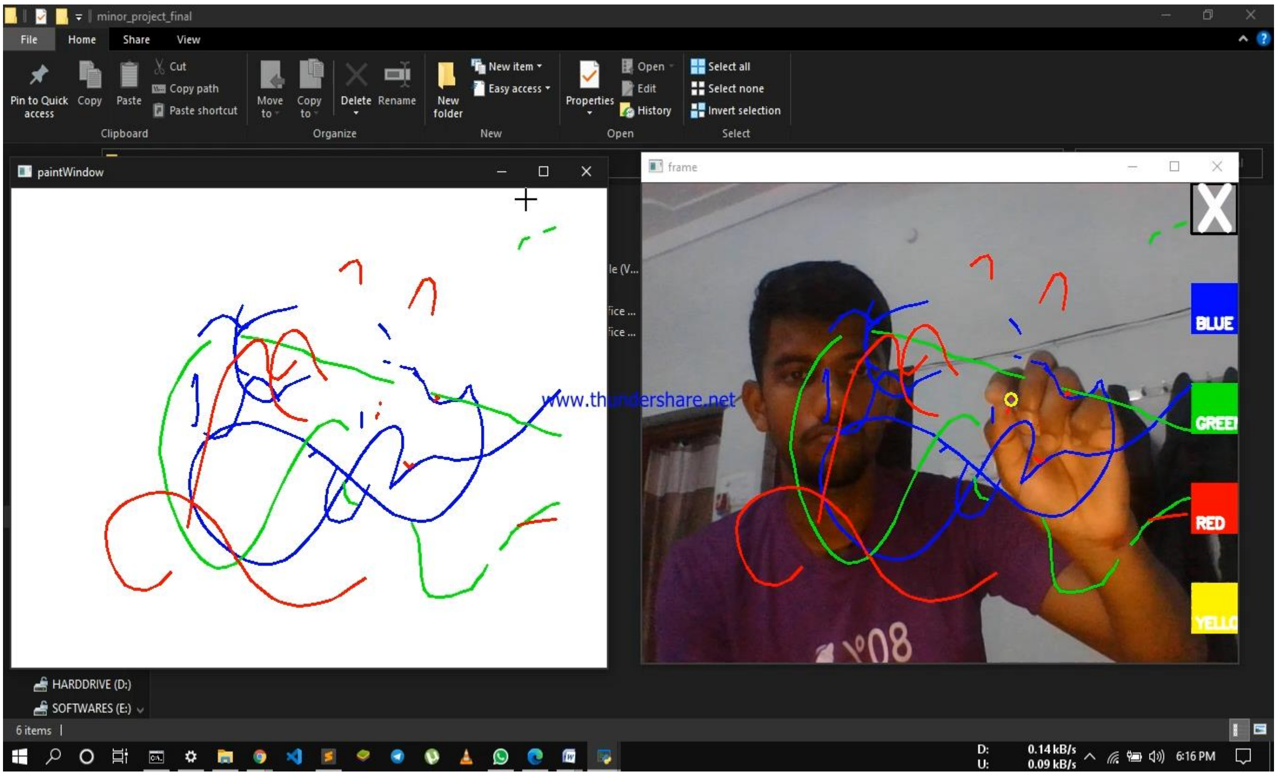


```
        cal_surface = font.render("calibrate", True, WHITE)
        rect_cal = cal_surface.get_rect(center=(260,200))
        screen.blit(cal_surface, rect_cal)
        pygame.display.flip()



        #If we hit button 17, change the color
        if not GPIO.input(17):
        change_color()

        #If we hit button 22, increase the drawn dot size
        if not GPIO.input(22):
        change_radius(True)

        #If we hit button 23, decrease dot size
        if not GPIO.input(23):
        change_radius(False)

      #If we hit button 27, end the program
        if not GPIO.input(27):
        print("End game")
        break

    except KeyboardInterrupt:
        break

  #OpenCV and PIO cleanup before program ending
    cv2.destroyAllWindows()
    capture.release()
    GPIO.cleanup()


# Run main()
if __name__ == '__main__':
    main()
```

```
  File "<tokenize>", line 253
    return None
    ^
IndentationError: unindent does not match any outer indentation level
```

Screenshots after running:





7. Conclusion

This project is relevant to Virtual Developers and Artificial Intelligence. OpenCV is at work in robotics systems—picking let- tuce, recognizing items on conveyor belts, helping self-driving cars see, flying quadrotors, doing tracking and mapping in virtual and augmented reality systems, helping unload trucks and pallets in distribution centers, and more—and is built into the Robotics Operating System (ROS). It is used in applications that promote mine safety, prevent swimming pool drownings, process Google Maps and streetview imagery, and implement Google X robotics .

8. Future Scope

OpenCV is the most popular open source computer vision library in the world. It implements over 2500 optimized algorithms, works on all major operating systems, is available in multiple languages and is free for commercial use. This talk will primarily provide a technical update on OpenCV: What's new in OpenCV 4.0? What is the Graph API? Why are we so excited about the Deep Neural Network (DNN) module in OpenCV? (Short answer: It is one of the fastest inference engines on the CPU.) We will also briefly share info on the new Open Source Vision Foundation (OSVF), and on OpenCV's sister organizations, CARLA and Open3D, and some of the initiatives planned by these organizations.

9. References

1. Learning OpenCV - Gary Bradski Adrian Kaehler
2. Implementation - http://infoaryan.com/
3. www.towardsdatascience.com
4. www.medium.com
5. OpenCV - https://opencv.org/
6. https://scikit-learn.org/stable/ Scikit learn documentation Python
7. Face Detection & Recognition Report 08/09/2017 Andreea Diana Dinca, IT Innovation Intern , mailto:itinnovation@contacts.bham.ac.uk