UNIVERSITY OF SINDH

Name & Roll No

1. *Muhammad Furqan:2K20/ITE/81*
2. *Owais Mustafa : 2K20/ITE/99*
3. *Hassam Ullah Khan : 2K20/ITE/53*
4. *Mujtaba Rajput : 2K20/ITE/88*

Class

BSIT Part 03 (Evening)

Teacher Name

Dr Sundar Ali Khowaja

Topic Name

Gesture Recognition

Data Science Assigment

Real-time Hand Gesture Recognition using OpenCV:

Gesture recognition is an active research field in Human-Computer Interaction technology. It has many applications in virtual environment control and sign language translation, robot control, or music creation. In this machine learning project on Hand Gesture Recognition, we are going to make a real-time Hand Gesture Recognizer using OpenCV and Python.

OpenCV is a real-time Computer vision and image-processing framework built on C/C++. But we'll use it on python via the OpenCV-python package.

Prerequisites for this project:

1. Python – 3.x (we used Python 3.8.8 in this project)

2. OpenCV – 4.5

Steps to solve the project:

- *Import necessary packages.*
- *Initialize models.*
- *Read frames from a webcam.*
- *Detect hand key points.*
- *Recognize hand gestures.*

Import necessary packages:

To build this Hand Gesture Recognition project, we'll need two packages. So first import these.

 # import necessary packages for hand gesture recognition project using Python OpenCV

import cv2

import numpy as np

Recognize hand gestures:

 # Predict gesture in Hand Gesture Recognition project

prediction = model.predict([landmarks])

print(prediction)

classID = np.argmax(prediction)

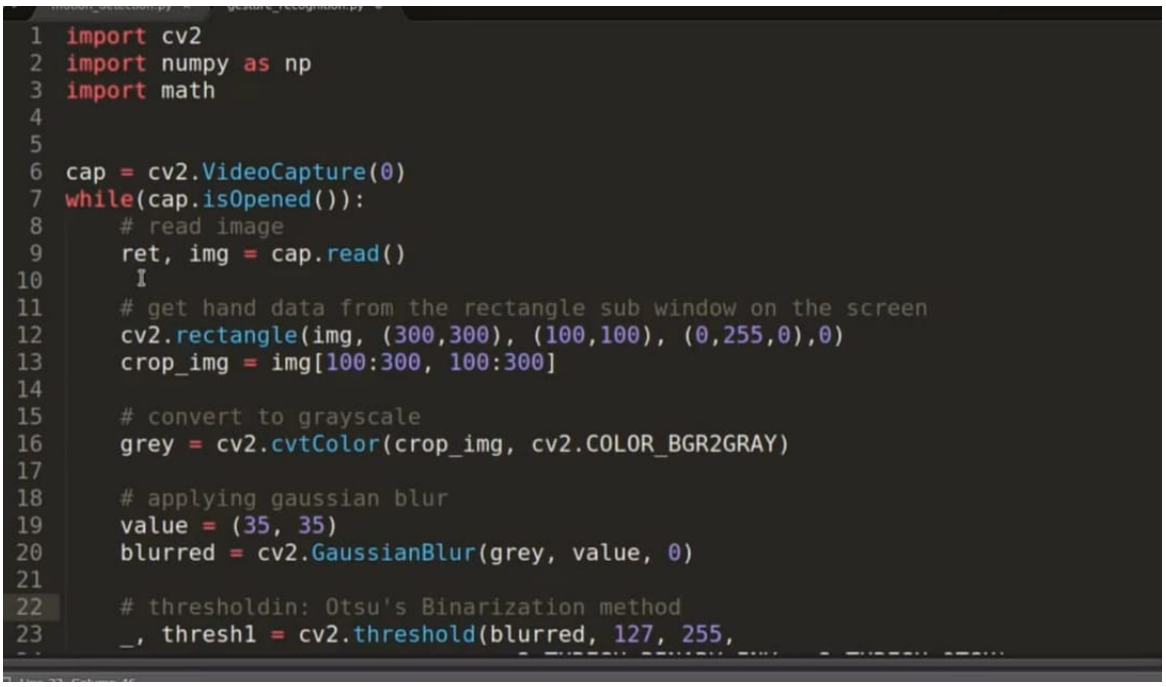
className = classNames[classID]

 # show the prediction on the frame

cv2.putText(frame, className, (10, 50), cv2.FONT_HERSHEY_SIMPLEX,

1, (0,0,255), 2, cv2.LINE_AA)

Coding :



```
import cv2
import numpy as np
import math


cap = cv2.VideoCapture(0)
while(cap.isOpened()):
    # read image
    ret, img = cap.read()

    # get hand data from the rectangle sub window on the screen
    cv2.rectangle(img, (300,300), (100,100), (0,255,0),0)
    crop_img = img[100:300, 100:300]

    # convert to grayscale
    grey = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)

    # applying gaussian blur
    value = (35, 35)
    blurred = cv2.GaussianBlur(grey, value, 0)

    # thresholdin: Otsu's Binarization method
    _, thresh1 = cv2.threshold(blurred, 127, 255,
```

File Edit Selection Find View Goto Tools Project Preferences Help

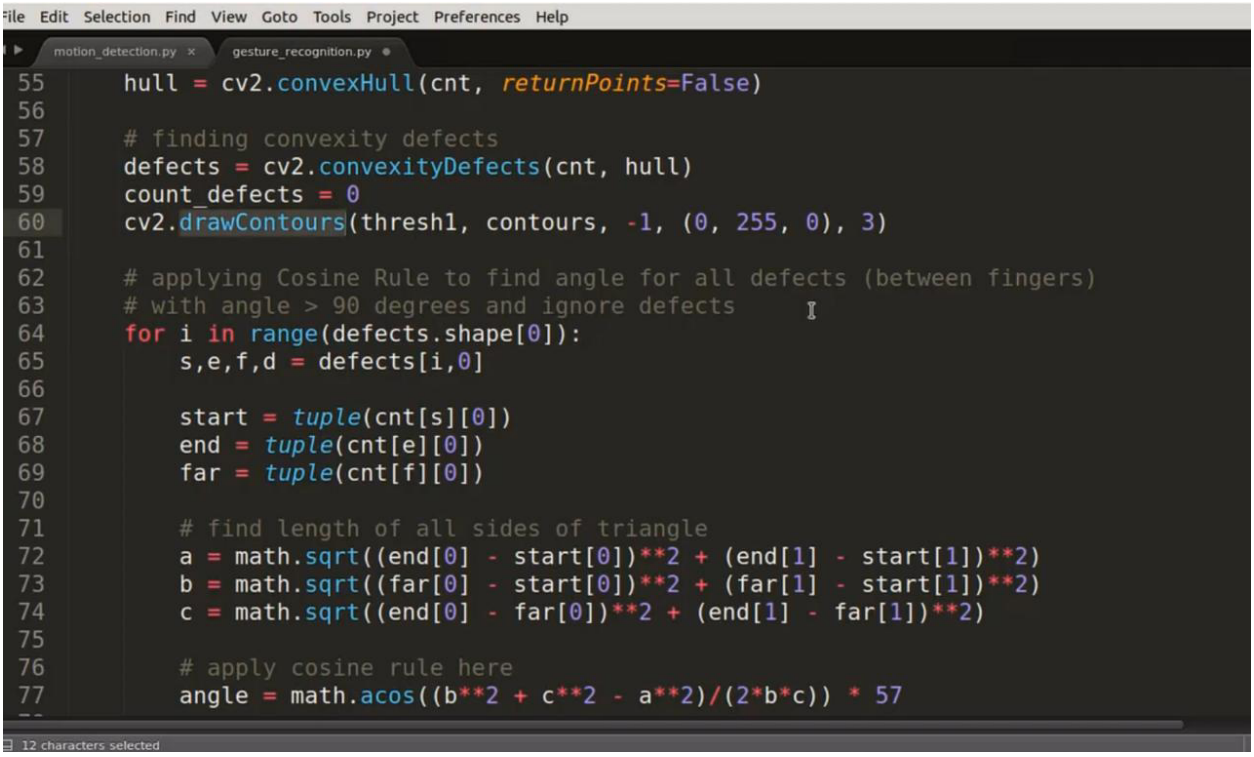
motion_detection.py × gesture_recognition.py •

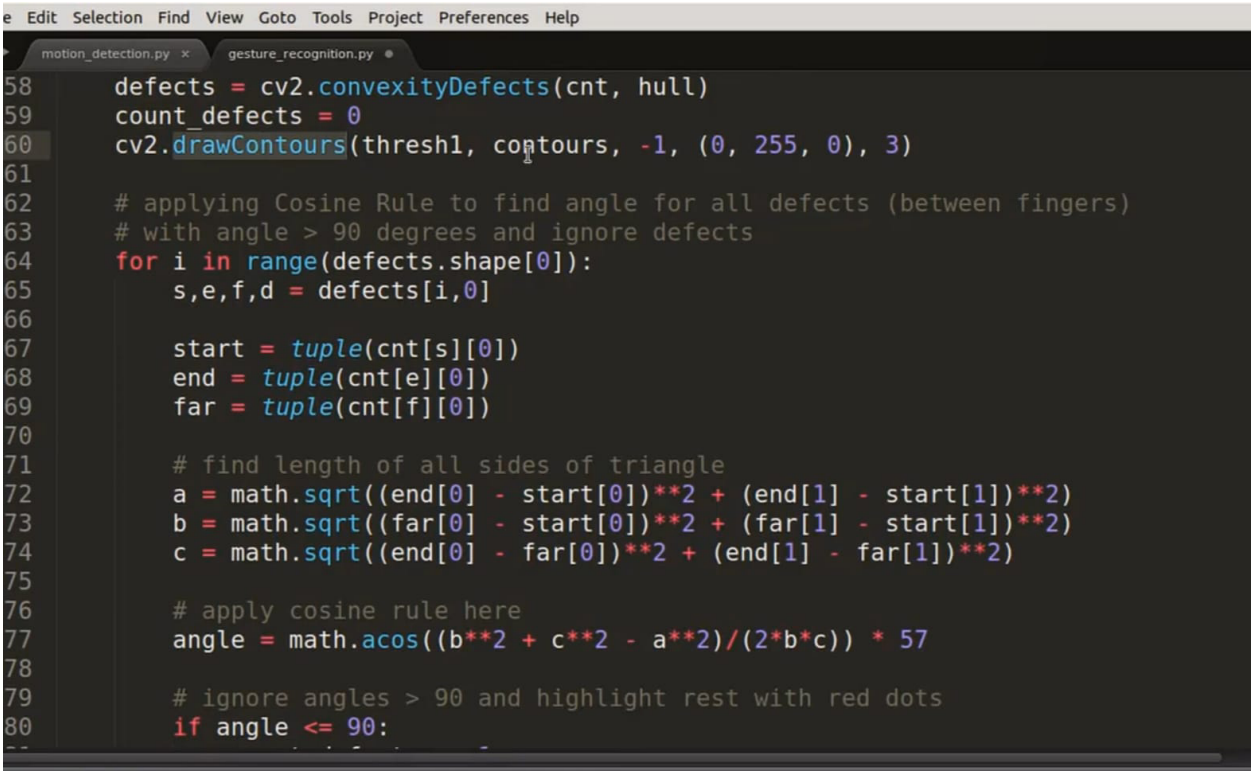
```
22      # thresholdin: Otsu's Binarization method
23      _, thresh1 = cv2.threshold(blurred, 127, 255,
24                                 cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
25
26      # show thresholded image
27      cv2.imshow('Thresholded', thresh1)
28
29      # check OpenCV version to avoid unpacking error
30      (version, _, _) = cv2.__version__.split('.')
31
32      if version == '3':
33          image, contours, hierarchy = cv2.findContours(thresh1.copy(), \
34                  cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
35      elif version == '4':
36          contours, hierarchy = cv2.findContours(thresh1.copy(),cv2.RETR_TREE, \
37                  cv2.CHAIN_APPROX_NONE)
38
39      # find contour with max area
40      cnt = max(contours, key = lambda x: cv2.contourArea(x))
41
42      # create bounding rectangle around the contour (can skip below two lines)
43      x, y, w, h = cv2.boundingRect(cnt)
44      cv2.rectangle(crop_img, (x, y), (x+w, y+h), (0, 0, 255), 0)
```

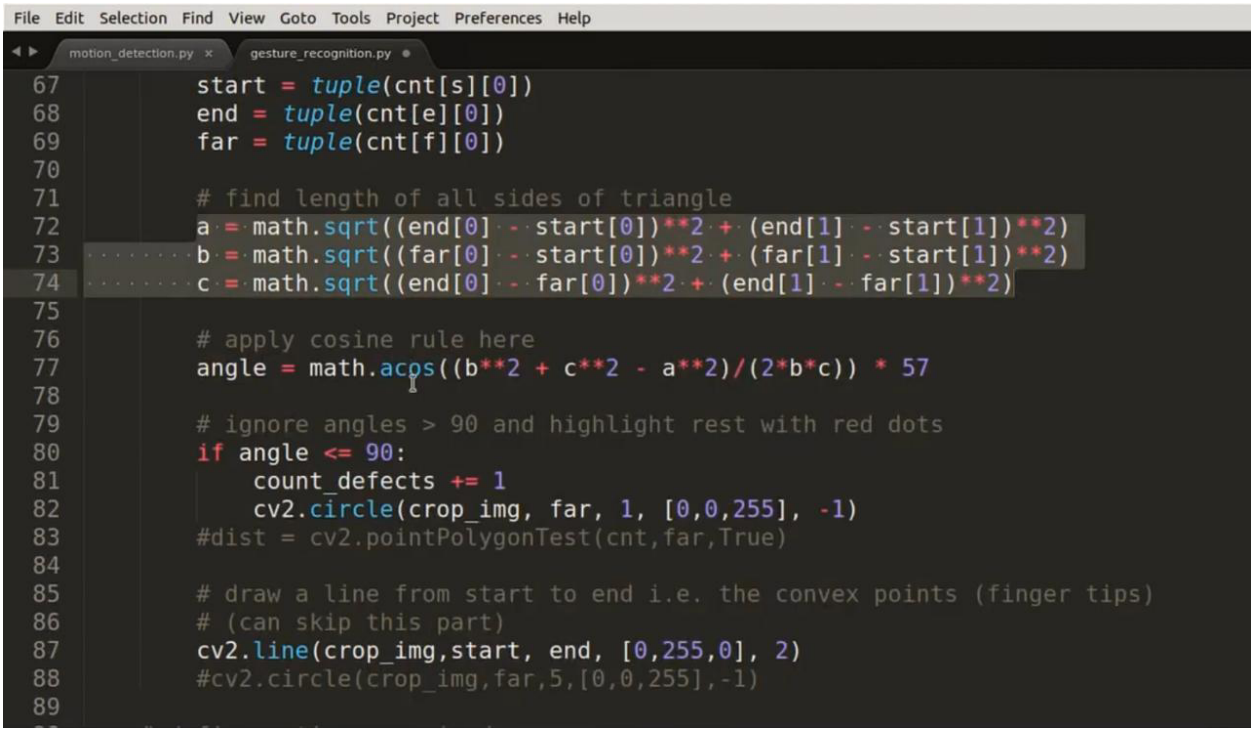
File Edit Selection Find View Goto Tools Project Preferences Help

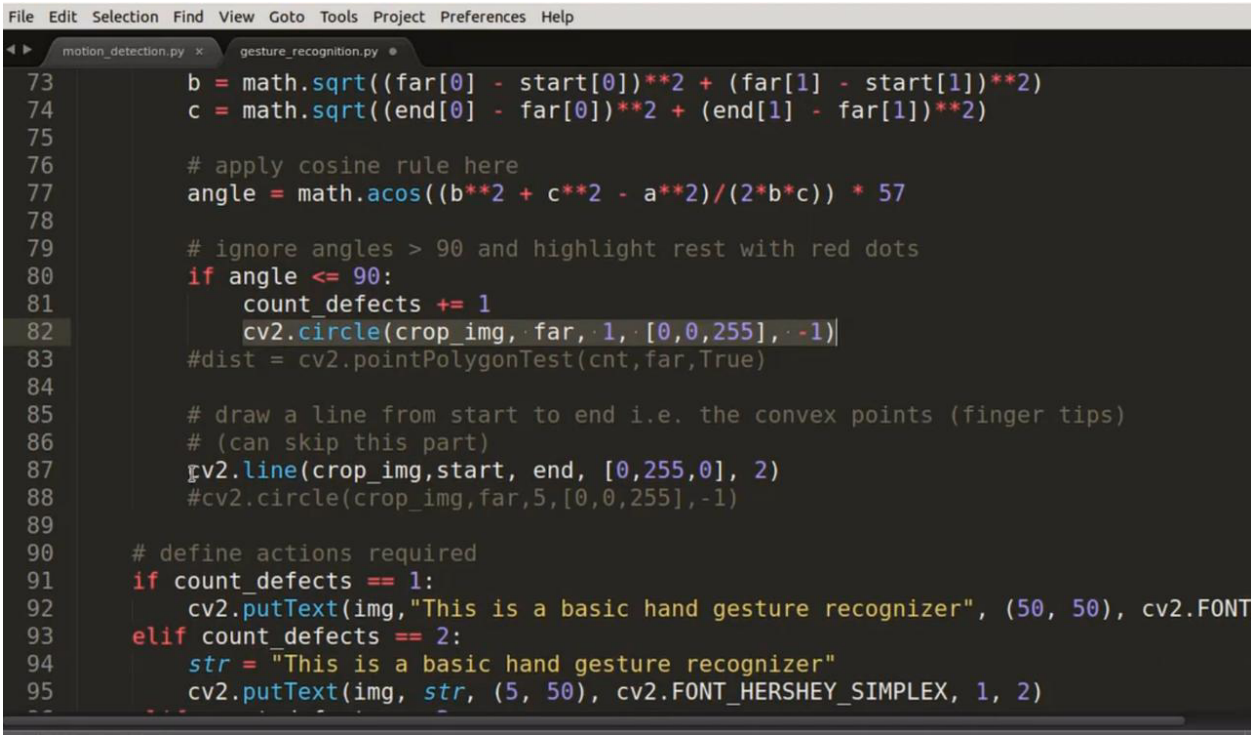
motion_detection.py × gesture_recognition.py •

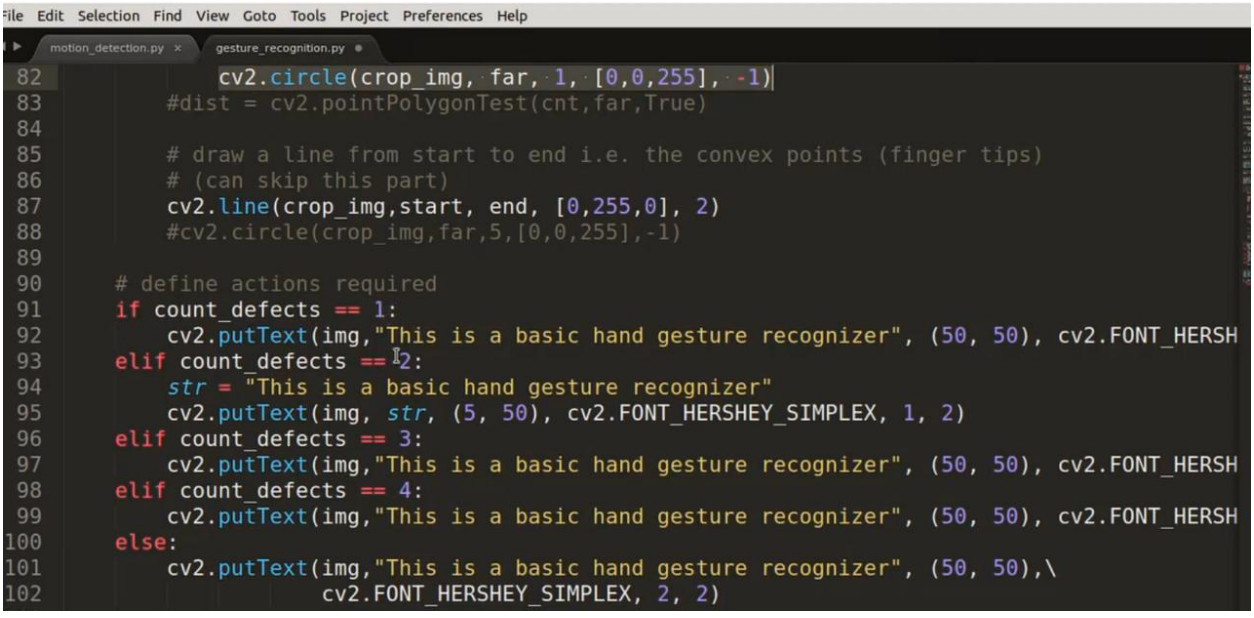
```
40      cnt = max(contours, key = lambda x: cv2.contourArea(x))
41
42      # create bounding rectangle around the contour (can skip below two lines)
43      x, y, w, h = cv2.boundingRect(cnt)
44      cv2.rectangle(crop_img, (x, y), (x+w, y+h), (0, 0, 255), 0)
45
46      # finding convex hull
47      hull = cv2.convexHull(cnt)
48
49      # drawing contours
50      drawing = np.zeros(crop_img.shape,np.uint8)
51      cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
52      cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)
53
54      # finding convex hull
55      hull = cv2.convexHull(cnt, returnPoints=False)
56
57      # finding convexity defects
58      defects = cv2.convexityDefects(cnt, hull)
59      count_defects = 0
60      cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)
61
62      # applying Cosine Rule to find angle for all defects (between fingers)
```


File Edit Selection Find View Goto Tools Project Preferences Help
motion_detection.py
gesture_recognition.py
hull = cv2.convexHull(cnt, returnPoints=False)
# finding convexity defects
defects = cv2.convexityDefects(cnt, hull)
count_defects = 0
cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)
# applying Cosine Rule to find angle for all defects (between fingers)
# with angle > 90 degrees and ignore defects
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])
# find length of all sides of triangle
a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
# apply cosine rule here
angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57
12 characters selected

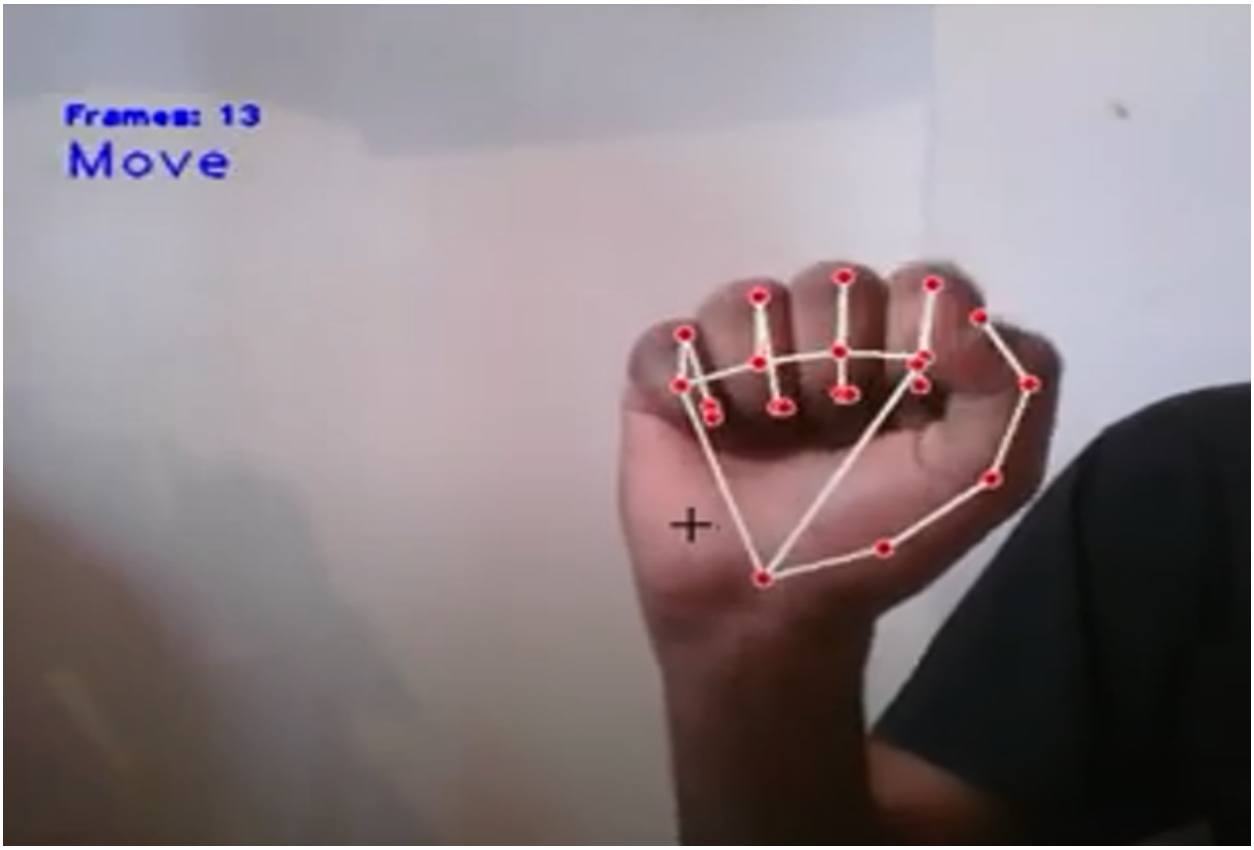

e Edit Selection Find View Goto Tools Project Preferences Help
motion_detection.py
gesture_recognition.py
defects = cv2.convexityDefects(cnt, hull)
count_defects = 0
cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)
# applying Cosine Rule to find angle for all defects (between fingers)
# with angle > 90 degrees and ignore defects
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])
# find length of all sides of triangle
a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
# apply cosine rule here
angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57
# ignore angles > 90 and highlight rest with red dots
if angle <= 90:

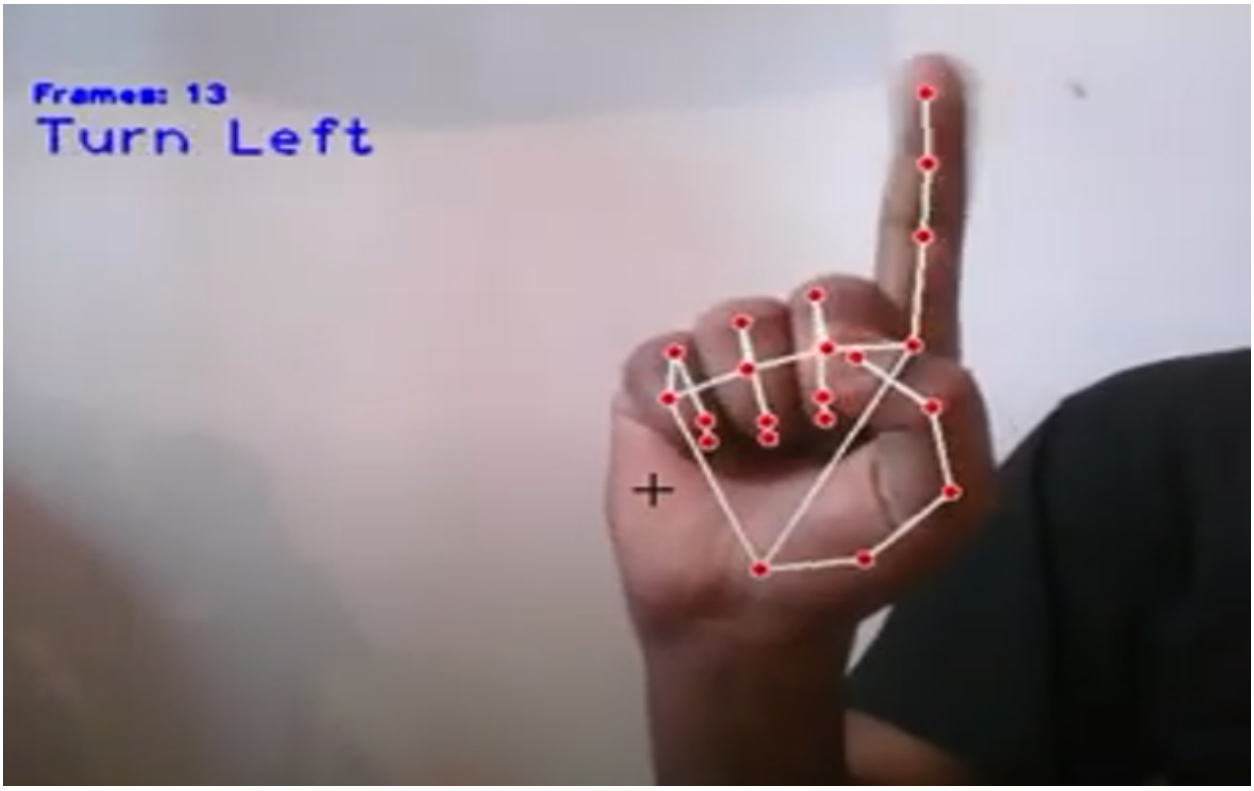

File Edit Selection Find View Goto Tools Project Preferences Help
motion_detection.py
gesture_recognition.py
start = tuple(cnt[s][0])
end = tuple(cnt[e][0])
far = tuple(cnt[f][0])
# find length of all sides of triangle
a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
# apply cosine rule here
angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57
# ignore angles > 90 and highlight rest with red dots
if angle <= 90:
count_defects += 1
cv2.circle(crop_img, far, 1, [0,0,255], -1)
#dist = cv2.pointPolygonTest(cnt,far,True)
# draw a line from start to end i.e. the convex points (finger tips)
# (can skip this part)
cv2.line(crop_img,start, end, [0,255,0], 2)
#cv2.circle(crop_img,far,5,[0,0,255],-1)

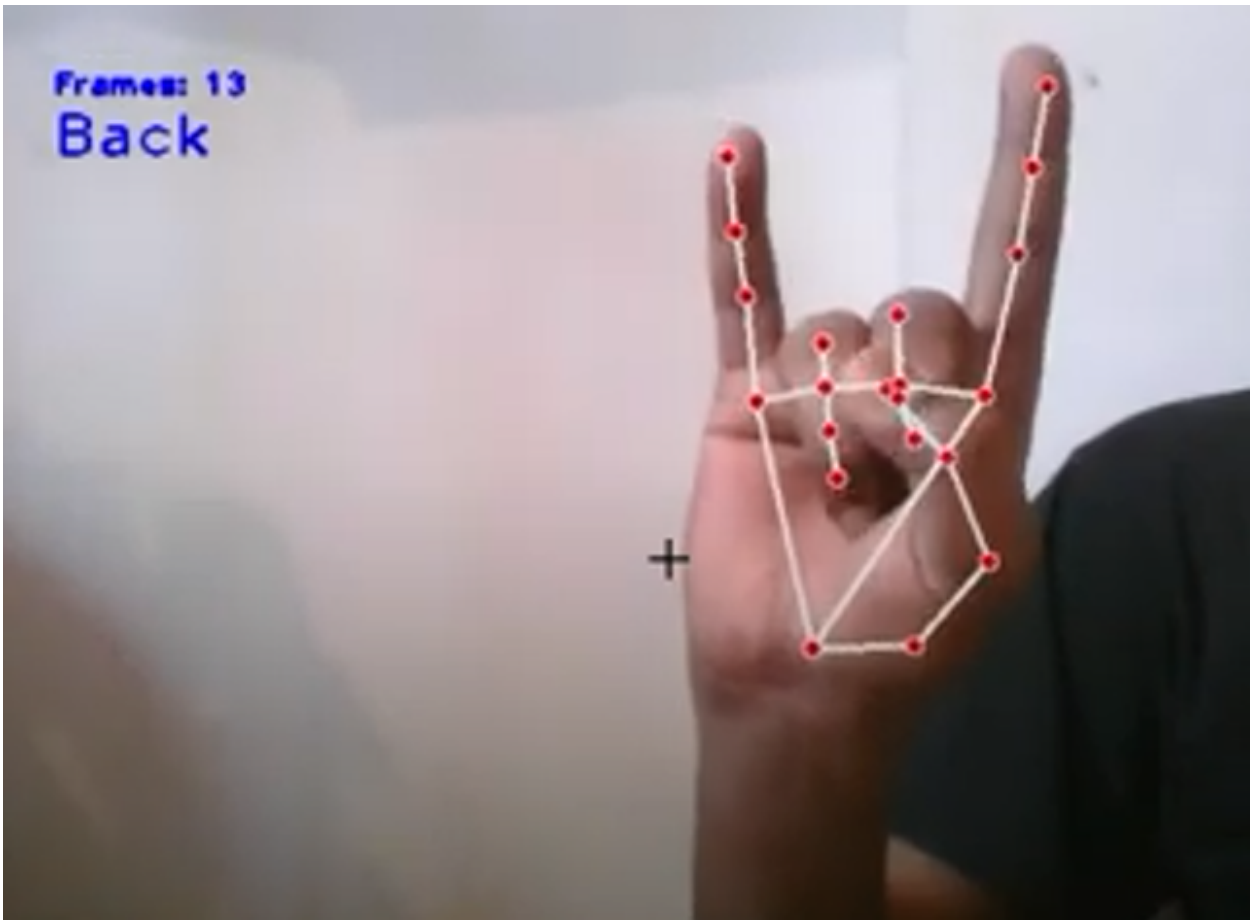

File Edit Selection Find View Goto Tools Project Preferences Help
motion_detection.py
gesture_recognition.py
b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2
# apply cosine rule here
angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57
# ignore angles > 90 and highlight rest with red dots
if angle <= 90:
count_defects += 1
cv2.circle(crop_img, far, 1, [0,0,255], -1)
#dist = cv2.pointPolygonTest(cnt,far,True)
# draw a line from start to end i.e. the convex points (finger tips)
# (can skip this part)
cv2.line(crop_img,start, end, [0,255,0], 2)
#cv2.circle(crop_img,far,5,[0,0,255],-1)
# define actions required
if count_defects == 1:
cv2.putText(img,"This is a basic hand gesture recognizer", (50, 50), cv2.FONT
elif count_defects == 2:
str = "This is a basic hand gesture recognizer"
cv2.putText(img, str, (5, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, 2)


File Edit Selection Find View Goto Tools Project Preferences Help
motion_detection.py
gesture_recognition.py
cv2.circle(crop_img, far, 1, [0,0,255], -1)
#dist = cv2.pointPolygonTest(cnt,far,True)
# draw a line from start to end i.e. the convex points (finger tips)
# (can skip this part)
cv2.line(crop_img,start, end, [0,255,0], 2)
#cv2.circle(crop_img,far,5,[0,0,255],-1)
# define actions required
if count_defects == 1:
cv2.putText(img,"This is a basic hand gesture recognizer", (50, 50), cv2.FONT_HERSH
elif count_defects == 2:
str = "This is a basic hand gesture recognizer"
cv2.putText(img, str, (5, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
elif count_defects == 3:
cv2.putText(img,"This is a basic hand gesture recognizer", (50, 50), cv2.FONT_HERSH
elif count_defects == 4:
cv2.putText(img,"This is a basic hand gesture recognizer", (50, 50), cv2.FONT_HERSH
else:
cv2.putText(img,"This is a basic hand gesture recognizer", (50, 50),\
cv2.FONT_HERSHEY_SIMPLEX, 2, 2)

Result :




Frames: 13
Turn Left



Summary:

In this Hand Gesture Recognition project, we've built a hand gesture recognizer using OpenCV and python.

We've used framework for the detection and gesture recognition respectively.

Here we've learned about the basics of File handling, some common image processing techniques, etc.