HW 4

Collaborators: Yuhan Xie, Zhirong Lin

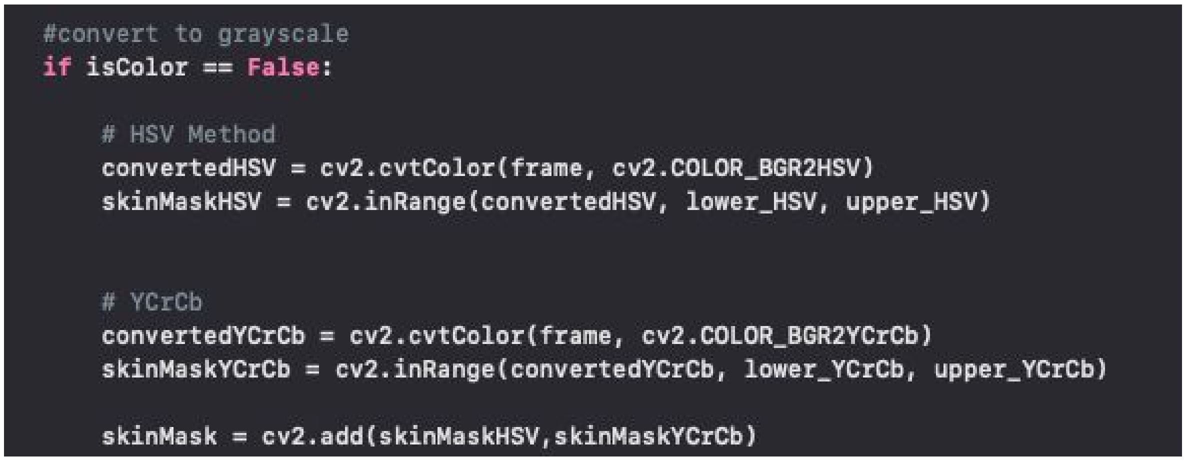
Video: https://youtu.be/jOgDHdC3v6A

Part 0

In part 0, we simply set up the camera using our webcams. We used cv2.imshow to display the hand image.

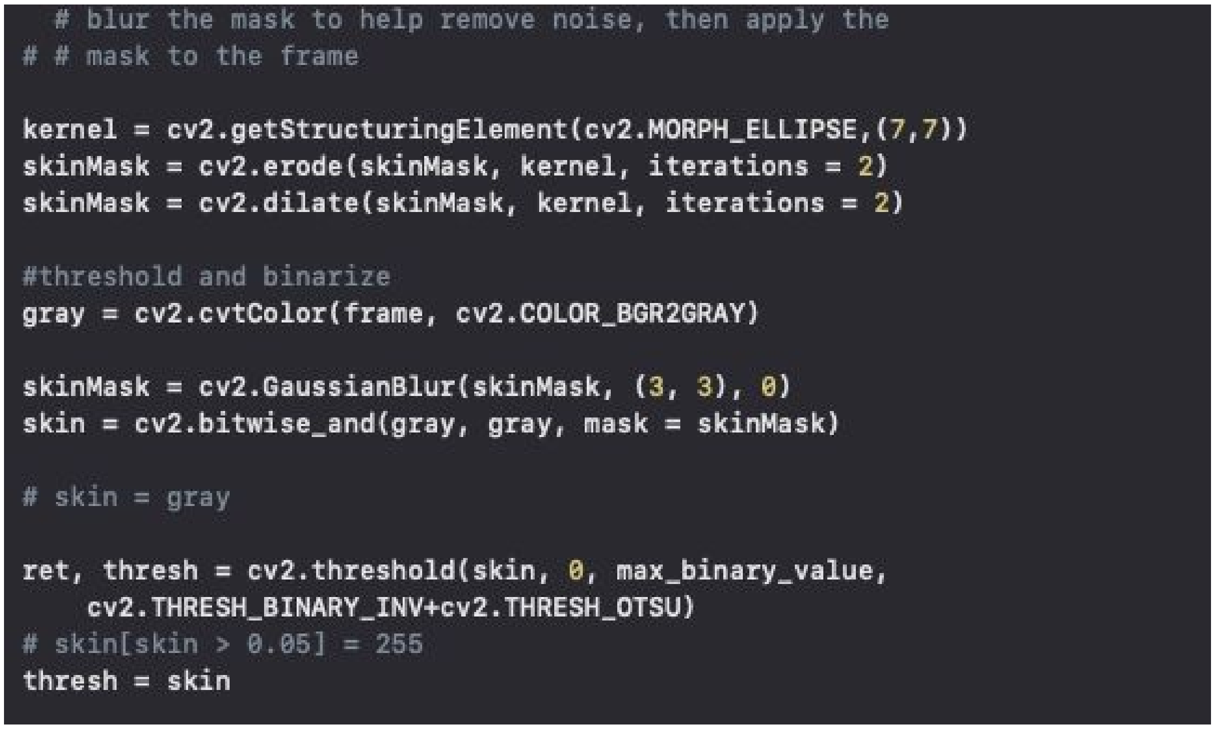
Part 1

In order to extract the hand from the feed, we first used the default upper and lower bound for HSV and YCrCb colors. If there's no color detected, we convert the skinmask HSV and YCrCb using cv2.cvtColor and cv2.inRange.



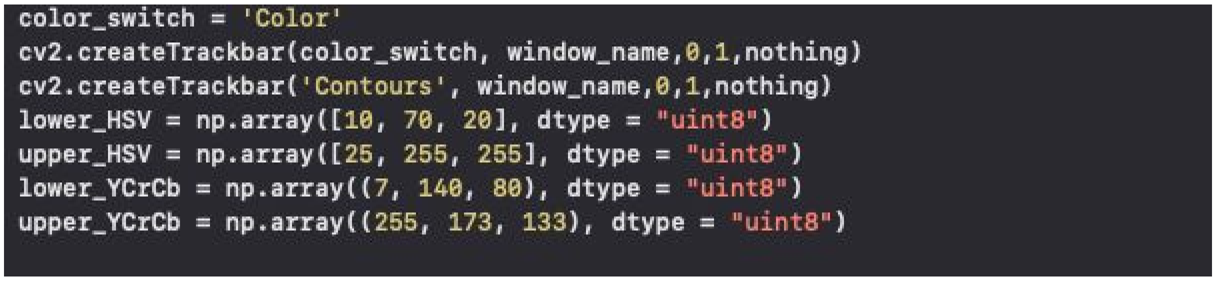
```
#convert to grayscale
if isColor == False:

    # HSV Method
    convertedHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    skinMaskHSV = cv2.inRange(convertedHSV, lower_HSV, upper_HSV)


    # YCrCb
    convertedYCrCb = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
    skinMaskYCrCb = cv2.inRange(convertedYCrCb, lower_YCrCb, upper_YCrCb)

    skinMask = cv2.add(skinMaskHSV,skinMaskYCrCb)
```

To tune the image, we applied erosion and dilation to the skinMask. We also applied Gaussian blur on the skin mask. To make the image gray, we converted the frame color to gray and then used cv2.bitwise_and with the gray image. Lastly, we used cv2.threshold to mark the background as 0 and skin color as 1.



```
  # blur the mask to help remove noise, then apply the
# # mask to the frame

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7))
skinMask = cv2.erode(skinMask, kernel, iterations = 2)
skinMask = cv2.dilate(skinMask, kernel, iterations = 2)

#threshold and binarize
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

skinMask = cv2.GaussianBlur(skinMask, (3, 3), 0)
skin = cv2.bitwise_and(gray, gray, mask = skinMask)

# skin = gray

ret, thresh = cv2.threshold(skin, 0, max_binary_value,
    cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# skin[skin > 0.05] = 255
thresh = skin
```

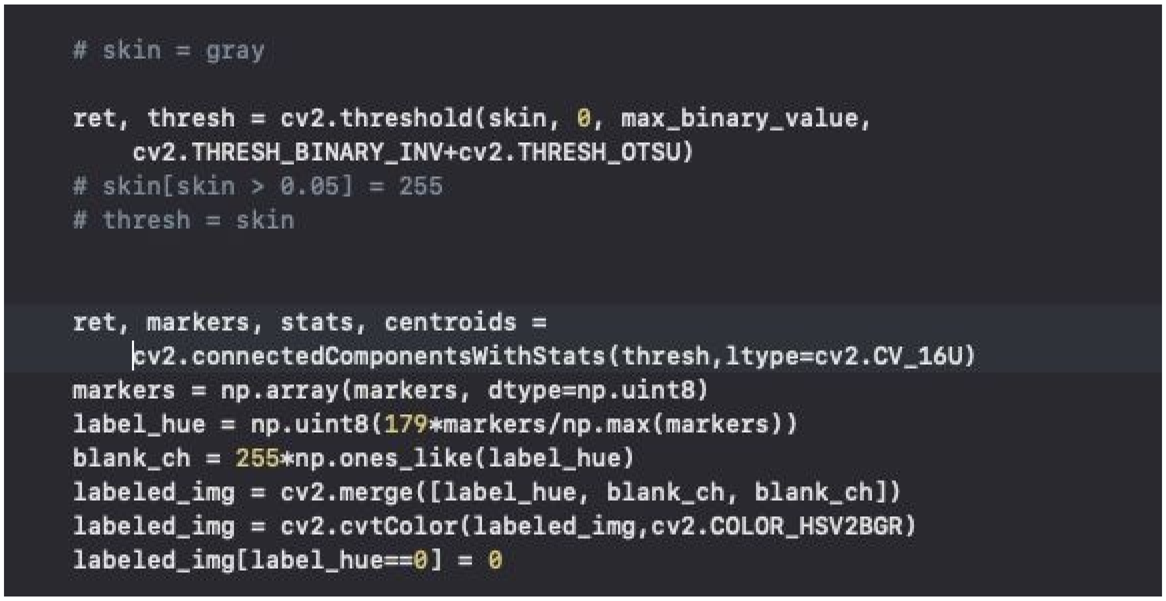
In order to tune the skin color, we spent a lot of time editing the lower bound. However, because we were doing the tuning both in the morning and evening, the color was unstable when the lighting changed a little bit, making the frame somehow noisy.



```
color_switch = 'Color'
cv2.createTrackbar(color_switch, window_name,0,1,nothing)
cv2.createTrackbar('Contours', window_name,0,1,nothing)
lower_HSV = np.array([10, 70, 20], dtype = "uint8")
upper_HSV = np.array([25, 255, 255], dtype = "uint8")
lower_YCrCb = np.array((7, 140, 80), dtype = "uint8")
upper_YCrCb = np.array((255, 173, 133), dtype = "uint8")
```

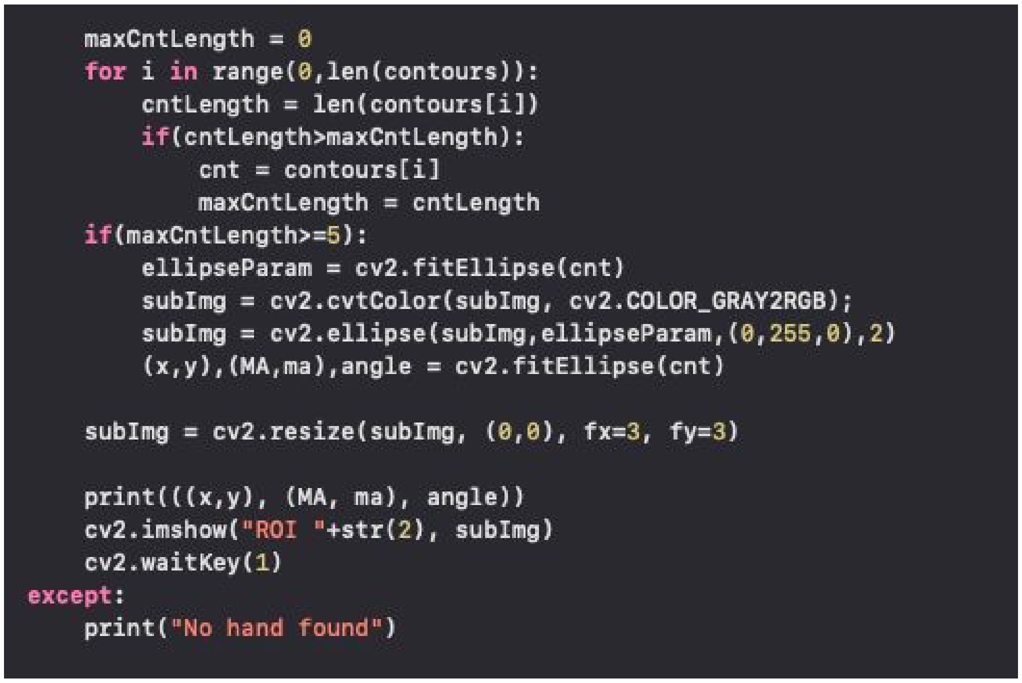
Part 2

In part 2, in order to do the connected component analysis, we first used Otsu's binarization to apply threshold and convert the image to black and white. Following the instruction, we first tried the simple approach using labeled images to mark the background color as 0. Then we sorted areas to find the region of interest and marked it as a sub image.

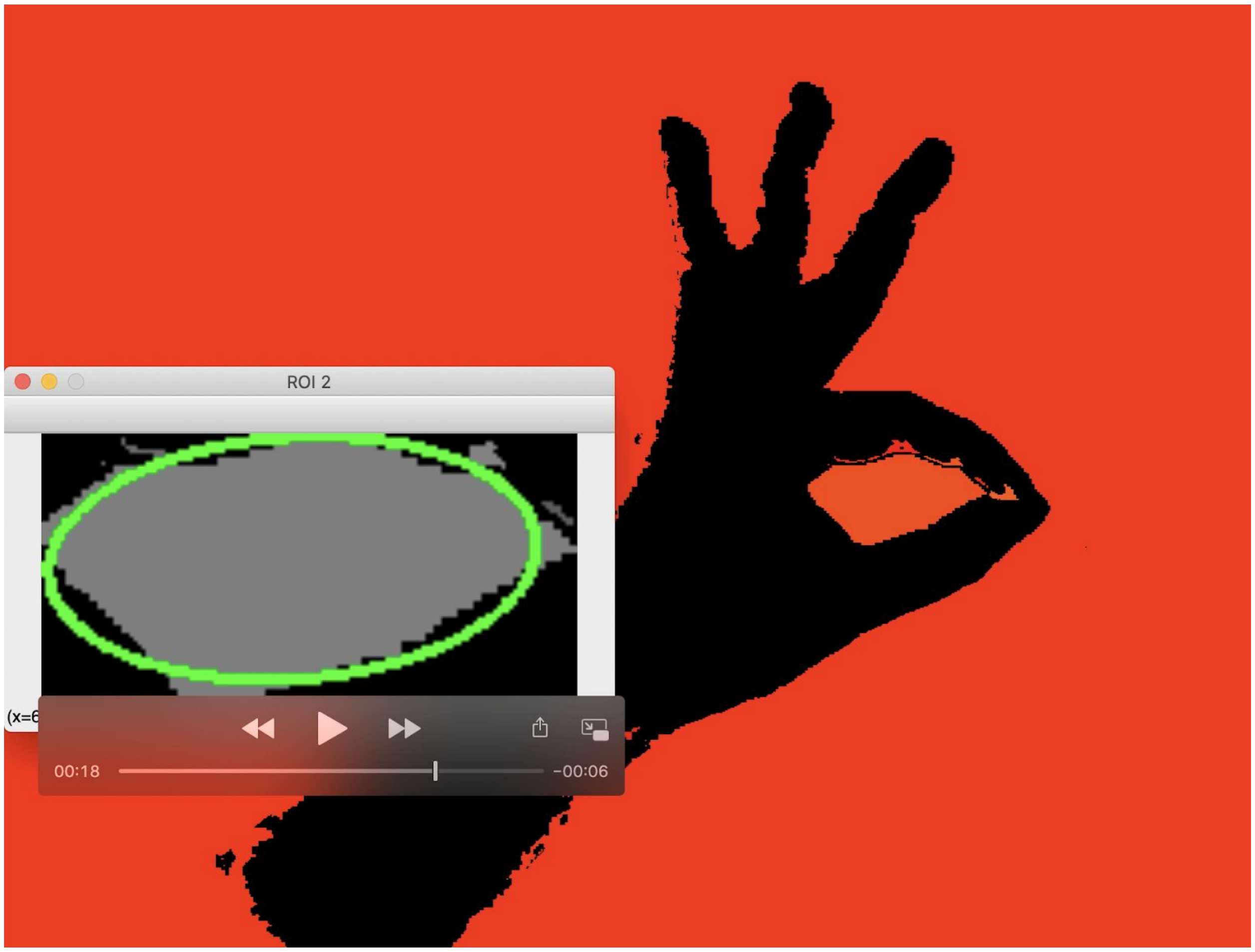


```
    # skin = gray

    ret, thresh = cv2.threshold(skin, 0, max_binary_value,
        cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    # skin[skin > 0.05] = 255
    # thresh = skin


    ret, markers, stats, centroids =
        cv2.connectedComponentsWithStats(thresh,ltype=cv2.CV_16U)
    markers = np.array(markers, dtype=np.uint8)
    label_hue = np.uint8(179*markers/np.max(markers))
    blank_ch = 255*np.ones_like(label_hue)
    labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
    labeled_img = cv2.cvtColor(labeled_img,cv2.COLOR_HSV2BGR)
    labeled_img[label_hue==0] = 0
```

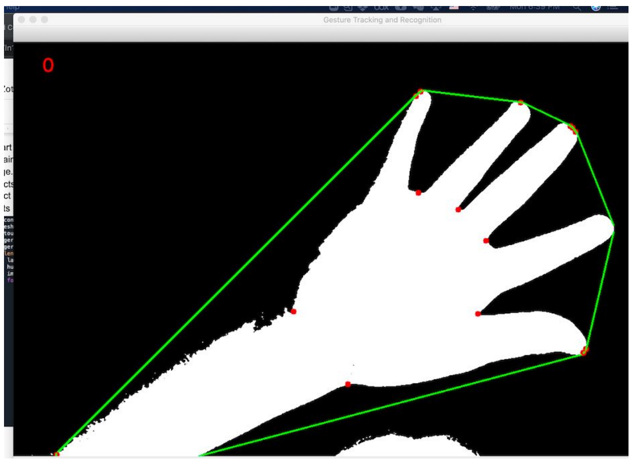
As we encountered the bug because when we moved the hand away, the contour[0] lost track of the hand but caught some noisy regions. Therefore, we used the if statement to check whether the image has at least 5 points. If so, we would print the (x ,y) (MA, ma) and the angle to fit the ellipse formed by the enclosed finger. However the picture looks a little fuzzy and we tried many methods but still could not figure out why.

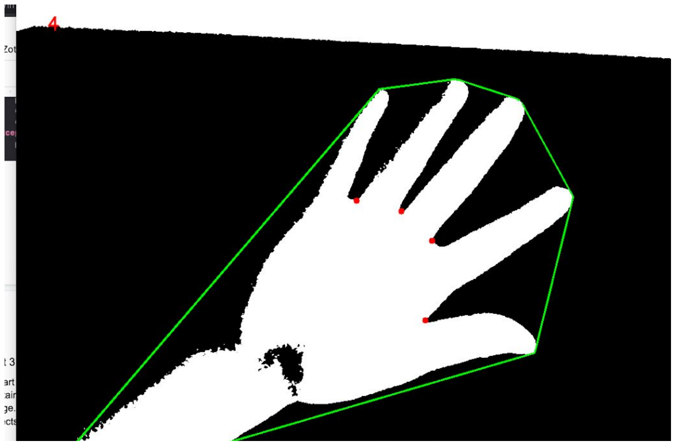


```
    maxCntLength = 0
    for i in range(0,len(contours)):
        cntLength = len(contours[i])
        if(cntLength>maxCntLength):
            cnt = contours[i]
            maxCntLength = cntLength
    if(maxCntLength>=5):
        ellipseParam = cv2.fitEllipse(cnt)
        subImg = cv2.cvtColor(subImg, cv2.COLOR_GRAY2RGB);
        subImg = cv2.ellipse(subImg,ellipseParam,(0,255,0),2)
        (x,y),(MA,ma),angle = cv2.fitEllipse(cnt)

    subImg = cv2.resize(subImg, (0,0), fx=3, fy=3)

    print(((x,y), (MA, ma), angle))
    cv2.imshow("ROI "+str(2), subImg)
    cv2.waitKey(1)
except:
    print("No hand found")
```

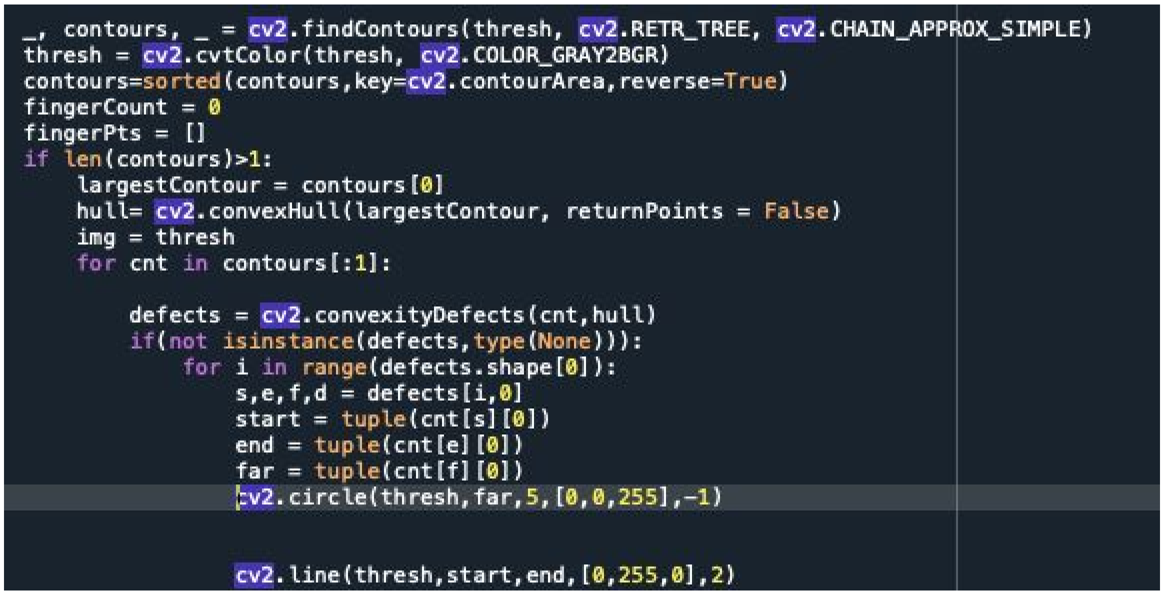

ROI 2
(x=6
00:18
-00:06

Part 3



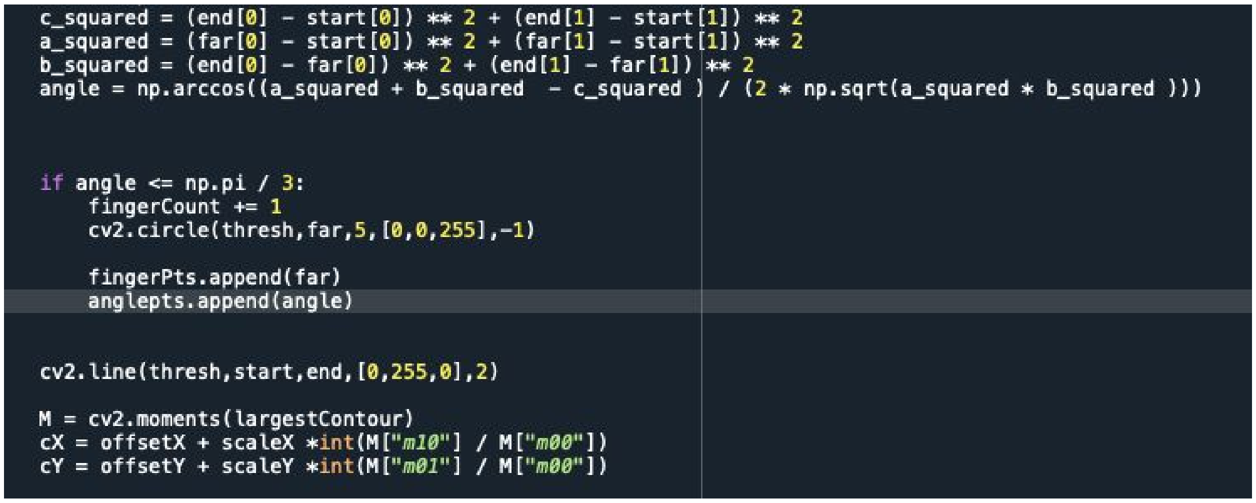


In part 3, we used gray skinmask from part 2 as the base layer. A well threshold image only contains the hand portions required to detect the hand contours, which is the largest one in the image. Then we need to compute a convex hull for the detected hand contours, and convexity defects. The left image above is the tested image that describes the hand contours in green and convexity defect points in red. The is the result before the heuristics filter.



```
_, contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
thresh = cv2.cvtColor(thresh, cv2.COLOR_GRAY2BGR)
contours=sorted(contours,key=cv2.contourArea,reverse=True)
fingerCount = 0
fingerPts = []
if len(contours)>1:
    largestContour = contours[0]
    hull= cv2.convexHull(largestContour, returnPoints = False)
    img = thresh
    for cnt in contours[:1]:

        defects = cv2.convexityDefects(cnt,hull)
        if(not isinstance(defects,type(None))):
            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])
                cv2.circle(thresh,far,5,[0,0,255],-1)


                cv2.line(thresh,start,end,[0,255,0],2)
```

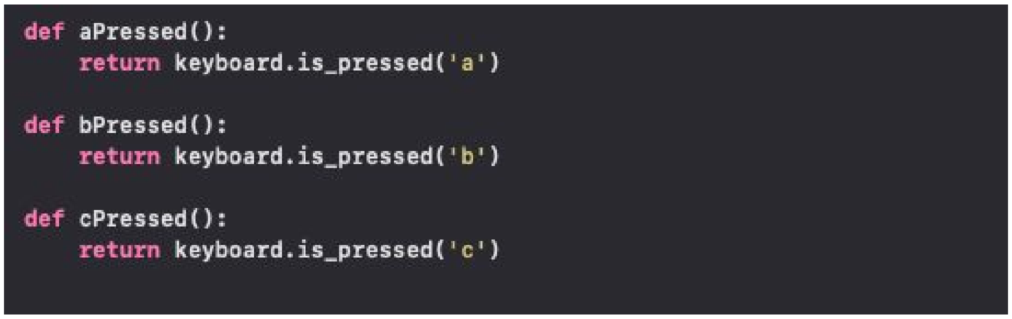
The above image in the top right is the result after the heuristics filter. It calculated the angle between two contour lines and only determined the cross points as a defect point when the angle theta is less than 60^0. The number in the top left image noted the number of the convexity defect points in the image.



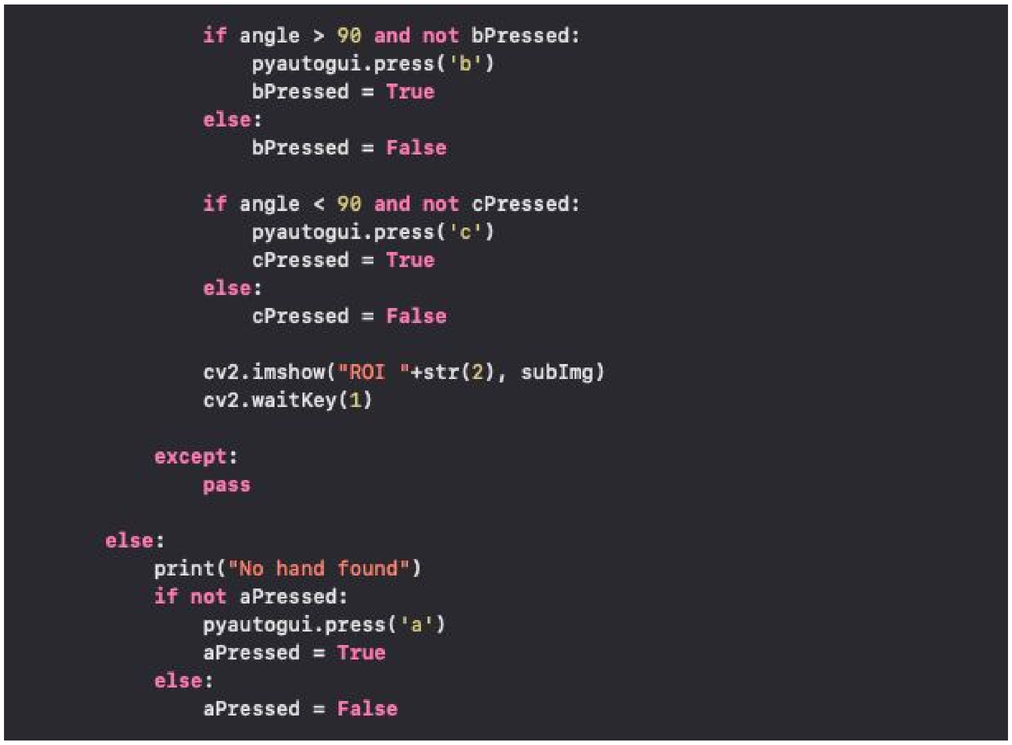
```
c_squared = (end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2
a_squared = (far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2
b_squared = (end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2
angle = np.arccos((a_squared + b_squared  - c_squared ) / (2 * np.sqrt(a_squared * b_squared )))



if angle <= np.pi / 3:
    fingerCount += 1
    cv2.circle(thresh,far,5,[0,0,255],-1)

    fingerPts.append(far)
    anglepts.append(angle)


cv2.line(thresh,start,end,[0,255,0],2)

M = cv2.moments(largestContour)
cX = offsetX + scaleX *int(M["m10"] / M["m00"])
cY = offsetY + scaleY *int(M["m01"] / M["m00"])
```

Part 4

In part 4, we used both part 2's connected component analysis and part 3's contour analysis to design new gestures. For part 2, we defined some simple gestures. For example, when there's no hand detected, we would press a. When the angle of the enclosed circle is more than 90, we would press b. When the angle of the enclosed circle is less than 90, we would press c.

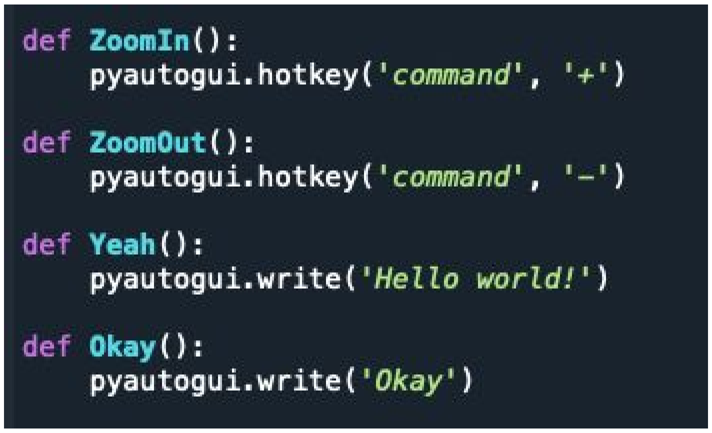


```
def aPressed():
    return keyboard.is_pressed('a')

def bPressed():
    return keyboard.is_pressed('b')

def cPressed():
    return keyboard.is_pressed('c')
```

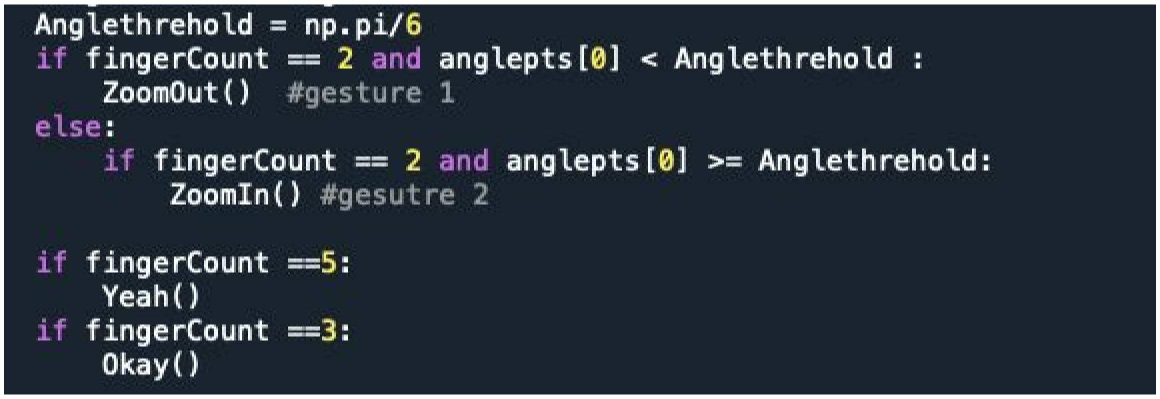


```
                if angle > 90 and not bPressed:
                    pyautogui.press('b')
                    bPressed = True
                else:
                    bPressed = False

                if angle < 90 and not cPressed:
                    pyautogui.press('c')
                    cPressed = True
                else:
                    cPressed = False

                cv2.imshow("ROI "+str(2), subImg)
                cv2.waitKey(1)

            except:
                pass

        else:
            print("No hand found")
            if not aPressed:
                pyautogui.press('a')
                aPressed = True
            else:
                aPressed = False
```

For part 3’s gesture , we defined some gestures by detecting the number of fingers and angles. If two fingers are detected and the angles are smaller than 30 degree, we would zoom out. If the detected angle was larger than or equal to 30 degree, we would zoom in. If five fingers or three fingers are detected, we would type “Hello World!” or “Okay” respectively.



```python
def ZoomIn():
    pyautogui.hotkey('command', '+')

def ZoomOut():
    pyautogui.hotkey('command', '-')

def Yeah():
    pyautogui.write('Hello world!')

def Okay():
    pyautogui.write('Okay')
```



```python
Anglethrehold = np.pi/6
if fingerCount == 2 and anglepts[0] < Anglethrehold :
    ZoomOut()  #gesture 1
else:
    if fingerCount == 2 and anglepts[0] >= Anglethrehold:
        ZoomIn() #gesutre 2

if fingerCount ==5:
    Yeah()
if fingerCount ==3:
    Okay()
```