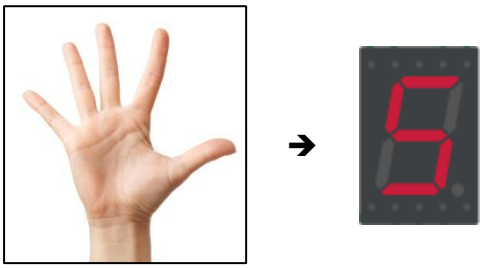
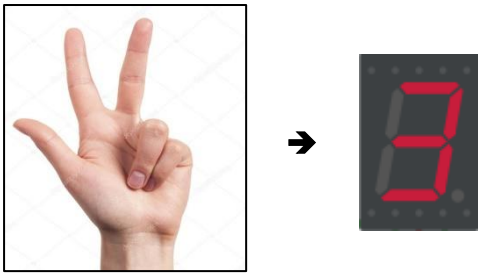
REAL TIME FINGER DETECTION

AIM:

To detect and display number of fingers shown in the camera using Python and Arduino.

Eg:

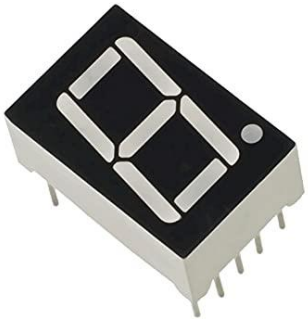




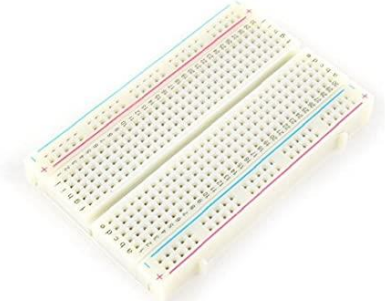
COMPONENETS REQUIRED:

1. Arduino UNO with Cable

2. 7 Segment display (CC)

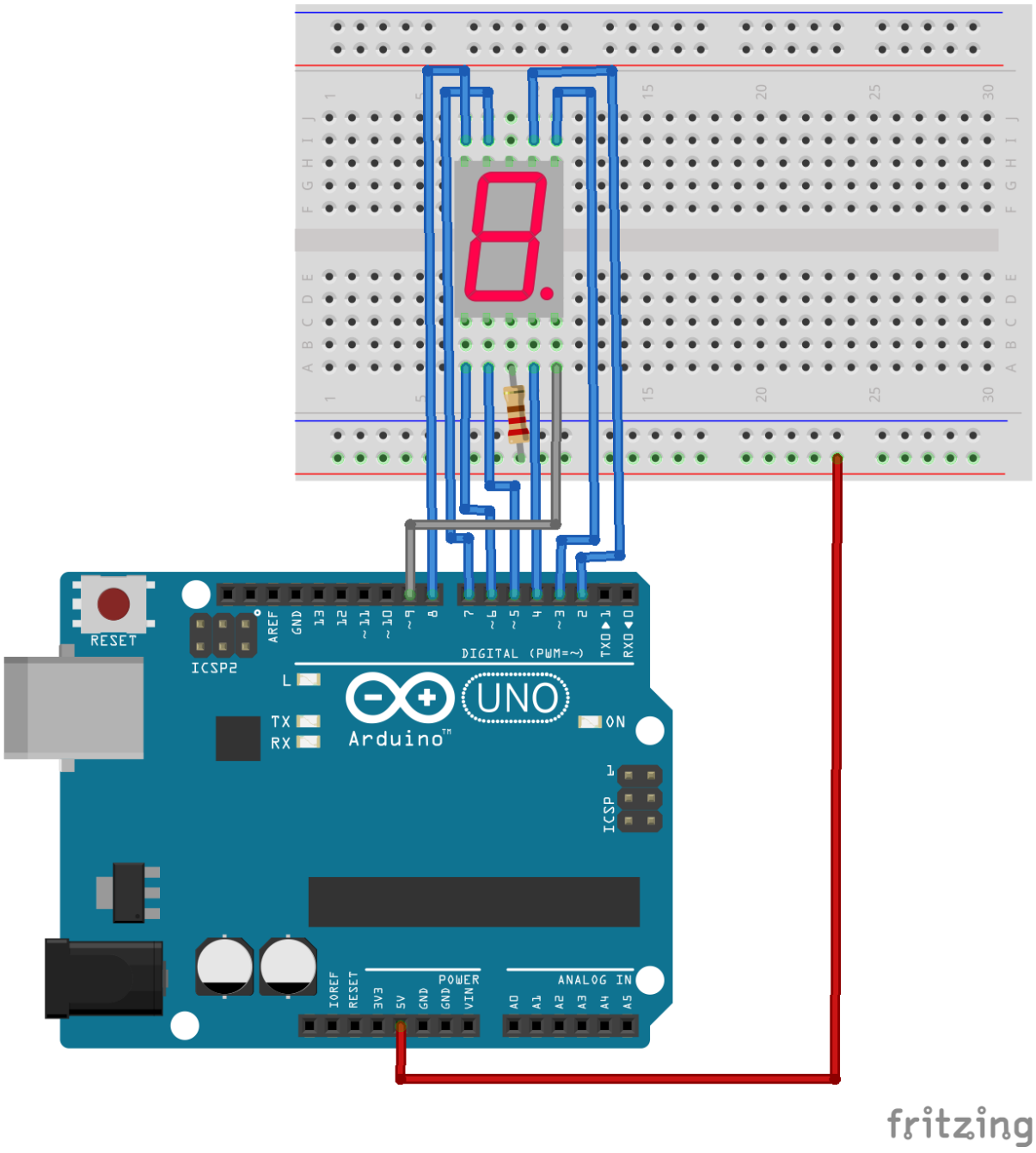


4. Breadboard



5. Jumper wires (Male – Male)

CIRCUIT DIAGRAM:



ARDUINO CODE:

```
#include <segment7.h>
segment7 disp(2,3,4,5,6,7,8,9); //a,b,c,d,e,f,g,dp

void setup()
{
  Serial.begin(9600);
  disp.PinSetup();
  disp.nothing();
  delay(100);
}
void loop()
{
  if(Serial.available())
  {
    char i = Serial.read();
    if(i == '0')
    {
      disp.zero(false);
      delay(200);
    }
    if(i == '1')
    {
      disp.one(false);
      delay(200);
    }
    if(i == '2')
    {
      disp.two(false);
      delay(200);
    }
    if(i == '3')
    {
      disp.three(false);
      delay(200);
    }
    if(i == '4')
    {
      disp.four(false);
      delay(200);
    }
    if(i == '5')
    {
      disp.five(false);
      delay(200);
    }
  }
  delay(100);
}
```

PYTHON CODE:

```
import cv2
import numpy as np
import math
import serial

myUno = serial.Serial('COM3',9600)

kernel1 = np.ones([1,1], np.uint8)
kernel4 = np.ones([3,3], np.uint8)
kernel = np.ones((5,5), np.float32)/25

def prepareFrame(img):
    wid = int(img.shape[1] * 1.25)
    hgt = int(img.shape[0] * 1.25)
    img = cv2.resize(img, (wid, hgt),
interpolation=cv2.INTER_AREA)
    img = cv2.flip(img, 1)
    cv2.rectangle(img, (565, 135), (765, 380), (0, 255, 0), 2,
cv2.LINE_8)
    cv2.putText(img, 'Show your HAND', (560, 130),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
    return img

def calFingers(res, drawing):
    global far
    hull = cv2.convexHull(res, returnPoints=False)
    if len(hull)>3:
        defects = cv2.convexityDefects(res, hull)
        if type(defects) != type(None):
            count = 0
            for i in range(defects.shape[0]):
                s, e, f, d = defects[i][0]
                start = tuple(res[s][0])
                end = tuple(res[e][0])
                far = tuple(res[f][0])
                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)
                angle = math.acos((b ** 2 + c ** 2 - a ** 2
)/(2*b*c))
                if angle<= math.pi /2 :
                    count += 1
                    cv2.circle(drawing, far, 3, [211,84,0], -
1)
            return  True, count
        return False,0
```

```
def getPos(x):
    pass

cv2.namedWindow('TrackBar')
cv2.resizeWindow('TrackBar', 400, 400)
cv2.createTrackbar('blurValue', 'TrackBar', 100, 255, getPos)
cv2.createTrackbar('LowHue', 'TrackBar', 0, 255, getPos)
cv2.createTrackbar('LowSat', 'TrackBar', 4, 255, getPos)
cv2.createTrackbar('LowVal', 'TrackBar', 125, 255, getPos)
cv2.createTrackbar('MaxHue', 'TrackBar', 10, 255, getPos)
cv2.createTrackbar('MaxSat', 'TrackBar', 140, 255, getPos)
cv2.createTrackbar('MaxVal', 'TrackBar', 255, 255, getPos)

cam = cv2.VideoCapture(0)
cam.set(3, 640)
cam.set(4, 480)

while(cam.isOpened()):
    ret, frame = cam.read()
    frame = prepareFrame(frame)
    img = frame[135:380, 565:765]
    cv2.imshow('Show the number', frame)
    blur_value = cv2.getTrackbarPos('blurValue', 'TrackBar')
    lh = cv2.getTrackbarPos('LowHue', 'TrackBar')
    ls = cv2.getTrackbarPos('LowSat', 'TrackBar')
    lv = cv2.getTrackbarPos('LowVal', 'TrackBar')
    mh = cv2.getTrackbarPos('MaxHue', 'TrackBar')
    ms = cv2.getTrackbarPos('MaxSat', 'TrackBar')
    mv = cv2.getTrackbarPos('MaxVal', 'TrackBar')
    l_hsv = np.array([lh, ls, lv])
    u_hsv = np.array([mh, ms, mv])
    # print(l_hsv, u_hsv, blur_value)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, l_hsv, u_hsv)
    res = cv2.bitwise_and(img, img, mask=mask)
    gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (9, 9), 0)
    blur = cv2.dilate(blur, kernel4)
    blur = cv2.erode(blur, kernel1, iterations=10)
    _, thresh = cv2.threshold(blur, blur_value, 255, 0)
    thresh = cv2.filter2D(thresh, -1, kernel)
    cv2.imshow('Thresh', thresh)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    index = 0
    maxArea = -1
    if len(contours) > 0:
        for i, cnt in enumerate(contours):
            area = cv2.contourArea(cnt)
            if area > maxArea:
                maxArea = area
```

```
                index = i
        larContour = contours[index]
        extTop = tuple(larContour[larContour[:, :,
1].argmin()][0])
        m = cv2.moments(larContour)
        cx = int(m['m10'] / m['m00'])
        cy = int(m['m01'] / m['m00'])
        hull = cv2.convexHull(larContour)
        drawing = np.zeros(img.shape, np.uint8)
        cv2.drawContours(drawing, [larContour], 0, (0, 255,
0), 2)
        cv2.drawContours(drawing, [hull], 0, (0, 0, 255), 3)
        font = cv2.FONT_HERSHEY_COMPLEX
        cv2.circle(drawing, extTop, 3, (255, 0, 0), -1)
        cv2.circle(drawing, (cx, cy), 3, (255, 255, 255), -1)
        cv2.putText(drawing, (str(extTop[0]) + ',' +
str(extTop[1])), (extTop), font, 0.5, (255, 255, 255), 1)
        cv2.putText(drawing, (str(cx) + ',' + str(cy)), (cx,
cy), font, 0.5, (255, 255, 255), 1)
        cv2.imshow('Num', drawing)
        isFinCal, count = calFingers(larContour, drawing)
        if count == 0:
            if cy - extTop[1] < 100:
                count = -1
            else:
                count = 0
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
    if k == ord('s'):
        c = str(count+1)
        print('The number is : ', c)
        myUno.write(c.encode('utf-8'))
        pass

cam.release()
cv2.destroyAllWindows()
```

DESCRIPTION:

In this project, we are going to detect how many fingers are raised and display the number in 7segment display. The libraries to be imported in python are **cv2** (opencv), **numpy**, **serial** and **math**. A camera window is opened and a green box is shown. Once the hand is placed inside the green box, using opencv the palm is alone filtered by setting **hsv** range and **blur** value manually. A closed contour is drawn around the palm.

The number of **convexity defects** is found. The number of fingers raised is one greater than number of convexity defects. For example, if the number of defects is 1, then number of fingers raised is 2. A condition is also used to filter out noise that the angle between two fingers must be less than 90^0. If the number of convexity defects found is 0, then the number of fingers raised may be 1 or 0 (no finger raised). To solve this problem, the top most point and center of the contour is found and if the distance between them are greater than a certain value, then one finger must be raised else no fingers are raised. The number of convexity defects is stored in variable **count**. If **esc** is pressed the loop **breaks**.

A communication between arduino and python is established using **serial** function at a baud rate of **9600**. If s is pressed the string **c** (count+1) is encoded and sent as char to the serial monitor of Arduino. In arduino IDE, the library imported is **segemnt7**. The value read in serial monitor is stored in character **i**. According to the character stored in i, the particular function is invoked. Here a common cathode 7segemnet display is used. The function **one** displays **1** in segment display.

Note:

- For learning purpose the coordinates of top most point and center of the contour is printed, if not needed that part of code can be removed.
- The argument of segment7 function should be **boolean**, if it is false the **dp** pin is set HIGH. Or if it is true, the dp pin glows.