Tracking Of Defined Symbols Using Intel Real Sense

Riddhi R. Dasani

Bachelor of Engineering
Instrumentation and control engineer
Government Engineering College, Rajkot, India

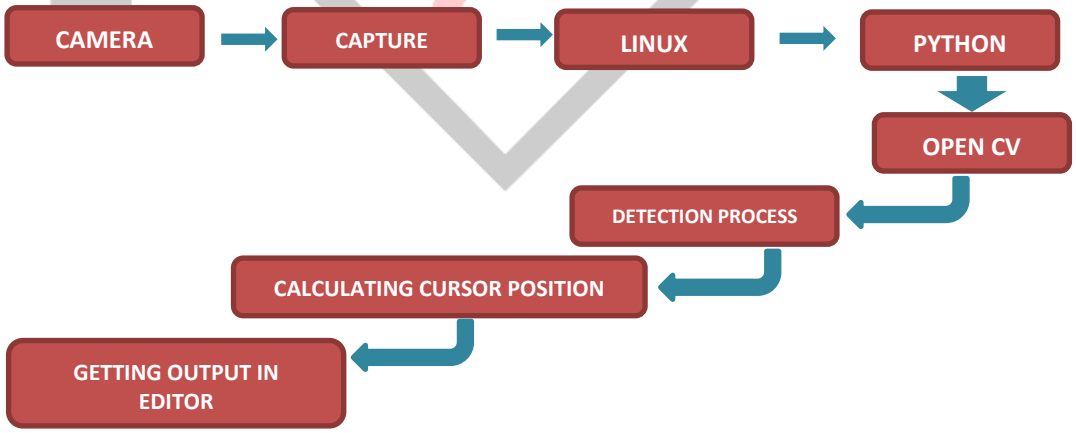
***ABSTRACT*: This paper is on the basis of Image Processing. It will be implemented using OpenCV Library and coded in Python scripting language. The purpose of the paper is to detect an object based on its color and shape. As well as to write in air which needs to display on the screen? I track its movements frame by frame, and based on these movements I manipulate the system. In accordance to the position of object the mouse will move on the system screen. The object will be detected using the basic algorithms of Image processing. This paper will increase the comfort of the user and make operations easier. Further many other systems can be manipulated using the same method, like a media Centre can be manipulated using the same algorithms. I have used OpenCV library as its open source and has large variants of functions that can be used. Initially I found it involves continuous writing only because of its 2 dimensional and unable to recognize third dimension. Hence, to write discontinuous letter isn't possible. So I have decided to work in 3 dimensional. To make it further convenient for the user I have used Intel Real Sense camera which is 3D camera. It senses 3rd dimension which is depth. It responds to natural movement in 3D. It elevates user interface to futuristic levels. Then it converts that 3D image into 2D image on display (Monitor Screen). After Converting I need to recognize finger for starting purpose therefore, a program is needed which can detect particular color in the image (Range is also specified). Then it recognizes Depth of the image. It allows the device to inter depth by detecting the reflected IR light. This data along with motion tracking software create touch free interface. For color detection I referred algorithms. Therefore with a single click a user can activate the program and start using the object to monitor their desktop.**

***Keywords:* python, OpenCV, hand detection 2D/3D, finger tracking**

INTRODUCTION:

I intended to do human computer interaction which involves interfacing between human and machines. Once I came to know about next generation ultra-books and notebooks involves inbuilt Intel Real Sense camera. For the development phase Intel has provided a separate Intel Real Sense Camera using this computer interaction can easily manipulate. When I start running my program camera captures an image. The image is processed in Linux. For that Python scripting and OpenCV is used. Then further processing has been done.

DATA FLOW DIAGRAM



(Fig: 1.1 Data Flow Diagram)

PROGRAM:

```
#!usr/bin/python
import serial
import cv2
import numpy as np
import cv2.cv as cv
import time

#cv2.namedWindow("camera", [illegible]IZE)
capture = cv2.VideoCapture(0)#capturing image from cam

while True:
    f, o_img = capture.read()
    o_img = cv2.flip(o_img, 1)#flipping the image
    img = cv2.GaussianBlur(o_img, (5,5), 0) #gaussian blur the image
    img = cv2.cvtColor(o_img, cv2.COLOR_BGR2HSV)#convert to HSV image
    im1 = cv2.cvtColor(o_img, cv2.COLOR_BGR2GRAY)#convert to gray image
    edges = cv2.Canny(im1,70,250)#canny edge detection
    cv2.imshow("edges",edges)
    red_lower = np.array([110, 60, 60],np.uint8)#setting lower limit for blue colour
    red_upper = np.array([130, 255, 255],np.uint8)#setting upper limit for blue colour
    red_binary = cv2.inRange(img, red_lower, red_upper)#converting to binary image
    dilation = np.ones((20, 20), "uint8")#creating structural element
    red_binary = cv2.dilate(red_binary, dilation)#dilating the image
    cv2.imshow("yoo", red_binary)
    contours, hierarchy = cv2.findContours(red_binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)#finding
contours
    max_area = 0
    largest_contour = None
    slargest_contour = None
    for idx, contour in enumerate(contours):#finding largest contour
        area = cv2.contourArea(contour)
        if area > max_area:
          max_area = area
          largest_contour = contour
               max_area2=0
               for idx, contour in enumerate(contours):#finding the second contour
              area2 = cv2.contourArea(contour)
              if area2 > max_area2 and area2< max_area:
                max_area2 = area2
                slargest_contour = contour
    if not largest_contour == None:
        moment = cv2.moments(largest_contour)
             cx = int(moment['m10']/moment['m00'])#finding centroid of contour
             cy = int(moment['m01']/moment['m00'])
#            print cx,cy
        if moment["m00"] > 5000:
          rect = cv2.minAreaRect(largest_contour)#drawing rectangle around the contour
          box = cv2.cv.BoxPoints(rect)
          box = np.int0(box)
          cv2.drawContours(o_img,[box], 0, (0, 0, 255), 2)
             if not slargest_contour == None:
          moment = cv2.moments(slargest_contour)
               cx1 = int(moment['m10']/moment['m00'])
               cy1 = int(moment['m01']/moment['m00'])
#              print cx1,cy1
          if moment["m00"] > 2000:
            rect = cv2.minAreaRect(slargest_contour)
            box = cv2.cv.BoxPoints(rect)
            box = np.int0(box)
            cv2.drawContours(o_img,[box], 0, (0, 0, 255), 2)
                 print "Bottle detected"
```

```
            length = 2*(cy-cy1)
            print "The length of the bottle is:"
            print length
        cv2.imshow("ColourTrackerWindow", o_img)
        else:
                print "Error"
    else:
        print "No bottle detected"
    if cv2.waitKey(20) ==27:
        cv2.destroyAllWindows()
        break
```

HAND DETECTION IN 2-DIMENSION:

- The main flow of the program follows the following path:
  - Detection of hand
  - Tracking of the hand
  - Linking cursor position with hand
  - Using it for creating a writing interface
  - Using an editor to implement writing process.

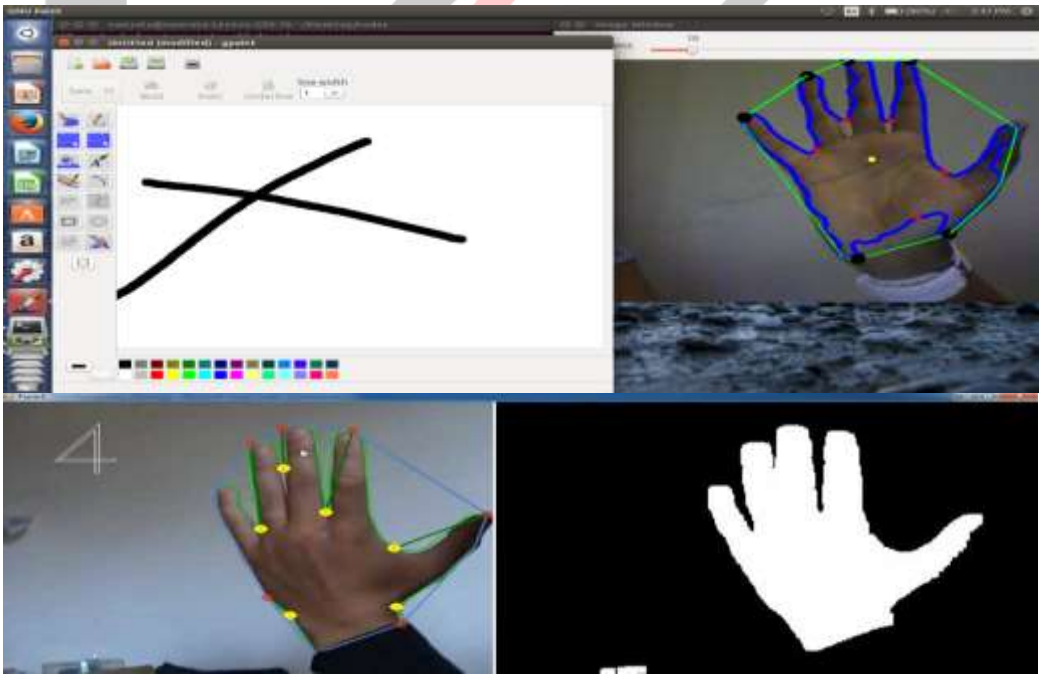
PROGRAM:

```
import cv
import random
from unix import PyMouse


minV=10
hc = cv.Load("haarcascade_frontalface_alt.xml")
def changeMinBrightness(x):
    global minV
    minV=x
    def pointInFace(xpos,ypos,x,y,w,h):
    return (xpos>x and xpos<x+w and ypos>y and ypos<y+h)
x=0
y=0
w=0
h=0
mouse = PyMouse()
width,height= mouse.screen_size()
capture=cv.CaptureFromCAM(1)
image=cv.QueryFrame(capture)
X,Y=mouse.position()
click=True
#writer=cv.CreateVideoWriter("output.avi", 0, 15, cv.GetSize(image), 1)
count=0
cv.NamedWindow("Image Window")
cv.CreateTrackbar("min-brightness", "Image Window", minV, 100, changeMinBrightness );

while True:
    image=cv.QueryFrame(capture)
    cv.Flip(image, flipMode=0)
    grey=cv.CreateImage(cv.GetSize(image),8,1)
    cv.CvtColor(image,grey,cv.CV_BGR2GRAY)

    #Detect face in image
    if count % 10 == 0:
        face = cv.HaarDetectObjects(grey, hc, cv.CreateMemStorage(), 1.2,2, cv.CV_HAAR_DO_CANNY_PRUNING,
(0,0) )

    #print face
```

```
    for [(x,y,w,h),k] in face:
        cv.Rectangle(image,(x ,y),(x+w,y+h),(0,255,0))

    window_width,window_height=cv.GetSize(image)
    hsv_image=cv.CreateImage(cv.GetSize(image),8,3)
    hsv_mask = cv.CreateImage( cv.GetSize(image), 8, 1);
    hsv_edge = cv.CreateImage( cv.GetSize(image), 8, 1);
    hsv_min = cv.Scalar(0, 30, minV, 0);
    hsv_max = cv.Scalar(20, 150, 255, 0);
    cv.CvtColor(image,hsv_image,cv.CV_BGR2HSV)
    cv.InRangeS (hsv_image, hsv_min, hsv_max, hsv_mask)
    element_shape = cv.CV_SHAPE_RECT
    pos=1
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Dilate(hsv_mask,hsv_mask,element,2)
    cv.Erode(hsv_mask,hsv_mask,element,2)
    cv.Smooth(hsv_mask, hsv_mask, cv.CV_MEDIAN, 25,0,0,0)
    cv.Canny(hsv_mask, hsv_edge, 1, 3, 5);
    contours=cv.FindContours(hsv_mask, cv.CreateMemStorage(), cv.CV_RETR_LIST,
cv.CV_CHAIN_APPROX_SIMPLE, (0, 0))
    des=cv.CreateImage(cv.GetSize(image),8,3)
    contours2 = False
    hull=False
    blank=cv.CreateImage(cv.GetSize(image),8,3)

    while contours:
        contours2=contours
        contours = contours.h_next()

    if contours2:
        hull=cv.ConvexHull2(contours2, cv.CreateMemStorage(),cv.CV_CLOCKWISE,0)
        defects=cv.ConvexityDefects(contours2, hull, cv.CreateMemStorage())
        cv.DrawContours(image, contours2, (255,0,0), (0,255,0), 0,5)
        noOfDefects=0
        comx,comy=0,0
        Z,(p,q),radius = cv.MinEnclosingCircle(contours2)

        if Z:
            cv.Circle(image,(int(p),int(q)),int(5),[0,255,255],-1)

        for defect in defects:
            start,end,far,d = defect
            xpos,ypos=far
            cv.Line(image,start,end,[0,255,0],2)

            if d > 20 and not pointInFace(xpos,ypos,x,y,w,h):
                cv.Circle(image,far,5,[0,0,255],-1)
                if noOfDefects:
                    cv.Circle(image,end,10,[0,0,0],-1)
                    #cv.Line(image,end,(int(p),int(q)),[0,0,0],3)
                    x,y=end
                    comx,comy=comx+x,comy+y
                else:
                    cv.Circle(image,start,10,[0,0,0],-1)
                    #cv.Line(image,start,(int(p),int(q)),[0,0,0],3)
                    x,y=start
                    comx,comy=comx+x,comy+y
                    cv.Circle(image,end,10,[0,0,0],-1)
                    #cv.Line(image,end,(int(p),int(q)),[0,0,0],3)
                    x,y=end
                    comx,comy=comx+x,comy+y
                noOfDefects=noOfDefects+1
        p,q=comx/(noOfDefects+1),comy/(noOfDefects+1)
```

```
    #print len(defects),noOfDefects
    initx,inity=X,Y
    X,Y = ((p-window_width/12)*width*6/(window_width*5)),((q-window_height/12)*height*6/(window_height*5))

    diffx,diffy=X-initx,Y-inity
    X,Y=initx+ diffx*(0.1+abs(diffx)/700), inity+ diffy*(0.1+abs(diffy)/300)

    if X < 0:
       X = 0
    else:
       X=min(X,width)

    if Y < 0:
       Y = 0
    else:
       Y=min(Y,height)

    if len(defects) > 10 and len(defects) < 35:
       if noOfDefects > 0 and noOfDefects < 7 :
          i=0.33
          while i<=1:
             mouse.move(initx+i*(X-initx),inity+i*(Y-inity))
             i+=0.33
          if noOfDefects > 1 and noOfDefects<3:
             if not(click):
                mouse.click(initx,inity)
                click=True
          else:
             click=False

  cv.ShowImage('Image Window',image)
#   cv.WriteFrame(writer, image)
  if cv.WaitKey(2)== 27:
     break
  count+=1-
```

OUTPUT :



HAND DETECTION IN 3-DIMENTION:

- The main flow of the program follows the following path:
  - Detection of the hand

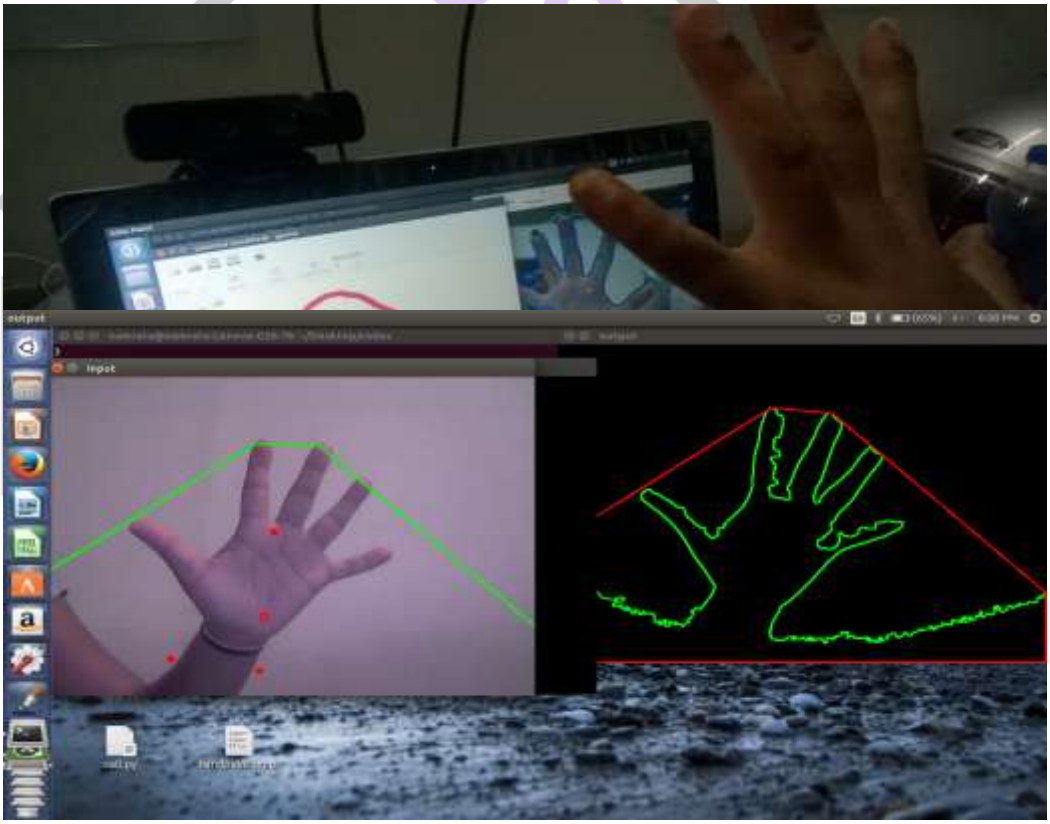
- Tracking of the hand
- Introduced depth of object mean z-coordinate.
- Linking cursor position with hand
- it for creating a writing interface
- Using an editor to implement writing process.

PROGRAM:

```
import cv2
import numpy as np
#from pymouse import PyMouse

#m = PyMouse()

#z=m.screen_size()
#print z
colour_db=input("Enter BGR values of the coulour u want to use: ")
width_db =input("Enter the width of ur pen: ")
d_img = np.zeros((479,639,3), np.uint8)

cap = cv2.VideoCapture(1)
while( cap.isOpened() ) :
   ret,img = cap.read()
   img = cv2.flip(img, 0)
   gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
   blur = cv2.GaussianBlur(gray,(5,5),0)
   ret,thresh1 = cv2.threshold(blur,70,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

   contours, hierarchy = cv2.findContours(thresh1,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
   drawing = np.zeros(img.shape,np.uint8)

   max_area=0
   try:
    for i in range(len(contours)):
        cnt=contours[i]
        area = cv2.contourArea(cnt)
        if(area>max_area):
            max_area=area
            ci=i
    cnt=contours[ci]
    hull = cv2.convexHull(cnt)
    moments = cv2.moments(cnt)
    if moments['m00']!=0:
            cx = int(moments['m10']/moments['m00']) # cx = M10/M00
            cy = int(moments['m01']/moments['m00']) # cy = M01/M00

    centr=(cx,cy)
    cv2.circle(img,centr,5,[0,0,255],2)
    cv2.drawContours(drawing,[cnt],0,(0,255,0),2)
    cv2.drawContours(drawing,[hull],0,(0,0,255),2)

    cnt = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
    hull = cv2.convexHull(cnt,returnPoints = False)

    if(1):
        defects = cv2.convexityDefects(cnt,hull)
        mind=0
        maxd=0
        for i in range(defects.shape[0]):
            s,e,f,d = defects[i,0]
            start = tuple(cnt[s][0])
```

```
            end = tuple(cnt[e][0])
            far = tuple(cnt[f][0])
            dist = cv2.pointPolygonTest(cnt,centr,True)
            cv2.line(img,start,end,[0,255,0],2)

            cv2.circle(img,far,5,[0,0,255],-1)
        print(i)
             if (i<=2 and i>0):
                            cx1= cx
                            cy1= cy
                            #m.move(cx1, cy1)

                            cv2.circle(d_img,(cx1,cy1), width_db, colour_db, -1)
                            cv2.imshow("Drawing board",d_img)
        i=0
  cv2.imshow('output',drawing)
  cv2.imshow('input',img)
except:
        pass
k = cv2.waitKey(10)
if k == 27:
   break
```

OUTPUT:



FINGER TRACKING:

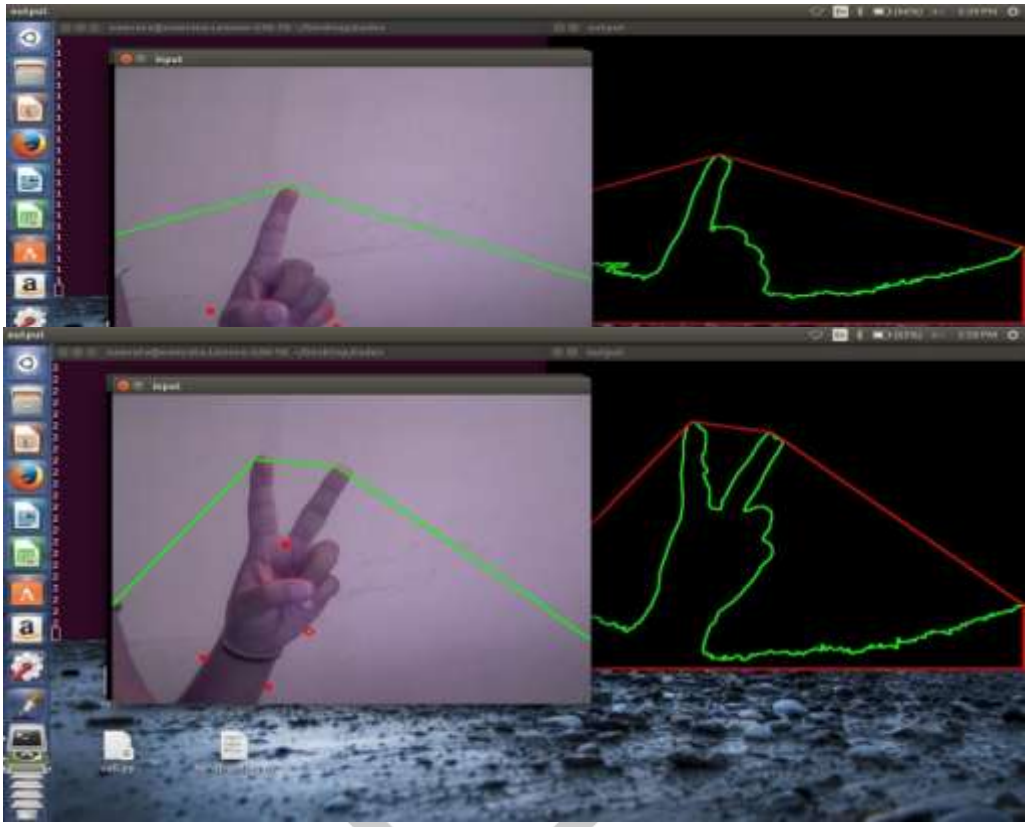
- The main flow of the program follows the following path:
  - Detection of the finger
  - Tracking of the finger
  - Introduced depth of object mean z-coordinate.
  - Making angle according finger.
  - Linking cursor position with finger
  - Using it for creating a writing interface
  - Using an editor to implement writing process.

PROGRAM:

```
import cv2
import numpy as np
#from pymouse import PyMouse

#m = PyMouse()

#z=m.screen_size()
#print z
colour_db=input("Enter BGR values of the coulour u want to use: ")
width_db =input("Enter the width of ur pen: ")
d_img = np.zeros((479,639,3), np.uint8)

cap = cv2.VideoCapture(1)
while( cap.isOpened() ) :
   ret,img = cap.read()
   img = cv2.flip(img, 0)
   gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
   blur = cv2.GaussianBlur(gray,(5,5),0)
   ret,thresh1 = cv2.threshold(blur,70,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

   contours, hierarchy = cv2.findContours(thresh1,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
   drawing = np.zeros(img.shape,np.uint8)

   max_area=0
   try:
    for i in range(len(contours)):
        cnt=contours[i]
        area = cv2.contourArea(cnt)
        if(area>max_area):
            max_area=area
            ci=i
    cnt=contours[ci]
    hull = cv2.convexHull(cnt)
    moments = cv2.moments(cnt)
    if moments['m00']!=0:
            cx = int(moments['m10']/moments['m00']) # cx = M10/M00
            cy = int(moments['m01']/moments['m00']) # cy = M01/M00

    centr=(cx,cy)
    cv2.circle(img,centr,5,[0,0,255],2)
    cv2.drawContours(drawing,[cnt],0,(0,255,0),2)
    cv2.drawContours(drawing,[hull],0,(0,0,255),2)

    cnt = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
    hull = cv2.convexHull(cnt,returnPoints = False)

    if(1):
            defects = cv2.convexityDefects(cnt,hull)
            mind=0
            maxd=0
            for i in range(defects.shape[0]):
                s,e,f,d = defects[i,0]
                start = tuple(cnt[s][0])
                end = tuple(cnt[e][0])
                far = tuple(cnt[f][0])
                dist = cv2.pointPolygonTest(cnt,centr,True)
                cv2.line(img,start,end,[0,255,0],2)

                cv2.circle(img,far,5,[0,0,255],-1)
            print(i)
                 if (i<=2 and i>0):
```

```
                    cx1= cx
                    cy1= cy
                    #m.move(cx1, cy1)

                    cv2.circle(d_img,(cx1,cy1), width_db, colour_db, -1)
                    cv2.imshow("Drawing board",d_img)
        i=0
  cv2.imshow('output',drawing)
  cv2.imshow('input',img)
 except:
        pass
 k = cv2.waitKey(10)
 if k == 27:
    break
```

OUTPUT:



THUMBS UP BOTTLE SYMBOL TRACKING:

In this what I did is camera will continuously detect a symbol and LED will glow if it detected properly. This can be implemented in industries where bottle is going through conveyor belt.

PROGRAM:

```
        box = cv2.cv.BoxPoints(rect) #!usr/bin/python
import serial
import cv2
import numpy as np
import cv2.cv as cv
import time

#cv2.namedWindow("camera", cv2.CV_WINDOW_AUTOSIZE)
capture = cv2.VideoCapture(0)#capturing image from cam

while True:
   f, o_img = capture.read()
   o_img = cv2.flip(o_img, 1)#flipping the image
   img = cv2.GaussianBlur(o_img, (5,5), 0) #gaussian blur the image
   img = cv2.cvtColor(o_img, cv2.COLOR_BGR2HSV)#convert to HSV image
```

```
im1 = cv2.cvtColor(o_img, cv2.COLOR_BGR2GRAY)#convert to gray image
edges = cv2.Canny(im1,70,250)#canny edge detection
cv2.imshow("edges",edges)
red_lower = np.array([110, 60, 60],np.uint8)#setting lower limit for blue colour
red_upper = np.array([130, 255, 255],np.uint8)#setting upper limit for blue colour
red_binary = cv2.inRange(img, red_lower, red_upper)#converting to binary image
dilation = np.ones((20, 20), "uint8")#creating structural element
red_binary = cv2.dilate(red_binary, dilation)#dilating the image
cv2.imshow("yoo", red_binary)
contours, hierarchy = cv2.findContours(red_binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)#finding contours
max_area = 0
largest_contour = None
slargest_contour = None
for idx, contour in enumerate(contours):#finding largest contour
   area = cv2.contourArea(contour)
   if area > max_area:
    max_area = area
    largest_contour = contour
        max_area2=0
        for idx, contour in enumerate(contours):#finding the second contour
       area2 = cv2.contourArea(contour)
       if area2 > max_area2 and area2< max_area:
        max_area2 = area2
        slargest_contour = contour
if not largest_contour == None:
   moment = cv2.moments(largest_contour)
       cx = int(moment['m10']/moment['m00'])#finding centroid of contour
       cy = int(moment['m01']/moment['m00'])
#       print cx,cy
   if moment["m00"] > 5000:
    rect = cv2.minAreaRect(largest_contour)#drawing rectangle around the contour
    box = cv2.cv.BoxPoints(rect)
    box = np.int0(box)
    cv2.drawContours(o_img,[box], 0, (0, 0, 255), 2)
       if not slargest_contour == None:
    moment = cv2.moments(slargest_contour)
        cx1 = int(moment['m10']/moment['m00'])
        cy1 = int(moment['m01']/moment['m00'])
#       print cx1,cy1
    if moment["m00"] > 2000:
     rect = cv2.minAreaRect(slargest_contour)
     box = np.int0(box)
     cv2.drawContours(o_img,[box], 0, (0, 0, 255), 2)
          print "Bottle detected"
          length = 2*(cy-cy1)
          print "The length of the bottle is:"
          print length
    cv2.imshow("ColourTrackerWindow", o_img)
       else:
                print "Error"
else:
       print "No bottle detected"
if cv2.waitKey(20) ==27:
       cv2.destroyAllWindows()
       break
```

LIMITATIONS:

- Constant Background is needed.
- Z-Coordinate having limitation.
- Depth range from which it can detect object is also limited.

CONCLUSION:

From this paper, I concluded that initially I did hand tracking and came to conclusion that hand tracking being very intuitive and I did finger tracking. With continuous observation and manipulation Human Computer Interaction is being executed. Symbol recognition of Thumbs up can be implemented in Industry. In addition to this I can also detect various color using this. We can also write on display too.

FUTURE SCOPE:

- By using Symbol recognition and detection I can detect one object. And we can find that particular object. It may be used for identification purpose.
- Mouse and Keyboard control transfers to our hand makes an alternative of touch screen.
- By increasing range of the camera we can write from the distance.

REFERANCES:

[1] http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-voice-technologies.html
[2] http://www.pcworld.com/article/2084810/hands-on-intels-realsense-is-both-productive-and-fun.html
[3] https://www.python.org/about/gettingstarted/
[4] http://www.codecademy.com/en/tracks/python
[5] http://www.tutorialspoint.com/python/
[6] http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html
[7] http://answers.opencv.org/question/10102/how-to-generate-haar-cascade-xml-file-for-finger-detection/
[8] http://www.slideshare.net/IJMER/ab2639964000
[9] http://www.ai5.uni-bayreuth.de/de/download/guthe/endler-2014-robust.pdf