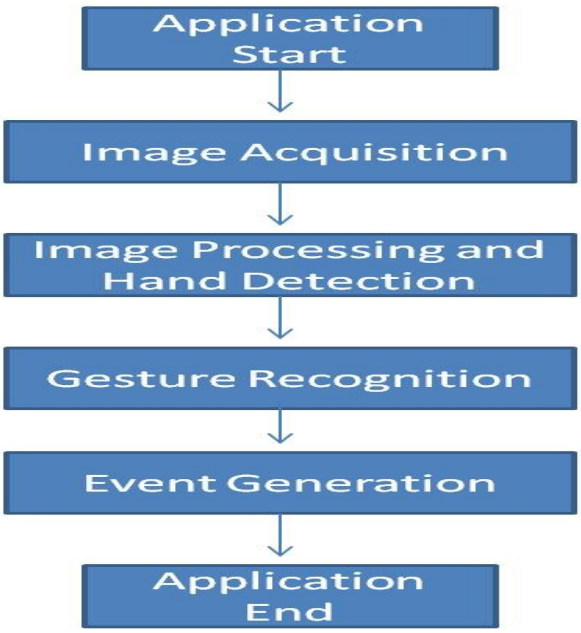
Gesture Recognition Virtual Mouse Using OpenCV

We all wonder it would have been so comfortable if we could control cursor through the use of hand gestures. Well our proposed project puts forward a hand gesture based system that allows user to control the pc mouse movements through the use of hand movements. Our system uses pc webcam in order to detect hand gesture movements. The system continuously scans the camera input finger hand like patterns.

Our algorithm works as follows:

- Webcam video is captures using python video capture libraries
- It is broken up into continuous image frames
- Each frame is converted to grayscale then HSV
- Each frame is now scanned to check finger design pattern
- Once detected it is flagged as an object
- System now tracks its movement within particular frame
- This is tracked in terms of x,y coordinates
- These coordinates are now mapped onto mouse cursor positioning on the screen



Libraries In Use

The external libraries that we will be using:

- openCV
- numpy
- wx
- pynput

Lets Start Coding

We will start by importing the libraries first

```
import cv2
import numpy as np
```

Now to detect color we need to know what is color in pixels of an image. Images are made of tiny dots of pixels each having a color and we can define those colors in terms of HSV -> Hue, Saturation, Value.

The hue of a pixel is an angle from 0 to 359 the value of each angle decides the color of the pixel the order of the color is same.

The Saturation is basically how saturated the color is, and the Value is how bright or dark the color is

So the range of these are as follows

•Hue is mapped – >0º-359º as [0-179]

•Saturation is mapped -> 0%-100% as [0-255]

•Value is 0-255 (there is no mapping)

So what does that mean.. It means for hue if we select for example 20 it will take it as 40º in terms of degree,
And for saturation 255 means 100% saturate and 0 means 0% saturate

Enough Talking lets jump back to code. We need to tell out program that we only want blue color object to be detected rest of the colors we are not interested in. to do that we need to decide a rage for HSV value for Blue (as there are lots of variation of blue color)

```
lowerBound=np.array([110,50,50])
upperBound=np.array([130,255,255])
```

So we declared these limits for the hsv values of each pixels. Now we will create a new binary image of same size a original image, we will call it mask and we'll make sure only those pixels that are in this hsv range will be allowed to be in the mask. that way only blue objects will be in the mask

Okay before doing that lets initialize our camera object

```
cam= cv2.VideoCapture(0)
```

Now lets Start The Main Processing

first we will read a frame from the camera

```
ret, img=cam.read()
```

we will resize it to make it a small fixed size for faster processing

```
img=cv2.resize(img,(340,220))
```

Now we will convert this image to hsv format

```
imgHSV= cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
```

after this we will be creating the filter which will create the mask for green color

```
mask=cv2.inRange(imgHSV,lowerBound,upperBound)

cv2.imshow("mask",mask)
cv2.imshow("cam",img)
cv2.waitKey(10)
```

Filtering the Mask

if we can see the output is quite great but we have some false positives in the mask. Those are the noises which are not good for object tracking. We have to clean to make out tracker work otherwise we will be seeing object marked in random places.

to do that we need to do some morphological operation called opening and closing

opening will remove all the dots randomly popping here and there and closing will close the small holes that are present in the actual object

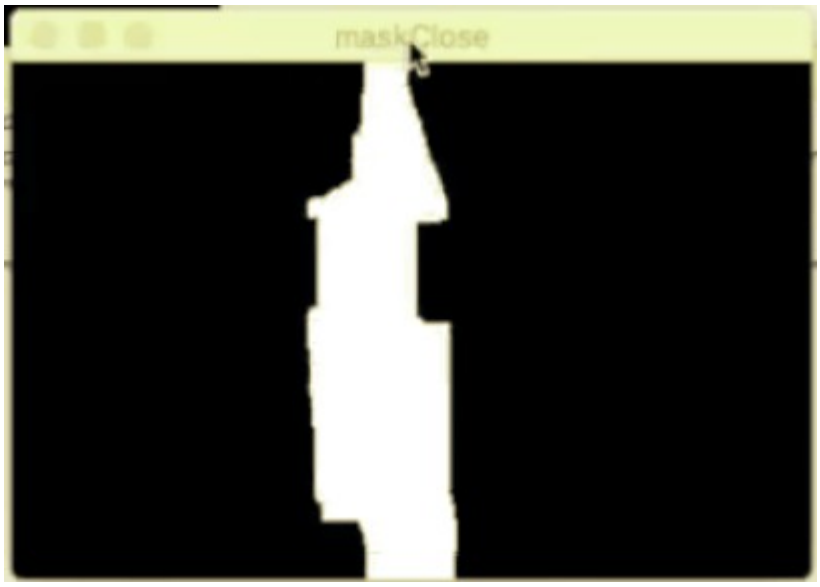
so before doing that we need a 2d matrix called kernal which is basically to control the effects of opening and closing

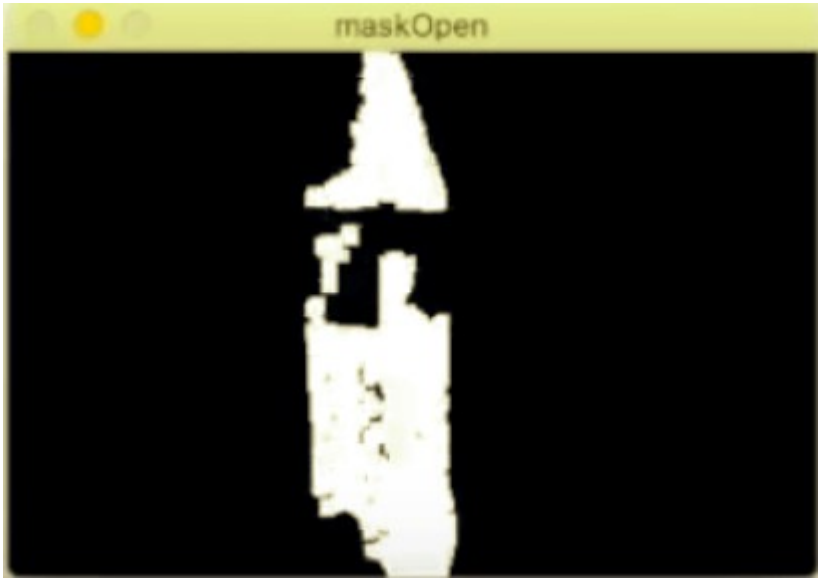
```
kernelOpen=np.ones((5,5))
kernelClose=np.ones((20,20))

maskOpen=cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernelOpen)
maskClose=cv2.morphologyEx(maskOpen,cv2.MORPH_CLOSE,kernelClose)

cv2.imshow("maskClose",maskClose)
cv2.imshow("maskOpen",maskOpen)
cv2.waitKey(10)
```


mask


maskClose



the result in the maskClose is the final form after cleaning all the noise now we know exactly where the object is so we can draw a contours from this mask

```
maskFinal=maskClose
conts,h=cv2.findContours(maskFinal.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
cv2.drawContours(img,conts,-1,(255,0,0),3)
```

and now the variable **conts** is a list of contours (in this case only one contour is present but if multiple objects are there it will contain all the contours). we will loop through all the contours and put a rectangle over it and we will mark them with a number for object tracking

```
for i in range(len(conts)):
   x,y,w,h=cv2.boundingRect(conts[i])
   cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255), 2)
      cv2.cv.PutText(cv2.cv.fromarray(img),  str(i+1),(x,y+h),font,(0,255,255))
```

```
import cv2
import numpy as np

lowerBound=np.array([33,80,40])
upperBound=np.array([102,255,255])

cam= cv2.VideoCapture(0)
kernelOpen=np.ones((5,5))
kernelClose=np.ones((20,20))

font=cv2.cv.InitFont(cv2.cv.CV_FONT_HERSHEY_SIMPLEX,2,0.5,0,3,1)

while True:
    ret, img=cam.read()
    img=cv2.resize(img,(340,220))

    #convert BGR to HSV
    imgHSV= cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
    # create the Mask
    mask=cv2.inRange(imgHSV,lowerBound,upperBound)
    #morphology
    maskOpen=cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernelOpen)
    maskClose=cv2.morphologyEx(maskOpen,cv2.MORPH_CLOSE,kernelClose
)

    maskFinal=maskClose
    conts,h=cv2.findContours(maskFinal.copy(),cv2.RETR_EXTERNAL,cv2
.CHAIN_APPROX_NONE)

    cv2.drawContours(img,conts,-1,(255,0,0),3)
    for i in range(len(conts)):
        x,y,w,h=cv2.boundingRect(conts[i])
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255), 2)
        cv2.cv.PutText(cv2.cv.fromarray(img), str(i+1),(x,y+h),font,
(0,255,255))
    cv2.imshow("maskClose",maskClose)
```

```
    cv2.imshow("maskOpen",maskOpen)
    cv2.imshow("mask",mask)
    cv2.imshow("cam",img)
    cv2.waitKey(10)
```

Lets Create a Virtual Mouse with Gesture Recognition

Okay lets start modifying the above code for gesture recognition

Import libraries

```
import cv2
import numpy as np
from pynput.mouse import Button, Controller
import wx
```

These are the libraries that we will be using. **Pynput** to control mouse movements and clicking and **wx** to get the display resolution of the monitor

Global variables Setup

now that we already have all the libraries lets setup all the variables and objects

```
mouse=Controller()
app=wx.App(False)
(sx,sy)=wx.GetDisplaySize()
(camx,camy)=(320,240)
```

we will need these variables and objects, mouse object is for mouse movements and to get the screen resolution we need an **wx** app then we can use the **wx.GetDisplaySize()** to get the screen resolution.

lastly we are setting some variables **camx**,**camy** to set the captured image resolution. we will be using it later in image resize function

Lets Start The Main Loop

The above code it the portion of the loop we wrote in our color detection program. We don’t need to modify the loop till this point. We will be adding our code after this.

```
while True:
    :
    :

    if(len(conts)==2):
        # logic for the open gesture, move mouse without click
        ....
        ....
    elif(len(conts)==1):
        # logic for close gesture
        ....
        ....
    cv2.imshow("cam",img)
    cv2.waitKey(5)
```

Above is the structure of our extended code. after getting the contours in conts variable we will check if there are contour of 2 object present in the frame we will move the mouse but we wont perform any click operation

similarly if there are only one object contour present we will move the mouse as well as we will perform click operations

Implement The Open Gesture Operation

To Implement the open gesture we need to do some calculation to find some coordinates. See the below image to get the idea

We have to first calculate the centre of both detected green object which we can easily do by taking the average of the bounding boxes maximum and minimum points. now we got 2 coordinate from the centre of the 2 objects we will find the avarage of that and we will get the red point shown in the image.. okay lets do this

```
while True:
    :
    :
    if(len(conts)==2):
        # logic for the open gesture, move mouse without click
        x1,y1,w1,h1=cv2.boundingRect(conts[0])
        x2,y2,w2,h2=cv2.boundingRect(conts[1])
        # drawing rectangle over the objects
        cv2.rectangle(img,(x1,y1),(x1+w1,y1+h1),(255,0,0),2)
        cv2.rectangle(img,(x2,y2),(x2+w2,y2+h2),(255,0,0),2)
        #centre coordinate of first object
        cx1=x1+w1/2
        cy1=y1+h1/2
        # centre coordinate of the 2nd object
        cx2=x2+w2/2
        cy2=y2+h2/2
        # centre coordinate of the line connection both points
        cx=(cx1+cx2)/2
        cy=(cy1+cy2)/2
        # Drawing the line
        cv2.line(img, (cx1,cy1),(cx2,cy2),(255,0,0),2)
        # Drawing the point (red dot)
        cv2.circle(img, (cx,cy),2,(0,0,255),2)
```

So the above code is the result of what I just explained earlier and with this we have the coordinate to position our mouse curser

Now we need to position our mouse curser according to the calculated coordinate okay lets do that

```
while True:
    :
    :
    if(len(conts)==2):
        :
        :
        mouse.release(Button.left)
        mouseLoc=(sx-(cx*sx/camx), cy*sy/camy)
        mouse.position=mouseLoc
        while mouse.position!=mouseLoc:
            pass
```

In the above code first we are doing a mouse release to ensure the mouse left button is not pressed. Then we are converting the detected coordinate from camera resolution to the actual screen resolution. After that we set the location as the mouse.position. but to move the mouse it will take time for the curser so we have to wait till the curser reaches that point. So we started a loop and we are not doing anything there we are just waiting will the current mouse location is same as assigned mouse location.That is for the open gesture

Implement Close Gesture/ Clicking

Now lets implement the close gesture where we will be clicking the object and dragging it

```
while True:
    :
    :
    if(len(conts)==2):
        :
        :
    elif(len(conts)==1):
        x,y,w,h=cv2.boundingRect(conts[0])
        #drawing the rectangle
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        cx=x+w/2
```

```
        cy=y+h/2
        cv2.circle(img,(cx,cy),(w+h)/4,(0,0,255),2)

        mouse.press(Button.left)
        mouseLoc=(sx-(cx*sx/camx), cy*sy/camy)
        mouse.position=mouseLoc
        while mouse.position!=mouseLoc:
            pass
```

The above code is similar to the open gesture, but the difference is we only have one object here so we only need to calculate the centre of it. And that will be where we will position our mouse pointer. also we are performing a mouse press operation instead of mouse release operation. rest of the part is same as earlier one.

Some Fine Tuning

We are almost done. The code is almost perfect except. we wont be able to drag anything. because in close gesture we are continuously performing **mouse.press** operation which will result in continuous multiple clicks while dragging.

To solve this problem what we can do is, we will be putting a flag called “**pinchFlag**” and we will set that **1** once we perform a click operation. and we wont perform mouse press operation anymore until the flag is **0** again

so the code will look like this

```
pinchFlag=0# setting initial value
while True:
    :
    :
    if(len(conts)==2):
        :
```

```
        :
        if(pinchFlag==1): #perform only if pinch is on
            pinchFlag=0 # setting pinch flag off
            mouse.release(Button.left)
        mouseLoc=(sx-(cx*sx/camx), cy*sy/camy)
        mouse.position=mouseLoc
        while mouse.position!=mouseLoc:
            pass
    elif(len(conts)==1):
        :
        :
        if(pinchFlag==0): #perform only if pinch is off
            pinchFlag=1 # setting pinch flag on
            mouse.press(Button.left)
        mouseLoc=(sx-(cx*sx/camx), cy*sy/camy)
        mouse.position=mouseLoc
        while mouse.position!=mouseLoc:
            pass
```