Mouse Control Using a Webcam based on Color Detection in OpenCV Python

1. Abstract:

In briefly computer vision is high - level understanding to computer from digital images and videos. That is an application on computer vision in machine vision, object identification and tracking. The whole project is based on python programming language and importing some libraries like open cv, pyplot from matplotlib, numpy and pyautogui module. Computer vision based mouse can control and command the cursor of a computer or a computerized system using a camera. This method mainly focuses on the use of a Web Camera to develop a virtual human computer interaction device in a cost.

2. Introduction:

The current generation is the touch generation. In this generation, everything works with the touch of a finger; be it our cell phones, computers or any day to day appliance. But the so called "touch generation" has reached its ultimate state and is soon likely to decline. The generation of “intangible interfaces" is just at its dawn and is soon going to take over the touch generation. Presently we can control our laptop without using a mouse or a keyboard by detecting colors and following it’s movement. In this project, we will practically know the concepts of computer vision technology and build computer vision based Mouse hands on and control the cursor by using the object tracking algorithm. The yellow color on the forefinger will be used for moving the cursor, the green color on the thumb will be used for the left clicking action and the red color on the thumb will be used for the right clicking action. It is a python code, when we run the program a grid that detects various colors gets activated. At first it is required to know well the basic concepts of an image, as the video is just representation of moving visual images. Computer vision Based Mouse is a system to control the cursor of your computer, that can be achieved by having colored markers on your palm, then capturing the video of the motion of your palm, using the webcam of your computer. You will track the colored markers, and use their motion, to control the cursor of your computer! You can use 3

colors, for the 3 typical actions of the mouse, Color 1 for cursor movement, Color 2 for left click, and color 3 for right click.

3. Keywords:

Based mouse, OpenCV, Color Detection, Web Camera, Computer Vision, Object Tracking, Cursor Control, Image Acquisition

4. OBJECTIVE:

Intangible interfaces are those which require no touch. They are soon going to take over the "touch" devices. Many developments have been seen in this field of Human-Computer Interaction lately. "Mouseless", an invisible computer mouse, provides the familiarity of interaction of a physical mouse without actually requiring a real hardware mouse. The device eliminates the requirement of having a physical mouse altogether but still provides the intuitive interaction of a physical mouse that users are familiar with. The vast scope of intangible interfaces was the key motivation for the project. The objective of this project is to make the human computer interaction more intuitive but keeping the entire process economic. The aim in each step is to make the interaction more and more natural to reduce the gap between a human mental model and the process of performing the task. The project focuses on mouse control. The aim is to design color detection and tracking for performing the mouse functions.

5. DIGITAL IMAGE PROCESSING

In image processing, a digital image is the input. This image is processed for noise reduction, filtering. Example, enhancing a blurred image to get a clear photo. Digital images are also used for identification purpose using image processing.

5.1 Image Processing:

For best performance, always use the NumPy library. A digital image is nothing more than data -- numbers indicating variations of red, green, and blue at a particular location on a grid of pixels. . You will remove any noise from the image, using Gaussian Smoothing.

Remember that, you are going to track the objects, based on their specific color. The default images in Open CV, are in the B G R color space. But, it is easier, and better, to identify colored objects, in the H S V space. So, you will first convert the image, from the B G R, to the H S V color space. Finally, you will separate the objects of a specific color, or a color range, from the image. You will do a thresholding operation, to identify the objects of the specific color range.

5.2 Image Analysis:

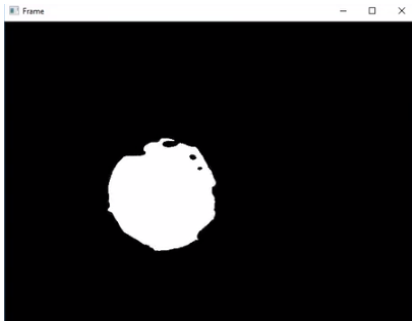
Image analysis is the extraction of meaningful information from images; mainly from digital images by means of digital image processing techniques. Image analysis tasks can be as simple as reading bar coded tags or as sophisticated as identifying a person from their face. In this project we do various image analyzing techniques. The main thing done is the color detection. At first we receive an image from the web cam. Then each pixel is retrieved from the image and extracts the red, green and blue values (RGB) from each pixel. Now we can easily detect a particular color since all the colors are combinations of RGB values. Here we just to try to detect only Red Green and Blue colors. This is done by traversing through the image, retrieving each pixel, extracting RGB values and then comparing the color values to the RGB values. Digital image processing is the use of computer algorithms to perform image processing on digital images. As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and signal distortion during processing.

6. SYSTEM DESCRIPTION:

6.1. Color Detection:

You will now track a color object, yellow in this case. You can use an object with a color of your choice. You will first try to define the HSV value of the colored object. To find the HSV value you

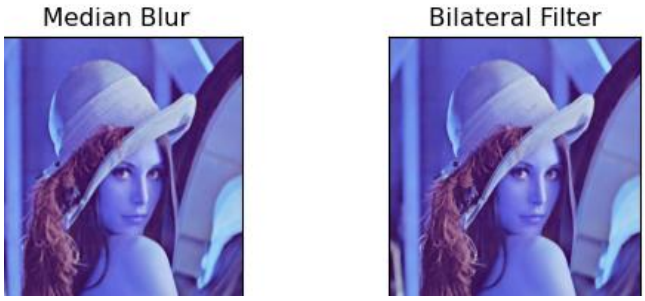
need the image to be plotted in RGB and use any online tool to convert the RGB. By this method you will be able to find the upper and lower range of the yellow color. You will now find the yellow colored object by extracting only the pixels which lie in a color range. Now to extract all the pixels in a particular region, you will use the inRange function



6.2 Filtering the Noise

The first step is to remove the noise in the image with a 5x5 Gaussian filter. Finding Intensity Gradient of the Image. The Smoothened image is then filtered with another technique, to get the first derivative of the pixel intensity in the horizontal direction and the vertical direction. Hysteresis Thresholding, This stage decides which are all edges are really edges and which are not.

Image blurring is useful for removing noise from the images. It actually removes the high frequency content like noise and edges from the image. Open CV provides mainly four types of blurring techniques. We will use the function cv.medianBlur to apply the median filter. Finally, we will be using a bilateral filter on an image. Bilateral Filtering is highly effective in noise removal while keeping edges sharp. So, we will use the function cv.bilateralfilter to apply the filter on the image.



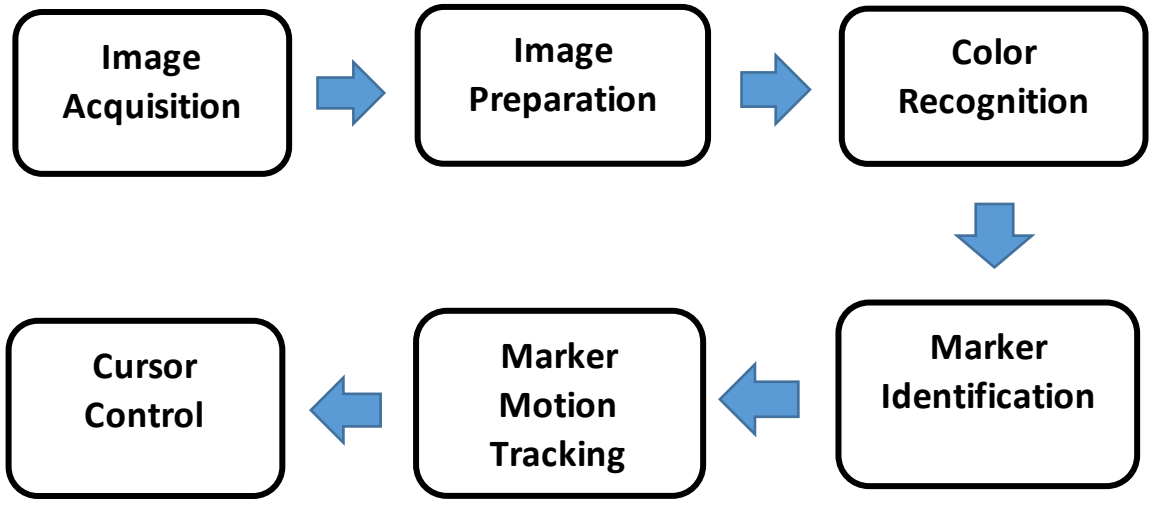
7. METHODOLOGY:

The implementation has been divided into various steps and each step will be explained below. The system flow explains the overview of the steps involved in the implementation of mouse control.

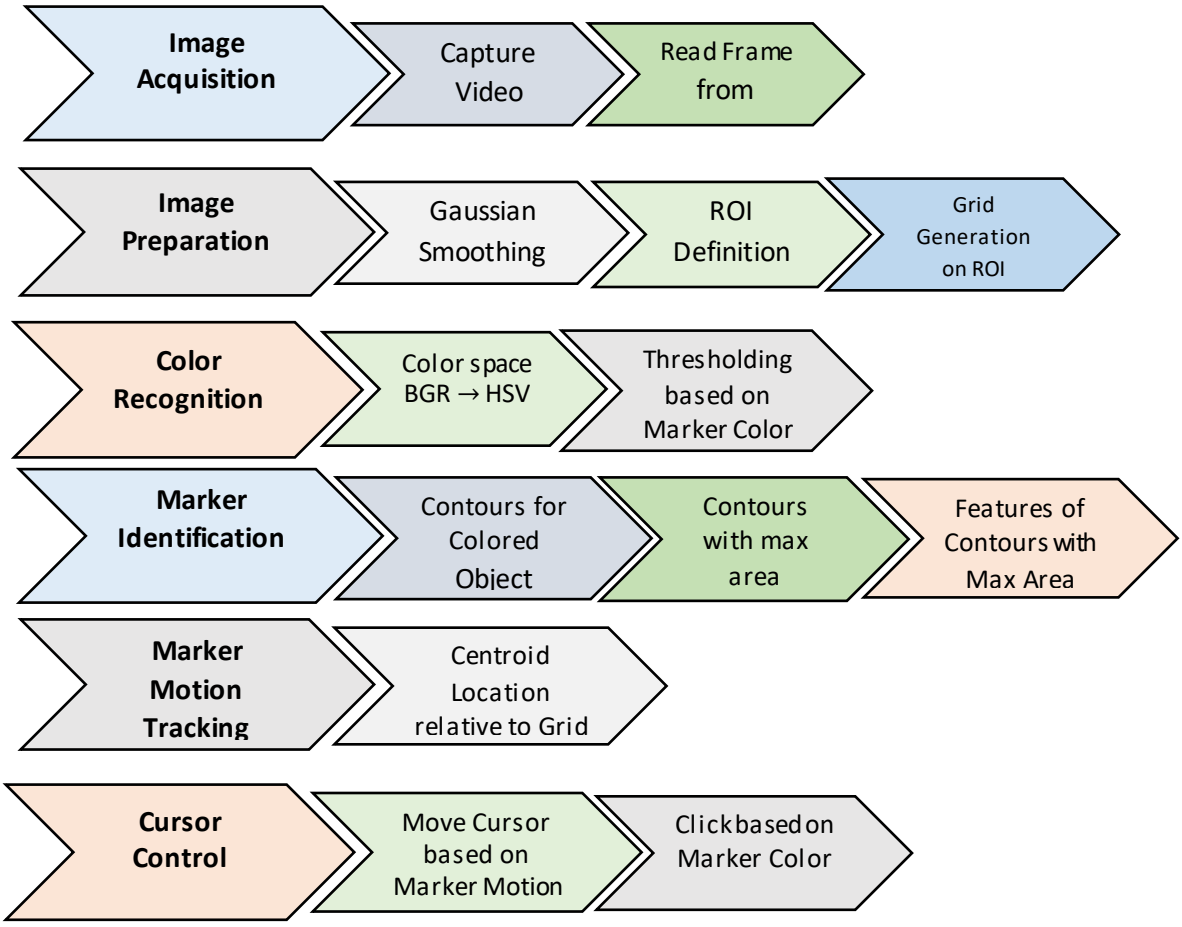
7.1 System Flow

We will now learn about the pipeline, for implementing the Mouse Control. The pipeline for CV based mouse, has 6 stages. They are:

Pipeline for computer mouse control



7.2 System Approach



7.2.1. Image Acquisition

Image Acquisition which involves acquiring the images, that will have the markers on your palm, that need to be tracked. To capture a video, you need to create a Video Capture object. Its argument can be either the device index, or the name of a video file. Device index, is just the number to specify which camera connected to your computer is being used for video capture. Normally one

Video Capture

- ***Cam = cv2.VideoCapture(index)***
- Creates Video Object
- Index → Index of Camera connected to computer or name of the video file
- cam → Video Capture object

camera will be connected. So you simply pass 0. You can select the second camera by passing 1, and so on.

After that, you can capture frame-by-frame. You will then use the read function, of the video capture object, to read the image from the video being captured. The function gives two outputs. It returns a boolean, ret, which is true, when the frame is read correctly. It also gives out the frame from the video captured, as an output. The captured image, is stored in the variable, frame. You can now perform the usual image functions on the variable, frame.

Read frame from Video

- ***ret, frame = cam.read()***
- Reads frame from the video
- ret → boolean output. True, if frame is read correctly. False otherwise
- frame → image with the frame of the video

7.2.2. Color Recognition

In the stage of Color Recognition, You will now convert the image, into a different color space. There are more than 150 color-space conversion methods, available in Open CV, but you will see the basic ones.

For color conversion, you will use the cv.cvtColor function. The first argument for the c v t Color function, is the image file that needs to be converted. The second argument, is the code for the conversion type. There are numerous color spaces that Open CV can handle, and therefore, quite a few conversion codes. You will convert the image from B G R, to H S V, to ensure better operations based on color. So you will use the code, cv.color_BGR2HSV. The output of the c v t Color function, is an image, in the H S V color space.

Color space
BGR → HSV

- img_hsv = cv2.cvtColor(img, conversion_code)
- convert one image from one color space to the other color space
- img → inputl image
- conversion_code → color conversion code based on color spaces. For ex: cv.COLOR_BGR2HSV
- img_hsv → converted image

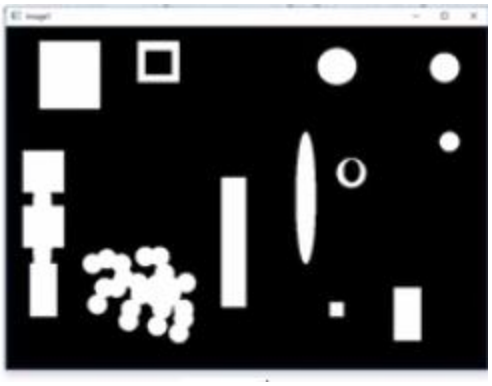
Now, to extract only the pixels of a particular color range, you will use the inRange function. You will first determine the range of H S V values, that your colored object lies in - the lower bound, and the upper bound. You will then use the in range function, to convert the H S V image, to a binary image, which has only the pixels of a particular color.

The in range function, takes three inputs: the original array, the lower bound, and, the upper bound. The function gives an array with the same shape as the original array the elements, corresponding to the elements of the original array, the elements, corresponding to the elements of the original array, lying within the bounds, are given the value of 1 and, the elements, corresponding to the elements of the original array, beyond the bounds, are assigned the value of 0.

Thresholding based on Object Color

- img_threshold = cv2.inRange(img, lower_bound, upper_bound)
- convert pixels based on their values
- lower_bound < pixel value (in img) < upper_bound → pixel value = 255
- Else, pixel value = 0

So, as you can see, the in range function, applied for green color, picks only the objects in green color, and removes the rest of the objects.



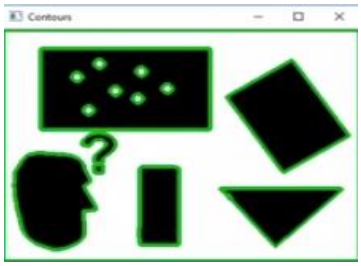
7.2.3. Object Identification

In the stage of Object Identification, Now that you converted the original image into a binary image, you will find the contours of the colored objects, from the binary image. To find the contours, you will use the cv.find contours function. There are three arguments in cv.find Contours function. The first one, is the source image in a binary form, from which the contours need to be found.
The second argument, is the con tour retrieval mode, which we had learnt about earlier. And, the third, is the contour approximation method.

Contours for Colored Object

- cv.findContours(img, mode, method)
- Finds contours in an image
- img → source image
- mode → contour retrieval mode
- method → contour approximation method
- Returns the following:
  - Img2 → resultant image with contours
  - contours → Array of detected contours
  - method → relationship between contours

The find con tours function, outputs a modified image, the contours, and the 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. Hierarchy, is the representation of the relationships between different con tours, found from an image. The con tours identified, can be displayed on an image, using the draw Contours function, as shown.



Finally, you will find the contour with the maximum area, as the contour of interest, and identify it as the object. You will ignore the smaller contours, as they might be due to some error in setting the bounds, or some extraneous pixels that do not belong to the specific object. You can then use con tour features, like centroid, bounding rectangle, to track the motion of the object. This finishes the programming logic, on how to track a colored object.

Find Contours with ax Area

- Ignore the smaller contours as errors

Calculate and use Features of contour with Max Area

- Centriod
- Bounding Rectangle

7.2.4. Color Recognition

In the 4th stage of marker identification, you will track the motion of the marker in different frames, by locating the centroid of the marker, in the Region of Interest. Finally, you will

move the cursor, based on the marker motion. You will also click the mouse, based on the color of the marker, as mentioned earlier. This finishes the detailed pipeline, for implementing a Computer Vision based Mouse.

In the stage of Image Preparation, after acquiring the image, and removing the noise, you will create a region of interest, and a grid, in that region. This will increase the accuracy of the marker identification, and tracking. The grid will help you in translating the position of the marker, to cursor movement. Let us now see how the motion of the marker is tracked.

7.2.5. Marker motion tracking

In the stage of Marker motion tracking, you will use the contour features, to find the contour around the markers on your palm. You will then find the centroid of the marker. Finally, you will find the location of the centroid, in the grid that you had created. It can fall in any of the 9 regions as shown.

- ***Find Location of Marker Relative to Grid***
- Top - Left
- Top – Center
- Top – Right
- Mid- Left
- Mid – Center
- Mid – Right
- Bottom - Left
- Bottom – Center
- Bottom – Right

If it is in the top left region, the cursor should move towards the top left part of your screen. If the marker is in the top center region of the grid, the cursor should move towards the left of your screen, and so on. If you hold the marker in the central region, the cursor should not move, and stay still. This is how you can control the cursor, based on the marker motion. Now, how do you actually control the cursor?

7.2.6. Cursor control

In the stage of Cursor control, you will use the **pyautogui** library of python, to control the mouse cursor, and also click, based on the color of the marker. The mouse functions of PyAutoGUI use x- and y-coordinates of the pixels on the screen, just like any image. You can also determine the mouse's current position by calling the pyautogui.position() function.

Finally, we can control the motion of the mouse cursor. The pyautogui.moveTo() function will instantly move the mouse cursor to a specified position on the screen. Integer values for the x- and y-coordinates make up the function's first and second arguments, respectively. An optional duration argument specifies the number of seconds it should take to move the mouse to the destination. If you leave it out, the default is 0 for instantaneous movement. Note that all of the duration keyword arguments in PyAutoGUI functions are optional.

moveTo:

Syntax: pyautogui.moveTo(x, y)

Output:

Current mouse cursor position.

You will now write for loop to move the mouse cursor clockwise in a square pattern for a total of ten times. As you had passed a duration as 0, to the pyautogui.moveTo() function, the mouse cursor would instantly teleport from one point to the next point..

8. EXPERIMENTAL RESULTS

We can control the mouse using the yellow marker, and we can just flash the green marker, whenever you need to left click, and we can just flash the red marker, whenever you need to right click. By making sure there is no interference of any background object, with the marker colors.

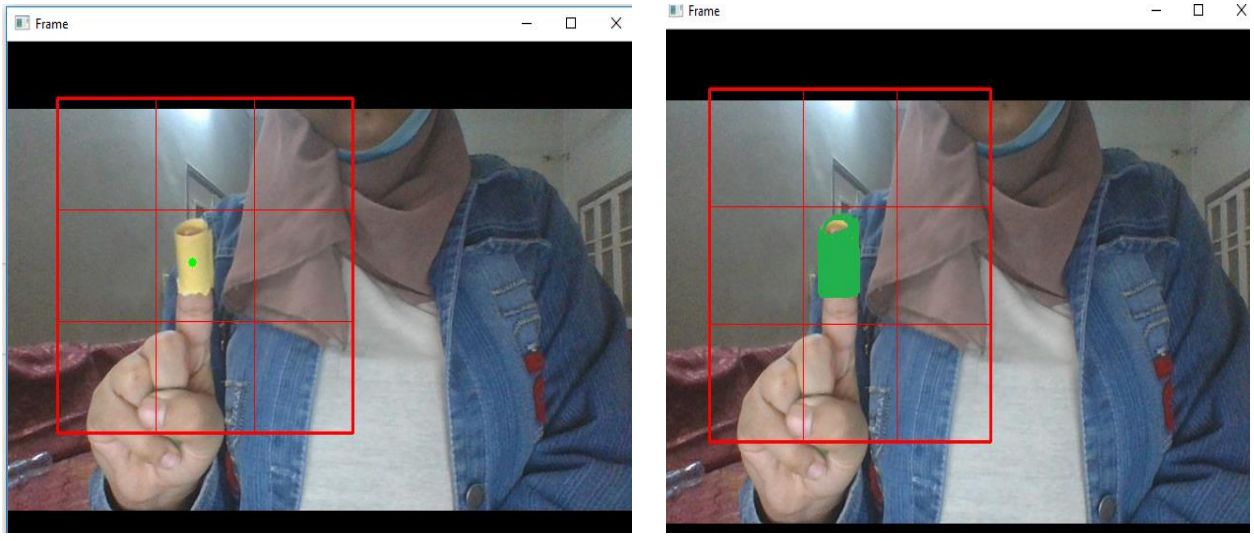
```
import cv2 as cv
import numpy as np
import pyautogui
from matplotlib import pyplot as plt
cam = cv.VideoCapture(0)
lower_yellow = np.array([20,100,100])    ##Yellow_Move Cursor Pointer
upper_yellow = np.array([40,255,255])
lower_green = np.array([50,100,100])    ##Green _for Clicking(left)
upper_green = np.array([80,255,255])
lower_red = np.array([0,125,125])    ##red _for Clicking(right)
upper_red = np.array([10,255,255])
while(True):
    ret, frame = cam.read()##in usual the cursor move in the opposite direction
    frame = cv.flip(frame, 1)  ## that to avoid that problem
#smothen the image
    image_smooth = cv.GaussianBlur(frame,(7,7),0)
```

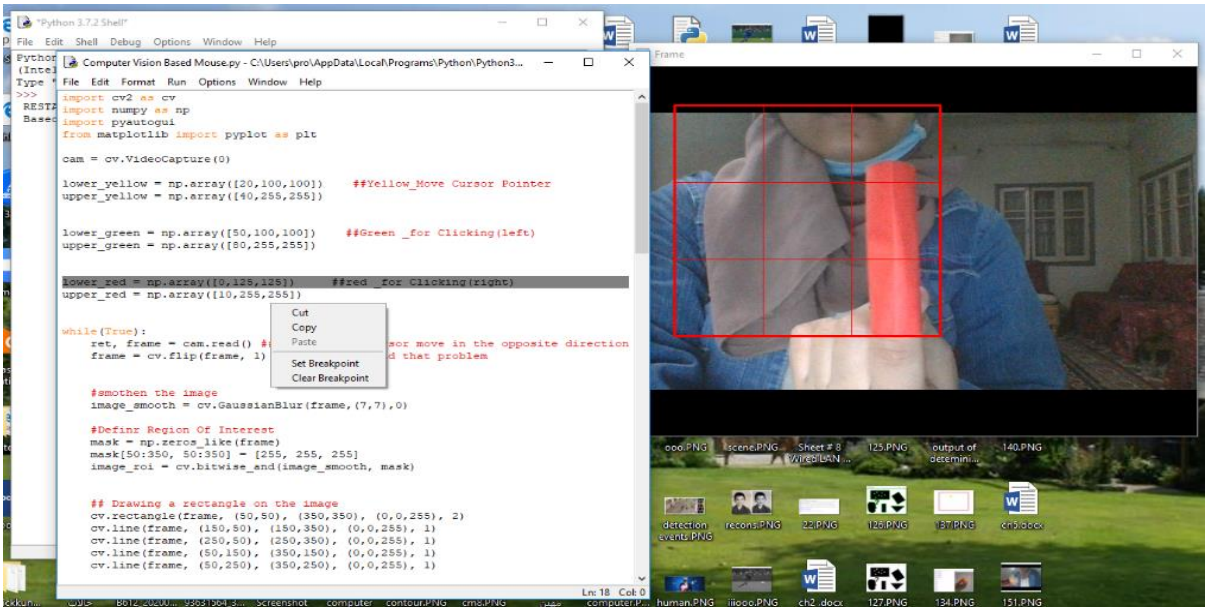
```
#Definr Region Of Interest
    mask = np.zeros_like(frame)
    mask[50:350, 50:350] = [255, 255, 255]
    image_roi = cv.bitwise_and(image_smooth, mask)
    ## Drawing a rectangle on the image
    cv.rectangle(frame, (50,50), (350,350), (0,0,255), 2)
    cv.line(frame, (150,50), (150,350), (0,0,255), 1)
    cv.line(frame, (250,50), (250,350), (0,0,255), 1)
    cv.line(frame, (50,150), (350,150), (0,0,255), 1)
    cv.line(frame, (50,250), (350,250), (0,0,255), 1)
    ##Threshold the image for yellow color
    #img_hsv = cv.cvtColor(image_smooth, cv.COLOR_BGR2HSV)
    image_hsv = cv.cvtColor(image_roi, cv.COLOR_BGR2HSV)
    image_threshold = cv.inRange(image_hsv, lower_yellow, upper_yellow)
    #find contours
    contours, heirarchy = cv.findContours(image_threshold, \
                                          cv.RETR_TREE, \
                                          cv.CHAIN_APPROX_NONE)
#find the index of the largest contour
    if(len(contours)!=0):
        areas = [cv.contourArea(c) for c in contours]
        max_index= np.argmax(areas)
        cnt = contours[max_index]
```

```
#Cursor Motion
        if cx < 150:       ##if the pointer is in the left sector, the cursor should move towards the left.
           dist_x = -20
        elif cx > 250:     ##when the pointer is in the right sector, the cursor should move towards the right.
           dist_x = 20
        else:              ## the marker is in the central zone.
           dist_x = 0
        if cy < 150:
           dist_y = -20
        elif cy > 250:
           dist_y = 20
        else:
           dist_y = 0
        pyautogui.moveRel(dist_x, dist_y, duration = 0.25)
    ## Check for (left)click
    image_threshold_green = cv.inRange(image_hsv, lower_green, upper_green)
    contours_green, heirarchy = cv.findContours(image_threshold_green, \
                                                cv.RETR_TREE, \
                                                cv.CHAIN_APPROX_NONE)
    if(len(contours_green) != 0): ##check if the contours list is not empty, which means, a green colored contour is detected.
        pyautogui.click()
        cv.waitKey(1000)
  ## Check for click(right)
    image_threshold_red = cv.inRange(image_hsv, lower_red, upper_red)
    contours_red, heirarchy = cv.findContours(image_threshold_red, \
                                                cv.RETR_TREE, \
                                                cv.CHAIN_APPROX_NONE)
    if(len(contours_red) != 0): ##check if the contours list is not empty, which means, a red colored contour is actually detected.
        pyautogui.click(button='right')
        cv.waitKey(1000)
```

```
cv.imshow('Frame', frame)
   #key = cv.waitKey(10)
   key = cv.waitKey(100)
   if key == 27:
      break
#img_RGB=cv.cvtColor(frame, cv.COLOR_BGR2RGB)
#plt.imshow(img_RGB)
#plt.show()
cam.release()
cv.destroyAllWindows()
```

The results as shown:





9. References:

1. D, Hall, J. Martin, and J. L. Crowley, " Statistical Recognition of Parameter Trajectories for Hand Gestures and Face Expressions" , Computer Vision and Mobile Robotics Workshop, Santorini, Greece, September 17-18, 1998.

2. I. Laptev and T. Lindeberg, "Tracking of multi-state hand models using particle filtering and a hierarchy of multi-scale image features", Tech. Rep, KTH , Sweden, March 2000.

3. F. Quek et. al, "Gesture cues for conversational interaction in monocular video" , Proc. of Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, Greece, Sept 1999.

4. O. F. Ozer, O. Oziin, C. O. Tuzel, V. Atalay, A. E. Cetin, "Vision Bas ed Single-Stroke Character Recognition for Wearable Computing", IEEE Intelligent Systems, Vol. 16, No. 3, pp. 33-37, May/June 2001. 5. http://www.keyalt.com/pointdevices/headmouse.htrn

Made By : Elham Hassan Goda Tammam Kedwany