Color space : HSV, YCC

Convex Hull

Convexity defect

EmguCV, csharp

1. Skin color detection
2. Convex hull
3. Convex defects
4. OpenCV documentation: <http://docs.opencv.org/2.4.13/>
5. Opencv python hand gesture recognition: <http://creat-tabu.blogspot.com/2013/08/opencv-python-hand-gesture-recognition.html>
6. Mahaveerverma's hand gesture recognition project: [hand-gesture-recognition-opencv](https://github.com/mahaveerverma/hand-gesture-recognition-opencv)

<http://www.handresearch.com/news/hand-gestures-control-your-tv.htm>

Talks about hand gestures for TV control have been since 2007

Bluetooth IR blaster

Raspberry Pi, skywriter HAT

Hackaday.com

Enough lighting , camera angle, inconvenient

Depth based hand tracking

Cameras, however, have a number of line of sight-related challenges that may prevent gesture recognition from being effective

 For example, poorly-lit environments may have a negative impact on the image quality and in turn degrade the performance of gesture detection through the camera. The other main issue with camera-based gesture recognition is privacy

Naturally, it becomes more difficult to remember the gestural command set as the number of gestures increase.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5422194/>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.7688&rep=rep1&type=pdf>

Doppler radar, UWB radar

K-means clustering algorithm

In order to differentiate the gesture motion from some random hand or body motion, we included a data-fitting algorithm. Since the gesture motion defined in our work is almost periodic, therefore we fit the received gesture signal into a sinusoid and check the R-square value. If the R-square value is above a certain threshold, then it is supposed to be periodic and, hence, classified as a gesture signal; otherwise, it is classified as a non-gesture motion.

The main contribution of our work is that it is the first real-time IR-UWB-based gesture recognition technique, which avoids the overfitting problem and shows robustness when a change in distance or orientation of the hand occurs, because of the selection of robust parameters and the integration of the TOA information into the clustering algorithm. Additionally, we proposed an algorithm for the detection of only intended gestures while ignoring any random movement in front of the radar sensor. Considering these advantages, this method would be an important technology of the car user interface as one of the core technologies of the future autonomous vehicles.

5.2 Natural Hand Gesture Interfaces At the simplest level, gesture interfaces can be developed which respond to natural dynamic hand motion. An early example is the Theremin, an electronic musical instrument from the 1920's. The Theremin responds to hand position using two proximity sensors, one vertical, and one horizontal. Proximity to the vertical sensor controls the music pitch, proximity to the horizontal one controls volume. The Theremin is successful because there is a direct mapping of hand motion to continuous audible feedback, enabling the user to quickly build a mental model of how to use the device. Myron Krueger's Videoplace [17] is another system that responds to natural user gesture. Developed in the late 1970's and early 80's, Videoplace uses real time image processing of live video of the user. Background subtraction and edge detection are used to create a silhouette of the user and identify relevant features. The feature recognition is sufficiently fine to distinguish between hand and fingers, whether fingers are extended or closed, and even which fingers. With this capability, the system has been programmed to perform a number of interactions, many of which closely echo our use of gesture in the everyday world. The key to its success is the recognition of dynamic natural hand gestures, so users require no training. These types of natural gesture that can be easily remembered, causing minimum cognitive driver workload, make them an attractive proposition for use with in-vehicle secondary controls.

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, hierarchy = 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()