Assignment 1 Visual Combination Lock Report

By Clarence Jiang (uni: yj2737)

The main goal of this assignment is to build a computation system that takes a sequence of hand gesture images to answer the "what" and "where" questions. In terms of "what", the system should recognize what the gesture is; for instance, is the hand a fist or a splay? In terms of "where", it should know where the hand is; for example, is it at the center of the background or at the corner?

In the **initial stage**, I did the domain engineering step.

I took images through the camera on my iPhone SE. The main benefit of using a movable device, rather than a webcam fixed on the top of a pc, is that it is more flexible to capture visual inputs since I could easily move around my camera.

In addition, the whole environment setup is using a large black mousepad as the background and 2 small lamp lights that face toward the ceiling on a flat desk, which would offer adequate lighting but do not post any shadow on the mousepad. As for the environmental aspect, I controlled that my images were taken around 11 am-2 pm when the natural sunlight should reach its best. As for my appearance, I wore a long sleeve black shirt, which matches the color of the mousepad background. Also, my hand is quite white, so I made sure the color of my environment setup is as black as possible to form a contrast. In this first stage, I took a total of 10 images, including 1 image of fist and 1 image of splay for each of the 5 locations (4 corners + center).

After I finished taking images, I sent my inputs to my MacBook air through Airdrop, and my entire system would be based on Mac OS. The main python package I used is OpenCV, and the programing language is Python. However, my images were taken originally in the HEIC format, which does not work with OpenCV, so I converted the image format to jpeg by saving a copy of the image into the default "Files" app on iPhone. It automatically stored a copy in jpeg format. Finally, I renamed each file with the "what" and "where". Then, my program would be able to access those files through the "imread" function of OpenCV.

In the **second stage** of data reduction, I first checked the shape of an original image through the numpy array's shape attribute, which has a height * width of 2585 * 2817. All of my images should share similar dimensions because of the similar environmental setup and using the same hand. When I was taking the images, I made sure my left hand, holding the phone, was maintained at a similar height, so the size of the image would be approximately the same. In other words, I believe a proper resize for one image would work for every image. After experiments, I applied the "resize" function in OpenCV to scale down the image to only 10% of the width and height of the original image, so the new dimension would be roughly 258*281. Using the"imshow" function shows that my hand is still clear in the resized image.

Then, I intended to find a quantitative way to find the "hand area", so I could convert my resized image into a binary form. I used a "splay-center" image to explore how the pixel value changed when it goes from a black background to a human hand. First, I created a new 2d numpy array, and the values are the average RGB values of the resized image. Second, I select the middle row of that numpy array, as I know there is a hand in the center and a "black-to-white" moment of this "splay-center" image. I then draw a line plot (shown in Figure 1) and see there is a clear jump of value in the middle, and I conclude that having an average RGB value greater than 131 (rounded up from 130.6…) is my "hand area". I used this value as a threshold. If the average RGB value is greater than 131, it is converted to (255, 255, 255), otherwise to (0,0,0). Since these images were taken around the same time, I suppose this pixel threshold would apply to all of my images. To confirm the accuracy of my picked threshold, I checked the display of the new binary image, and it clearly outlines my hand.

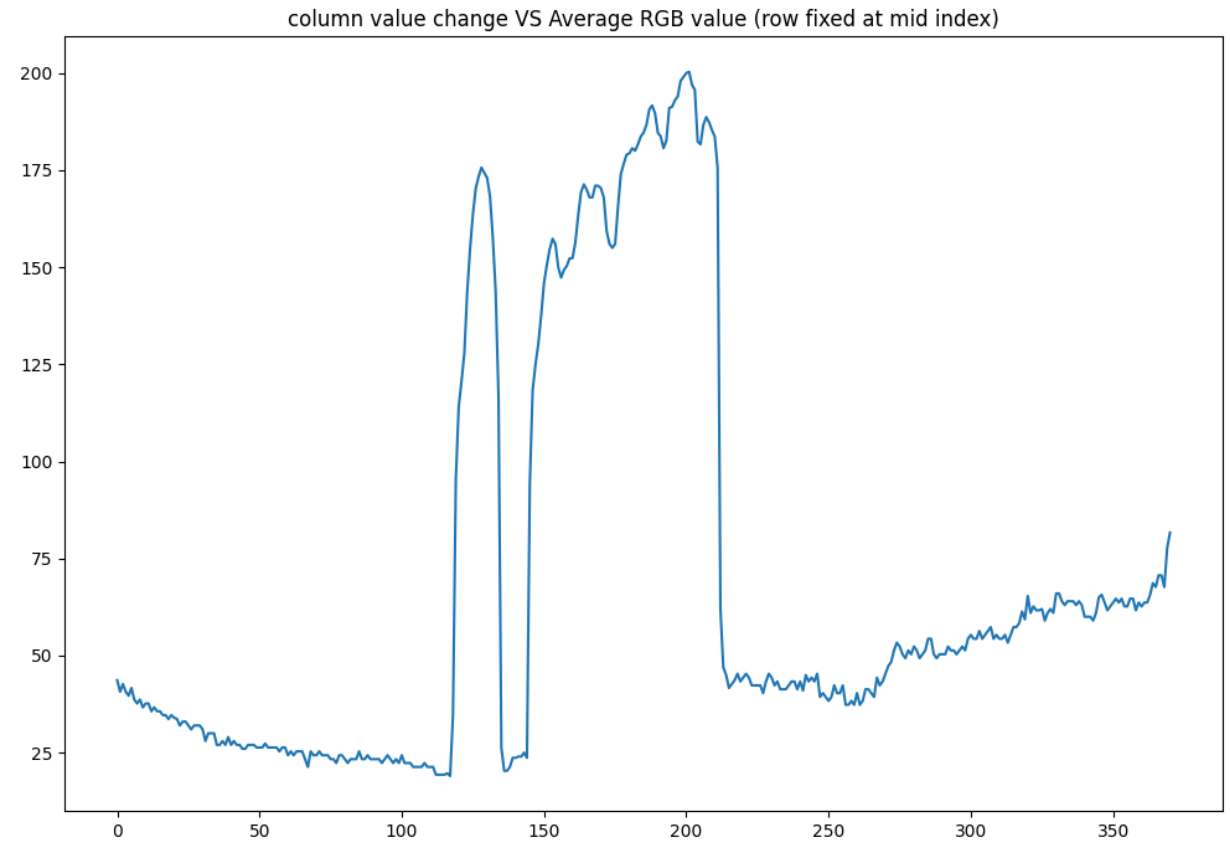


Figure 1

With the binary image, I developed 2 algorithms to find the "what" and "where". To find where the hand is, I originally wanted to apply typical built-in OpenCV functions to draw a bounding rectangle, but it seems not working well with my images, so I iterated through my images with 2 for-loops. Going row by row from index 0, I extracted the x-y-coordinates (imagine the image is in a Cartesian Coordinate; width calculated via x-dimension and height calculated via y-dimension) of the first white pixel, called the "top point". Regardless of fist or splay, the first pixel is a good sign of the upper bound of the bounding rectangle. Similarly, by going row by row reversely, I obtained the x-y-coordinates of the first bottom white pixel, called the “bottom point”. With also the “left point” and “right point” (calculate in the same fashion) as 2 other parameters, I drew a bounding rectangle with the “cv2.rectanlge” function.

Also to detect hand locations, my logic is first using the x-coordinate of “top point”. If it is in the range of (0.35*width, 0.65*width), my system regards the hand as being in the center. This method works well, as regardless of being a fist or splayed hand, the x-coordinate of top point is likely to be around the middle of the hand. If it is less than 0.3*width, it is a left image. If it’s greater than 0.7 * width, it’s a right image. Then to detect if it’s at an upper corner, I used the y-coordinate of “top point” to check. If the hand is at an upper corner, its top part is guaranteed to be at top as well. Similarly, I used the y-coordinate of “low point” to check if it’s at a lower corner. Again, I used the “less than 0.3 and great than 0.7” standard. As a result, it accurately depicts the white hand area.

Then, to find the shape, I used the area of the bounding rectangle and the proportion of black pixels as 2 metrics. It is intuitive that most areas of a fist should overlap with a splayed hand because the central palm area is the same, but a splayed hand has stretched figures that increase the height of the bounding rectangle, resulting in a much larger area. Through experiments, I used 8100 (81 * 100) as a threshold value. However, only using the area to distinguish is not rigorous, so I also counted the number of black pixels in the bounding rectangle. A fist is closer to a regular rectangle shape, so the bounding rectangle should contain fewer black pixels, which are gaps between fingers. By contrast, a splayed hand will definitely leave a lot of black spaces between fingers. As a result, in my system, a “fist” is a shape that has an area less than 8100 and a “less-than-45%” proportion of black pixels within the bounding rectangle. A “splay” is a shape that has an area greater than 8100 and a “greater-than-45%” proportion of black pixels. If a shape does not fall into these 2 categories, it is defined as “unknown”.

I annotated the image result by putting the location and shape labels on the image through OpenCV’s “putText” function. I also made sure the color of the bounding rectangle is clear and its boundary is thick enough to be sufficiently legible. Two image pairs of center fist and two image pairs of cornered splay results are shown from Figure 2 to Figure 9. My program correctly identifies both the location and gesture of the hand.

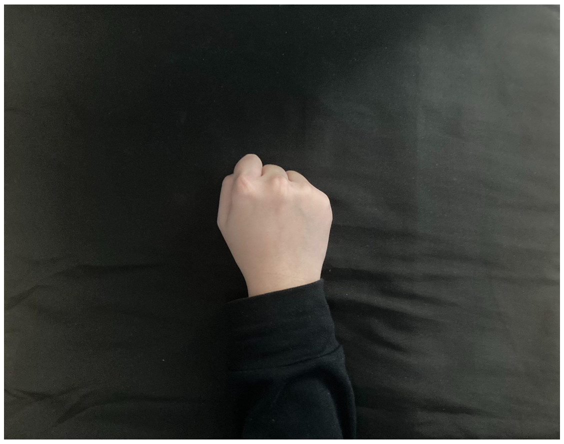


Figure 2

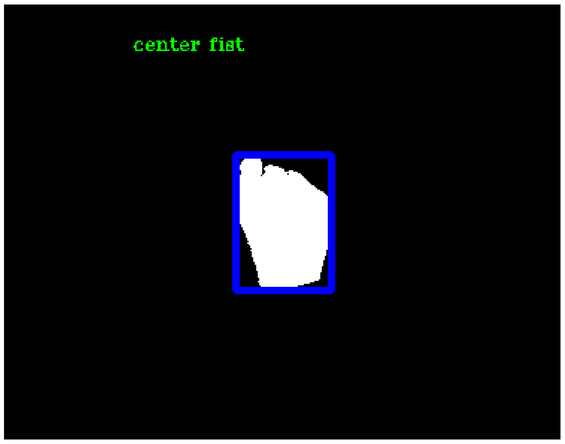


Figure 3

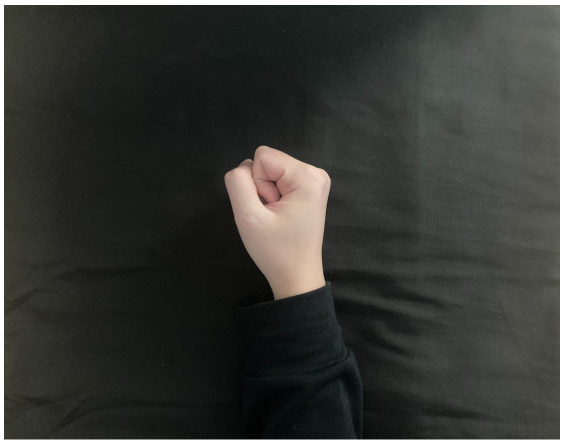


Figure 4

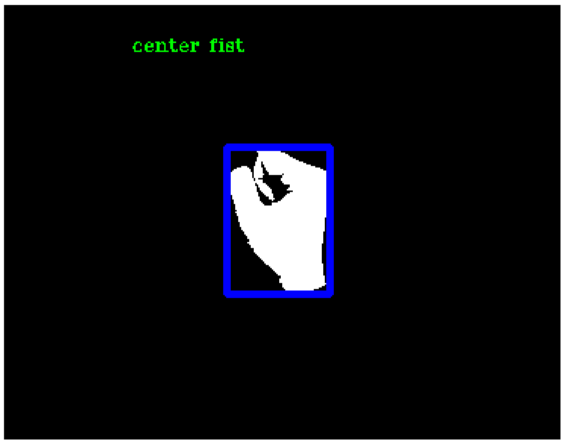


Figure 5

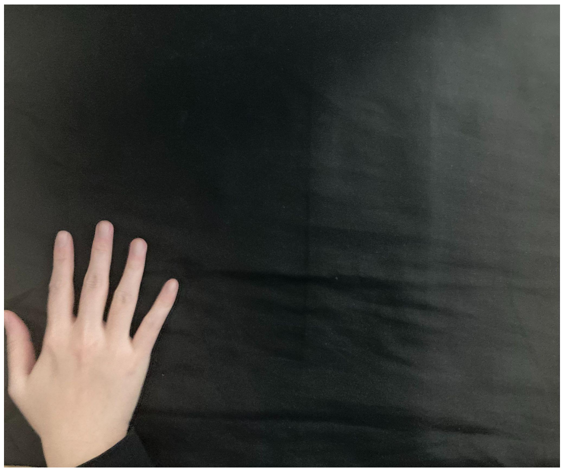


Figure 6

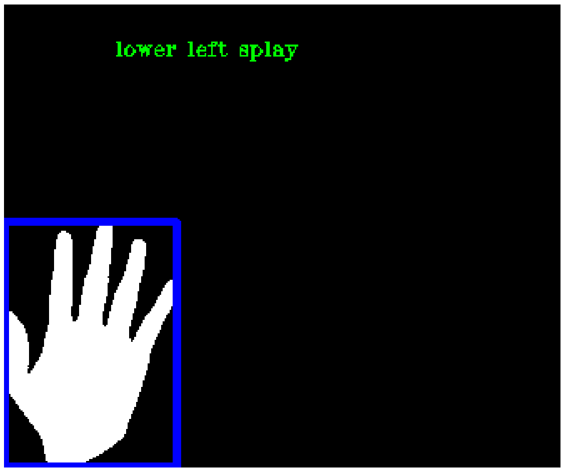


Figure 7

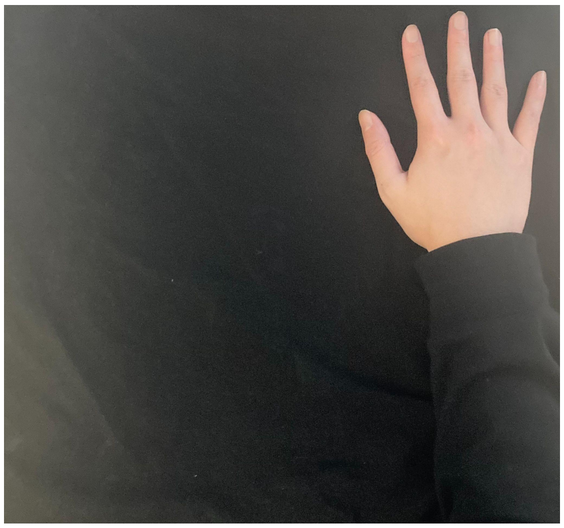


Figure 8

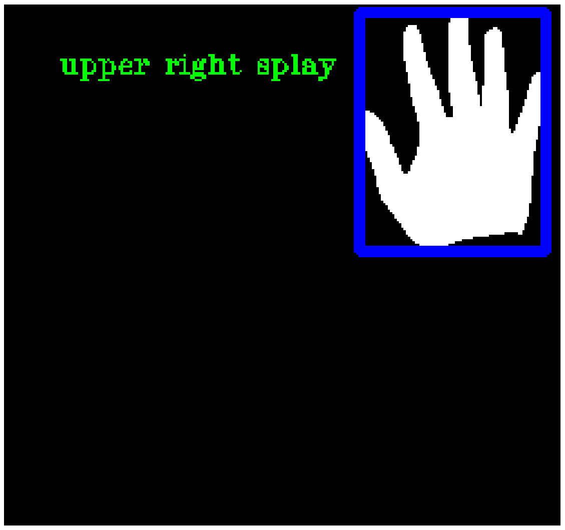
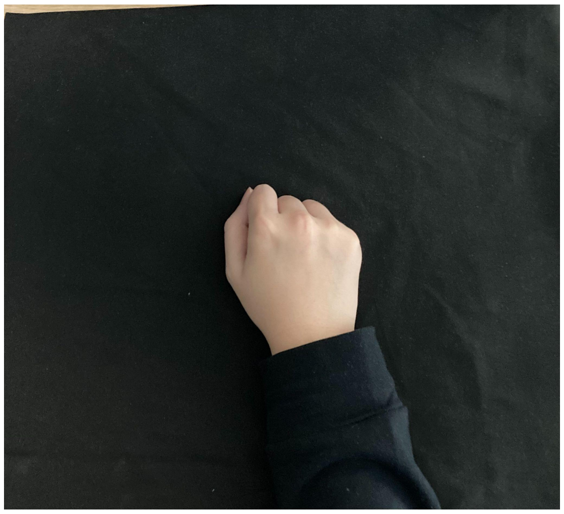


Figure 9

In the **third stage, edge and extension**, the 1st false negative example of a centered fist is shown below in Figure 11. This photo is indeed a centered fist which could be easily told from its white contour, but this image has a small white pixel area on the upper left corner (see Figure 10). Since my bounding rectangle system is based on the "top point", an incorrect "top point" caused by the white small pixel eventually generates a wrong area. This misclassification is caused by the domain engineering aspect of not taking image well.



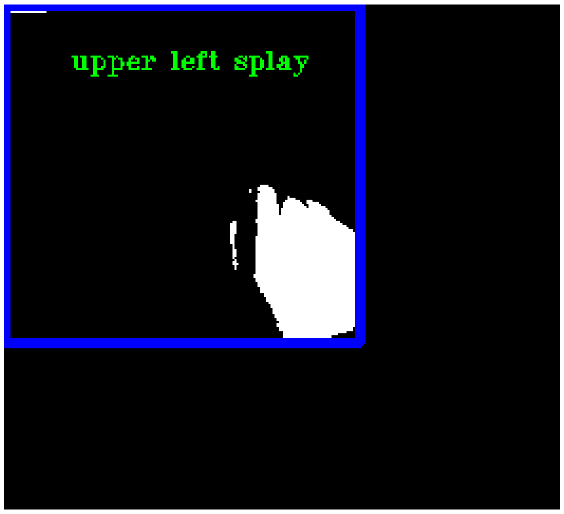
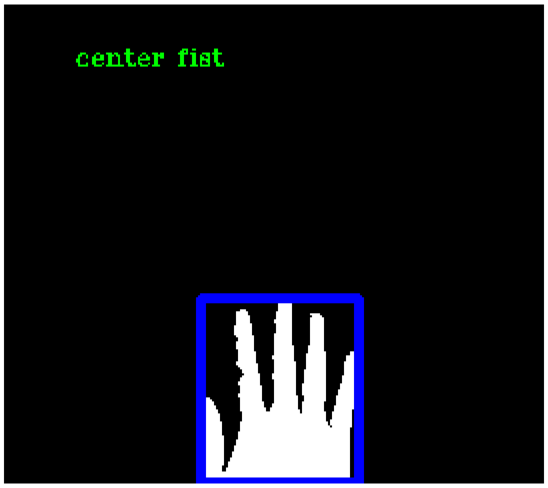


Figure 10 Figure 11

The 2nd false positive example of "fist, center" is shown in Figure 12. From the clear 5 fingers, this image is actually a splayed hand, but it is misclassified as a centered fist because part of the hand is out of frame. Since my system uses the total area as a metric to distinguish between splay and fist, a "partial" splayed hand could be seen as a fist as well. The 3rd false negative example of "splayed, upper right" is shown in Figure 13. It has the same area problem as the 2nd example shown in Figure 12.



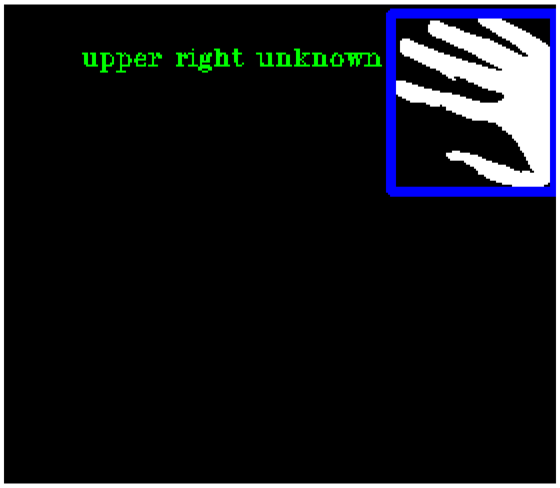


Figure 12 Figure 13

The 4th false positive example of "splay, upper right" is shown in Figure 14. This hand gesture is in fact a Spock sign, and not every finger is stretched, but it is mispredicted as a splay, because a Spock-sign hand would have a really similar area compared to a splayed hand and it also contains a lot of black pixels in the bounding rectangle.

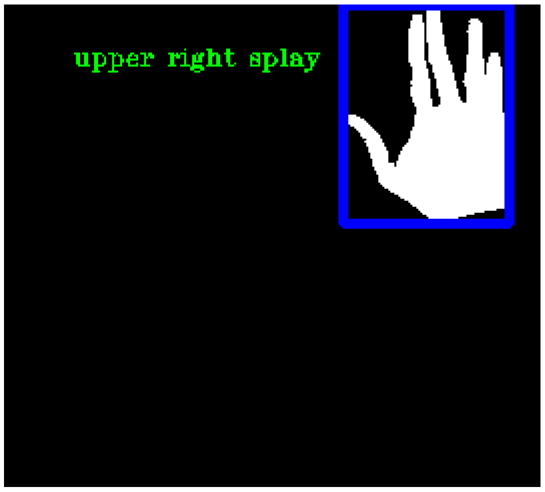


Figure 14

For the 5th question that talks about true negative situation of “unknown”, my 1st example is a “c-shaped hand, center” shown in Figure 15. It is not recognized as a splayed hand since it has a small area less than 8100, which is a good sign of a fist. However, it is also not considered a fist, since there are too many black pixels within the bounding rectangle. Thus, it falls into the “unknown” category, which is correctly predicted. The second example is shown in Figure 16, where the hand gesture is more like a “bird-mouth-shape”, and it accurately depicted as “unknown” for the same reason as Figure 15.

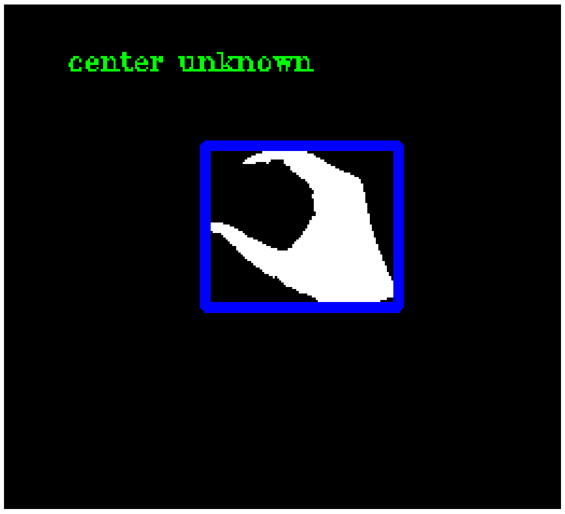


Figure 15

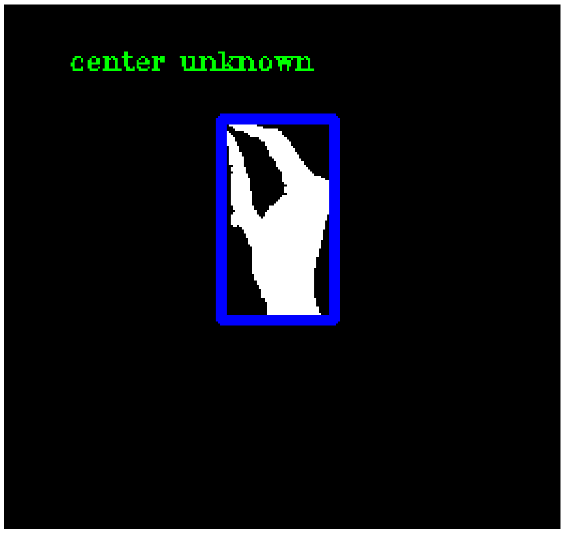
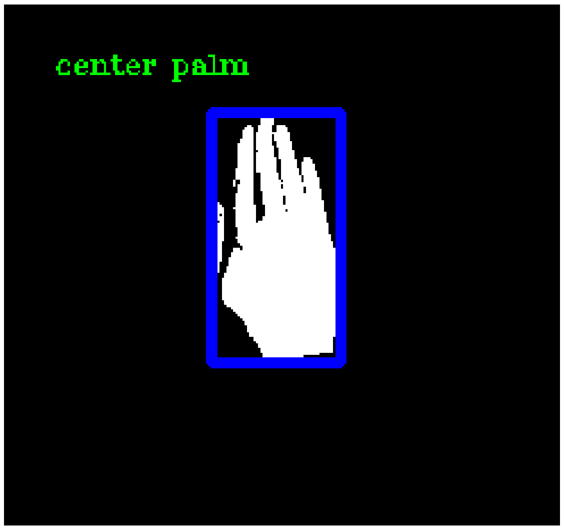


Figure 16

Furthermore, as “palm” get introduced, I tested 2 images of a “palm” and discovered that it is actually easy to distinguish between a “palm” and a “splayed hand”. Within my system, because the fingers of a “palm” are tightened with each other, it significantly reduces the area because of a smaller width. Though the height of a palm is similar to that of a splayed hand, the width (between the leftmost and rightmost finger) of a palm is only half of a splayed hand. Thus, with only the area condition is sufficient to distinguish a normal “palm” and “splayed”, so the difficulty is distinguishing between "palm" and "fist". I found it is obvious that the ratio of height and weight is different from that of fist. For a palm, the result of using the greater one between height and weight to divide by the smaller one would be greater than 1.8, whereas a fist is around 1.4. Thus, I incorporated the new vocabulary by adding an extra step to check the length-width ratio. Two successful recognitions are shown in Figure 17 and 18.



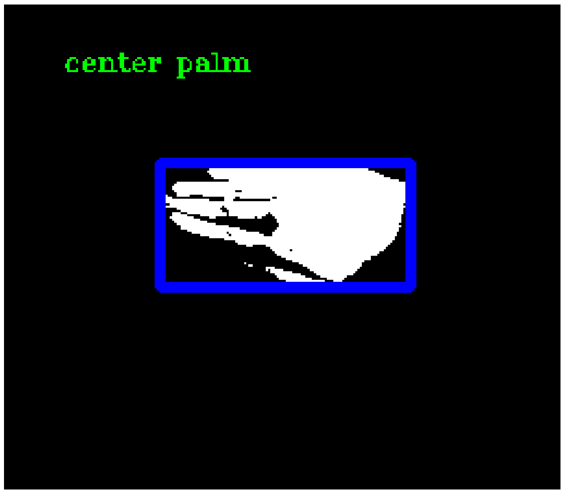


Figure 17 Figure 18

Again all these threshold values, which are used to find the hand shape, fit more to my own hands. This phenomenon is unavoidable. Even if I changed my system from using exact numbers like 8100 to a percentage, this percentage is also subject to the variety of human hand. Even if I do not use my chosen metrics, other alternative metrics have to be also quantitative with some numeric value (otherwise it is not scientific), and these numeric values will also be subject to the fact that human hands are different. Thus, I argue that as this project does not focus on a pre-training phase that makes the system understand each user's hand, my overall methodology of using the area of the bounding rectangle, the proportion of black pixels within the rectangle, and the "height-width" ratio is sufficiently designed.

An example of a palm incorrectly recognized as a fist is shown in Figure 19. My system first checked the area of the image. Since a part of the hand is out of frame, the system thinks its area is small and it could not be a "splayed hand". Also because a part is missing, the system obtained a small height-width ratio, which made it think the hand is a fist.

An example of a splayed hand incorrectly as a palm is shown in Figure 20. We could tell it is a splayed hand because of the stretched fingers. The system first thinks the area is too small, and then it checks the "height-width" ratio. Since the missing part is the right 2 fingers, rather than losing the bottom half like some of my previous examples, this actually increased the "height-width" ratio, and the system recognized it as a "palm".

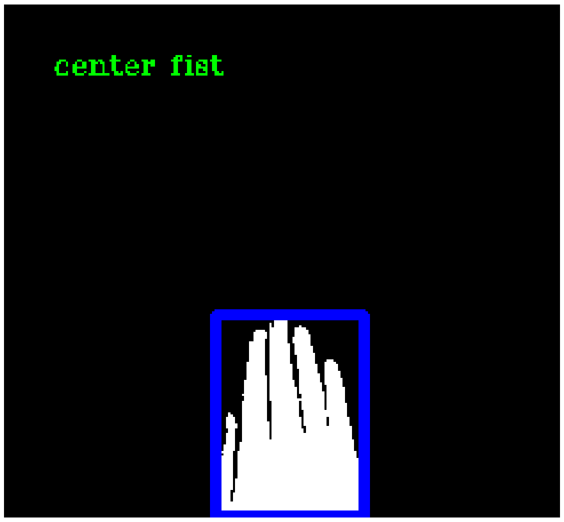


Figure 19

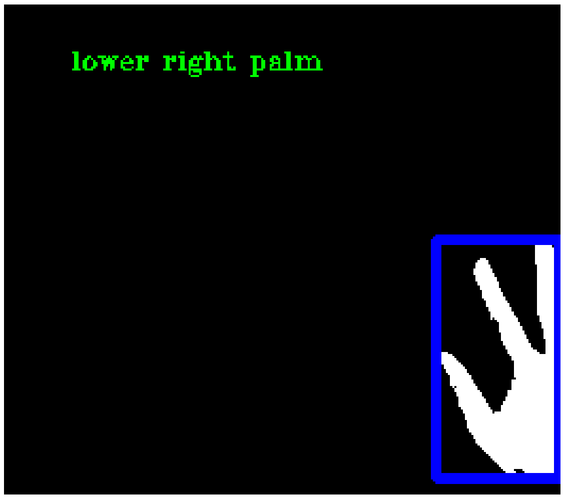


Figure 20

The 2 examples of unknown gestures under the vocabulary of palm, splay, and fist are shown in Figure 21 and Figure 22. Figure 21 is just a pose of a circle, while Figure 22 is a sign of "ok". Both images are recognized as "unknown" for the same reason. First, the system found that their areas are small, so it could not be a "splayed hand". Then, they found their height-width ratio does not match a palm. Finally, they found there were too many black pixels for them to be a fist image, which eventually gives them the "unknown" label.

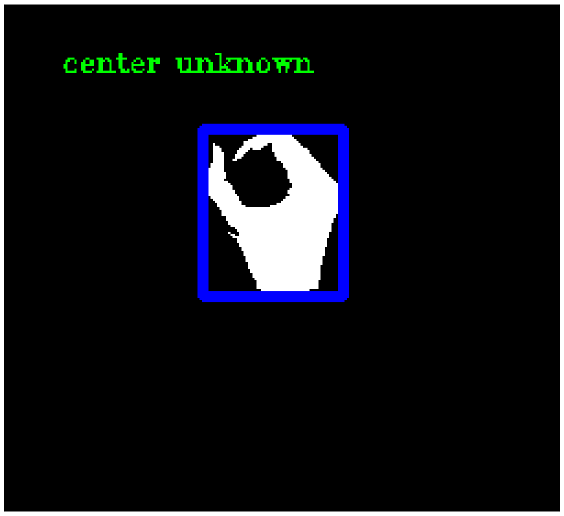


Figure 21

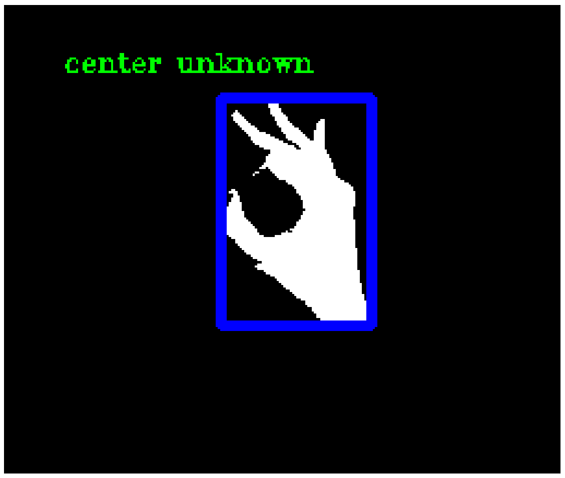


Figure 22

Finally, for the **last step**, **evaluation**, my methodology in this step is creating 3 ground truth labels first, inputting 3 captured images into my system, and finally comparing the results recognized by the system with the ground truth. It is considered correct only if both the location and shape labels are matched. The lock is opened only if all 3 comparisons are correct. In this section, there are not many new codes, I just re-utilized the same "find_shape" and "find_location" functions developed previously. The only difference is now that my input is a sequence of images, so I created a for-loop that iteratively compared results. The project requires printing the predicted result of each of the 3 images. Thus, even if the first image is already recognized incorrectly, the for-loop should still continue. I had an “open_lock” boolean variable. If there is a wrong recognition, it will be set to False, and eventually, I will print out if the lock is opened at the end of the code written for this section.

My easy sequence would be giving my system the same task 3 times. In other words, my ground-truth label is “splay, center”, “splay, center”, and “splay, center”, and I will test with 3 new different “splay, center” images. This is the easiest sequence, as the system essentially only needs to make one “location-shape” pair work, and my system successfully captures the 3 “splay, center” images. The binary intermediate results are shown in Figure 23, 24, and 25.

My friend’s easy sequence is just making each gesture of splay, fist, and palm at the center. He thought making standard gestures is easy for the machine to recognize. Again, the results are shown at Figure 26, 27, and 28, and my system successfully captures all the easy images.

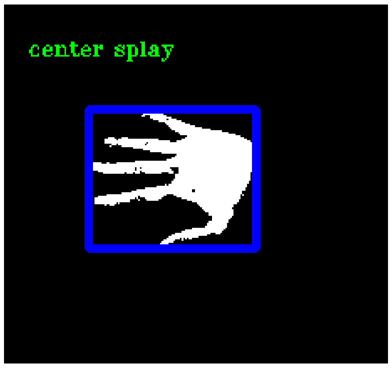


Figure 23

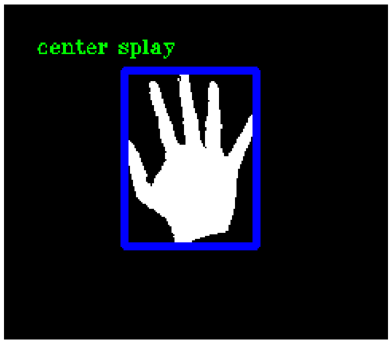


Figure 24

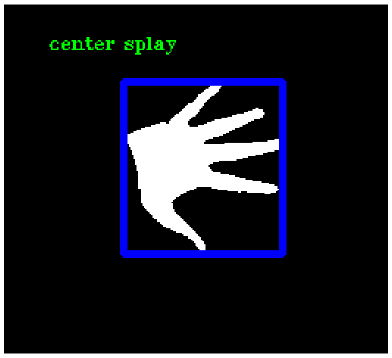


Figure 25

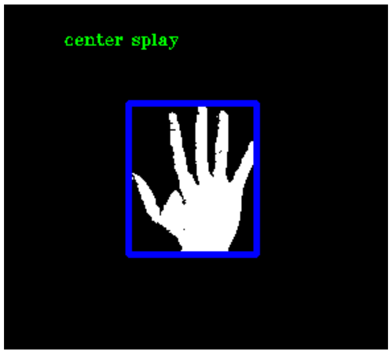


Figure 26

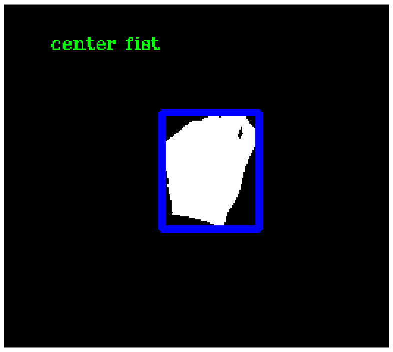


Figure 27

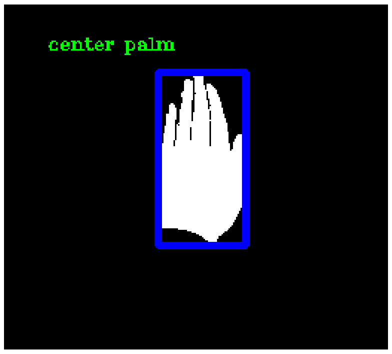


Figure 28

As for the difficult sequence, since each image has to be one of “palm”, “fist”, and “splay” and it has to make the system fail frequently, I came up with the idea of putting down my long black sleeve, so more of my human skin was exposed in the image. The 3 images are shown in Figures 29, 30, and 31. My system used the area of the bounding rectangle as a key metric of finding a splayed hand, so the areas of these 3 images are incorrectly predicted due to the large bounding rectangle. Thus, this difficult sequence reveals a problem of my system; it is hard to find the correct area of a hand if it is affected by the skin color of arms. My system 2.0 would tackle this problem, which will be explained in later paragraphs.

My friend used both hands in his difficult sequence shown in Figure 32, 33, and 34. My system never considered the situation where users put both of their hands together in the background, so the area calculation mechanism would mislead the result again; two hands will definitely cause a much larger bounding rectangle. The system thought Figure 34 was a splayed hand because of its large area. I will explain how my system 2.0 will tackle this problem.

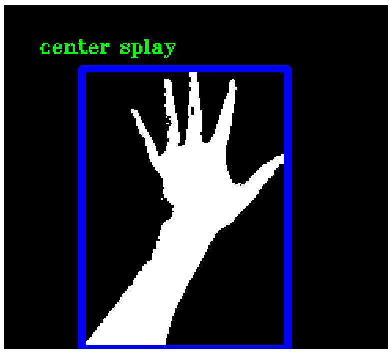


Figure 29

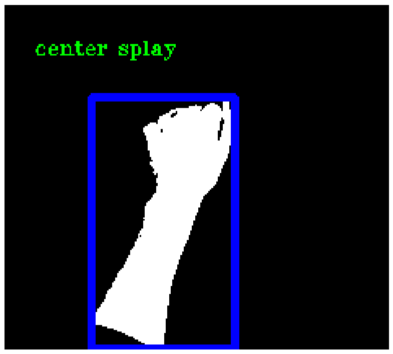


Figure 30

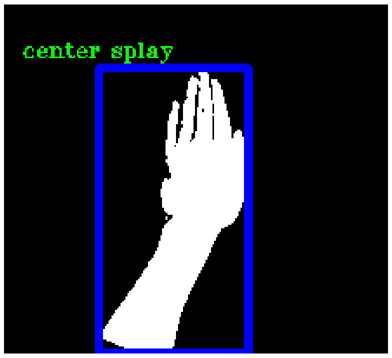


Figure 31

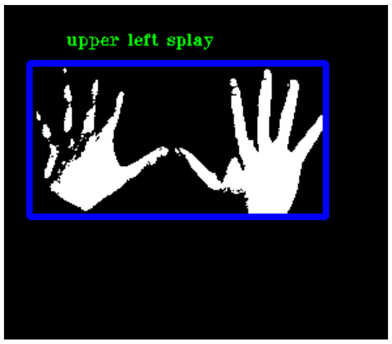


Figure 32

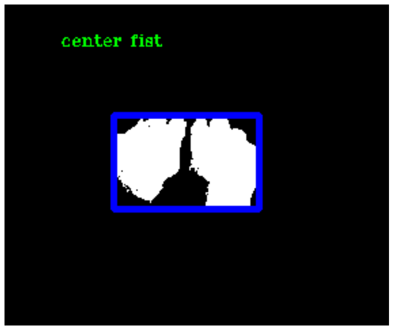


Figure 33

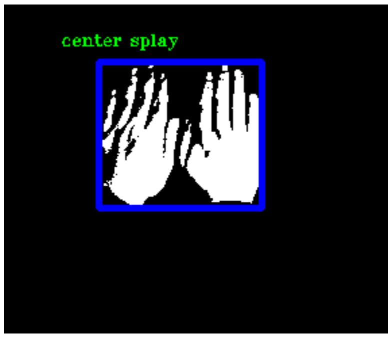
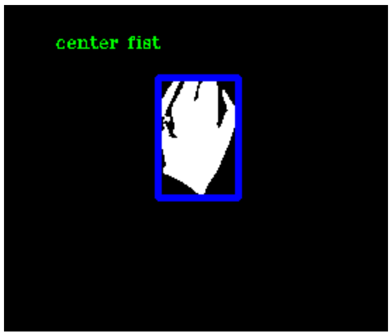
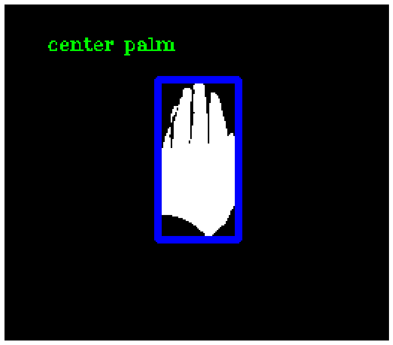


Figure 34

Finally, for the interesting ones, I tried a crawling version of each of the 3 gestures. It is most obvious in Figure 37; there are some curvatures on my finger indicating that this is a crawling gesture. A crawling gesture will make the overall height of the bounding rectangle smaller, and the fingers will be twisted a little bit. I believe this is an interesting alternative version to test, and my system successfully distinguished them; results are shown in Figure 35, 36, and 37.

My friend first tried a 4-finger version palm, shown in Figure 38. He hid his pinky finger by bending it. He told me a person could lose a finger due to accidents, so it will be interesting to hide a finger and see how the system responds. He also suggested testing 2 other alternative gestures of fist that I have not tested before. One is showing the inside of a fist outward, demonstrated in Figure 39, and the other way is having the inside to face to the right shown in Figure 40. My system correctly identified all of them.





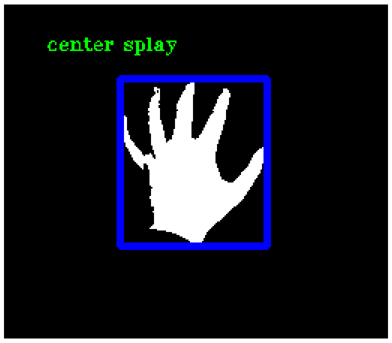
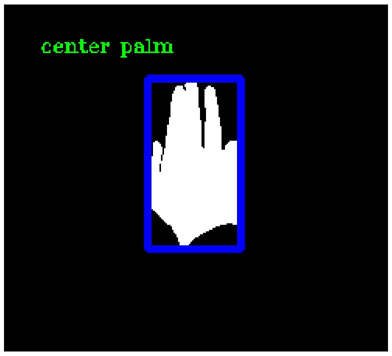
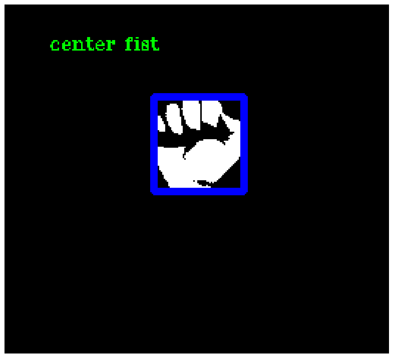


Figure 35 Figure 36 Figure 37





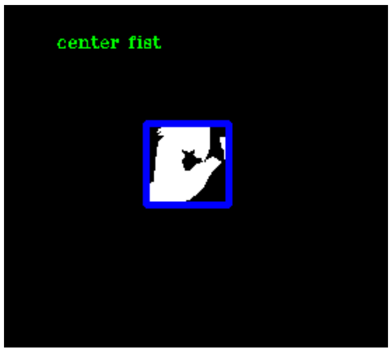


Figure 38 Figure 39 Figure 40

In summary, my overall success rate is 4/6, and the 2 failure cases are all difficult sequences. It is reasonable as those 2 cases are specifically designed to harshly challenge my system. My friend claimed that my system is straightforward and easy to use, given that the hand is already encircled with the rectangle, and there is the caption that directly shows the predicted result. Thus, it is easy to make comparisons with ground truth results.

As for my System 2.0, I would keep everything I have now. I will consider dropping some parts only if I run into some problems that could not be solved with new methods incorporated.

I need to first address the 2 failures in my current exploration. For the problem that my system is likely to be affected by the skin color under the wrist, I will add another algorithm to find the wrist location of the user's hand. There is a pattern that the width of a human hand will decrease from the central palm to the wrist, and then increase (or maintain the same width) from the wrist to the forearm. Thus to solve the problem in Figures 29, 30, and 31, I plan to scan through the bounding rectangle row by row and search for the white section. The row, with the lowest width of the white section, should be really close to the exact wrist location. Thus, the part that is above the wrist will be the entire human hand.

On the other hand, to tackle the failure caused by having 2 hands, my system needs to improve on the algorithm that develops the bounding rectangle. My system assumes that there will be only 1 hand, so it is reasonable to find the top point, low point, left point, and right point as the basis of a bounding rectangle. However, if there are 2 hands, the bounding box will include both hands. I have a straightforward solution; consider the rectangle as an x-y coordinate system, set the x-coordinate equal to the midpoint of the bottom edge of the rectangle, the pixels on this column are largely composed of black pixels. By contrast, if the image has only one hand, the middle column will be composed of much more white pixels. By checking the middle column, the system could further distinguish between the “one-hand” and “two-hand” conditions and adjust its bounding rectangle.

In the future, another problem is I find that human hands are different for each individual and are probably greatly affected by the person’s body size. A short person is likely to have a small hand, and some system parameters do not work well with hands that are too small or large. There is no standard on the value of “area_splay/area_fist”. Even using a percentage area metric rather than a fixed value does not solve the problem, because there is no guarantee on how big the image input is. Thus, to make this system perfectly generic, there has to be a training procedure that makes the system understand a set of images and obtain a set of parameters. This set of parameters will work well for every image this particular user takes since each user has the same hands and the same image-taking habits. When another user starts to use this system, the training process will be run again before analyzing images. However, this assignment is not aiming to overfit this system, so I will explore these improvements as a side project later.