# **Virtual White Board: With movement detection of an object**

## Abstract

We have come up with a project which is titled “Virtual White-Board”, As the title itself indicates, we are not going to touch the surface, but we are going to draw on air and what we have drawn on air will be displayed on the screen/monitor.

We see people use ATMs which is very risky and there are chances of spreading of the virus at such point of times we can use this as a prevention method to a great extent. Not only in ATMs but also in offices for biometrics or in cyber cafes and many more areas we can use this project.

## Objectives

Our project's goal is to create a framework that functions as a virtual whiteboard by combining PC vision with drawing recognition.

Our model can detect movements. Text is created from writing that is done in the air.

We may do “movements” in front of a webcam continuously or in advance, and those signals would be turned into writing/converted art.

Acknowledgment of this project is that it considers clients who have defined requirements and uses an optional/elective form of communication.

Virtual reality platforms, such as the HTC Vive, offer a virtual whiteboard experience for a high cost and necessitate a complex monitoring system.

Our solution is more accessible and affordable because it only needs a PC and a camera.

## Assumptions

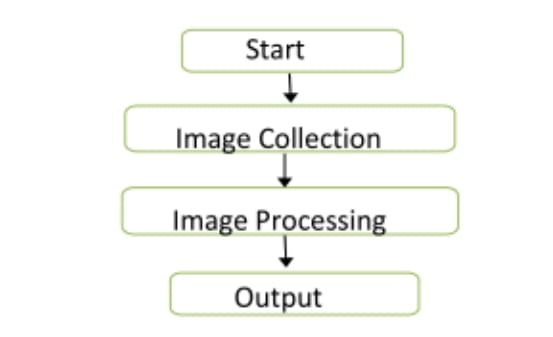
All we want in this undertaking is a PC with a web camera introduced in it. We will prepare our PC or our screen to peruse anything that the client will compose Infront of the screen. We have involved Open CV for object identification that is with which the client will be writing on air, and we have used Python language for coding.

The solution will work for only certain pen colours blue, green, yellow and red. You can readjust the hue, saturation etc properties. First and foremost, after code is executed, we get a white screen shown in the screen and anything we write on air Infront of camera it tracks the article whose properties we have proclaimed in the code. After that tracked points are connected and projected on the screen.

System Architecture

To develop this project, we'll be utilising OpenCV's computer vision methods. The preferred language is python because of its extensive libraries and simple syntax, but by grasping the fundamentals, it can be implemented in any language that supports OpenCV.

The colour whose characteristics we define in the code is recognised by the screen. H stands for hue, S for saturation, and V for value. The RGB colour coding is comparable to this one. In order to obtain the HSV value, you must first run the object tracking code.



The First step i.e., Start starts the beginning of the task. It demonstrates the running of the whole program.

The Second step i.e., Image Collection expresses that the catching of picture by means of web camera in PC gadget.

The Third step i.e., Image Processing states that getting to the datapoints Of the Images from the camera.

The Final advance is the Projection of Output on the screen which is finished by associating the followed focuses and extending it on the screen.

Requirements: python3, NumPy, OpenCV installed on your system.

OpenCV:

OpenCV (Open-Source Computer Library) is a programming featured library explicitly focused on at computer vision in real time. In the Python interface, all the most recent progressions and algorithms show up. It is significant open-source computer vision, machine learning and image processing library and now it plays an important part in real time activity in today’s systems. We will utilize this to process photos and recordings to perceive individuals, faces or even in handwriting.

Python:

Python is a high-level programming language. It is basically, renowned for code reusability and simplicity. Even though it is slower it has a significant

attribute of python that it very well may be handily

reached out inside c. It comprises of NumPy library. NumPy is exceptionally strong and upgraded for numerical tasks. To take benefits of multi-core figuring.

Modules:

A trackbar function to set values of x

Created trackbars for the color detection object with functions:

Every color has a particular hsv range with that we are detecting color.

# cv2.createTrackbar("Upper Hue", "Color detectors", 153, 180,setValues)

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

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

Created different dequeue to handle the colour points.

All the coordinates we get are stored in these blue, green, yellow and red dequeues.

We implemented the dilation process to remove the noise:

used a 5x5 kernel with full of ones I.e., if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases.

Based on the color we get we use the colors if its blue then (255,0,0)

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

colorIndex = 0

Creation of Paint Window which of white color:

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

With np.zeros we have created a array with the dimensions and then we added up 255 to show it as white sheet.

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)  
Then we subdivided them into further rectangles I.e., whatever the color we give or clear option to clear the color will be implemented in these rectangles

ret, frame = cap.read()

Created a camera reading instance

Upper\_hsv = np.array([u\_hue,u\_saturation,u\_value])

Lower\_hsv = np.array([l\_hue,l\_saturation,l\_value])

Converted the first frame to hsv and based on the trackbar position we have set lower and upper hsv values.

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

In the Live frame we have to add color buttons in every successive frame

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)

Created a mask with proper erosion and dilation help of inrange() function

cnts,\_ = cv2.findContours(Mask.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

center = None

Find the contour to create a center, if the length of that contour is greater than 0 then we find the min and max circle of the point

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

# Calculating the center of the detected contour

M = cv2.moments(cnt)

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

Then with the help of movements of the circle we find the center

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)]

If center is less than 65 I.e., it is telling we are going to choose any of the option between red, blue, green, yellow and clear.

bpoints.append(deque(maxlen=512))

blue\_index += 1

In the else part we draw with the help of color index, here we get to know with which color to draw

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)

Draw lines of all the colors on the canvas and frame, we go through all the deques and find the points to know which has to be colored.

cv2.imshow("Tracking", frame)

cv2.imshow("Paint", paintWindow)

cv2.imshow("mask",Mask)

With imshow() function we show paint, window and mask.

Contribution of each member of the team

**Sai Sarvagna B (AM.EN.U4CSE20316)**

Worked on:

Canvas setup

Reading the frame from the camera

Adding the colour buttons to the live frame for colour access

Identifying the pointer by making its mask

Find contours for the pointer after idetifying it

**Devesh Kumar V V(AM.EN.U4CSE20321)**

Worked on:

If the contours are formed, Draw the circle around the contour

Calculating the center of the detected contour

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

Append the next deques when nothing is detected to avois messing up

Draw lines of all the colors on the canvas and frame

**Shashank Y(AM.EN.U4CSE20363)**

Worked on:

trackbar function

Creating the trackbars needed for adjusting the marker colour

different arrays to handle colour points of different colour

The kernel to be used for dilation purpose

Show all the windows( Tracking, paint and mask)

Input to the system

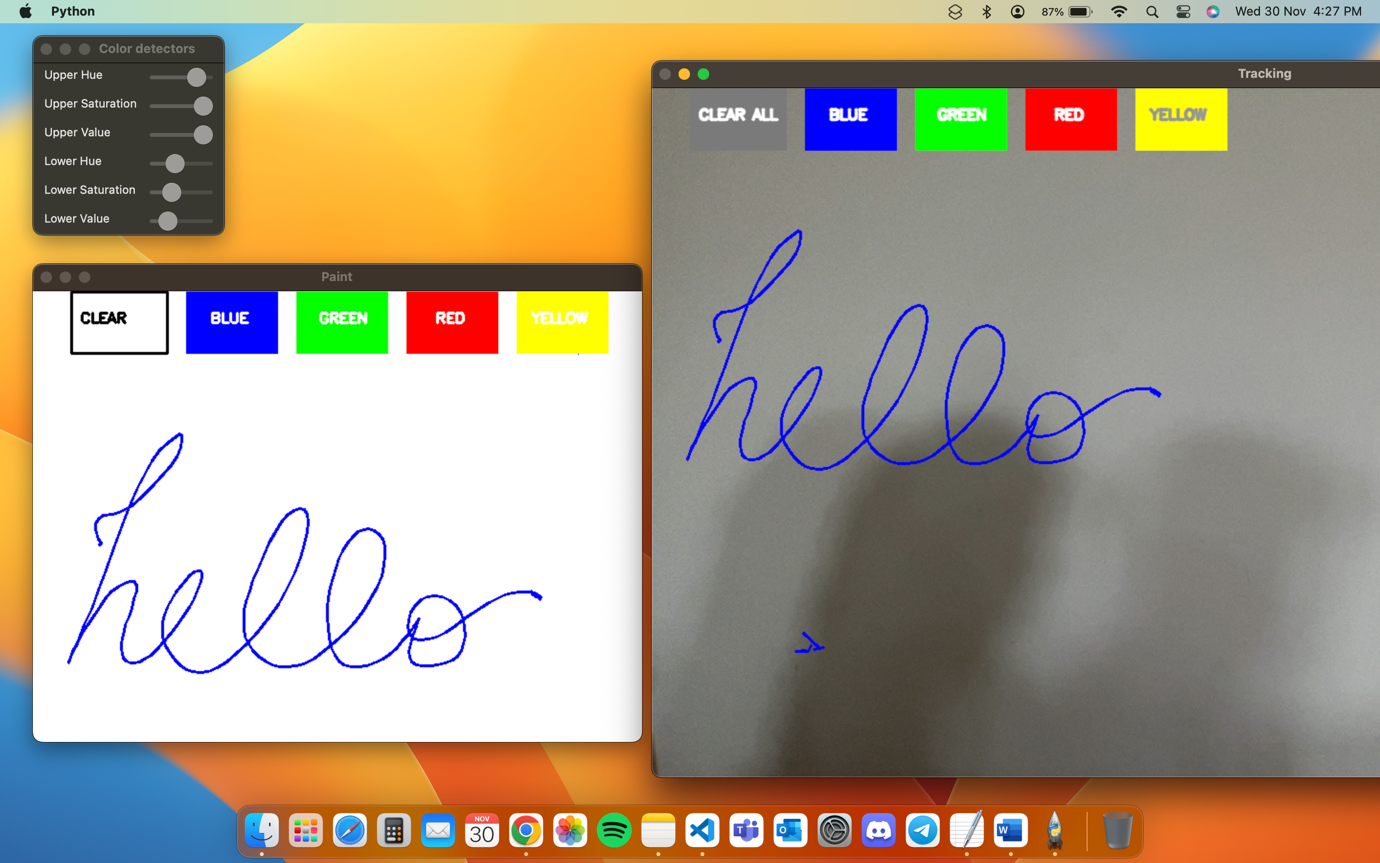
Images or Video - Describe what kind of images/videos(also mentions any limitations);

Any other input

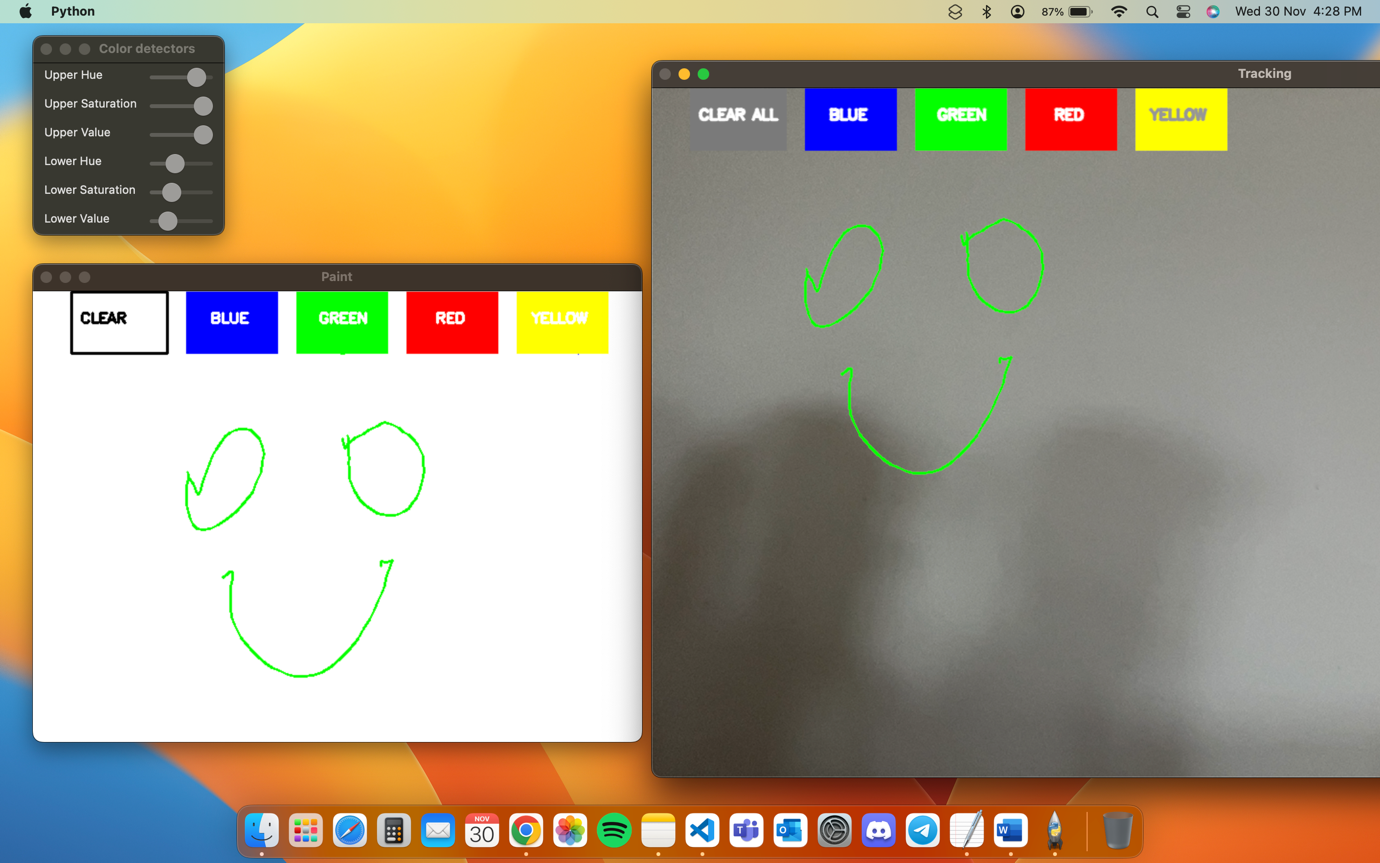
Provide snapshots of each category of input

INPUT with “BLUE” as the Colour Detector

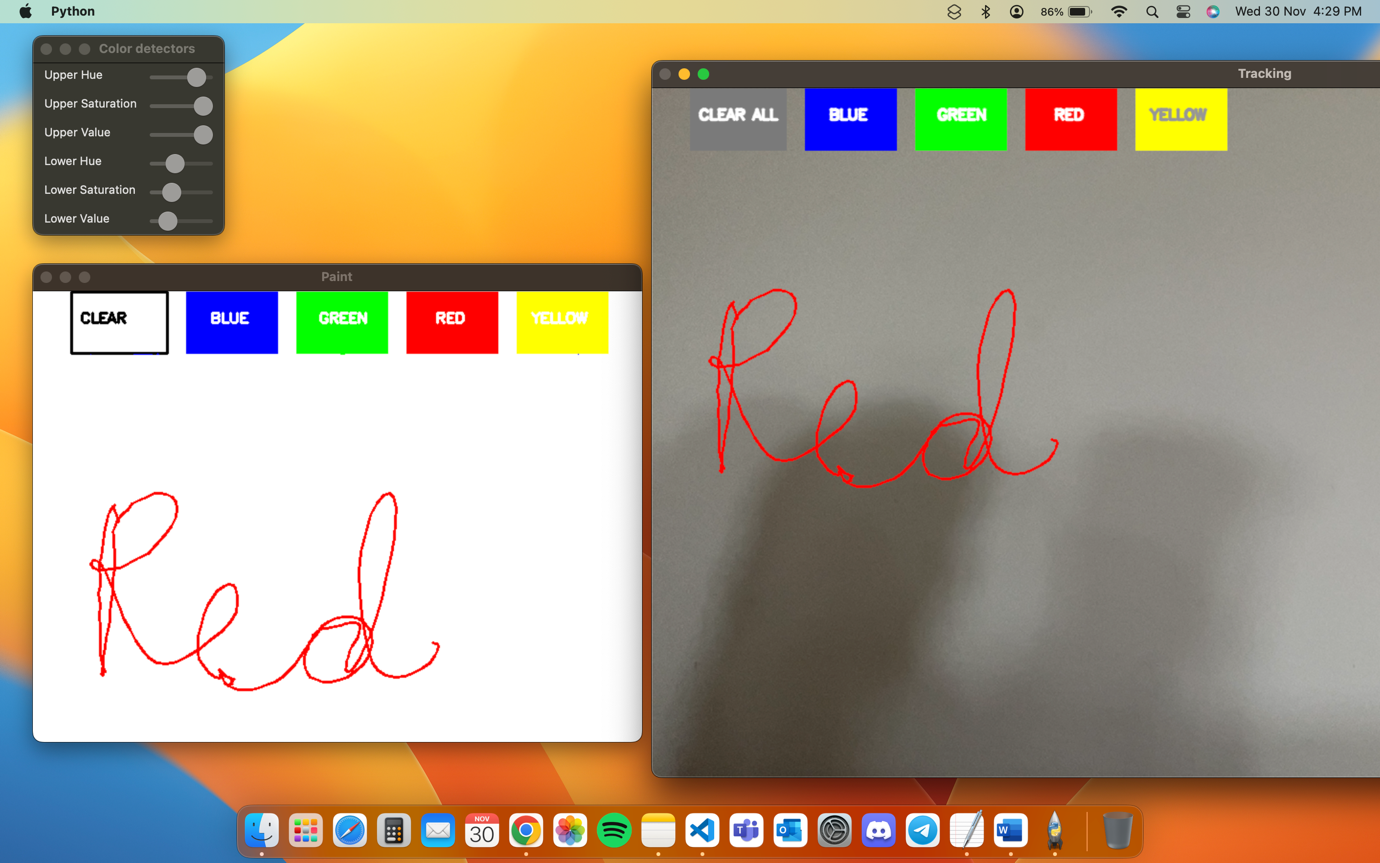
Canvas Colour Blue



Canvas Colour Green



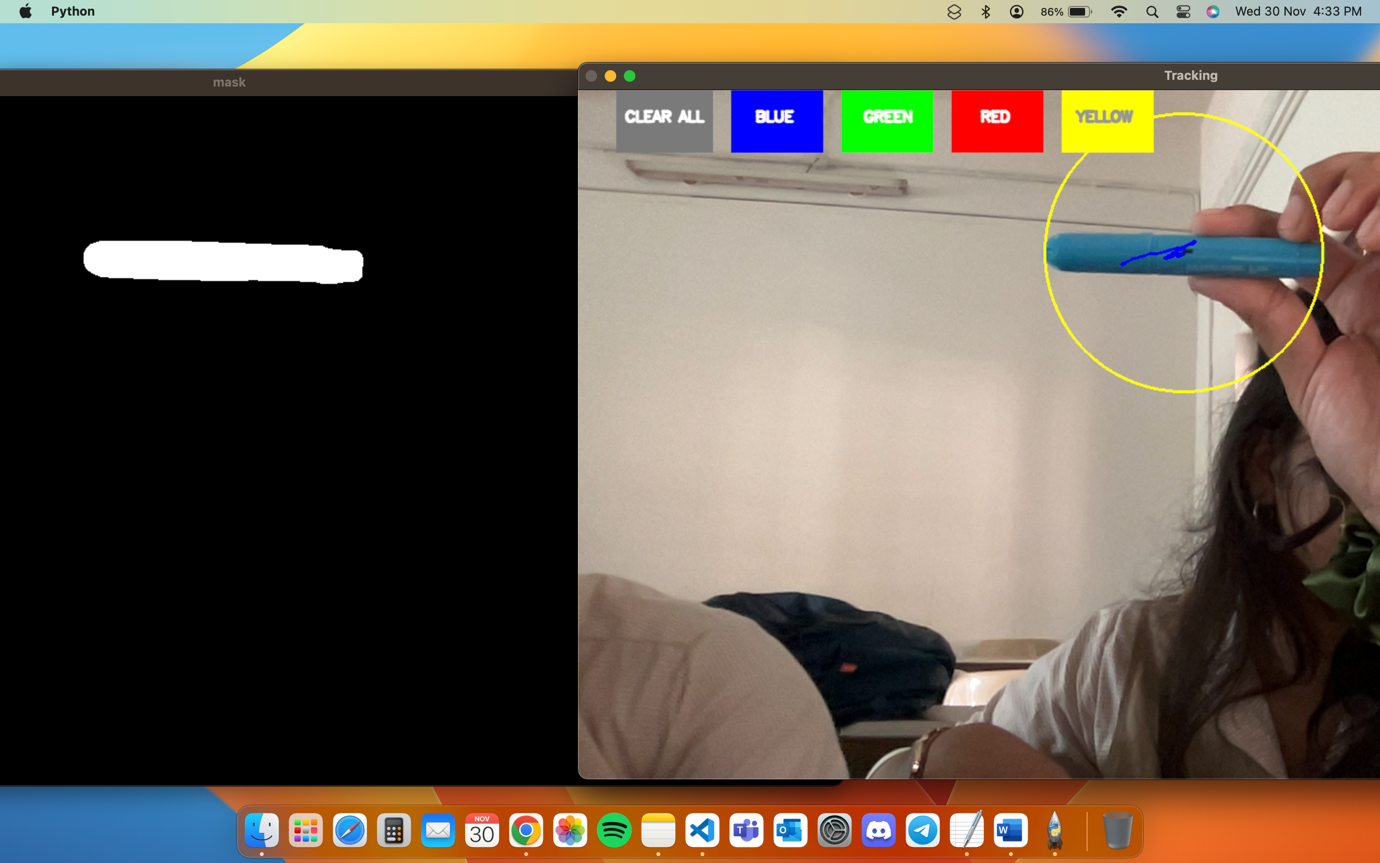
Canvas Colour Red



Canvas Colour Yellow



Object detection with blue As the Colour Detector



Object detection with RED As the Colour Detector

