COMS W4753

Assignment 1

Daisy Ye

UNI: yy3131

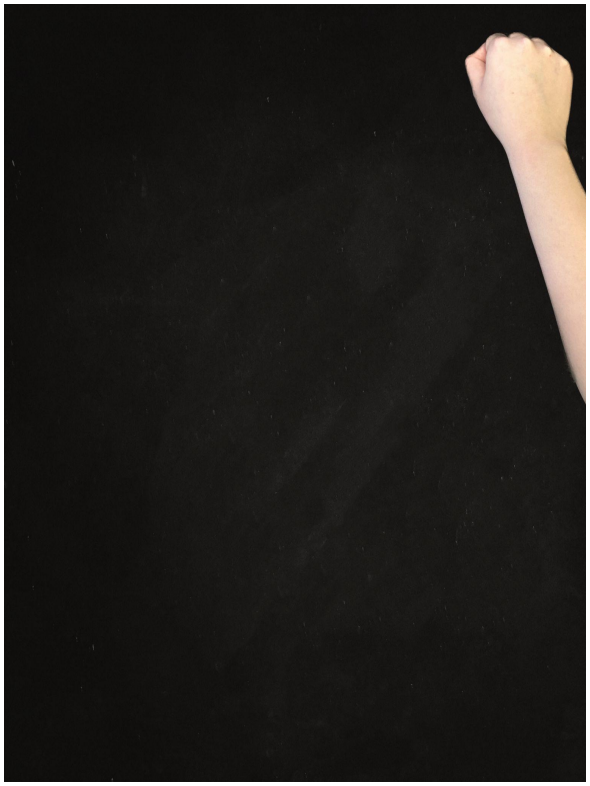
Step 1: Domain Engineering

To set up the photo environment, I used my iPhone 14 camera, with a big light on top directly shining at a black tablecloth background. The light is directly above my hand, such that the shadow of the hand is minimized.

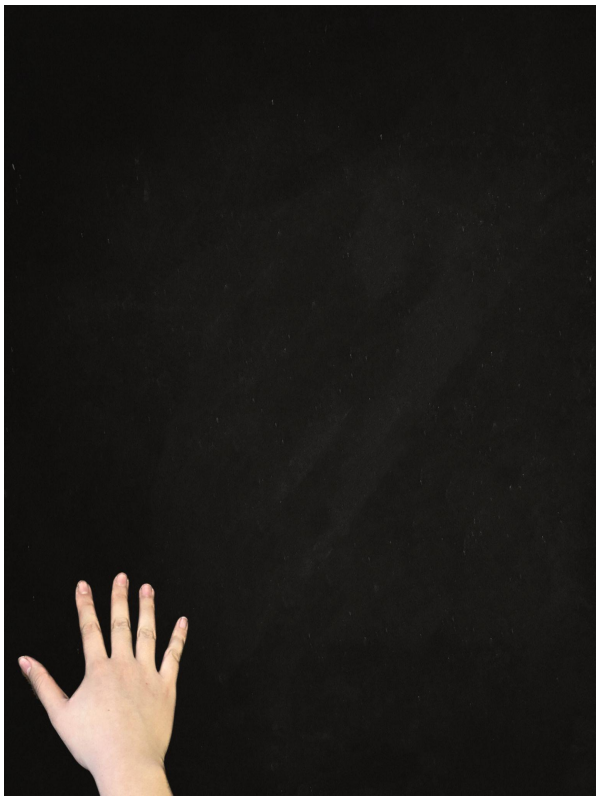
The analysis of the photos was run on MacOS Ventura 13.2 of my MacBook Pro 15-inch, 2019. I used Python 3.8 and the packages include OpenCV, matplotlib, imutils, and math. OpenCV is the main computer vision analysis tool; matplotlib is used to plot the images; imutils is used to capture the largest contour in the image; and math is used to calculate the Euclidean distance between points.

My photo library has a total of 37 images and they are all stored in .png format. Below are some examples:

Intended to capture a closed fist in the center



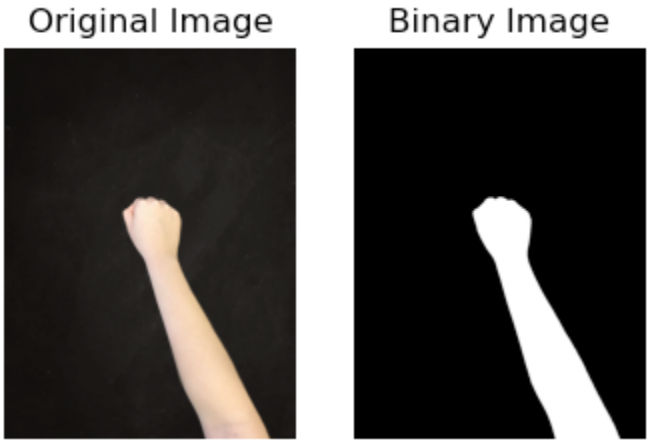
Intended to capture a close fist in the upper right corner



Intended to capture a flat palm with fingers outwardly splayed in the bottom left

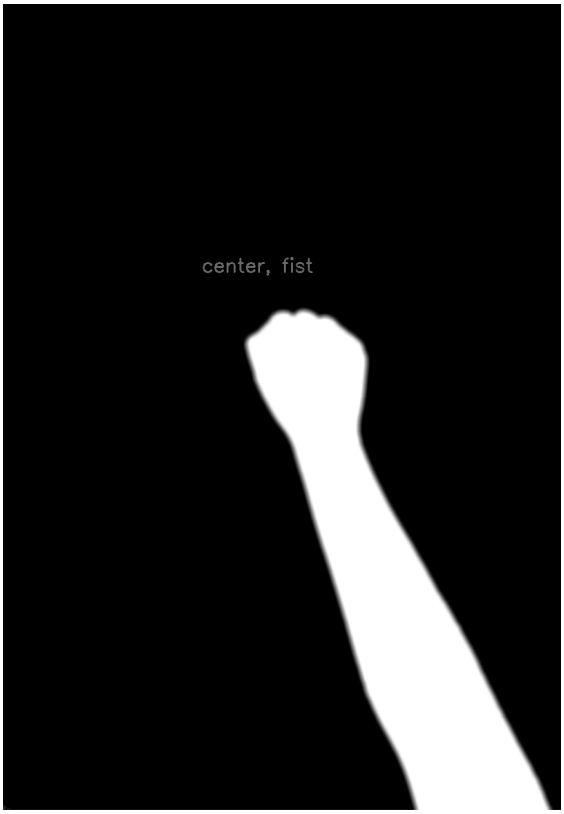
Step 2: Data Reduction Step

1. **Centered Fist 1**
    a. Original Image and Binary Image



    i.
    ii. Binary image is computed from a grayscale image of the original image and then adding a threshold of 70 to turn it to binary

    b. Label

i. 

ii. Algorithm Explanation:

First the image is converted into binary and blurred.

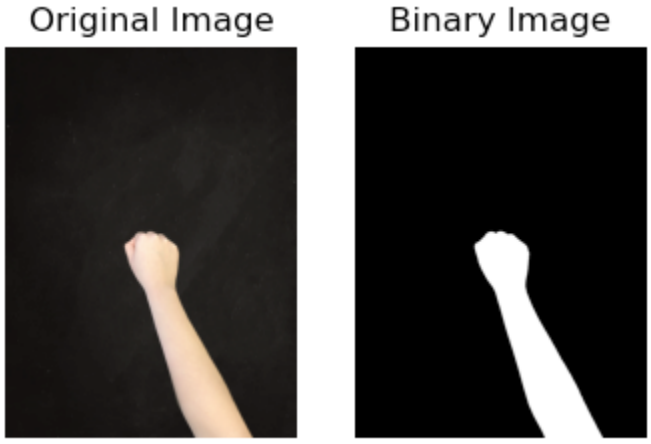
The contour of the hand is found by finding the largest contour among all the contours found on the binary image. There are a lot of smaller contours originally because the image background isn't ideal. But by finding the largest contour, the image is cleaned up as it assumes the largest object in the image is the hand+arm.

The position of the fist is found by dividing the height and width of the image by three. If the object is in the "upper middle", "center left", "center right", and "bottom middle" boxes, it is in an unknown location. If it is in "upper left", "upper right", "center", "bottom left", and "bottom right" boxes, it is in a recogized location. In order to locate the hand, I use the topmost point of the contour. I also experimented with using the center of mass of the contour, however, with the arm extended into the screen, it messes up with the center of mass as it moves outside of the hand.

Whether the hand is a fist or a splay is determined by the number of defects. I found an optimum threshold (40) such that irrelevant defects are ignored and only defects due to the hand

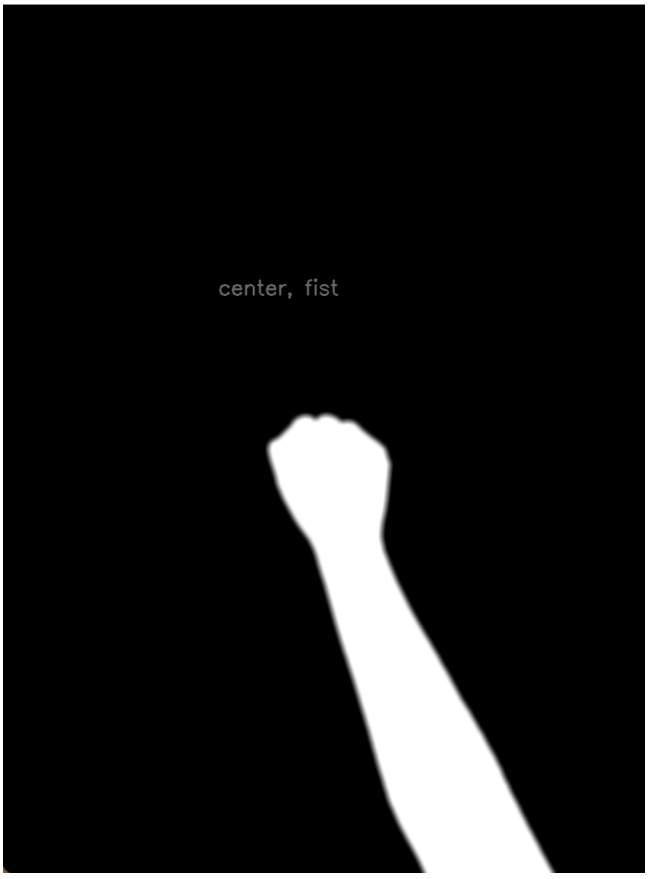
will be detected. If there are four detects, then the hand is a splay. If no defect, then the hand is a fist.

2. **Centered Fist 2**
    a. Original Image and Binary Image



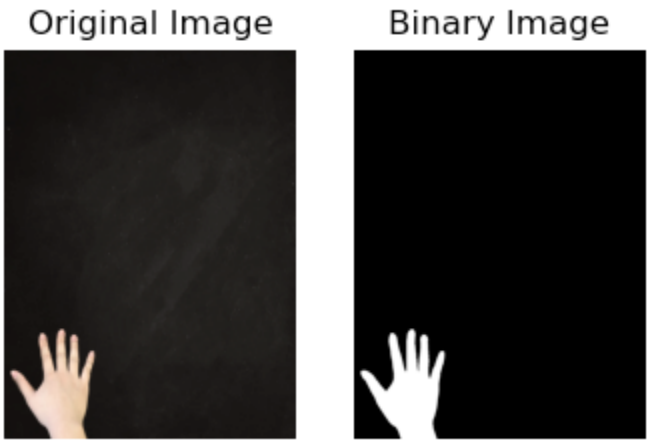
    i.

    b. Label



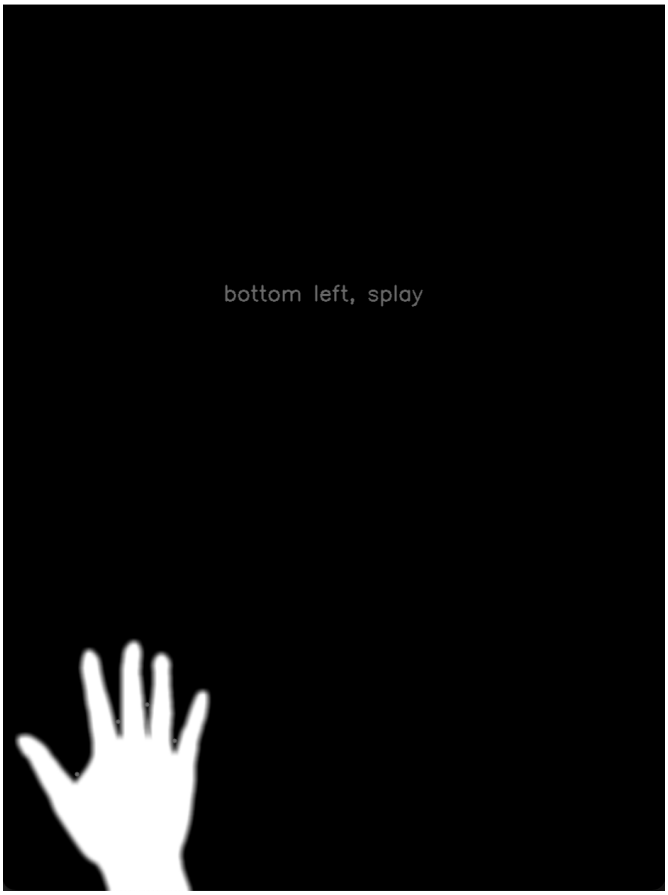
    i.

3. **Bottom left corner splay**
    a. Original Image and Binary Image

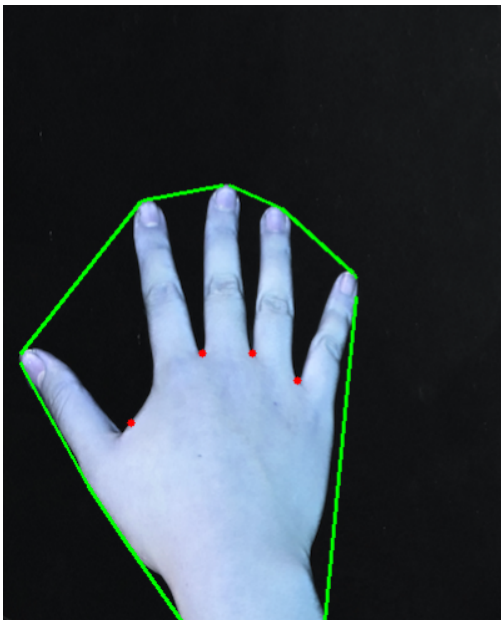


    i.

    b. Label

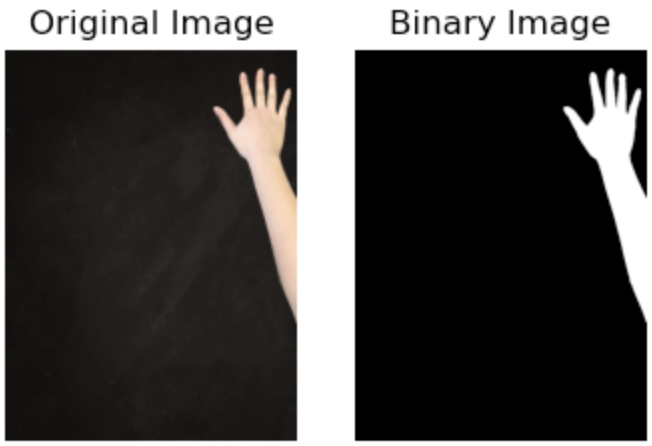
    i. 

c. On the binary image, it is impossible to see the defect dots that are used to tell a splay apart from a fist. In the image below, the red dots show how the program counts the number of gaps between fingers in order to determine the “what”.



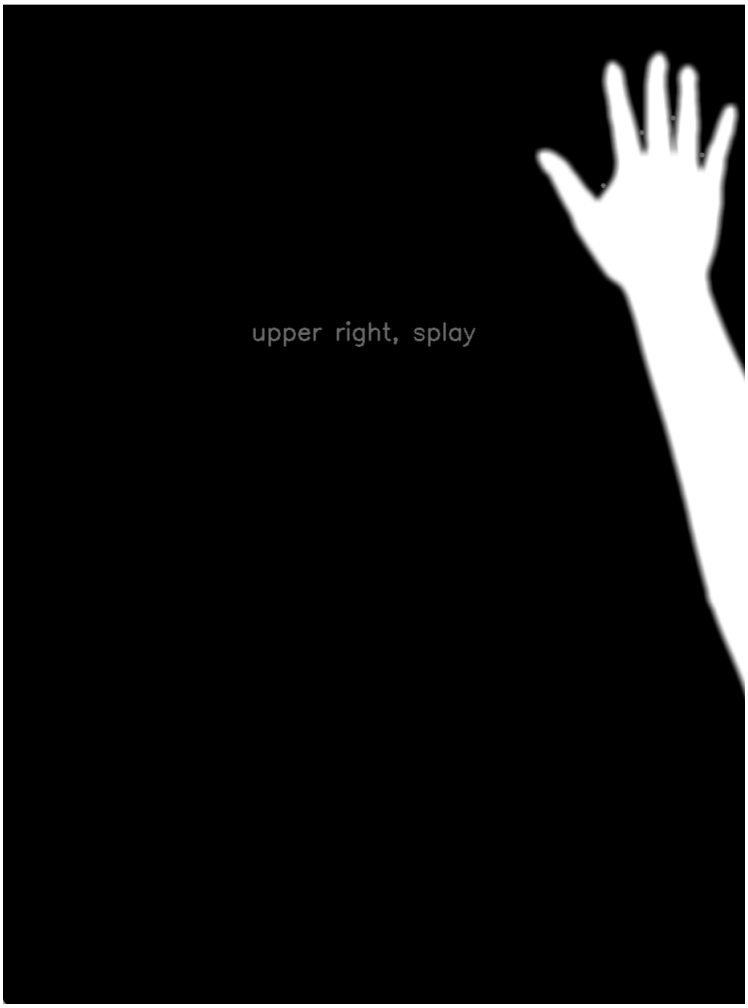
4. Upper right corner splay

a. Original Image and Binary Image



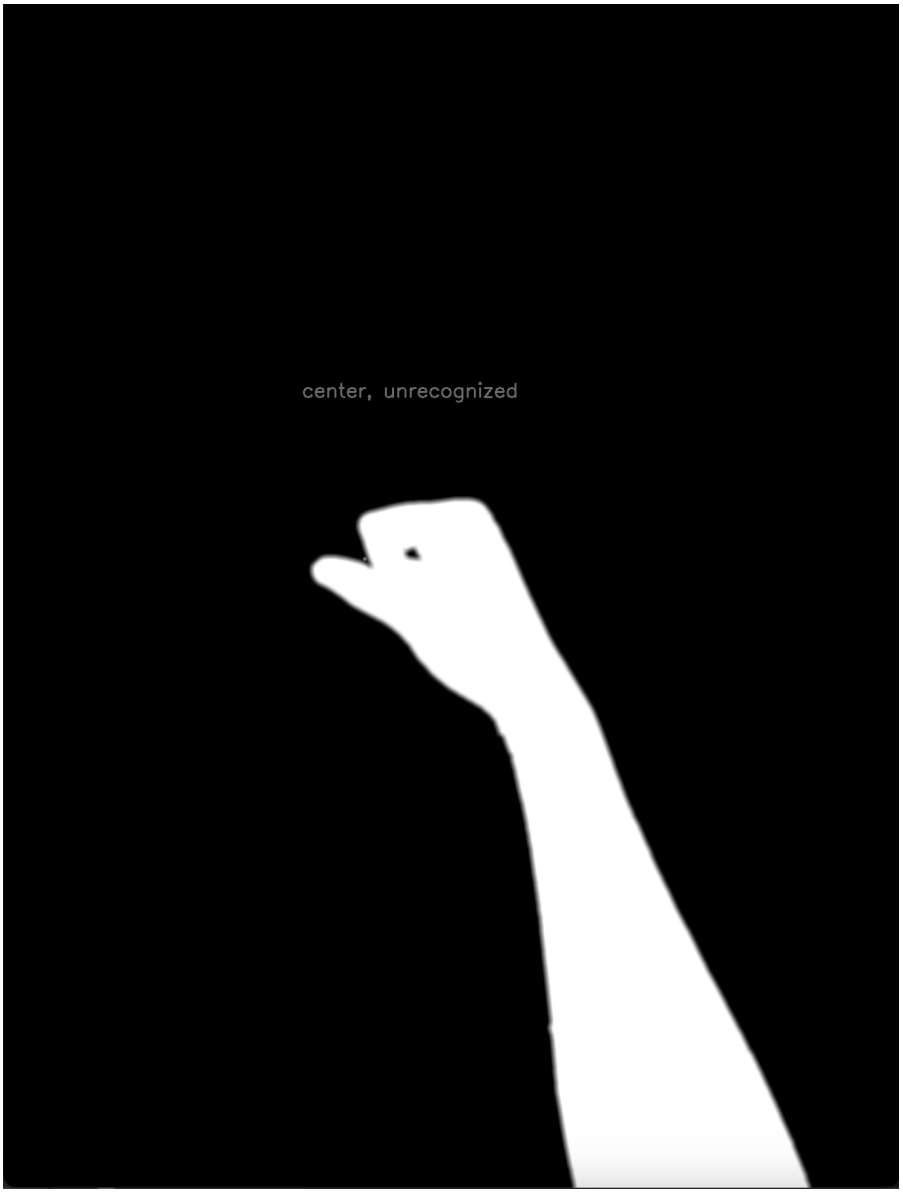
i.

b. Label

i. 

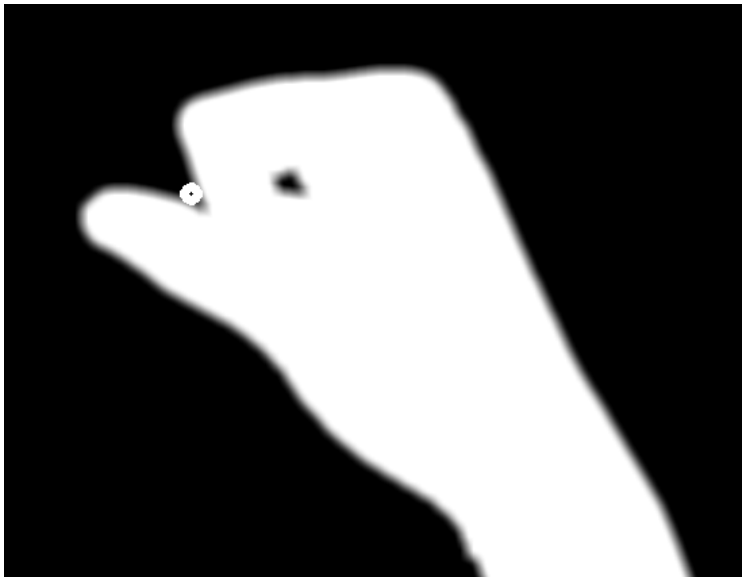
Step 3: Edge Cases and Extension

1. **Centered Fist False Negative**

a. 

Even though the program successfully recognized that the fist was in the center, it failed to recognize it was fist because it found a defect dot, which also refers to the gap between the thumb and other fingers. This defect dot is between the thumb and the rest of the fist. The program currently only realizes a fist when there is no dot, and a splay when it has four dots. In this case, it is because the fist looks more like a punch and it detects a gap between the fingers.

Here is a picture that shows the defect dot (in white) found on the hand:

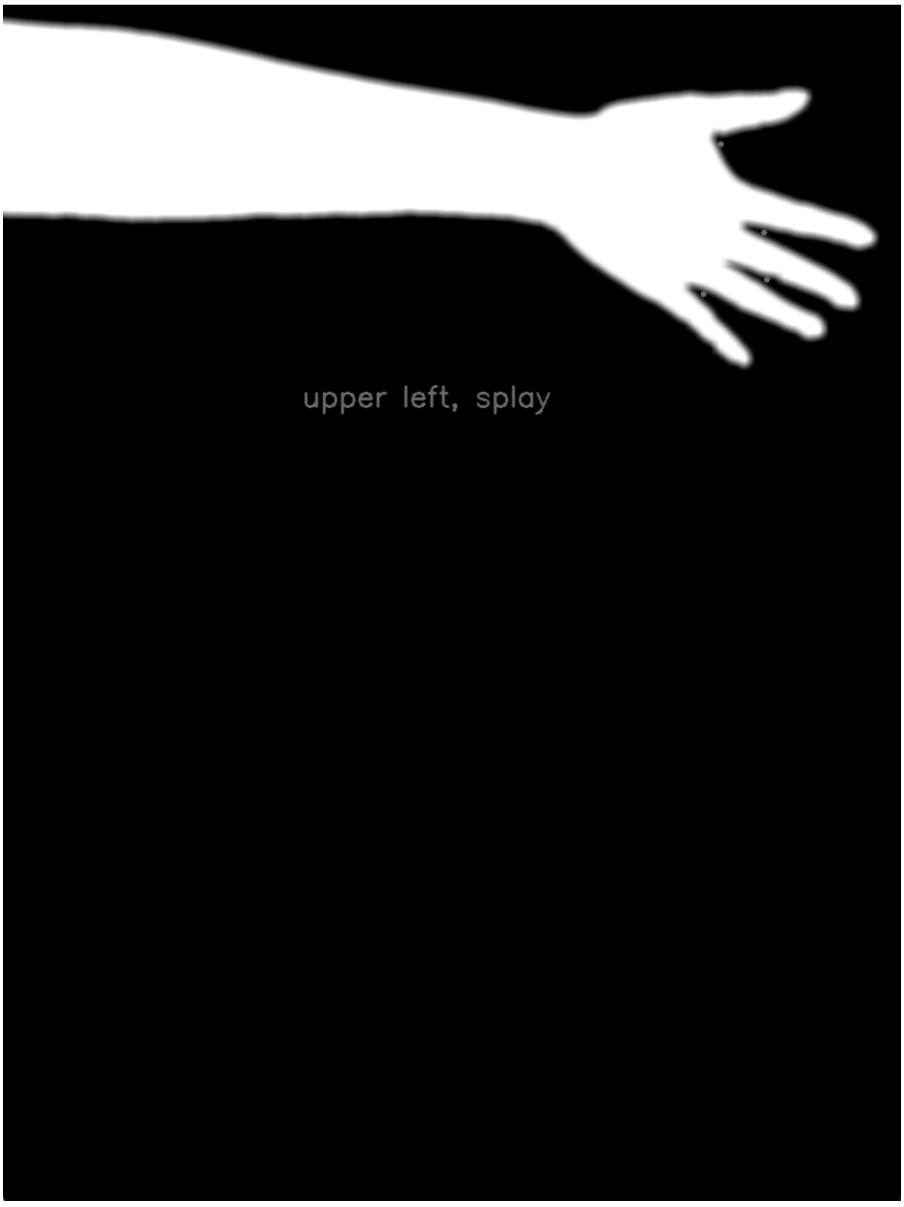


2. **Centered Fist False Positive**

a. 

Even though the program successfully recognized that the hand was in the center, it recognized it as fist because it found no defect dots, which also means it found no gaps between the fingers. The program currently only realizes a fist when there is no dot, and a splay when it has four dots. If the program puts a constrain on how long the fingers can be, it will solve this issue as the hand will be categorized as “unrecognized”.

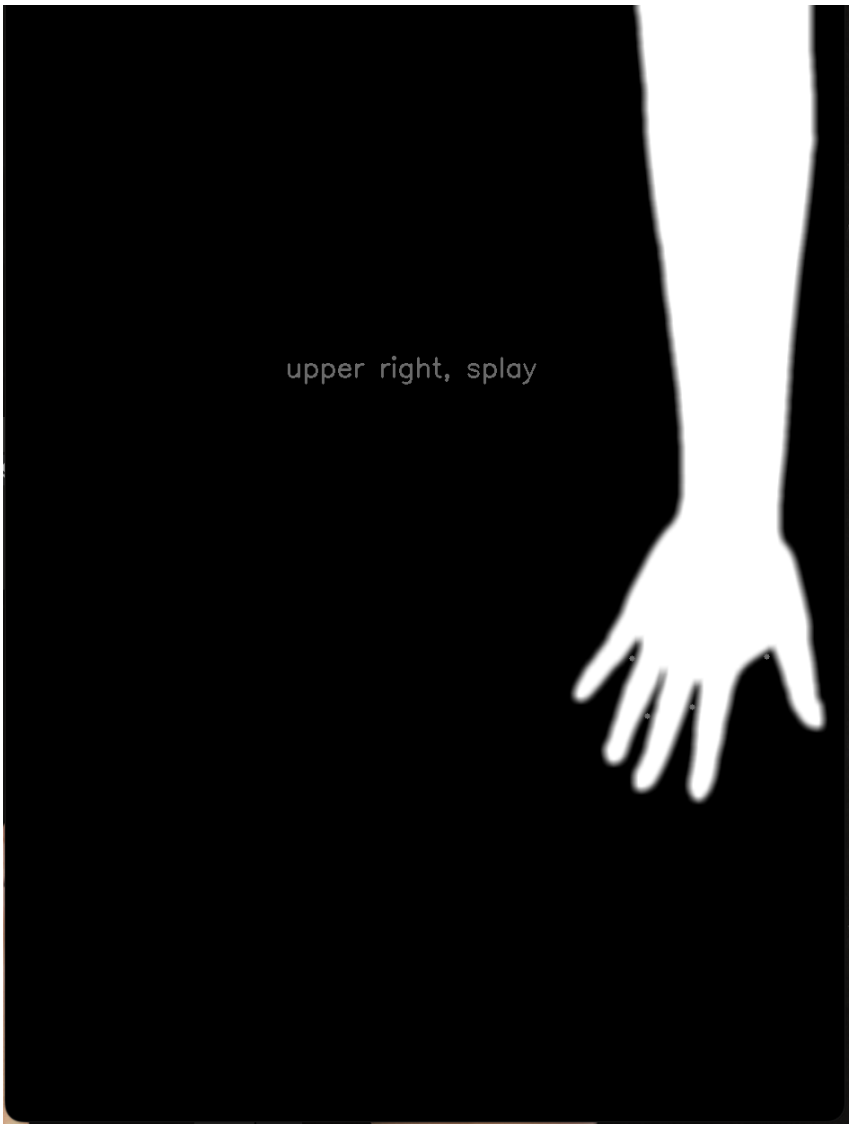
3. **Upper Right Splay False Negative**

a. 

Even though the program successfully recognized that the hand is a splay, it recognized it as the upper left corner even though it should be the upper right corner. This is because the program uses the top-most point of the hand+arm as the coordinate of the entire hand. In this case,

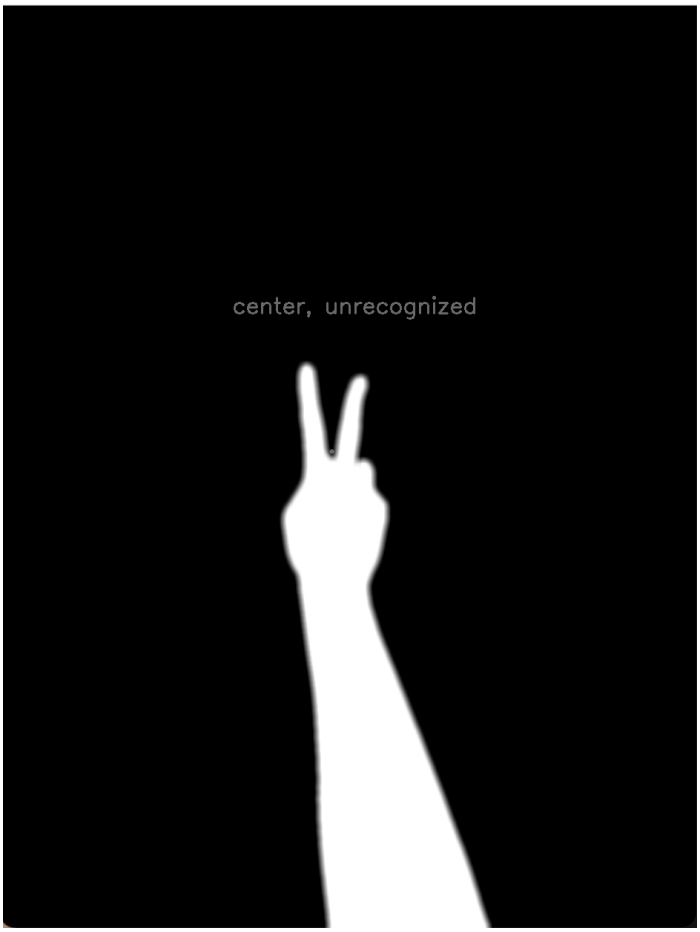
because the arm reaches from the left and is higher than the hand, the program is locating the top left corner of the arm instead of the hand.

4. **Upper Right Splay False Positive**

a. 

Even though the program successfully recognized that the hand is a splay, it recognized it as upper right corner even though its location should be unknown (because it is in middle right region, which is not defined). This error is caused by the program using the top-most point as the coordinate of the entire hand. In this case, because the arm reaches from the top, it is recognizing the location using the arm instead of the hand.

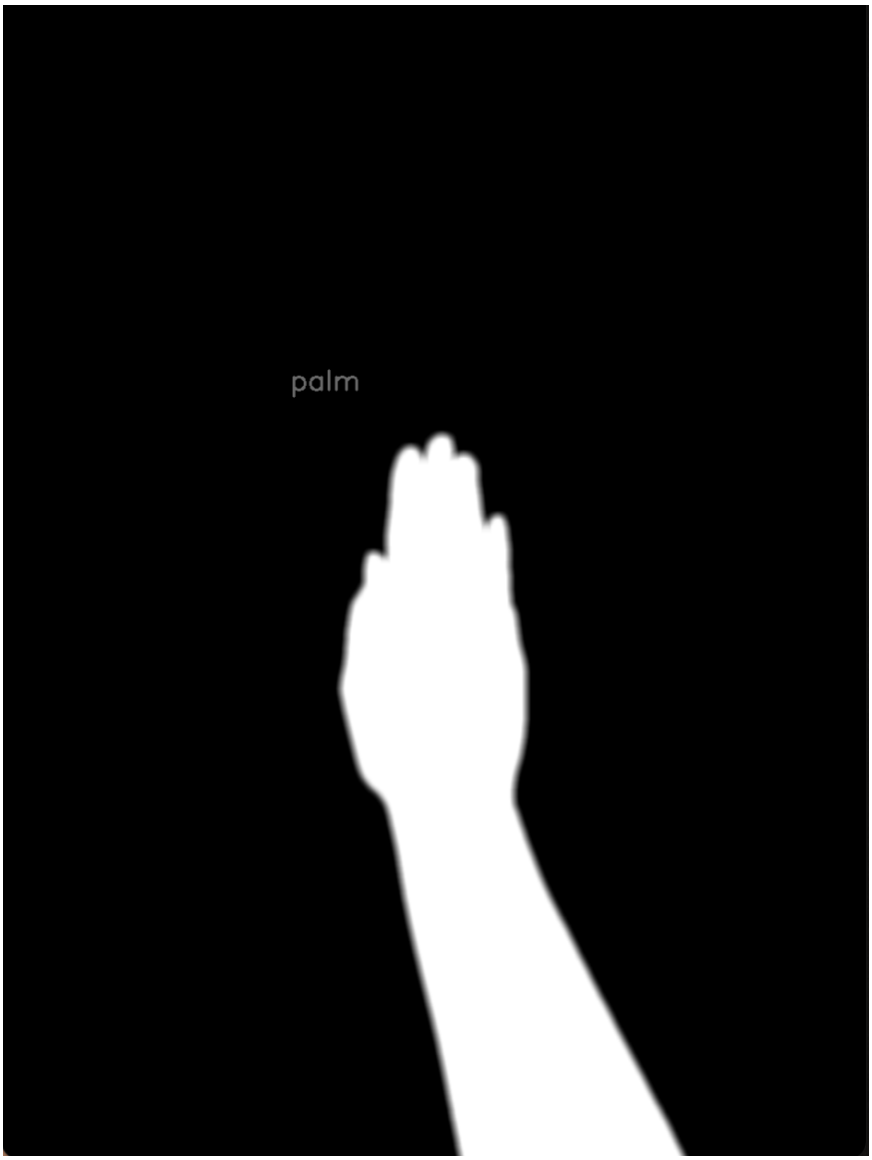
5. **Unknown 1**

a. 

6. Unknown 2

a. 

7. Palm True Positive 1

a. 

8. Palm True Positive 2

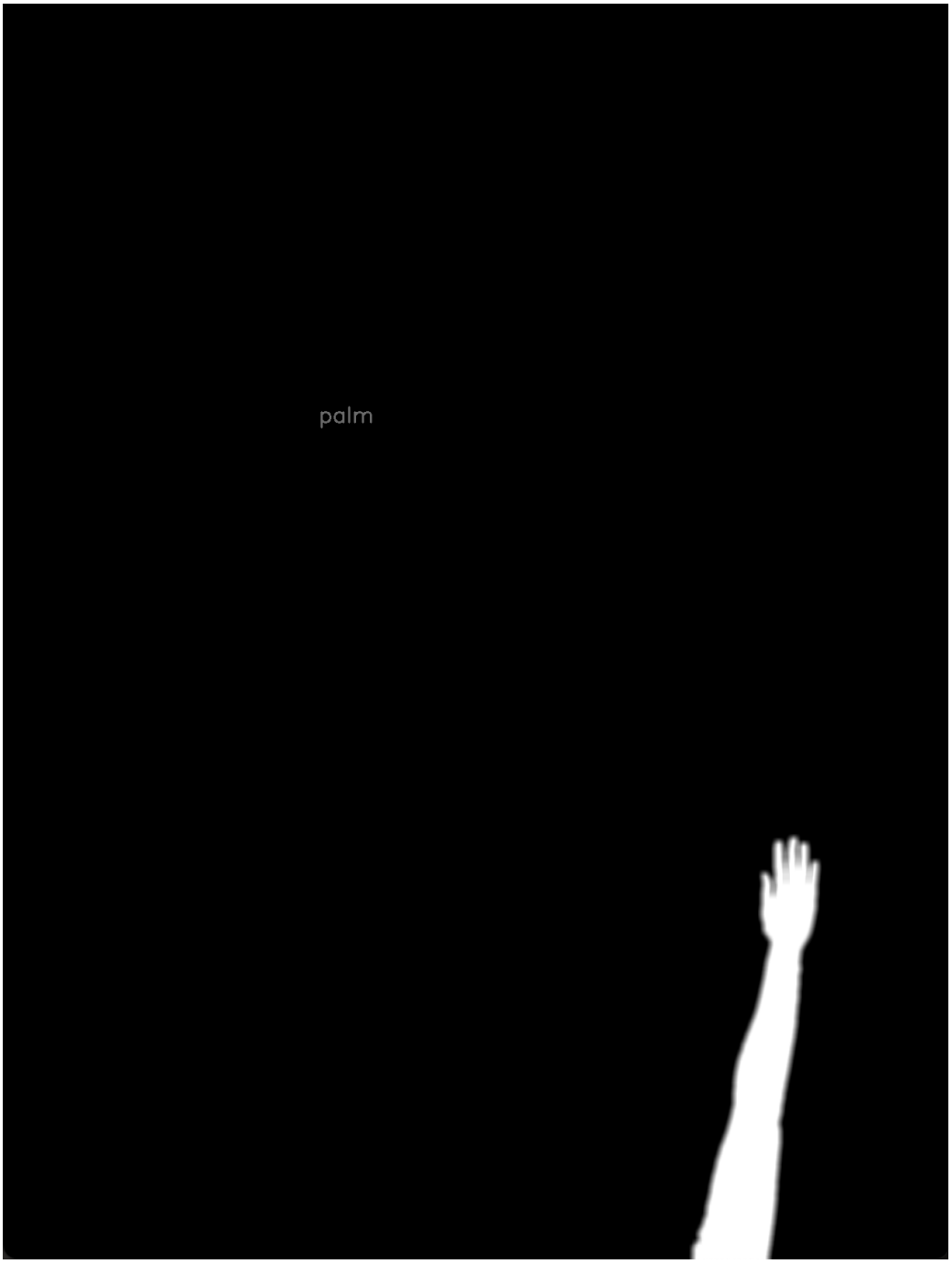
a. 

9. Palm False Negative

a. 

This is recognized as a fist because when differentiating between a fist and a palm, the program computes the distance between the left-most point and the top-most point of the hand and see if it surpasses the threshold (200). The logic behind it is that if a hand is extended, the top-most point and the left-most point would be larger. However, this is assuming that the arm is only extending from the bottom of the screen up instead of downwards. In this case, because the left-most point is the top-most point, the program fails to recognize it as a palm as the threshod is zero.

10. Palm False Positive

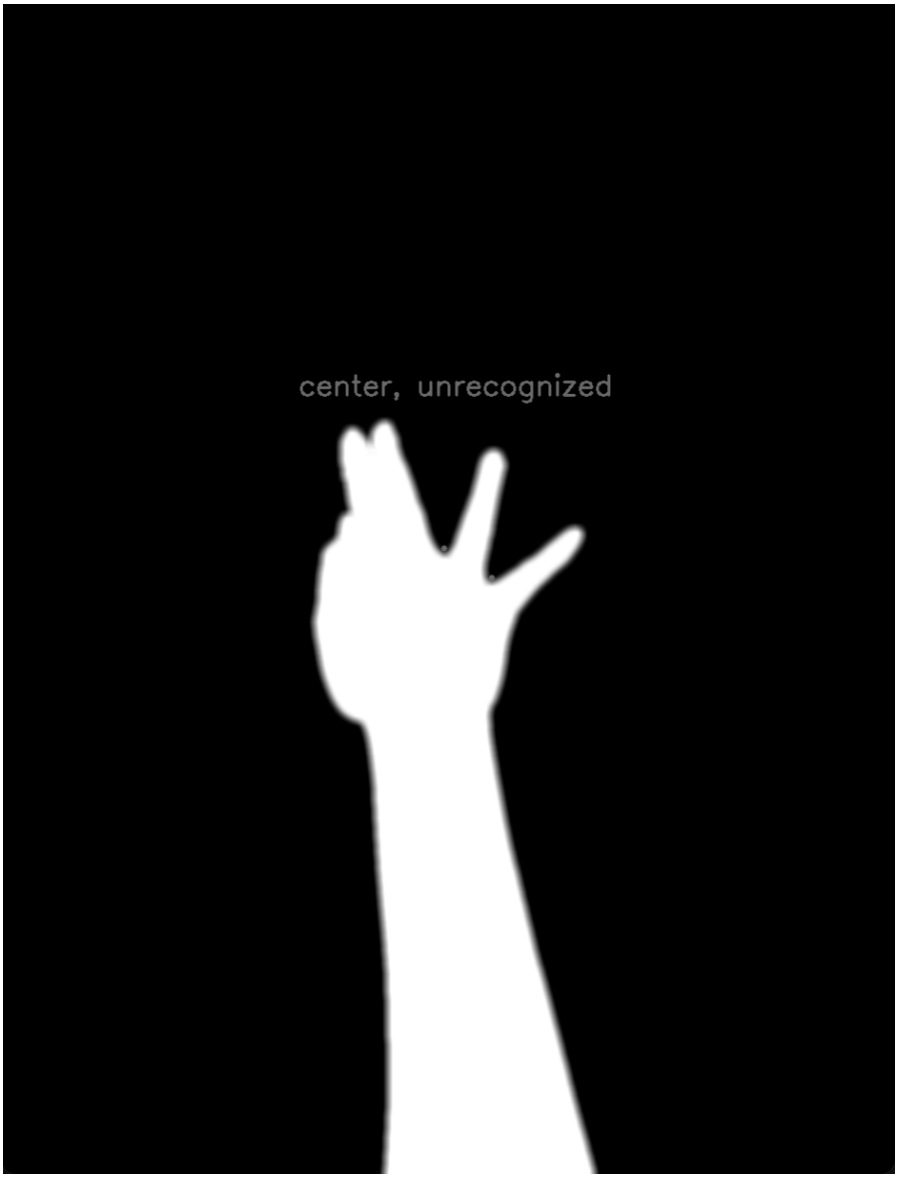
a. 

This is seen as a palm even though it is a splay because of how far away the arm is from the camera. The human eyes can still see that there are gaps between the fingers. However, because the image is blurred and the hand is really small. It is hard for the program to tell if there are any defect spots on the hand. One way to solve this is to detect the size of the hand and adjust the blurring based on distance.

11. Unknown 1

a. 

12. Unknown 2

a. 

Step 4: Evaluation Step

1. **How I determined the sequences**
    a. **Easy sequence**: < ‘fist, center’ , ‘splay, bottom left’, ‘splay, bottom right’>

This is an easy sequence because it involves only the bottom and center position. After the previous steps, I have realized that my program has a harder time determining the “where” than the “what” because the “what” involves more processing while the “where” depends heavily on where the arm is extending from as it looks at the top-most point of the contour. Therefore, the bottom positions and the center position are the easiest to control as the user most likely will show the least amount of arms. This means that when calculating the distance between the topmost point and the leftmost point, the leftmost point of the gesture is unlikely to be the arm because the arm is barely visible. With only two out of the three hand gestures in this sequence, I also limit the chance of when there is a gap in a palm that gets detected, causing the gesture to become unrecognized.

    b. **Difficult sequence:** < ‘fist, upper right’, ‘splay, upper left’, ‘palm’ >

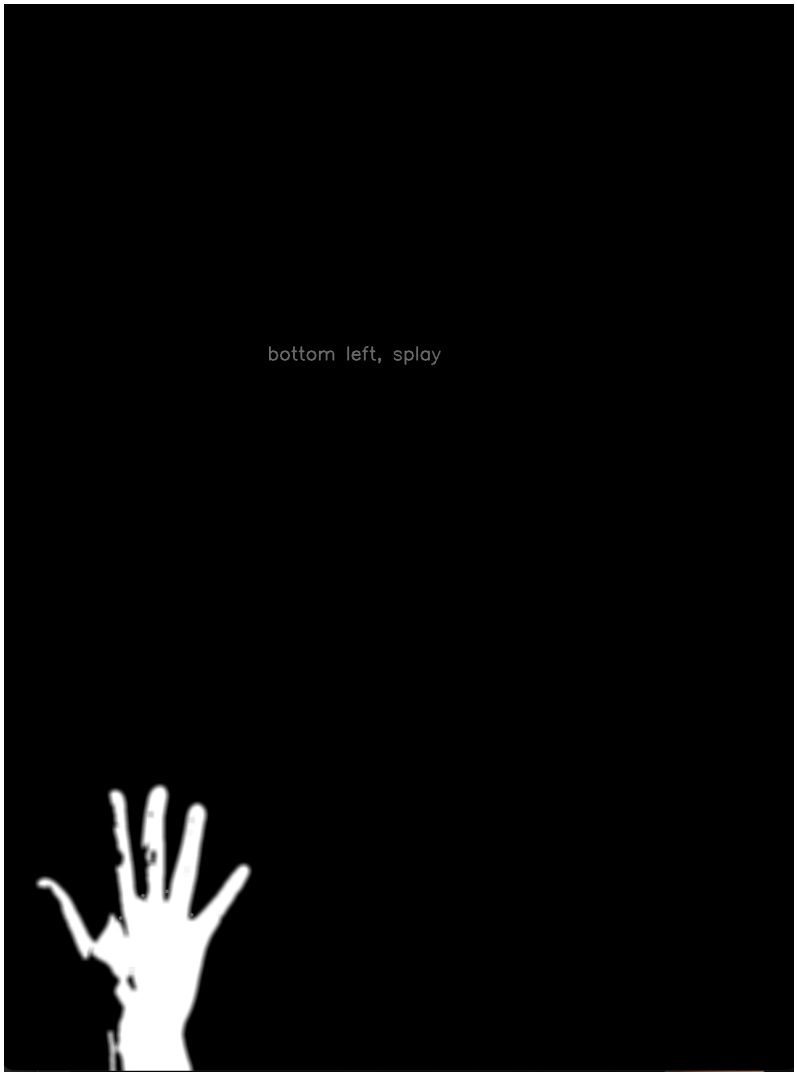
I chose this as a difficult sequence because my program has more difficulty recognize the “where” than the “what”. And this sequence involves more complicated positions in the upper region that involves more arms. When user is doing these cornered positions, it is more likely that the arm is extending from the sides of the screen instead of the bottom of the screen, which will cause the program to fail to recognize the location. Also, this sequence involves all of “fist”, “palm”, and “splay”. If the user is also extending the arm from the side of the screen, it is possible that a fist becomes a punch, and a splay becomes vertical to the camera and looks like a palm. Therefore, there are more ways this sequence can fail.

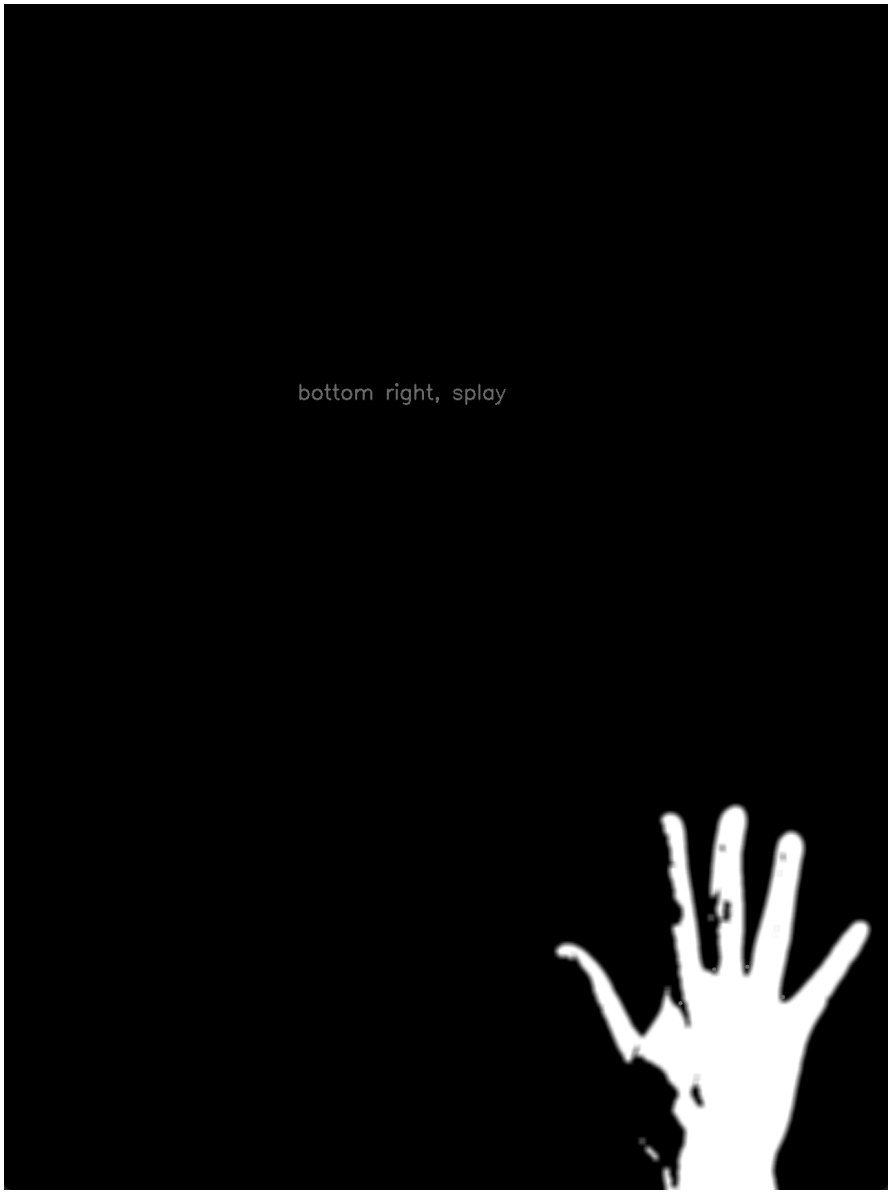
    c. **Good sequence**: < ‘fist, bottom left’, ‘splay, center’, ‘palm’ >

This is a good sequence because it involves all the hand gestures, an easy position, “bottom left”, and a harder position “center”. The difficulty here is determined by the amount of arm involved. “Fist” is easier to distinguish from “palm” than “splay” because it does not involve the extension of fingers. However, with a harder position “upper left”, it makes the sequence more interesting.

2. **My reduced binary intermediates**
    a. Easy sequence (CORRECT)


center, fist

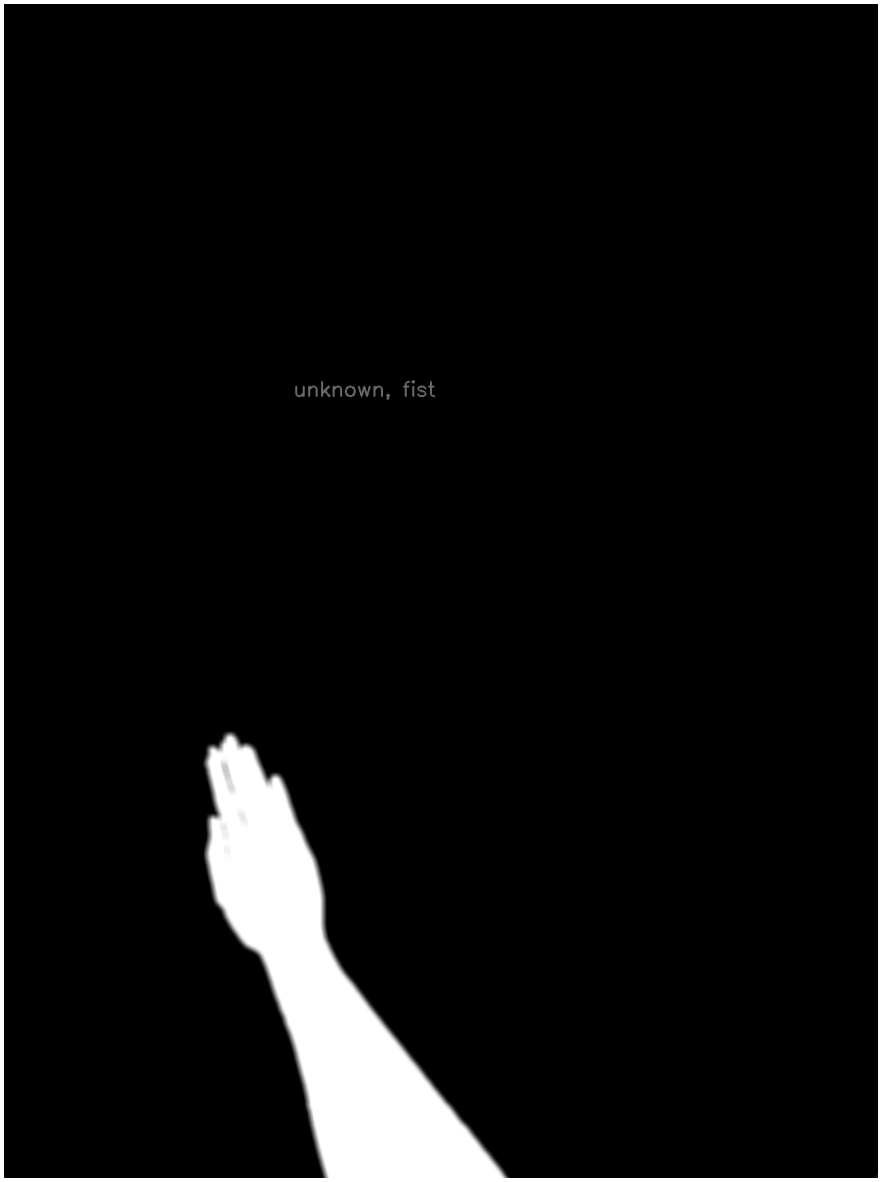

bottom left, splay



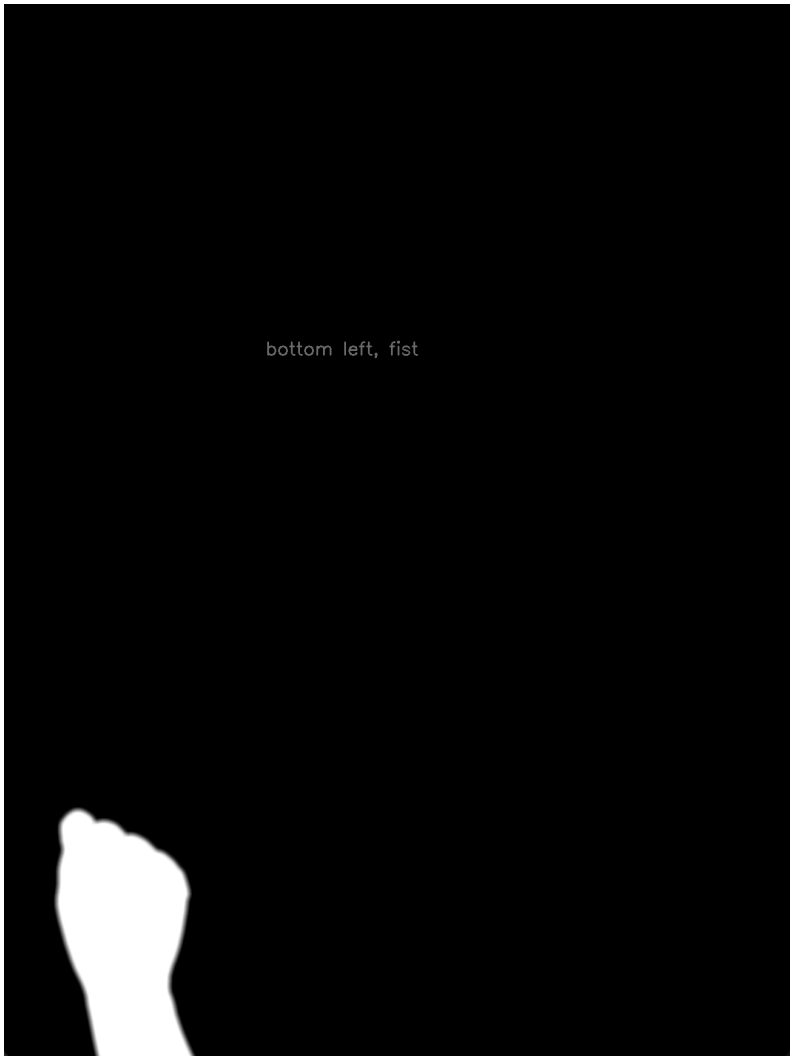
b. Difficult sequence (FALSE)


palm


upper left, splay



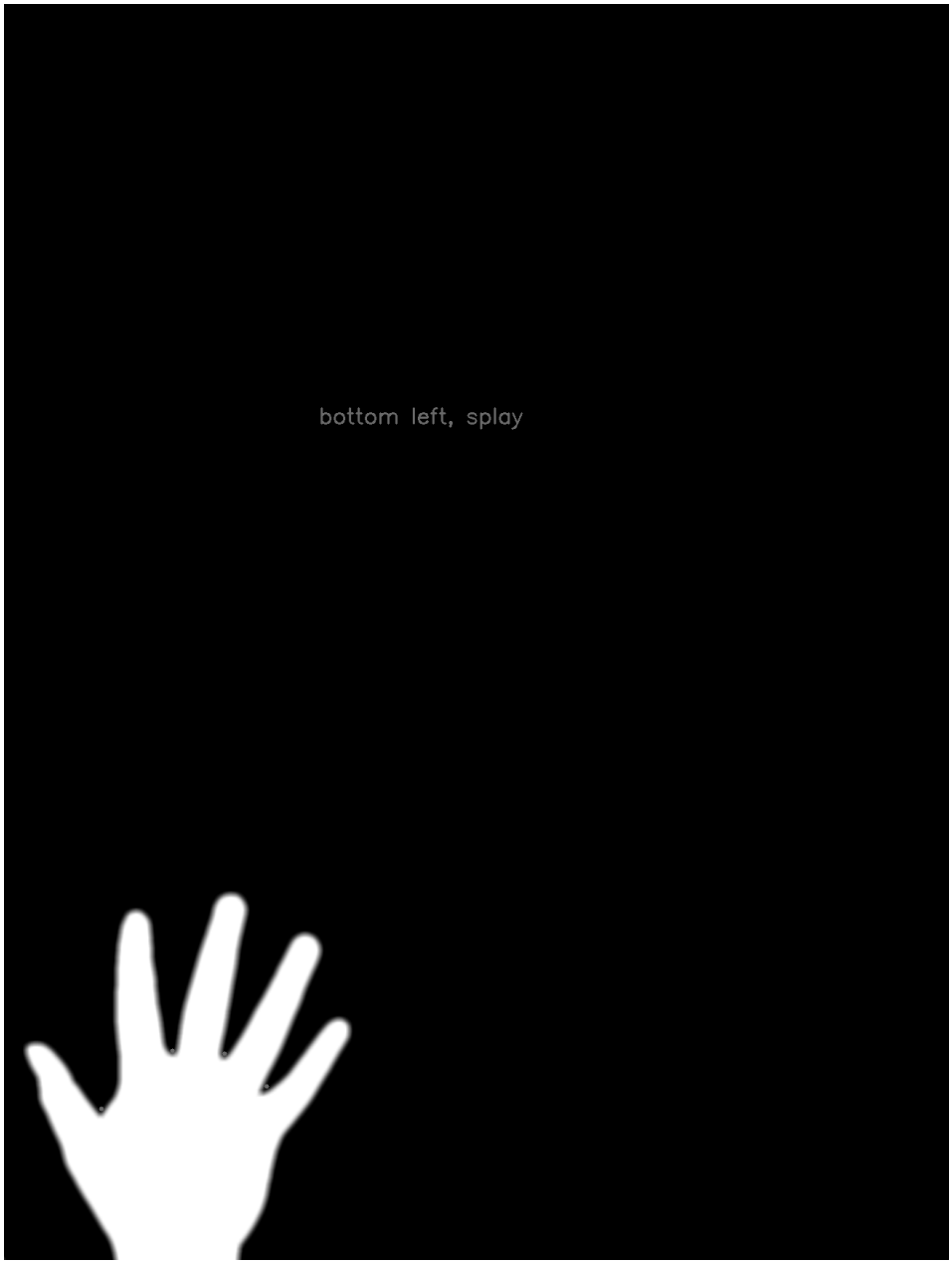
c. Good sequence (CORRECT)


bottom left, fist

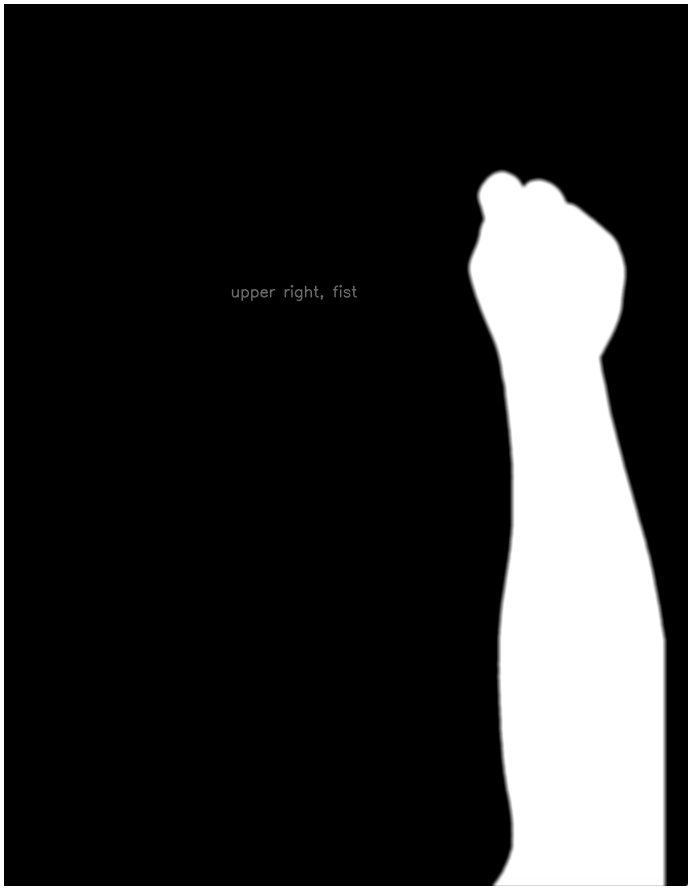

center, splay

3. **My friend's reduced binary intermediates**
    a. Easy sequence (CORRECT)


center, fist

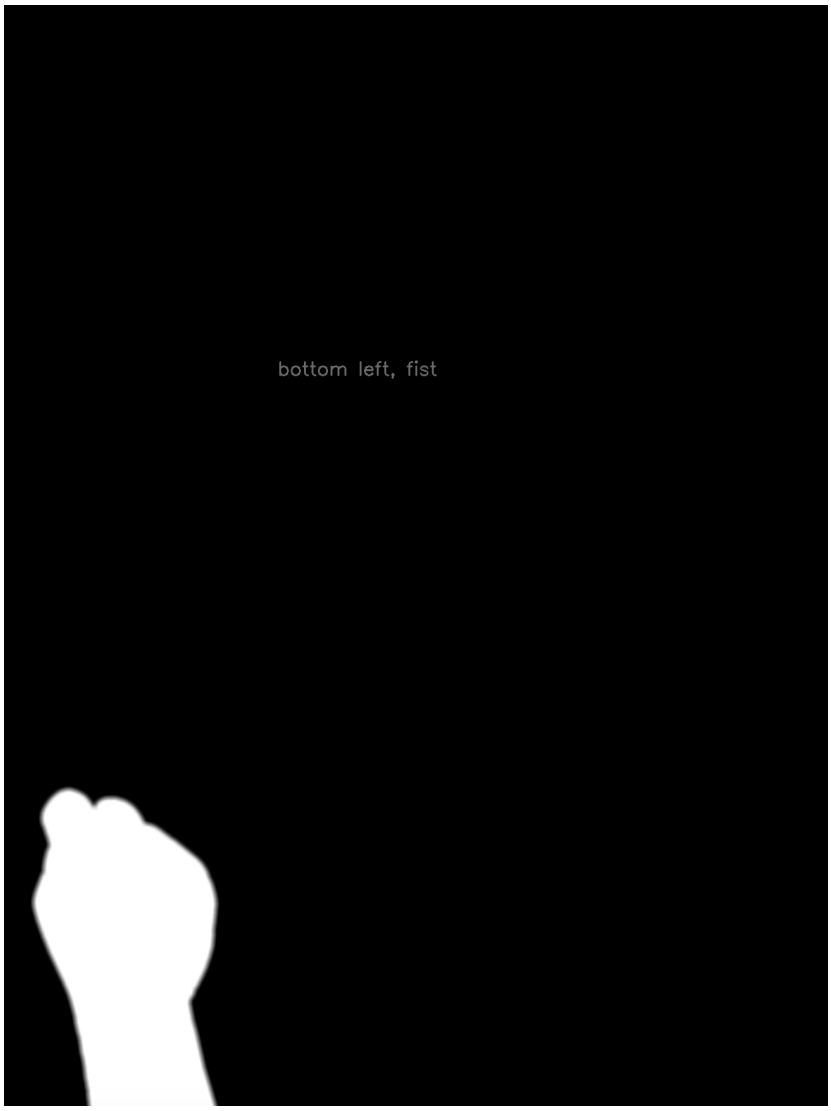

bottom left, splay

b. Difficult sequence (CORRECT)

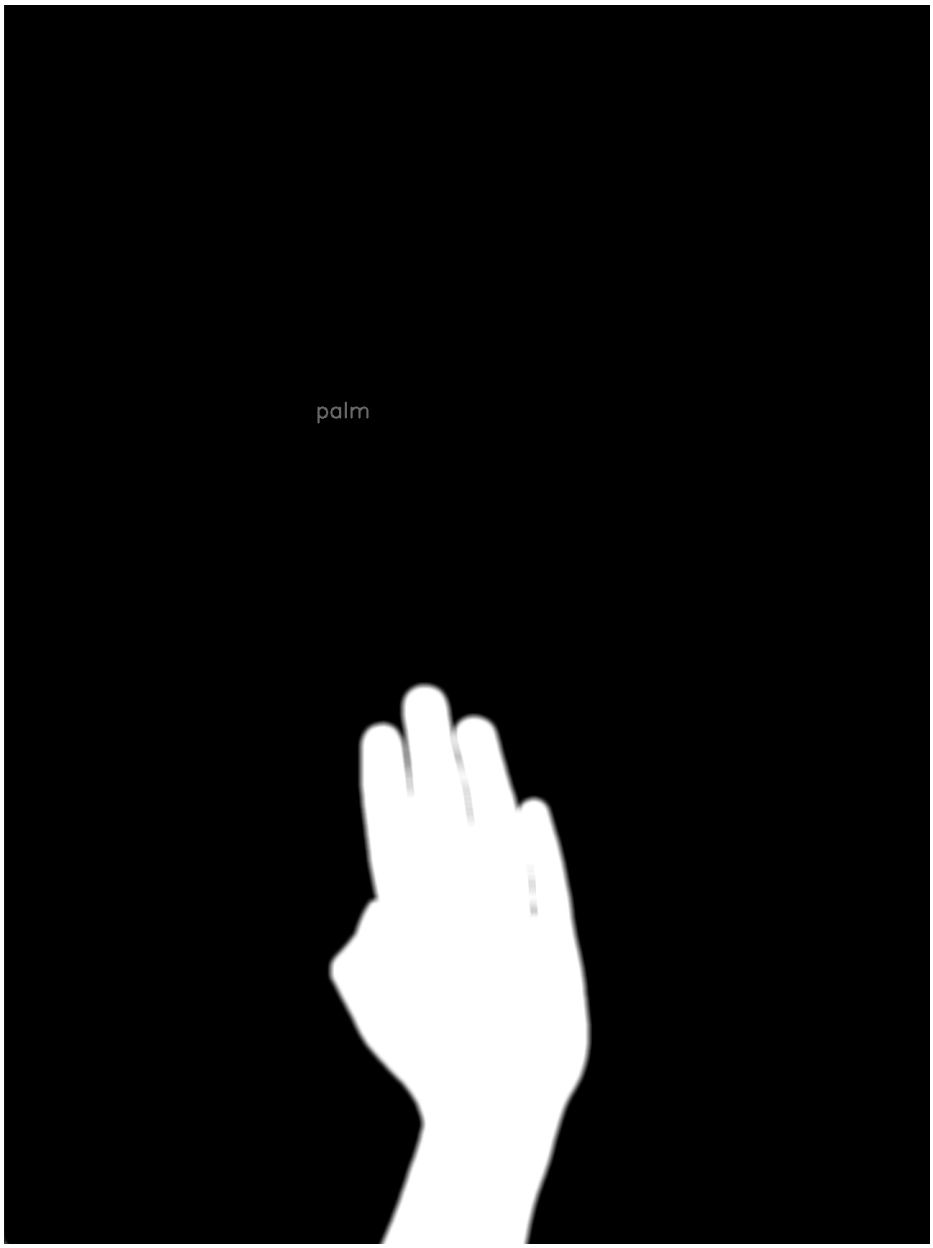

upper right, fist


upper left, splay

c. Good sequence (CORRECT)


bottom left, fist


center, splay



4. **Overall success rate**
    a. The overall success rate is 5/6 because five cases are successfully unlocked
5. **Feedback from my friend**

In terms of ease of use, my friend liked how the program does not restrict his movement or where he places his hand. However, he hopes that the text on the image can be more dynamically located. As currently, its location is always in the center. And it is hard to see sometimes if the object is in the center too.

My friend agrees to me that the "where" is more unpredictable than the "what". He believes that I should either write a function that separates the hand and the arm or force the users to put the arm upwards. He also suggests me to find the center of mass of the hand instead of finding the topmost point of the entire hand+arm.

6. **Where to improve upon?**

The biggest root problem of my current program is that I cannot locate the wrist, therefore, my program cannot separate the arm from the hand. If I can find where the wrist is, some key issues will be fixed. The first issue that is solved is the location of the hand as I can use the center of mass of the "chopped off" hand instead of the topmost point. Therefore, even when the arm is reaching downward, the program does not care because it focuses on the hand. Second, I can find where the leftmost, rightmost, topmost, and bottommost points are on the hand. Then I can differentiate a fist and a palm by finding the ratio using this formula: $\frac{math.dist(topmost,bottommost)}{math.dist(leftmost,rightmost)}$ and see if it surpasses a certain threshold. The logic behind this is that palm has a higher ratio than fist because the distance between the topmost and bottommost points are much larger. Currently I cannot use this formula because the bottommost point can be the bottom of the arm. I believe that one way to locate the wrist is to find where the distance between the horizontal ends of the hand/arm suddenly shrink.

Another area of improvement is the blurring function. Currently, all images are blurred equally. However, the smaller the object is, the less blurred it should be as the program will have a harder time differentiating what is there. One way to allow the program to detect how small the object is is to first find the hand and then see what is the ratio of its dimension to the image's dimension.

The last area of potential improvement is when I am ignoring the defect dots that are not caused by the gaps between the hand. Currently, I have the threshold set to 40, meaning that the program will think the dot is valid if the distance between the convex hull and the hand exceeds 40. However, when I tested my program on my friend's hand. I realized that it sometimes is too low of a threshold because his hand is different than mine. Something that can be improved is to create a formula that computes the threshold instead of using a hardcoded value.

Something that I would like to keep is how the defect points are determined. Currently it is able to ignore the noise dots that do not represent the gaps between the fingers. After testing it with many pictures, I have realized that it works well on top of the blurred images.

My last potential solution to the problems I have encountered is to force the users to put their hands inside a square. That way, the object will not be too small and the arm is assumed to be outside of the square.

Citations

1. https://www.tutorialspoint.com/opencv-python-how-to-convert-a-colored-image-to-a-binary-image
    a. I used this website to learn about how to convert a colored image to a binary image
    b. I am allowed to use it because it is a public tutorial website
2. https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html
    a. I used this website to learn about how to find the contour of the object
    b. I can use code from this website because it is the official OpenCV documentation
3. https://theailearner.com/2020/11/09/convexity-defects-opencv/
    a. I used this website to understand about convexity defects and I am allowed to use it because it is a public tutorial website
4. https://gist.github.com/Dhruv454000/dce6491280e09ff8d920ed46fc625889
    a. I used this github repo to learn about how the cosine rule can be used to find the defect dots
    b. I can use this github because it is public and is served as a tutorial codebase for learners