|  |
| --- |
|  |
| import cv2 |
|  | import numpy as np |
|  | import math |
|  | import RPi.GPIO as GPIO |
|  | import time |
|  |  |
|  | GPIO.setmode(GPIO.BCM) |
|  |  |
|  | #set GPIO Pins |
|  | GPIO\_TRIGGER = 18 |
|  | GPIO\_ECHO = 24 |
|  |  |
|  | menu= [False,False] |
|  | sub\_menu = [[False,False],[False,False]] |
|  | indicator = [ 2, 3] |
|  | appliance = [(22,10),(14,15)] |
|  | no\_menu = 4 |
|  | aalert = 0 |
|  | #set GPIO direction (IN / OUT) |
|  | GPIO.setup(GPIO\_TRIGGER, GPIO.OUT) |
|  | GPIO.setup(GPIO\_ECHO, GPIO.IN) |
|  | GPIO.setup(22, GPIO.OUT) |
|  | GPIO.setup(10, GPIO.OUT) |
|  | GPIO.setup(14, GPIO.OUT) |
|  | GPIO.setup(15, GPIO.OUT) |
|  | GPIO.setup(2, GPIO.OUT) |
|  | GPIO.setup(3, GPIO.OUT) |
|  | GPIO.setup(4, GPIO.OUT) |
|  |  |
|  | def project(): |
|  | 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(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, 2, 255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU) |
|  | cv2.imshow('Thresholded', thresh1) |
|  | 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: |
|  | result = 2 |
|  | elif count\_defects == 2: |
|  | result = 3 |
|  | elif count\_defects == 3: |
|  | result = 4 |
|  | elif count\_defects == 4: |
|  | result = 5 |
|  | else: |
|  | result = 1 |
|  |  |
|  | # 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 |
|  | return result |
|  |  |
|  | def distance(): |
|  | # set Trigger to HIGH |
|  | GPIO.output(GPIO\_TRIGGER, True) |
|  |  |
|  | # set Trigger after 0.01ms to LOW |
|  | time.sleep(0.00001) |
|  | GPIO.output(GPIO\_TRIGGER, False) |
|  |  |
|  | StartTime = time.time() |
|  | StopTime = time.time() |
|  |  |
|  | # save StartTime |
|  | while GPIO.input(GPIO\_ECHO) == 0: |
|  | StartTime = time.time() |
|  |  |
|  | # save time of arrival |
|  | while GPIO.input(GPIO\_ECHO) == 1: |
|  | StopTime = time.time() |
|  |  |
|  | # time difference between start and arrival |
|  | TimeElapsed = StopTime - StartTime |
|  | # multiply with the sonic speed (34300 cm/s) |
|  | # and divide by 2, because there and back |
|  | distance = (TimeElapsed \* 34300) / 2 |
|  |  |
|  | return distance |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | GPIO.cleanup() |
|  | while True: |
|  | #try: |
|  | #dist = distance() |
|  | #if dist < 50: |
|  | finger = project() |
|  | print finger |
|  | time.sleep(1) |
|  | #except KeyboardInterrupt: |
|  | #print("Measurement stopped by User") |
|  | if finger == 1 or finger == 4 or finger == 5: |
|  | GPIO.output(4, GPIO.HIGH) |
|  | elif finger == 2 or finger == 3: |
|  | i = finger - 2 |
|  | GPIO.output( indicator[i], menu[i]) |
|  | #dist = distance() |
|  | #while(dist > 50): |
|  | #dist = distance() |
|  | finger = project() |
|  | if finger == 2 or finger == 3: |
|  | j=finger-2 |
|  | sub\_menu[i][j] = not sub\_menu[i][j] |
|  | menu[i] = not menu[i] |
|  | GPIO.output(appliance[i][j], sub\_menu[i][j]) |
|  | GPIO.output(indicator[i], menu[i]) |
|  | else: |
|  | menu[i] = not menu[i] |
|  | GPIO.output(indicator[i], menu[i]) |
|  | GPIO.output(no\_menu,GPIO.LOW) |
|  |  |
|  | time.sleep(1) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |