|  |
| --- |
| 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, (400,400), (100,100), (0,255,0),0) |
|  | crop\_img = img[100:400, 100:400] |
|  |  |
|  | # convert to grayscale |
|  | grey = cv2.cvtColor(crop\_img, cv2.COLOR\_BGR2GRAY) |
|  |  |
|  | # applying gaussian blur |
|  | value = (35, 35) |
|  | blurred = cv2.GaussianBlur(grey, value, 0) |
|  | cv2.imshow('blurred',blurred) |
|  |  |
|  | # thresholdin: Otsu's Binarization method |
|  | \_, thresh1 = cv2.threshold(blurred, 2, 255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU) |
|  |  |
|  | # show thresholded image |
|  | cv2.imshow('Thresholded', thresh1) |
|  |  |
|  | image, contours, hierarchy = cv2.findContours(thresh1.copy(),cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE) |
|  |  |
|  | # find contour with max area |
|  | cnt = max(contours, key = lambda x: cv2.contourArea(x)) |
|  |  |
|  | # create bounding rectangle around the contour |
|  | x, y, w, h = cv2.boundingRect(cnt) |
|  | cv2.rectangle(crop\_img, (x, y), (x+w, y+h), (0, 0, 255), 0) |
|  |  |
|  | # finding convex hull |
|  | hull = cv2.convexHull(cnt) |
|  |  |
|  | # drawing contours |
|  | drawing = np.zeros(crop\_img.shape,np.uint8) |
|  | cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0) |
|  | cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0) |
|  |  |
|  | # finding convex hull |
|  | 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 |
|  |  |
|  | # 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 two!", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2) |
|  | elif count\_defects == 2: |
|  | str = "This is three!" |
|  | cv2.putText(img, str, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2) |
|  | elif count\_defects == 3: |
|  | cv2.putText(img,"This is four!", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2) |
|  | elif count\_defects == 4: |
|  | cv2.putText(img,"This is five!", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2) |
|  | else: |
|  | cv2.putText(img,"This is one!", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2) |
|  |  |
|  | # show appropriate images in windows |
|  | cv2.imshow('Gesture', img) |
|  | all\_img = np.hstack((drawing, crop\_img)) |
|  | cv2.imshow('Contours', all\_img) |
|  |  |
|  | k = cv2.waitKey(10) |
|  | if k == 27: |
|  | break |