GESTURE RECOGNISATION TO CREATE VIRTUAL MOUSE

A Report for the Evaluation-1 of Project-2

KRISHNAKANT

15SCSE105049

Under the Supervision of

Dr. D. Rajesh Kumar

School of Computing Science and Engineering
Greater Noida, Uttar Pradesh
2018

Table of Content

Abstract

The project “Gesture Recognition using colour segmentation to use it as a virtual mouse”, is based on identifying a particular coloured object and use the movement of that object to perform mouse operations. There are two mouse operations possible through this project i.e. mouse pointer movement and mouse left click operation.

The idea is based on converting a each frame of the webcam real-time video stream and convert it into equivalent HSV (Hue, Saturation, Value) format in a Numpy array. And filter out the HSV value that lies in the range of the colour that that we are going to detect. Once the filtered frame is obtained thereafter we can apply some morphological operations to make the filtered mask even more sharper. Once filtering is done, then we find the all the contours and store it in list of contours.

After the object detection part is done, we focus on using the detected contours to perform muse operations.

The mouse operation are decided on the following basis -

- If the length of contours list is two then only mouse pointer movement will be done.
- If the length of the list is one then mouse pointer movement and left click operation is also performed.

The external libraries that we will be using:

- openCV
- numpy
- wx
- pynput

OpenCV : OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

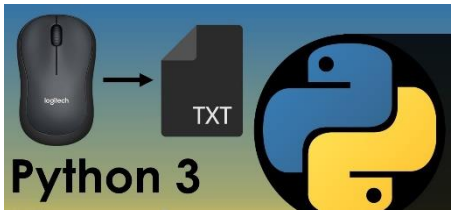
OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell and Ruby have been developed to encourage adoption by a wider audience.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality



Introduction

Description

"Gesture Recognition using colour segmentation to use it as a virtual mouse" is a project based on real time image frame procession based on pixel parameters.

We can divide this project into two parts, the first part will deal with the object detection and the second part will deal with the mouse operations using the detected objects.

PART-I

Libraries needed for Part-I are :

- OpenCV
- NumPy

So, in order to understand the complete working of the project we need to understand the basic fundamentals of pixels.

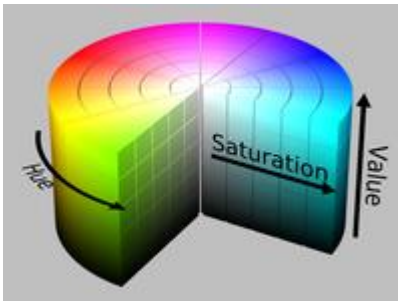
Now to detect color we need to know what is color in pixels of an image. Images are made of tiny dots of pixels each having a color and we can define those colors in terms of HSV -> Hue, Saturation, Value.

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 but i reverse as the order in rainbow order from red to violet and again back to red

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

We need to tell out program that we only want green 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 Green (as there are lots of variation of green color)

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

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 green objects will be in the mask

Firstly, we initialize our camera object

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

and create a font for the text we will be printing in the screen

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

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)
```

now lets see how the mask looks

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

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,kernelClos
e)
```

PART-II

Libraries needed for part-II (other than part-I) are :

- Pynput
- Wx

pynput is used to control mouse movements and clicking and ***wx*** to get the display resolution of the monitor.

Global variables needed:

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

Logic for mouse operations:

```
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.

To Implement the open gesture we need to do some calculation to find some coordinates.

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 average of that and we will get the red point shown in the image.

We will be 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.

After this the project is done and may be possible if it is necessary then we need to do some fine tuning.

Purpose

The main purpose of this project is perform mouse input through a camera device. A very sophisticated version of this program can we used for drawing precisely, hand writing on screen without touching the screen.

This project can have multiple applications such as:

- Used as a basic gaming console : It can be used as a basic gaming console for games like Jardians, TuxCart etc. where only mouse operations are sufficient.
- Writing on screen: It can also be used for writing on the screen without physically touching the screen. Mostly, handwriting on the screen is done by using a pen and a touch sensitive screen, but this program can allow users to write on a non-touch sensitive screens as well.
- Wireless mouse: It can be considered as a wireless mouse that can work up to longer distances depending upon the camera resolution and aperture.
- Motion detection of a particular colour : We can also detect motion of a particular coloured object by observing the mouse pointer movements and slightly changing the code.

Motivation and Scope

The idea for this project came from colour detection project which I saw in an article. The colour detection code was improved and extended to support mouse operations via colour detection. It support pinch gesture for left click, movement gesture for pointer movement.

The first half of the project is almost similar to the colour detection program with a minor modification. The second half of the project is the application of the result of first part to perform input operations through a virtual mouse.

The project could be fine tuned even more to perform much sophisticated works like animation object drawing, sketching, playing games etc. So, it has a very broad scope of extension and usability.

System Architecture

System architecture requirement for this project is not much high. I just requires some basic hardware components that a basic household PC has.

The hardware and software used for building this project are :

- 64 bit intel core i3 processor
- 4 GB RAM
- 1366x768 resolution screen
- 0.3 MP webcam
- Python 2.7
- OpenCV 2
- NumPy
- Pynput
- WxPython

Implementation Part:

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

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

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))
pinchFlag=0
```

This part is the initialisation of variables.

```
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
```

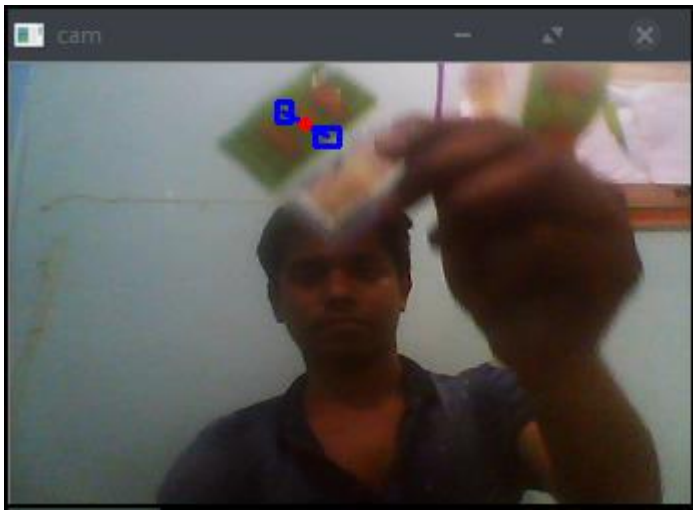
```
    maskOpen=cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernelOpen)
    maskClose=cv2.morphologyEx(maskOpen,cv2.MORPH_CLOSE,kernelClose)

    maskFinal=maskClose

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

if(len(conts)==2):
    if(pinchFlag==1):
        pinchFlag=0
        mouse.release(Button.left)
    x1,y1,w1,h1=cv2.boundingRect(conts[0])
    x2,y2,w2,h2=cv2.boundingRect(conts[1])
    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)
    cx1=x1+w1/2
    cy1=y1+h1/2
    cx2=x2+w2/2
    cy2=y2+h2/2
    cx=(cx1+cx2)/2
    cy=(cy1+cy2)/2
    cv2.line(img, (cx1,cy1),(cx2,cy2),(255,0,0),2)
    cv2.circle(img, (cx,cy),2,(0,0,255),2)
    mouseLoc=(sx-(cx*sx/camx), cy*sy/camy)
    mouse.position=mouseLoc
    while mouse.position!=mouseLoc:
        pass
```

This part contains mouse pointer movement operation.

```
elif(len(conts)==1):
    x,y,w,h=cv2.boundingRect(conts[0])
    if(pinchFlag==0):
        pinchFlag=1
        mouse.press(Button.left)
    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)
    mouseLoc=(sx-(cx*sx/camx), cy*sy/camy)
    mouse.position=mouseLoc
    while mouse.position!=mouseLoc:
        pass
cv2.imshow("cam",img)
cv2.waitKey(5)
```

This part contains the mouse click operations.

Output Screenshot :



References

- https://opencv.org/ (for documentation reference purpose)
- Geeksforgeeks.org (various articles on Numpy)
- Github.com (various github projects based on OpenCV).
- Various blogs (specially Codacus blog)
- Wikipedia (https://en.wikipedia.org/wiki/OpenCV)
- Youtube (Codacus Channel on Gesture recognisation.)

PPT

tinyurl.com/kk1561