Digital Image Processing

VIRTUAL WHITE BOARD

TEAM MEMBERS

B SAI SARVAGNA - AM.EN.U4CSE20316

DEVESH KUMAR V V - AM.EN.U4CSE20321

SHASHANK YAKKANTI - AM.EN.U4CSE20363

ABSTRACT

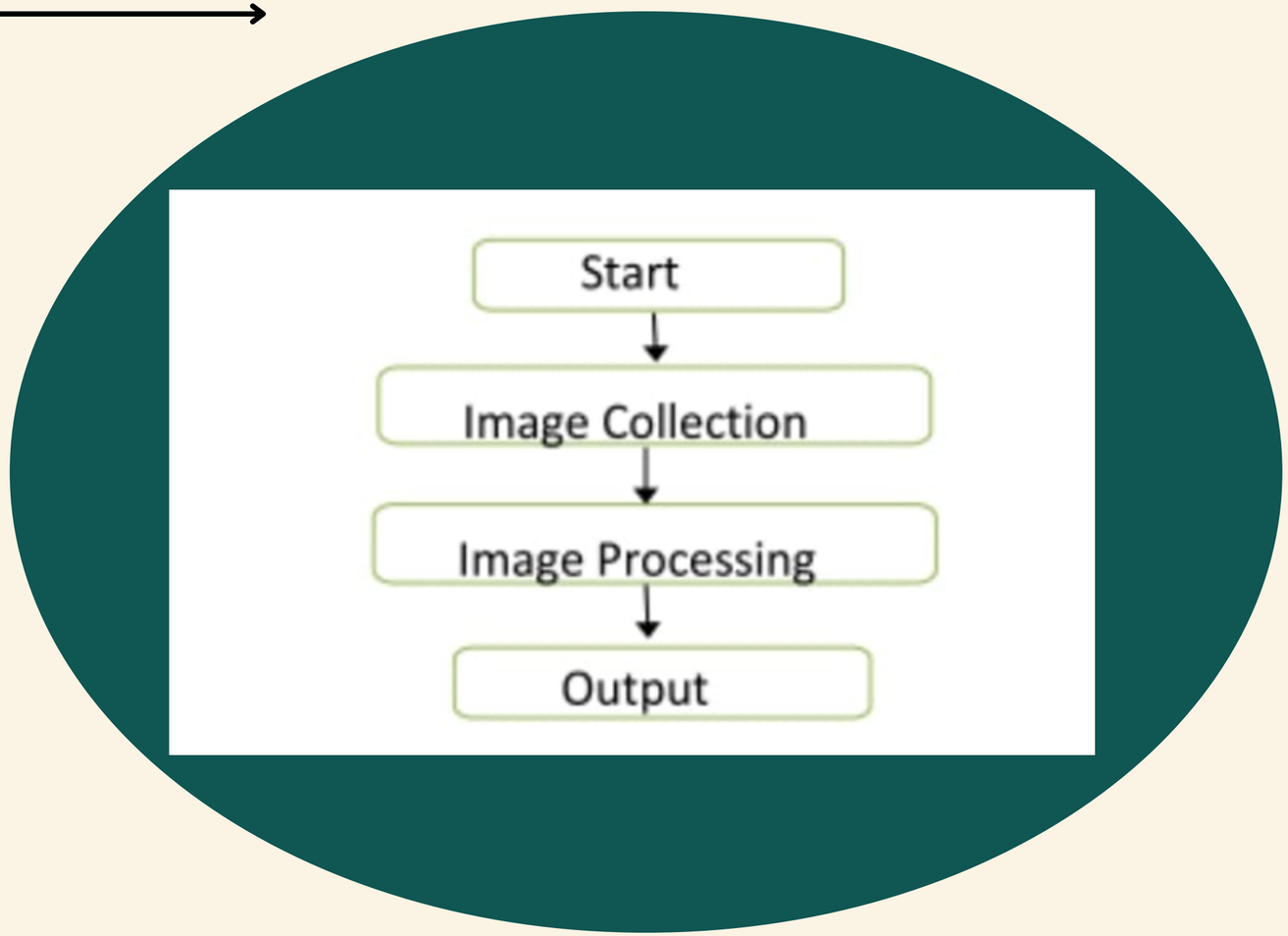
We have come up with a project which is titled “Virtual White-Board”, As the title itself indicates, we are not going to touch the surface, but we are going to draw on air and what we have drawn on air will be displayed on the screen/monitor.

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.

SYSTEM ARCHITECTURE



- 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.

MODULES EXPLANATION

***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 th white region in the image or size of foreground object increases.

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

Created a camera reading instanc

ret, frame = cap.read()

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

```
Upper_hsv = np.array([u_hue,u_saturation,u_value])
  Lower_hsv = np.array([l_hue,l_saturation,l_value])
```

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

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

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

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

```
cnts,_ = cv2.findContours(Mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
center = None
```

```
# Ifthe 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']))
```

- 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.

```
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)]
```

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

```
bpoints.append(deque(maxlen=512))
    blue_index += 1
```

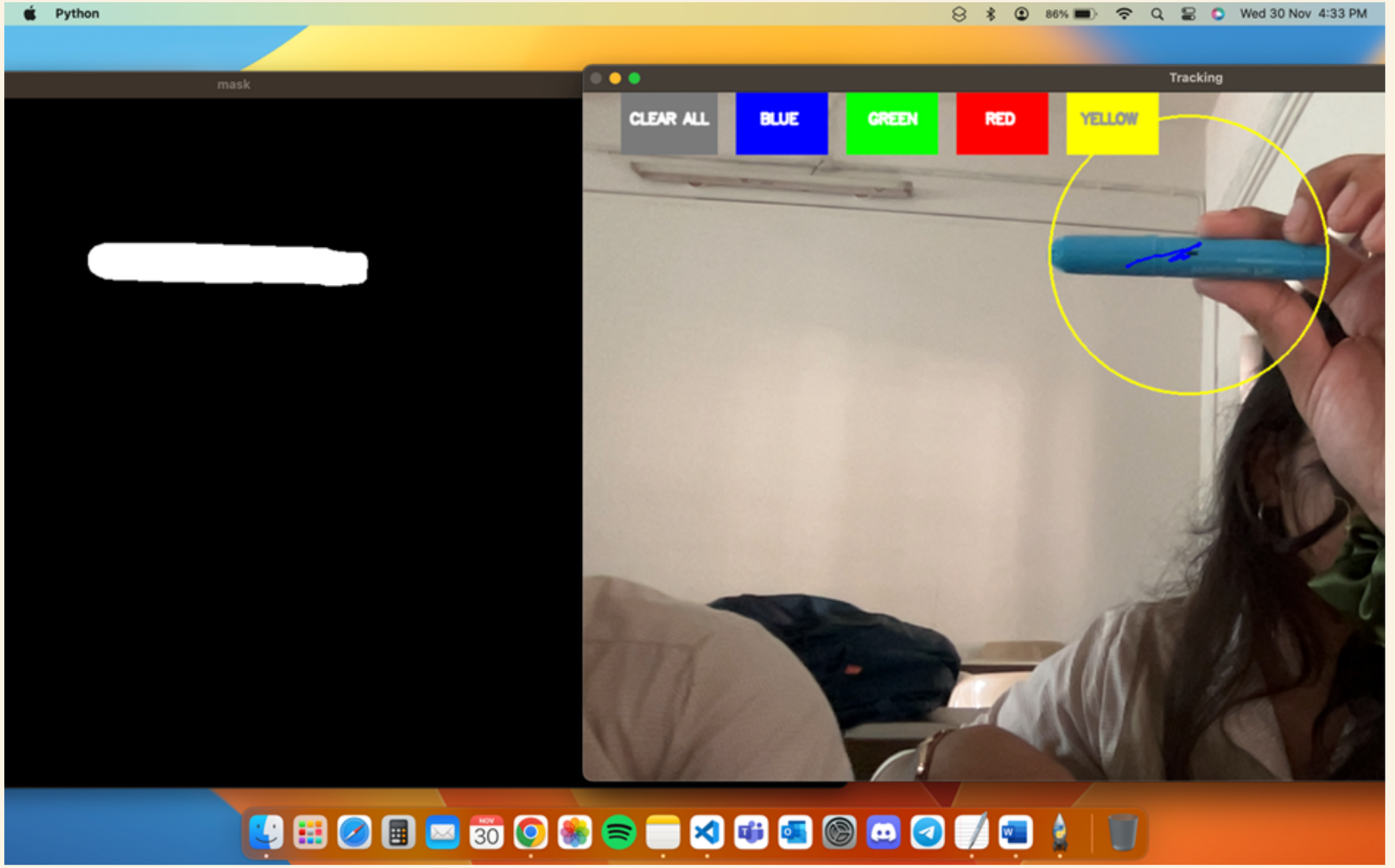
So far we stored all the points in their respective color deques. Now we just join them using a line of their own color. The OpenCV function cv2.line() comes in handy for us to do that. The following code does the same.

```
    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)
```

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

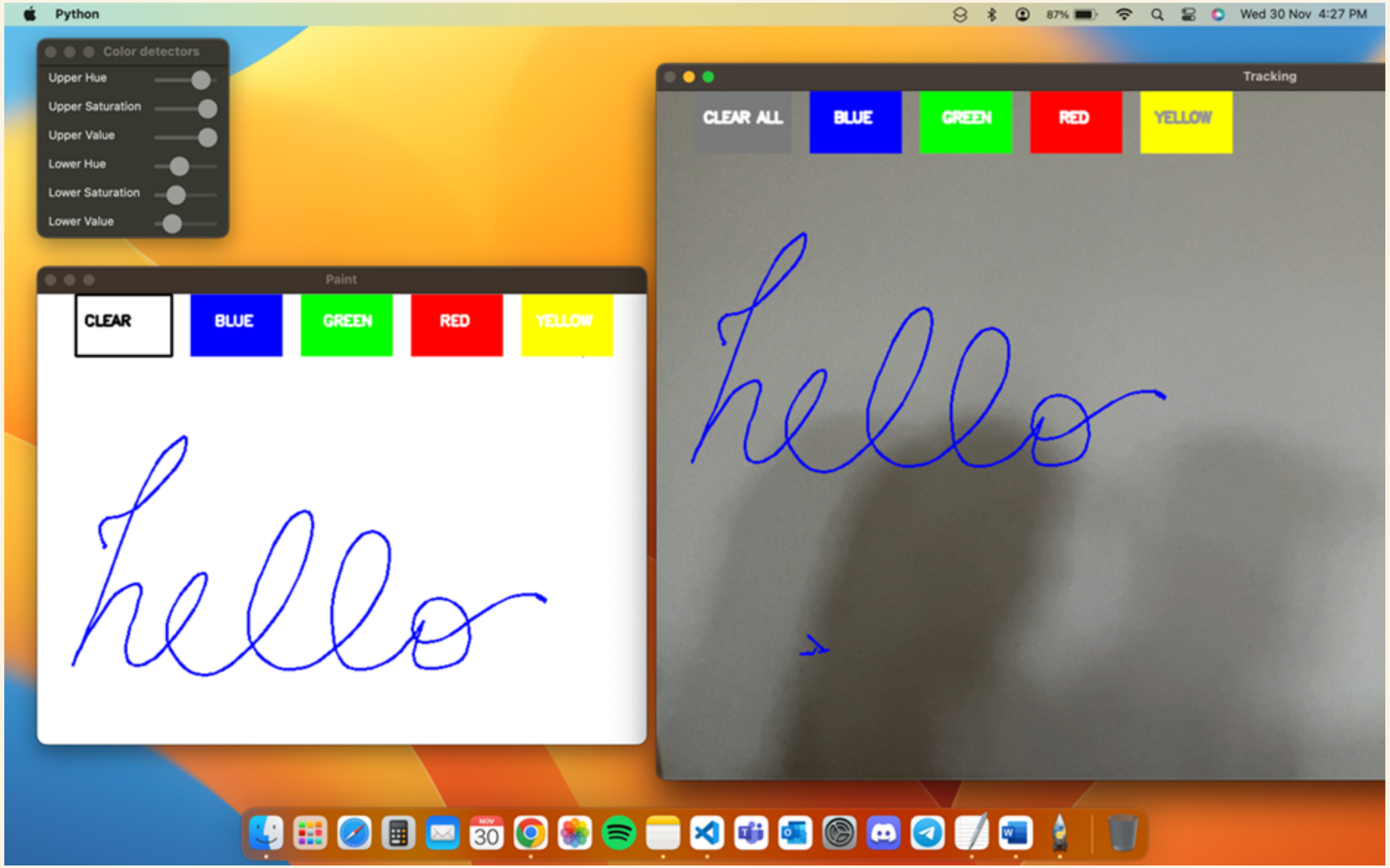
```
cv2.imshow("Tracking", frame)
    cv2.imshow("Paint", paintWindow)
    cv2.imshow("mask",Mask)
```

Object detection with BLUE As the Colour Detector



Input to the system

INPUT with “BLUE” as the Colour Detector



Input to the system

INPUT with “GREEN” as the Colour Detector

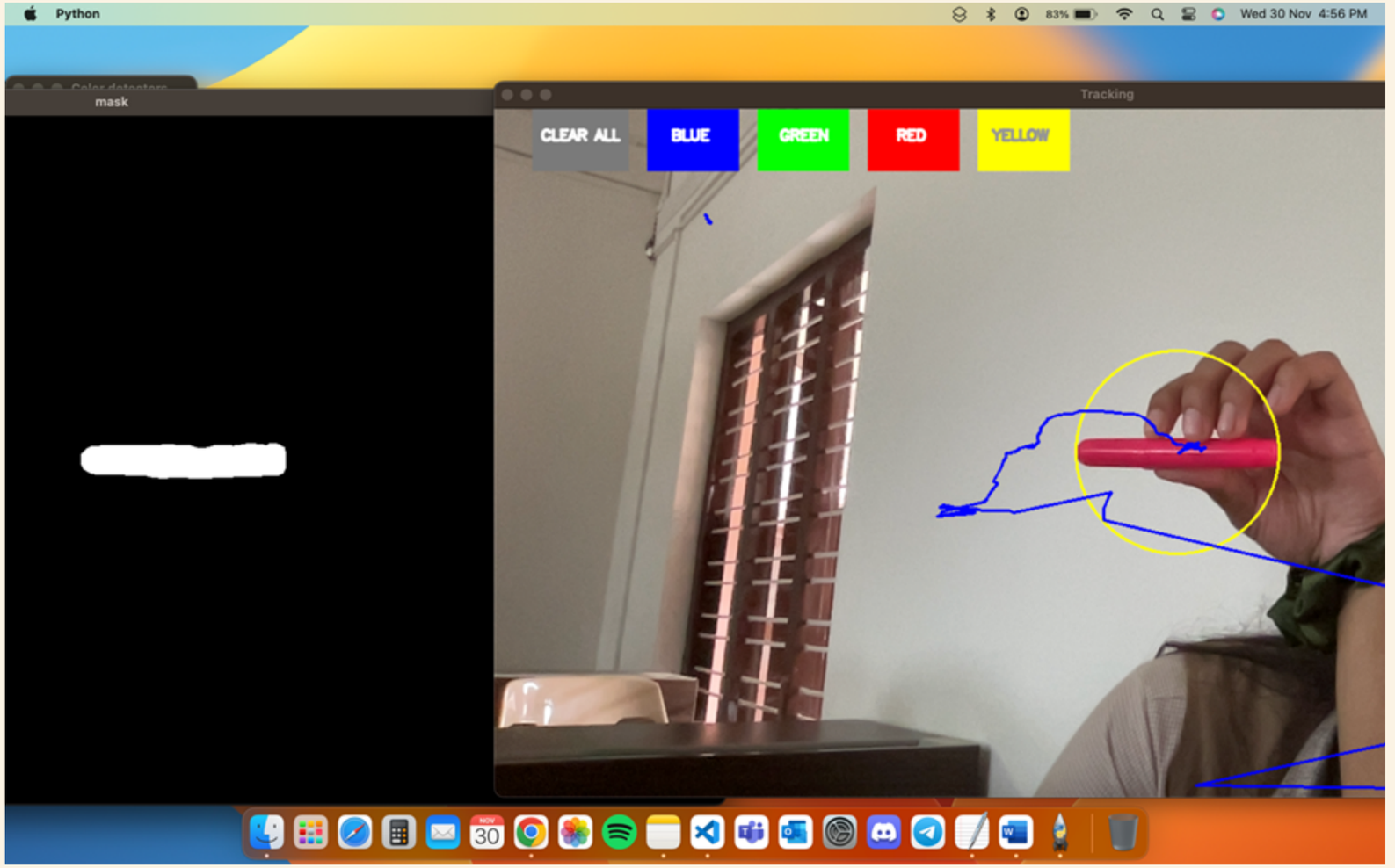
Input to the system

INPUT with “RED” as the Colour Detector

Input to the system

INPUT with “YELLOW” as the Colour Detector

Object detection with RED As the Colour Detector



Thank You!!