**CHALLENGES**

Background color and skin color should be much different

Hand is the largest object in the captured image

A certain angle infront of webcam

In mathematics, Convex Hull is the smallest convex set that contains a set of points. And a convex set is a set of points such that, if we trace a straight line from any pair of points in the set, that line must be also be inside the region. The result is then a nice, smooth region, much easier to be analised than our contour, that contains many imperfections.

there are “gaps” between the convex hull region and our contour region. The “convexDefects” will try to approximate those gaps using straight lines. We can then use that information to find the points where our fingertips are placed.

**NumPy**

NumPy is the fundamental package for scientific computing with Python. NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

The code captures frames using a web camera (tested on mac's camera) and outputs a video.

The code captures frames using a web camera (tested on mac's camera) and outputs a video with a number designates the number of pointed finger. For example, a fist corresponds to 0 and an open hand to 5.

import cv2

import numpy as np

import time

#Open Camera object

cap = cv2.VideoCapture(0)

#Decrease frame size

#cap.set(cv2.cv.CV\_CAP\_PROP\_FRAME\_WIDTH, 1000)

#cap.set(cv2.cv.CV\_CAP\_PROP\_FRAME\_HEIGHT, 600)

def nothing(x):

pass

# Function to find angle between two vectors

def Angle(v1,v2):

dot = np.dot(v1,v2)

x\_modulus = np.sqrt((v1\*v1).sum())

y\_modulus = np.sqrt((v2\*v2).sum())

cos\_angle = dot / x\_modulus / y\_modulus

angle = np.degrees(np.arccos(cos\_angle))

return angle

# Function to find distance between two points in a list of lists

def FindDistance(A,B):

return np.sqrt(np.power((A[0][0]-B[0][0]),2) + np.power((A[0][1]-B[0][1]),2))

# Creating a window for HSV track bars

cv2.namedWindow('HSV\_TrackBar')

# Starting with 100's to prevent error while masking

h,s,v = 100,100,100

# Creating track bar

cv2.createTrackbar('h', 'HSV\_TrackBar',0,179,nothing)

cv2.createTrackbar('s', 'HSV\_TrackBar',0,255,nothing)

cv2.createTrackbar('v', 'HSV\_TrackBar',0,255,nothing)

while(1):

#Measure execution time

start\_time = time.time()

#Capture frames from the camera

ret, frame = cap.read()

#Blur the image

blur = cv2.blur(frame,(3,3))

#Convert to HSV color space

hsv = cv2.cvtColor(blur,cv2.COLOR\_BGR2HSV)

#Create a binary image with where white will be skin colors and rest is black

mask2 = cv2.inRange(hsv,np.array([2,50,50]),np.array([15,255,255]))

#Kernel matrices for morphological transformation

kernel\_square = np.ones((11,11),np.uint8)

kernel\_ellipse= cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(5,5))

#Perform morphological transformations to filter out the background noise

#Dilation increase skin color area

#Erosion increase skin color area

dilation = cv2.dilate(mask2,kernel\_ellipse,iterations = 1)

erosion = cv2.erode(dilation,kernel\_square,iterations = 1)

dilation2 = cv2.dilate(erosion,kernel\_ellipse,iterations = 1)

filtered = cv2.medianBlur(dilation2,5)

kernel\_ellipse= cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(8,8))

dilation2 = cv2.dilate(filtered,kernel\_ellipse,iterations = 1)

kernel\_ellipse= cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(5,5))

dilation3 = cv2.dilate(filtered,kernel\_ellipse,iterations = 1)

median = cv2.medianBlur(dilation2,5)

ret,thresh = cv2.threshold(median,127,255,0)

#Find contours of the filtered frame

\_,contours, \_ = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

#Draw Contours

#cv2.drawContours(frame, cnt, -1, (122,122,0), 3)

#cv2.imshow('Dilation',median)

#Find Max contour area (Assume that hand is in the frame)

max\_area=100

ci=0

for i in range(len(contours)):

cnt=contours[i]

area = cv2.contourArea(cnt)

if(area>max\_area):

max\_area=area

ci=i

#Largest area contour

cnts = contours[ci]

#Find convex hull

hull = cv2.convexHull(cnts)

#Find convex defects

hull2 = cv2.convexHull(cnts,returnPoints = False)

defects = cv2.convexityDefects(cnts,hull2)

#Get defect points and draw them in the original image

FarDefect = []

for i in range(defects.shape[0]):

s,e,f,d = defects[i,0]

start = tuple(cnts[s][0])

end = tuple(cnts[e][0])

far = tuple(cnts[f][0])

FarDefect.append(far)

cv2.line(frame,start,end,[0,255,0],1)

cv2.circle(frame,far,10,[100,255,255],3)

#Find moments of the largest contour

moments = cv2.moments(cnts)

#Central mass of first order moments

if moments['m00']!=0:

cx = int(moments['m10']/moments['m00']) # cx = M10/M00

cy = int(moments['m01']/moments['m00']) # cy = M01/M00

centerMass=(cx,cy)

#Draw center mass

cv2.circle(frame,centerMass,7,[100,0,255],2)

font = cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(frame,'Center',tuple(centerMass),font,2,(255,255,255),2)

#Distance from each finger defect(finger webbing) to the center mass

distanceBetweenDefectsToCenter = []

for i in range(0,len(FarDefect)):

x = np.array(FarDefect[i])

centerMass = np.array(centerMass)

distance = np.sqrt(np.power(x[0]-centerMass[0],2)+np.power(x[1]-centerMass[1],2))

distanceBetweenDefectsToCenter.append(distance)

#Get an average of three shortest distances from finger webbing to center mass

sortedDefectsDistances = sorted(distanceBetweenDefectsToCenter)

AverageDefectDistance = np.mean(sortedDefectsDistances[0:2])

#Get fingertip points from contour hull

#If points are in proximity of 80 pixels, consider as a single point in the group

finger = []

for i in range(0,len(hull)-1):

if (np.absolute(hull[i][0][0] - hull[i+1][0][0]) > 80) or ( np.absolute(hull[i][0][1] - hull[i+1][0][1]) > 80):

if hull[i][0][1] <500 :

finger.append(hull[i][0])

#The fingertip points are 5 hull points with largest y coordinates

finger = sorted(finger,key=lambda x: x[1])

fingers = finger[0:5]

#Calculate distance of each finger tip to the center mass

fingerDistance = []

for i in range(0,len(fingers)):

distance = np.sqrt(np.power(fingers[i][0]-centerMass[0],2)+np.power(fingers[i][1]-centerMass[0],2))

fingerDistance.append(distance)

#Finger is pointed/raised if the distance of between fingertip to the center mass is larger

#than the distance of average finger webbing to center mass by 130 pixels

result = 0

for i in range(0,len(fingers)):

if fingerDistance[i] > AverageDefectDistance+130:

result = result +1

#Print number of pointed fingers

cv2.putText(frame,str(result),(100,100),font,2,(255,255,255),2)

#show height raised fingers

#cv2.putText(frame,'finger1',tuple(finger[0]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger2',tuple(finger[1]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger3',tuple(finger[2]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger4',tuple(finger[3]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger5',tuple(finger[4]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger6',tuple(finger[5]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger7',tuple(finger[6]),font,2,(255,255,255),2)

#cv2.putText(frame,'finger8',tuple(finger[7]),font,2,(255,255,255),2)

#Print bounding rectangle

x,y,w,h = cv2.boundingRect(cnts)

img = cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

cv2.drawContours(frame,[hull],-1,(255,255,255),2)

##### Show final image ########

cv2.imshow('Dilation',frame)

###############################

#Print execution time

#print time.time()-start\_time

#close the output video by pressing 'ESC'

k = cv2.waitKey(5) & 0xFF

if k == 27:

break

cap.release()

cv2.destroyAllWindows()