Table of Contents

PROBLEM STATEMENT 2
CHAPTER 1: INTRODUCTION 3
CHAPTER 2: LITERATURE SURVEY 4
CHAPTER 3: DETAILED DESIGN 8
CHAPTER 4: PROJECT SPECIFIC REQUIREMENTS 12
CHAPTER 5: IMPLEMENTATION 13
CHAPTER 6: RESULTS 24
CHAPTER 7: CONCLUSION AND FUTURE SCOPE 26
REFERENCES 27

PROBLEM STATEMENT

Build an Air Canvas which can draw anything on it by just capturing the motion of a coloured marker with camera. Here a coloured object at tip of finger is used as the marker.

CHAPTER 1: INTRODUCTION

Despite numerous recent advances in the field of deep learning for artistic purposes,the integration of these state-of-the-art machine learning tools into applications for drawing and visual expression has been an under explored field. Bridging this gap has the potential to empower a large subset of the population, from children to the elderly, with a new medium to represent and visualize their ideas

Air Canvas is a hands-free digital drawing canvas that utilizes OpenCV. The user's "brush" can change color by using built-in buttons. The direction of the brush is controlled completely using open source OpenCV software with which a marker of certain color is tracked and mapped to the screen.

The idea for Air Canvas was a result of prevailing problems and inconveniences like need for a dustless classroom for the students to study in or the current situation of online education.

In addition to this, there are substantial concrete benefits to such a system that encourages creative expression. For children, encouraging creativity from a young age is correlated with mood improvement and development of crucial social skill.

For many elderly populations, encouraging creative activity is associated with emotional and physical responses such as increased optimism, alertness, and perceptions of self-worth; furthermore, for older people with dementia, art therapy is associated with positive changes in mood and increased sociability.

With all these potential benefits in mind, this project was born.

CHAPTER 2: LITERATURE SURVEY

Object tracking is considered as an important task within the field of Computer Vision. The invention of faster computers, availability of inexpensive and good quality video cameras and demands of automated video analysis has given popularity to object tracking techniques. Generally video analysis procedure has three major steps: firstly detecting of the object, secondly tracking its movement from frame to frame and lastly analyzing the behavior of that object. For object tracking, four different issues aretaken into account; selection of suitable object representation, feature selection for tracking, object detection and object tracking. In real world, Object tracking algorithms are the primarily part of different applications such as: automatic surveillance, video indexing and vehicle navigation etc.

Another application of object tracking is human computer interaction. Different researchers proposed many algorithms which are categorically divided into two main approaches: image based approach and Glove based approach. Image based approach requires images as input in order to recognize the hand (object) movements. On the other hand, Glove based approach require specific hardware which includes special sensors etc. Such applications are beneficial for disabled people.

In this project, a real time fast video based finger tip tracking and recognizing algorithm is presented. The proposed algorithm has two major tasks: first it detects the motion of colored marker in video sequence and then draws the line accordingly. Proposed method is software based approach while in literature, almost all existing finger tracking based character recognition system require extra hard ware e.g. Light Emission Diode (LED) pen, Leap Motion controllerdevice etc.

Libraries Used:

- OpenCV**:** It is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.
- NumPy**:** It is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

Masking:

Image masking means to apply some other image as a mask on the original image or to change the pixel values in the image. masking involves these steps for the color detection.

- Reading the image and convert it into HSV (Hue, Saturation and Value)
- creating a NumPy array for the lower green values and the upper green values

- Use the inRange() method of cv2 to check if the given image array elements lie between array values of upper and lower boundaries
  <masking = cv2.inRange(hsv_img, lower_green, upper_green)>
  hence this will detect the green color.

Why do we need to convert the image into HSV?

RGB color space describes colors in terms of the amount of red, green, and blue present. HSV color space describes colors in terms of the Hue, Saturation, and Value. In situations where color description plays an integral role, the HSV color model is often preferred over the RGB model. The HSV model describes colors similarly to how the human eye tends to perceive color. RGB defines color in terms of a combination of primary colors, where as, HSV describes color using more familiar comparisons such as color, vibrancy and brightness.

R, G, and B components of an object's color in a digital image are all correlated with the amount of light hitting the object, and therefore with each other, image descriptions in terms of those components make object discrimination difficult. Descriptions in terms of hue/lightness/chroma or hue/lightness/saturation are often more relevant.

Finding contour and its center for moment detection:

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. Finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

- findContours()
- cv.drawContours function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

Morphological operations are a set of operations that process images based on shapes. They apply a structuring element to an input image and generate an output image.

What is kernel?

A Kernel tells you how to change the value of any given pixel by combining it with different amounts of the neighboring pixels. The kernel is applied to every pixel in the image one-by-one to produce the final image.

Working of erosion:

- A kernel(a matrix of odd size(3,5,7) is convolved with the image.
- A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).
- Thus all the pixels near boundary will be discarded depending upon the size of kernel.
- So the thickness or size of the foreground object decreases or simply white region decreases in the image.

Opening:

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as we explained above. Here we use the function, <cv2.morphologyEx()>

Why kernel size should be only odd?

The kernel doesn't need to be odd. When the kernel size is even, it is less obvious which of the pixels should be at the origin, but this is not a problem. You have seen mostly odd-sized filter kernels because they are symmetric around the origin, which is a good property.

Related Work:

Automatic object tracking has many applications such as; computer vision and human-machine interaction etc. Generally, object could be a text or person which needs to be tracked. In literature, different applications of tracking algorithm are proposed. One group of researchers are used it for translating the Sign Languages, other used it for hand gesture recognition, another group used it for text localization and detection, tracing full body motion of object for virtual reality and finger tracking based character recognition etc.

There have been works in this field for developing softwares using OpenCV and Object Tracking. Models have been suggested for Sign language translation, Hand gesture recognition, text localization and recognition in real world images, Color based motion capturing system and so on. Despite of all these, very less work has been made towards artistic field using these technologies. The entire field of OpenCV, AI/ML, deep learning has contributed in a very less amount to such field and left a whole area of possibilities and opportunities unexplored.

On the other hand, there are various softwares primarily focusing on canvas based approaches like Microsoft Paint and the wide variety of similar pixel-based drawing canvases.
Also there are Vector Focused Software like Adobe Photoshop and Adobe Illustrator and GIMP. But all of these require some kind of input device like mouse or Graphic tablets, the latter also have a steep learning curve.

Though there are some Machine Learning integrated softwares for similar purposes like Google - Magenta and NVIDIA - GauGAN, they usually are complex to implement and require some dedicated and expensive hardware for the same.

Vision-based drawing tool tracking is usually employed when an application requires 3D pose of the tool, e.g., 3D interaction with a tangible tool in virtual or augmented reality environments. For instance, ARPen is introduced for a 3D modeling task, where a 3D-printed pen combined with a smartphone is used. The interaction was done in the mid-air, which enables drawing and interacting with virtual objects. Visual markers and ArUco, an Open Source library are utilized to track the position of the pen tip. Milosevicet al.proposed a SmartPen for the sketch-based surface modeling in, where a stereo webcam and Inertial Measurement Unit (IMU) are utilized. These sensors provide sequences of sorted 3D points, which are used to estimate the absolute position and orientation of the pen tip. Their work

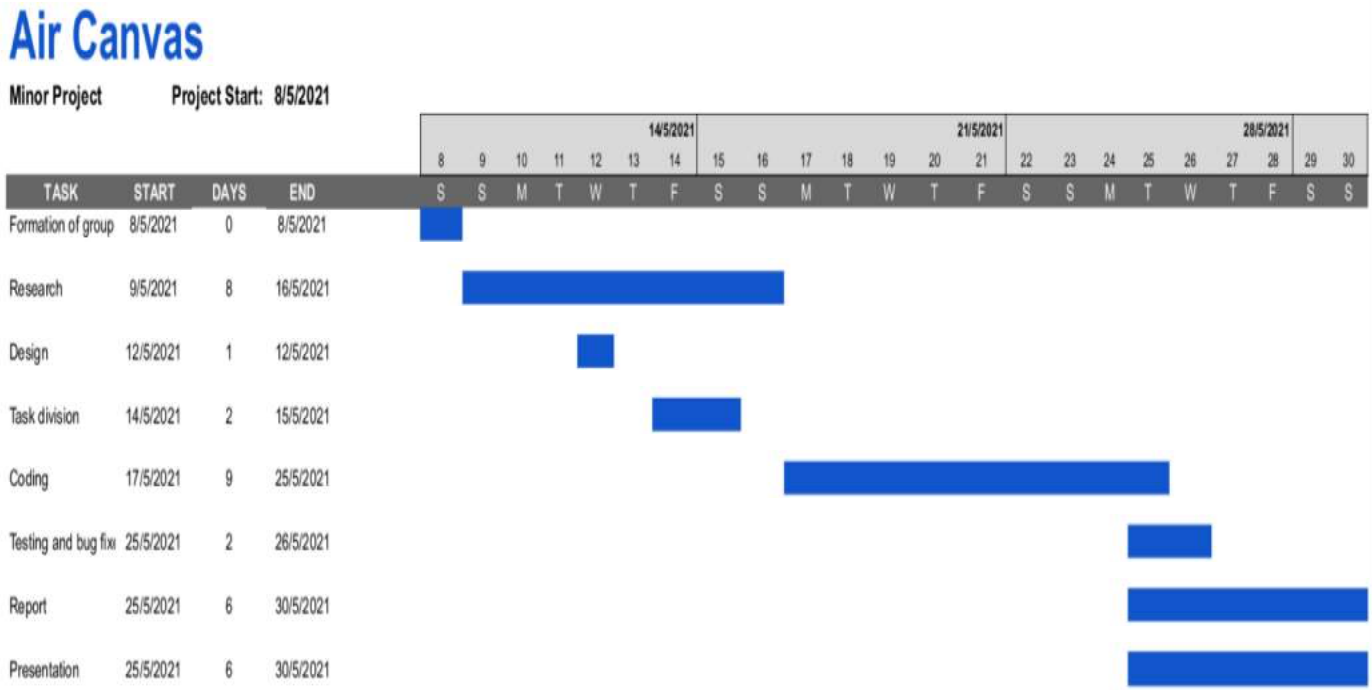
permits to obtain the style lines of actual objects, including concave parts and shapes. Moreover, they presented sketch-based modeling for automatically producing the 3D virtual model using an interactive surface sketching approach. Similarly,Wuet al. introduced a system for tracking of a passive stylus for drawing in augmented reality and virtual reality environments. In their work, a square marker on the 3D printed pen was applied and inter-frame corner tracking is performed. In their work, they applied the pyramidal LK optical flow algorithm to track the marker corners on each frame. In, a 6DOF digital pen was designed for performing drawing in the tablet as well as mid-air, where a Vicon motion capturing camera is employed for tracking the pen position and orientation. Although tangible tools gave a solid feeling in interaction, these systems sometimes require visually distracting markers that may disturb artists'creativity and did not consider the tracking of a brush with actual deforming bristles.

DRAWING ON AIR

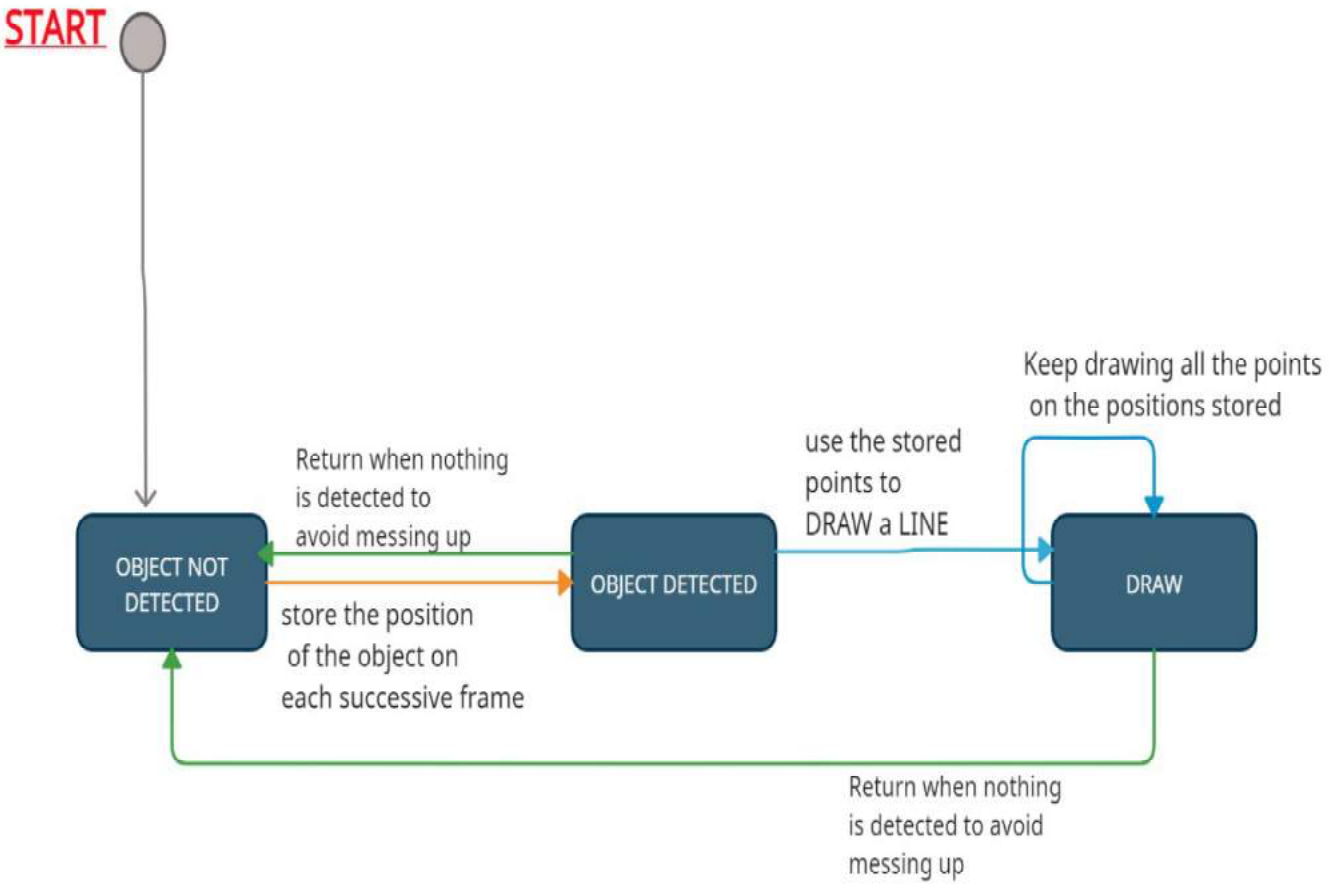
Drawing on Air integrates two complimentary approachesto drawing 3D curves, one-handed drag drawing and two-handed tape drawing. Both techniques have advantages.One-handed is generally easier to learn to control than two-handed, whereas two-handed feels more controllable to expert users. One-handed is also more appropriate for circular shapes that would require one's arms to cross ifdrawn with the two-handed approach. The key to both techniques is providing the user withexplicit control of the tangent of the curve being drawn. This direction-explicit approach to drawing can be de-scribed in terms of two subtasks: 1) defining the direction(tangent) of the drawing and 2) advancing the line alongthis direction. These factors aid control:1) The visual guidelines displaying the direction of drawinghelp to measure space and plan drawing. 2) Artists are able to work deliberately without introducing additional jitter, advancing the line only when they see it is going in the right direction.

CHAPTER 3: DETAILED DESIGN

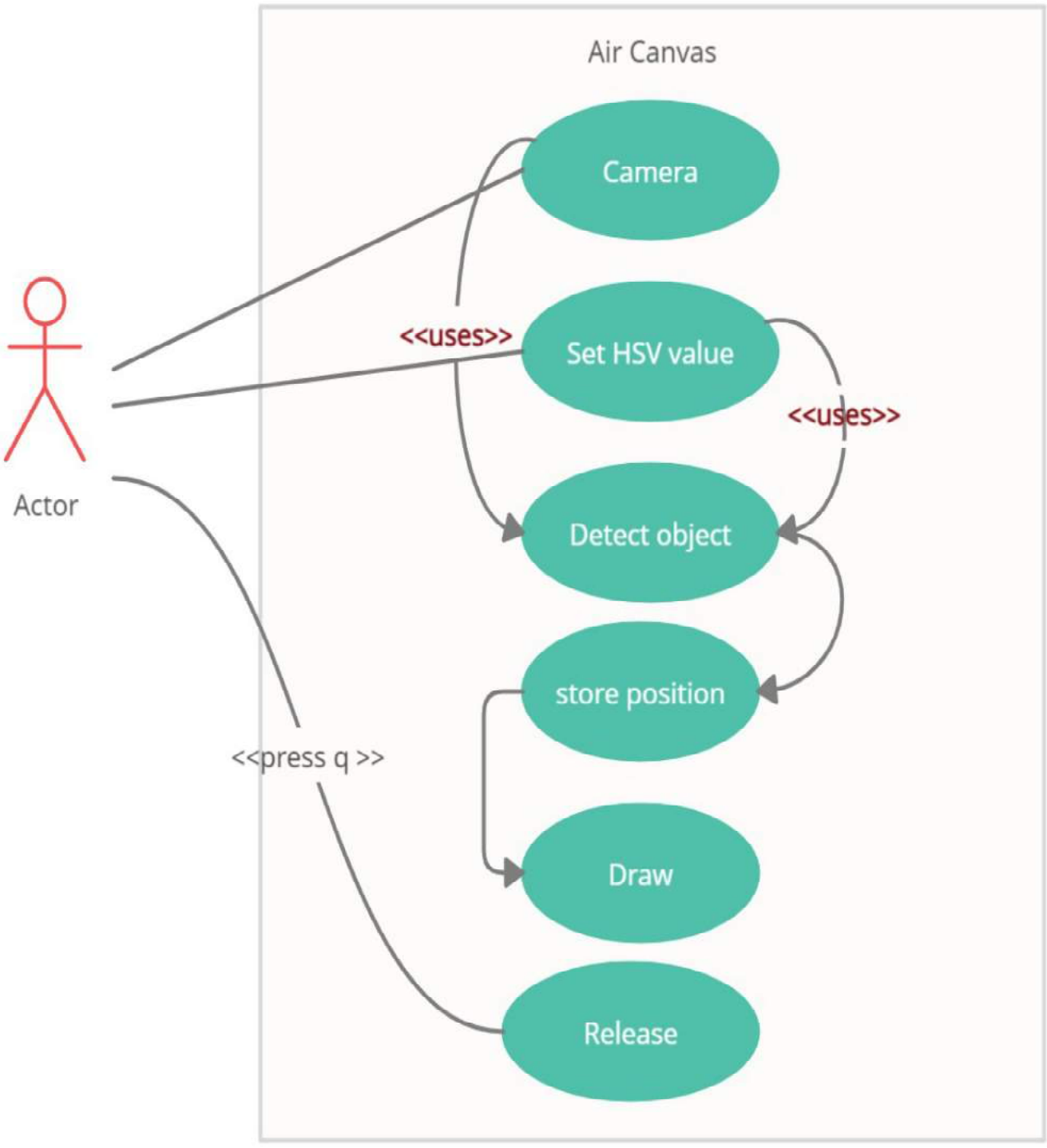
Gantt Chart:



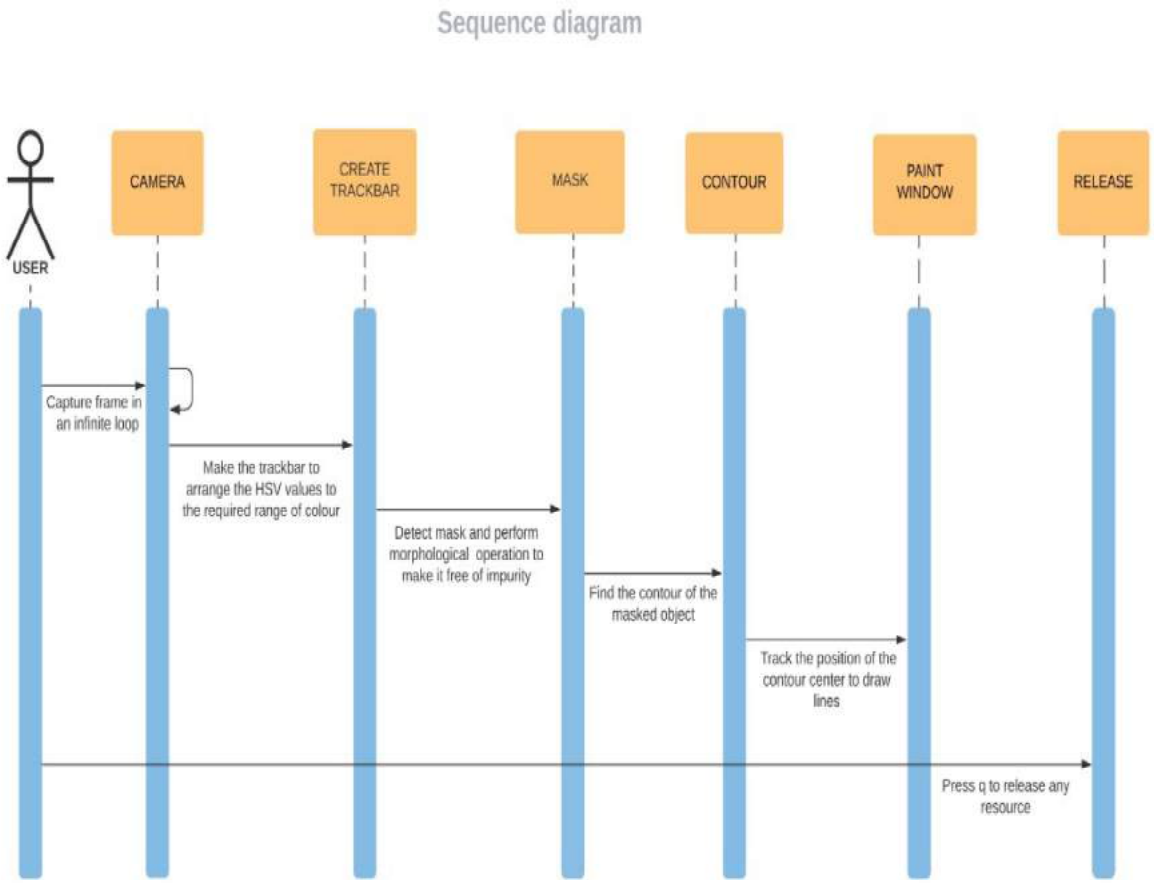
State Diagram:



Use Case Diagram:



Sequence Diagram:



CHAPTER 4: PROJECT SPECIFIC REQUIREMENTS

Software Requirements:

- Python 3.6 or above
- pip installer
- Text editor (or a python IDE)

Hardware Requirements:

- Any modern OS (Windows10, Linux, MacOS)
- 32-bit or 64-bit CPU (64-bit preferable)
- 2GB RAM (4GB preferable)
- Minimum Disk Space of 3GB
- Camera (Integrated or external)

Libraries Required:

- OpenCV library - To read and manipulate the images.
- NumPy library - For Numerical Computation

Working of the Air canvas project:

1. Colour Tracking of Object at fingertip:

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

```
# Loading the default webcam of PC.

cap = cv2.VideoCapture(0)

# Keep looping

while True:

# Reading the frame from the camera

ret, frame = cap.read()

# Flipping the frame to see same side of yours

frame = cv2.flip(frame, 1)

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

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

```
# Creating the trackbars needed for adjusting the marker colour
cv2.namedWindow("Color detectors")
cv2.createTrackbar("Upper Hue", "Color detectors", 179, 180, setValues)
cv2.createTrackbar("Upper Saturation", "Color detectors", 255, 255, setValues)
cv2.createTrackbar("Upper Value", "Color detectors", 255, 255, setValues)
cv2.createTrackbar("Lower Hue", "Color detectors", 77, 180, setValues)
cv2.createTrackbar("Lower Saturation", "Color detectors", 104, 255, setValues)
cv2.createTrackbar("Lower Value", "Color detectors", 84, 255, setValues)
```

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 coloured object. This Mask is a black and white image with white pixels at the position of the desired colour.

```
#Getting the Realtime values from the trackbars for colour detection
    u_hue = cv2.getTrackbarPos("Upper Hue", "Color detectors")
    u_saturation = cv2.getTrackbarPos("Upper Saturation", "Color detectors")
    u_value = cv2.getTrackbarPos("Upper Value", "Color detectors")
    l_hue = cv2.getTrackbarPos("Lower Hue", "Color detectors")
    l_saturation = cv2.getTrackbarPos("Lower Saturation", "Color detectors")
    l_value = cv2.getTrackbarPos("Lower Value", "Color detectors")

#Upper and lower are range within which color can be found
    Upper_hsv = np.array([u_hue, u_saturation, u_value])
    Lower_hsv = np.array([l_hue, l_saturation, l_value])

# Position of the pointer by making its mask
    Mask = cv2.inRange(hsv, Lower_hsv, Upper_hsv)
```

2. Contour Detection of the Mask of Colour Object:

After detecting the Mask in Air Canvas, Now is the time to locate its centre 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.

```
# Operations to remove impurities and false predictions
Mask = cv2.erode(Mask, kernel, iterations=1)
Mask = cv2.morphologyEx(Mask, cv2.MORPH_OPEN, kernel)
Mask = cv2.dilate(Mask, kernel, iterations=1)

# Find contours for the pointer
cnts, _ = cv2.findContours(Mask.copy(), cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)
center = None

# If the contours are formed, find center of contour (Position)
    if len(cnts) > 0:
            # sorting the contours to find biggest
            cnt = sorted(cnts, key=cv2.contourArea, reverse=True)[0]
            # Get the radius of the enclosing circle around the found contour
            ((x, y), radius) = cv2.minEnclosingCircle(cnt)
            # Draw the circle around the contour
            cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
            # Calculating the center of the detected contour
            M = cv2.moments(cnt)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

3. 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.

```
# Giving different arrays to handle colour points of different colour
bpoints = [deque(maxlen=1024)]
gpoints = [deque(maxlen=1024)]
rpoints = [deque(maxlen=1024)]
ypoints = [deque(maxlen=1024)]

# These indexes will be used to mark the points in particular arrays of specific colour
blue_index = 0
green_index = 0
red_index = 0
yellow_index = 0

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]
colorIndex = 0
```

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.

```
        # Now checking if the user wants to click on any button above the screen
        if center[1] <= 65:
                if 40 <= center[0] <= 140: # Clear Button
                        bpoints = [deque(maxlen=512)]
                        gpoints = [deque(maxlen=512)]
                        rpoints = [deque(maxlen=512)]
                        ypoints = [deque(maxlen=512)]

                        blue_index = 0
                        green_index = 0
                        red_index = 0
                        yellow_index = 0

                        paintWindow[67:, :, :] = 255
                elif 160 <= center[0] <= 255:
                        colorIndex = 0 # Blue
                elif 275 <= center[0] <= 370:
                        colorIndex = 1 # Green
                elif 390 <= center[0] <= 485:
                        colorIndex = 2 # Red
                elif 505 <= center[0] <= 600:
                        colorIndex = 3 # Yellow
                else:
                        if colorIndex == 0:
                                bpoints[blue_index].appendleft(center)
                        elif colorIndex == 1:
                                gpoints[green_index].appendleft(center)
                        elif colorIndex == 2:
                                rpoints[red_index].appendleft(center)
                        elif colorIndex == 3:
                                ypoints[yellow_index].appendleft(center)

        # Append the next deques when nothing is detected to avois messing up
        else:
                bpoints.append(deque(maxlen=512))
                blue_index += 1
                gpoints.append(deque(maxlen=512))
                green_index += 1
                rpoints.append(deque(maxlen=512))
                red_index += 1
                ypoints.append(deque(maxlen=512))
                yellow_index += 1
```

4. Drawing the points:

We will draw all the points on the positions stored in the deques, with respective colour

```
# Draw lines of all the colors on the canvas and frame
points = [bpoints, gpoints, rpoints, ypoints]
            for i in range(len(points)):
                for j in range(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)

# Show all the windows
cv2.imshow("Tracking", frame)
cv2.imshow("Paint", paintWindow)
cv2.imshow("mask", Mask)
```

Air Canvasing

Complete Code:

```
import numpy as np

import cv2

from collections import deque

# default called trackbar function

def setValues(x):

  print("")

# Creating the trackbars needed for adjusting the marker colour

cv2.namedWindow("Color detectors")

cv2.createTrackbar("Upper Hue", "Color detectors", 179, 180, setValues)

cv2.createTrackbar("Upper Saturation", "Color detectors", 255, 255, setValues)

cv2.createTrackbar("Upper Value", "Color detectors", 255, 255, setValues)

cv2.createTrackbar("Lower Hue", "Color detectors", 77, 180, setValues)

cv2.createTrackbar("Lower Saturation", "Color detectors", 104, 255, setValues)

cv2.createTrackbar("Lower Value", "Color detectors", 84, 255, setValues)

# Giving different arrays to handle colour points of different colour

bpoints = [deque(maxlen=1024)]

gpoints = [deque(maxlen=1024)]

rpoints = [deque(maxlen=1024)]

ypoints = [deque(maxlen=1024)]

# These indexes will be used to mark the points in particular arrays of specific colour

blue_index = 0

green_index = 0

red_index = 0

yellow_index = 0

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]

colorIndex = 0
```

Air Canvasing

```
# Code for Canvas setup

paintWindow = np.zeros((471, 636, 3)) + 255

paintWindow = cv2.rectangle(paintWindow, (40, 1), (140, 65), (0, 0, 0), 2)

paintWindow = cv2.rectangle(paintWindow, (160, 1), (255, 65), colors[0], -1)

paintWindow = cv2.rectangle(paintWindow, (275, 1), (370, 65), colors[1], -1)

paintWindow = cv2.rectangle(paintWindow, (390, 1), (485, 65), colors[2], -1)

paintWindow = cv2.rectangle(paintWindow, (505, 1), (600, 65), colors[3], -1)

cv2.putText( paintWindow, "CLEAR", (55, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2,

cv2.LINE_AA)cv2.putText( paintWindow, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255, 255,

255), 2, cv2.LINE_AA)

cv2.putText( paintWindow, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,

255), 2, cv2.LINE_AA)

cv2.putText( paintWindow, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),

2, cv2.LINE_AA)

cv2.putText( paintWindow, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (150, 150,

150), 2, cv2.LINE_AA)

cv2.namedWindow("Paint", cv2.WINDOW_AUTOSIZE)


# The kernel to be used for dilation purpose

# unit8-unsigned integer from 0to 255

kernel = np.ones((5, 5), np.uint8)


# Loading the default webcam of PC.

cap = cv2.VideoCapture(0)

# Keep looping

while True:

  # Reading the frame from the camera

  ret, frame = cap.read()
```

Air Canvasing

```
 # Flipping the frame to see same side of yours

 frame = cv2.flip(frame, 1)

 hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)


u_hue = cv2.getTrackbarPos("Upper Hue", "Color detectors")

u_saturation = cv2.getTrackbarPos("Upper Saturation", "Color detectors")

u_value = cv2.getTrackbarPos("Upper Value", "Color detectors")

l_hue = cv2.getTrackbarPos("Lower Hue", "Color detectors")

l_saturation = cv2.getTrackbarPos("Lower Saturation", "Color detectors")

l_value = cv2.getTrackbarPos("Lower Value", "Color detectors")

Upper_hsv = np.array([u_hue, u_saturation, u_value])

Lower_hsv = np.array([l_hue, l_saturation, l_value])

# Adding the colour buttons to the live frame for colour access

frame = cv2.rectangle(frame, (40, 1), (140, 65), (122, 122, 122), -1)

frame = cv2.rectangle(frame, (160, 1), (255, 65), colors[0], -1)

frame = cv2.rectangle(frame, (275, 1), (370, 65), colors[1], -1)

frame = cv2.rectangle(frame, (390, 1), (485, 65), colors[2], -1)

frame = cv2.rectangle(frame, (505, 1), (600, 65), colors[3], -1)

cv2.putText( frame, "CLEAR ALL", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,

255), 2, cv2.LINE_AA)

cv2.putText( frame, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2,

cv2.LINE_AA)

cv2.putText( frame, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),

2, cv2.LINE_AA)

cv2.putText( frame, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2,

cv2.LINE_AA)cv2.putText( frame, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (150,
150,

150), 2, cv2.LINE_AA )
```

Air Canvasing

```
# Identifying the pointer by making its mask

Mask = cv2.inRange(hsv, Lower_hsv, Upper_hsv)

Mask = cv2.erode(Mask, kernel, iterations=1)

Mask = cv2.morphologyEx(Mask, cv2.MORPH_OPEN, kernel)

Mask = cv2.dilate(Mask, kernel, iterations=1)

# Find contours for the pointer after identifying it

cnts, _ = cv2.findContours(Mask.copy(), cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)

center = None

# If the contours are formed

if len(cnts) > 0:

  # sorting the contours to find biggest

  cnt = sorted(cnts, key=cv2.contourArea, reverse=True)[0]

  # Get the radius of the enclosing circle around the found contour

  ((x, y), radius) = cv2.minEnclosingCircle(cnt)

  # Draw the circle around the contour

  cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)

  # Calculating the center of the detected contour

  M = cv2.moments(cnt)

  center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

# Now checking if the user wants to click on any button above the screen

if center[1] <= 65:

  if 40 <= center[0] <= 140: # Clear Button

    bpoints = [deque(maxlen=512)]

    gpoints = [deque(maxlen=512)]

    rpoints = [deque(maxlen=512)]

    ypoints = [deque(maxlen=512)]

    blue_index = 0

    green_index = 0
```

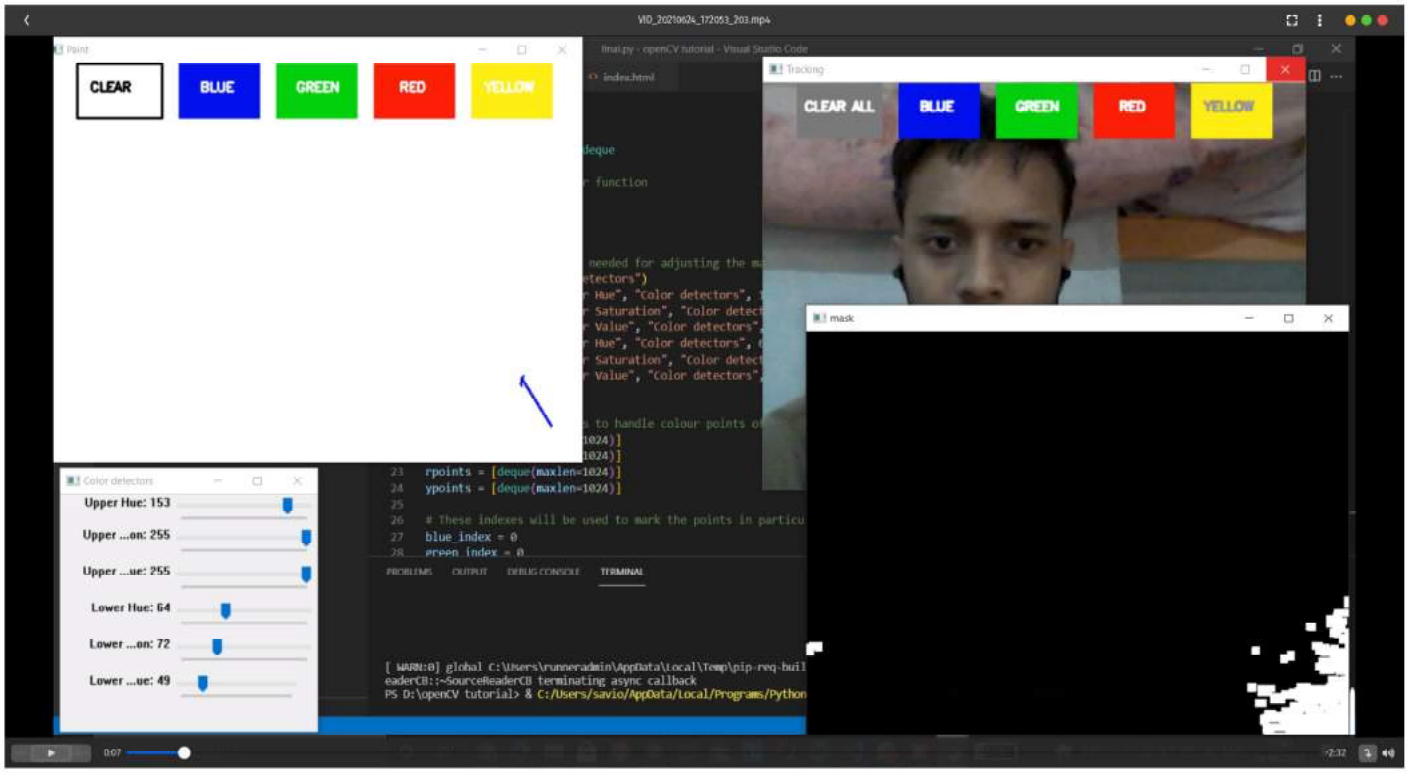
```
    red_index = 0
    yellow_index = 0
    paintWindow[67:, :, :] = 255
  elif 160 <= center[0] <= 255:
    colorIndex = 0 # Blue
  elif 275 <= center[0] <= 370:
    colorIndex = 1 # Green
  elif 390 <= center[0] <= 485:
    colorIndex = 2 # Red
  elif 505 <= center[0] <= 600:
    colorIndex = 3 # Yellow
  else:
    if colorIndex == 0:
      bpoints[blue_index].appendleft(center)
    elif colorIndex == 1:
      gpoints[green_index].appendleft(center)
    elif colorIndex == 2:
      rpoints[red_index].appendleft(center)
    elif colorIndex == 3:
      ypoints[yellow_index].appendleft(center)
# Append the next deques when nothing is detected to avois messing up
  else:
    bpoints.append(deque(maxlen=512))
    blue_index += 1
    gpoints.append(deque(maxlen=512))
    green_index += 1
    rpoints.append(deque(maxlen=512))
    red_index += 1
    ypoints.append(deque(maxlen=512))
```

```
    yellow_index += 1

# Draw lines of all the colors on the canvas and frame

points = [bpoints, gpoints, rpoints, ypoints]

for i in range(len(points)):

  for j in range(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)

# Show all the windows

cv2.imshow("Tracking", frame)

cv2.imshow("Paint", paintWindow)

cv2.imshow("mask", Mask)

# If the 'q' key is pressed then stop the application

if cv2.waitKey(1) & 0xFF == ord("q"):

  break

# Release the camera and all resources

cap.release()

cv2.destroyAllWindows()
```

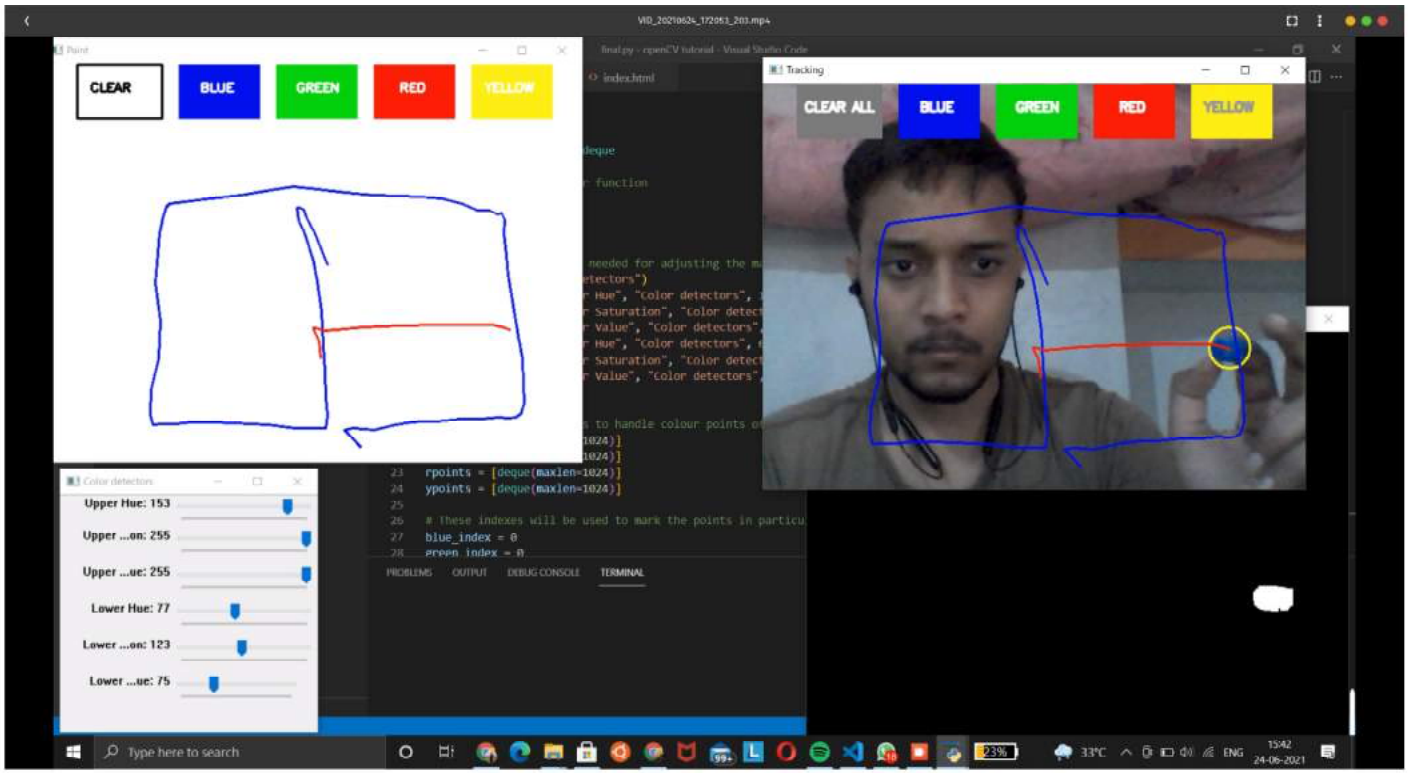
CHAPTER 6: RESULTS

Below are the screenshots of working/output of the program.

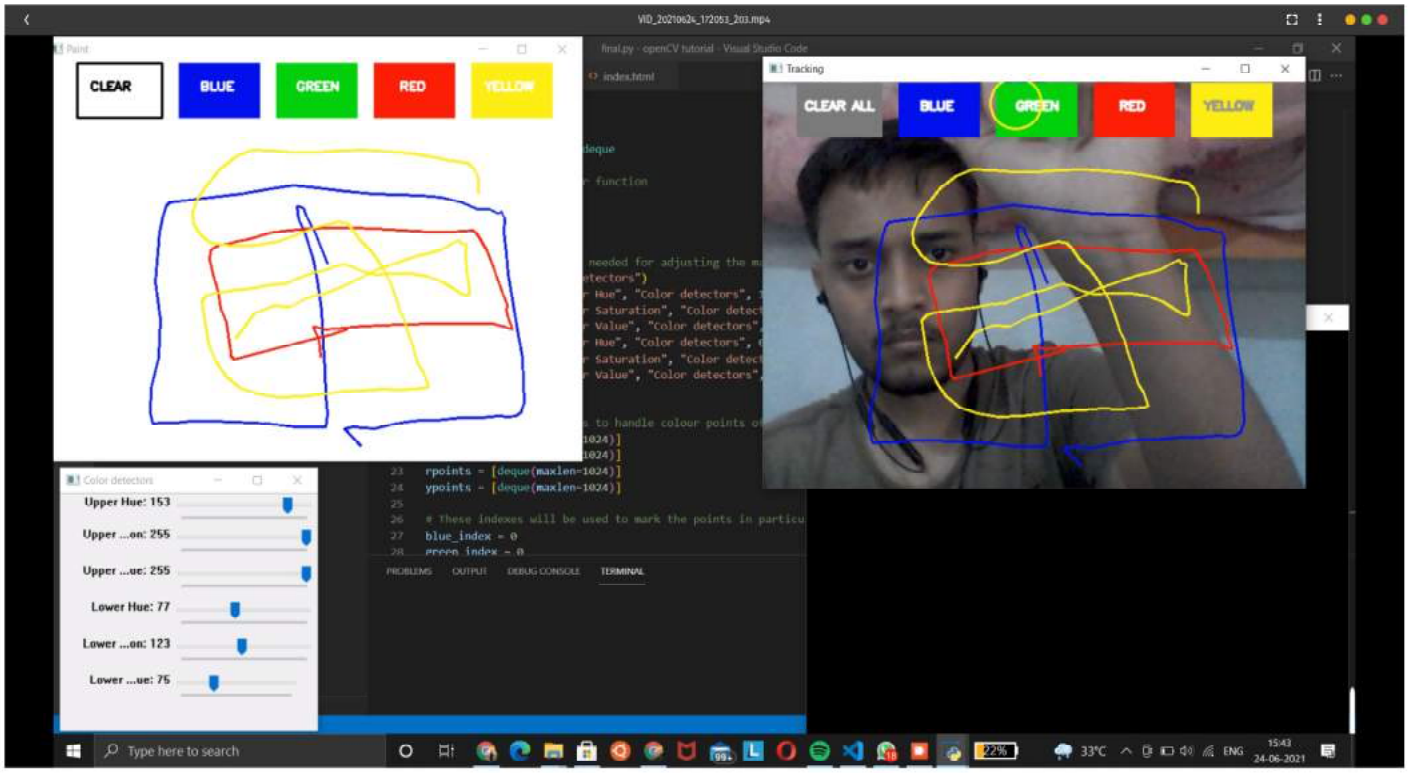
In the first picture you can see the four windows being created as we run the program.



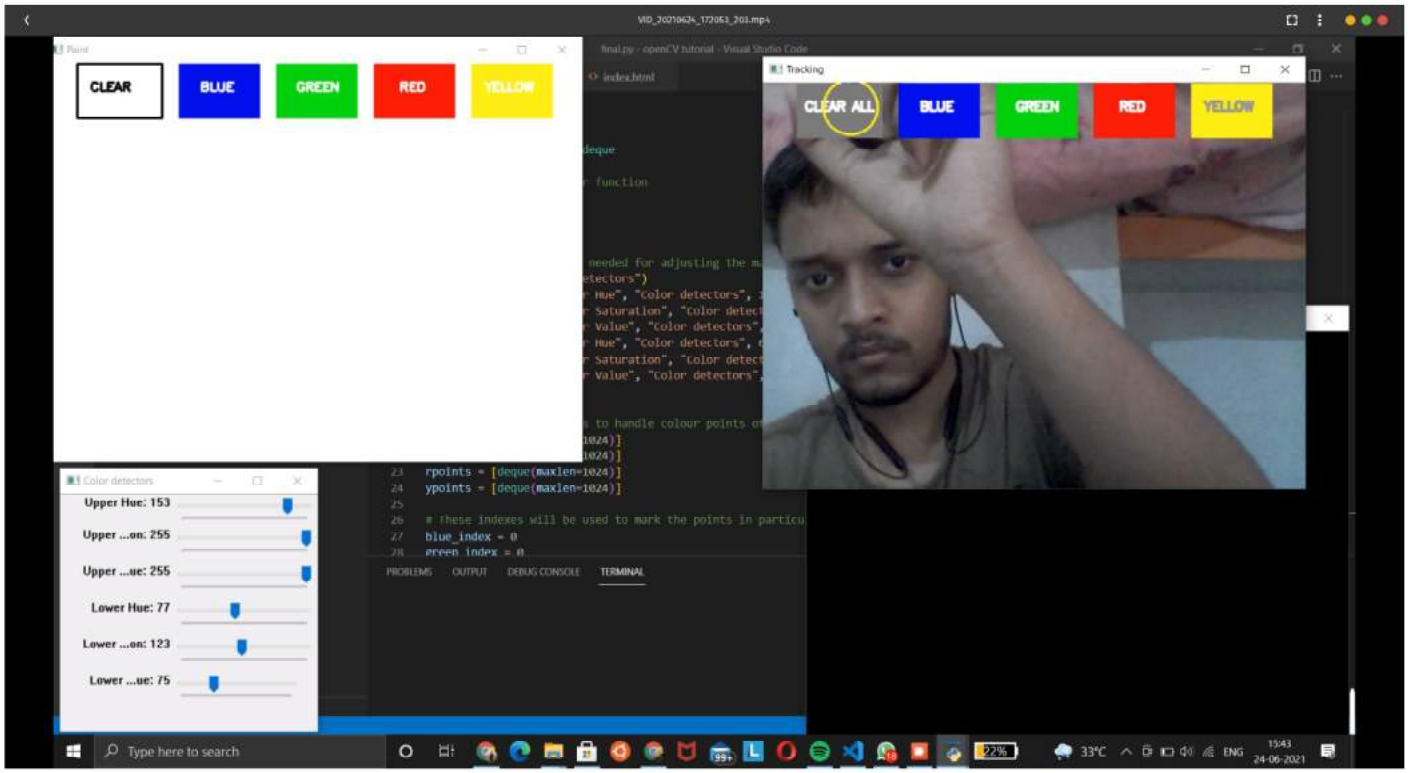
In the second picture we see how the marker is being tracked and how we are able to draw using different colors



In the picture below we see how we are able to change the color by selecting the corresponding button (Here green).



Similarly, we see how we can use the “CLEAR ALL” button to clear the screen/canvas



CHAPTER 7: CONCLUSION and FUTURE SCOPE

Conclusion:

While there has been an explosion of artificial intelligence-based software in the last few years, individuals such as children and the elderly often do not have access to these state-of-the-art machine learning models that are currently being developed. This project takes one step toward filling this gap by creating an intuitive and simple program that is customized for aiding the user in their visual expression.

Not only that, we are also trying to help the teachers with simple tools that will help them to carry on with their teaching during this pandemic with a bit more ease.

We are taking a step towards welfare of students and teachers too with trying to create a dustless learning environment(classroom).

Future Scope:

- Future work can be done on this project to let the user select a wider range of colors.
- We could work on the user being able to change the size of the “brush”.
- We could integrate deep learning for better hand tracking and try to track a particular finger of a user instead of a marker and to make it more accurate.
- In the end, we could build and deploy the project on some system like Raspberry Pi with required hardware to make this project work as a standalone model/unit which can be either used directly or used remotely.

REFERENCES

1. Books:
    i. Learn Python the hard way - Zed Shaw
    ii. Learning OpenCV - Adrian Kaehler and Gary Bradski
2. Research papers:
    i. Text writing in air - COMSATS Institute of Information and Technology
    ii. Paper Dreams: an Adaptive Drawing Canvas - MIT
3. Websites:
    i. https://theailearner.com/tag/cv2-inrange-opencv-python/#:~:text=Thresholding%20using%20cv2.&text=A%20pixel%20is%20set%20to,it%20returns%20the%20thresholded%20image.&text=Play%20around%20with%20the%20trackbars,inRange%20function
    ii. https://pythonprogramming.net/morphological-transformation-python-opencv-tutorial/
    iii. https://docs.python.org/3/tutorial/index.html