Visual Combination Lock

Gesture Recognition

Author: Amee Assad

I Domain engineering

A. Imagery capture

The basic equipment required for this system is first and foremost a mode of capturing images. I used an iPhone X, which features a 12-megapixel wide-angle camera. While high quality images are not necessary for the program to work, the camera on my phone was the most accessible way for me to capture images.
The most important part of the gesture detection software is having a green background. I decided to use green as a requirement of a background because it stands out against the red skin tones of the hand and is thus easier to detect than a color like black or white. The lighting for the image capture must thus not be green, and the setup must be well lit in order to detect the greens of the background in contrast to the the colors of the hand. This also means that there cannot be dark shadows throughout, but some weaker shadows will pass.
No other objects must be in the image other than the green background and the hand that is doing the gesturing. While jewelry is fine, the jewelry must not be green in order for the program to work. Moreover, the background doesn't have to be a completely matte green. For example, I used a green notebook and a green folder, and both had creases, writing, and small holes in them; however large extraneous objects are not OK.

B. Environment

I am running the system on a 2016 Macbook Pro 13-inch, which is using version 10.15.1 of macOS Catalina. I went with Python as a programming language because this is my first computer-vision project and Python code is definitely the most readable. My program relies on the functions within OpenCV. At first, I tried to run some starter code on PyCharm, but was having a lot of difficulty downloading the cv2 package on my platform. As a result, I switched over to Spyder, which didn't have issues downloading cv2. I chose OpenCV because it has some powerful image processing functions that are especially useful when detecting hand gestures. Other packages I downloaded are numpy (to edit arrays that are used within cv2), and imutils simply to resize the image in one line.

C. Library

The images are intended to capture gestures shows using a single hand, face down, with the arm at the bottom of the image and the hands upwards placed within a green background. The images are stored as png files that are kept in the input "amee" folder within my directory. Output png files are saved in the output folder, and the mid-step processing of the image contours are saved in the contoured directory.

Within the input folder, I captured about 60 photos of hand gestures, with multiple within each position and using different hand gestures, as well as those not within my vocabulary.

II Data reduction

The vocabulary is divided into the "what" and the "where".

A. "What"

The "what" includes the terms "fist", "splay" and "unknown gesture". These terms are saved as strings under the gesture variable in my program. In order to label the gesture, the program first blurs the input image using a gaussian blur method as part of OpenCV to smooth out the photo and prepare it for contour extraction. Then, I detect the green pixels, with green being determined if the green rgb is above 35, or if the green rgb is greater than the blue rbg or red rgb. This allows me to create a binary image, which I save and blur and smooth again before using cv2's findContour() method to extract the contours from the binary image. After doing so, I get the largest contour based on area and save that contour as the variable cnt. In order to focus on the hand and not the arm of the user, I figure out if the contour's topmost point is in the top quarter of the image or not in order to determine if the user is trying to reach a top corner or not. If the topmost point is within this area of the image, I remove the bottom third of contour points because that is most probably part of the arm and not the actual hand of the user. Otherwise, I'll just remove the bottom tenth of the contour points because it means less arm is showing. Next, I smooth out the contours by taking an approximate contour with the approxPolyDP() method, as otherwise it can create a "spiky-looking" contour.

In order to determine the actual gesture, I use convexHull(), which allows me to figure out the convexity defects based on the approximate contour and the hull. Using a similar method as that within OpenCV's documentation, I get each convexity defect, and if it is above the moment point then I count it as a possible finger (by incrementing the variable *count*). Defects below the centre of the contour most probably occur because of the arm, which is what I do not want to include.

"fist" -- describes pictures that include a hand curled in fist-form facing down. The way to detect a fist is if the count variable is below a 2. I allow a margin of error because sometimes a fist can still create a convexity defect above the moment point because of irregularities in finger lengths.

"splay" -- is a hand that is face down and in a "hi-five" gesture. A count of 3, 4 or 5 will result in a splay gesture, which allows for some margin of error.

"unknown gesture" -- used when the program cannot identify the hand gesture shown in the photo. This safety margin occurs when there are absolutely no contours due to incorrect environment setup (there is no hull detected) or the contours give results that don't match a fist or splay in terms of the count.

B. “Where”

“Where” includes the terms, “centre”, “top-left”, “top-right”, “bottom-left”, “bottom-right” and “unsure position.
The position string value is determined using the top_half boolean, which was used to figure out whether the top point of the contour was high up in the image or not. However, this is not enough, as it doesn’t account for when the hand is *not* in a corner or the centre of the image (such as the centre-left or centre-right). Because the arm brings in error opportunities, I only make the x axis stricter and leave the top_half boolean to determine if it is up or down.

“top-left” -- the moment point is in the top-half and left-third of the image.

“top-right” -- the moment point is in the top half and right-most third of the image.

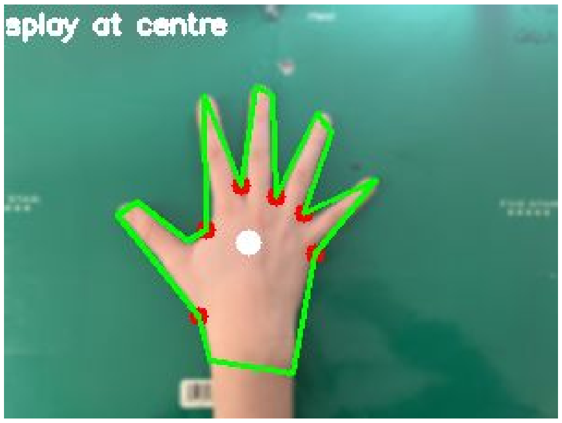
“bottom-left” -- the moment point is in the bottom half and left-third of the image.

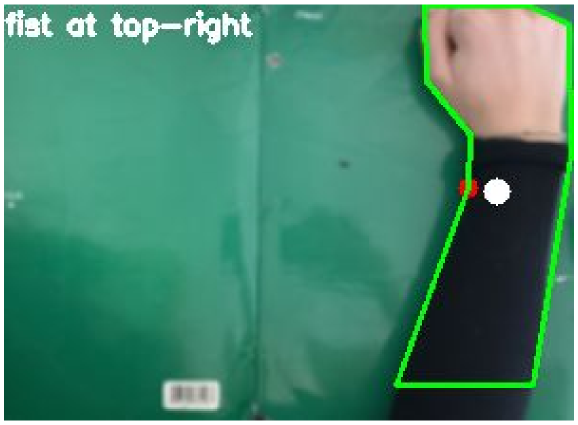
“bottom-right” -- the moment point is in the bottom half and right-most third of the image.

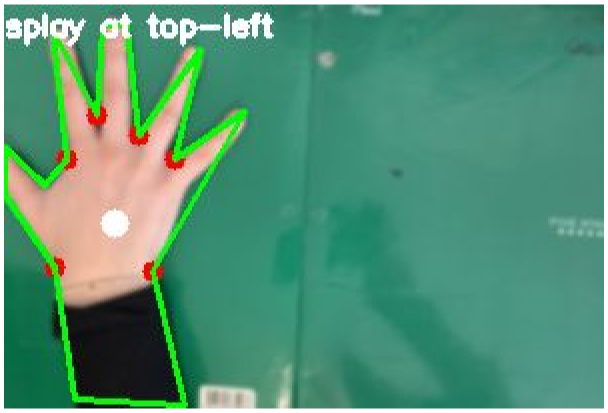
“centre” -- occurs when the moment is within the ⅖ and ⅗ of the x axis of the image and within the middle third of the vertical of the image. I gave more leeway for the vertical than the horizontal because the arm adds more room for error.

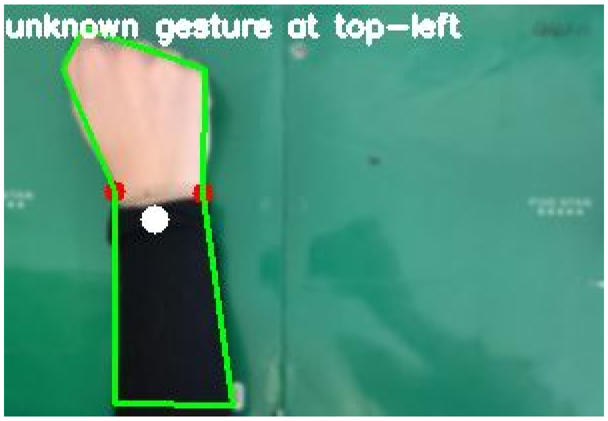
“unsure position” -- when the moment point does not fall within any of these regions, or the centre-- so it is top centre or bottom centre.

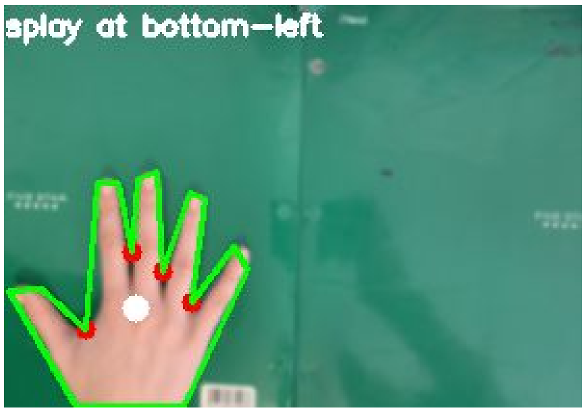
Below are some examples of gestures and positioning:

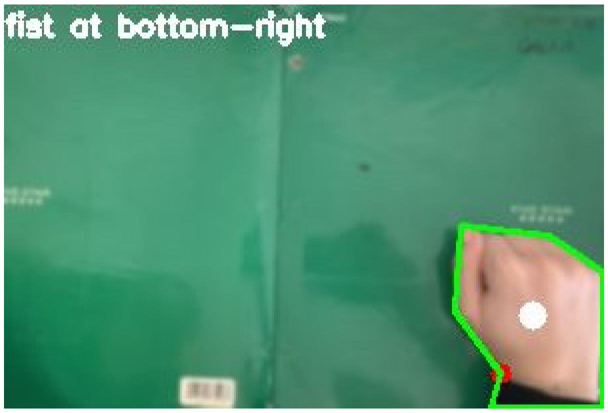

splay at centre

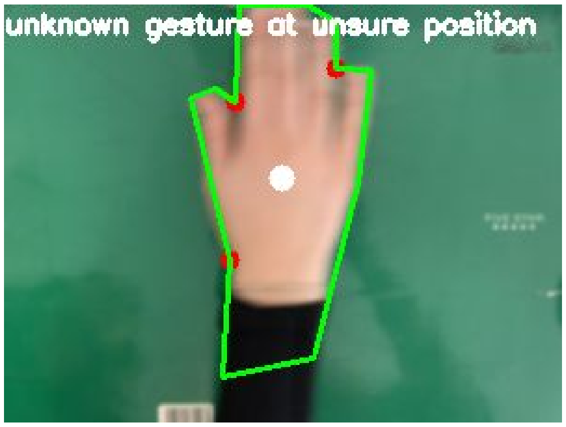

fist at top-right

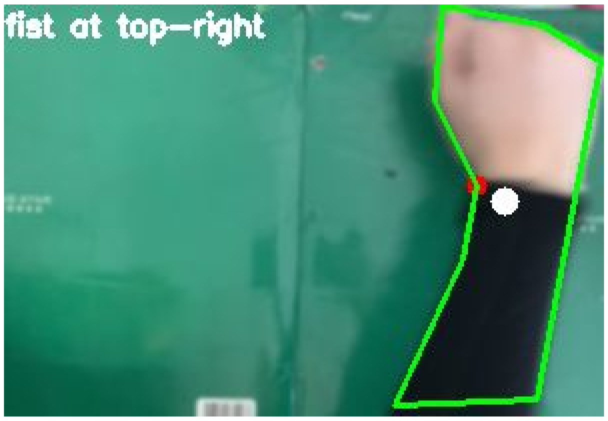

splay at top-left


unknown gesture at top-left


splay at bottom-left


fist at bottom-right


unknown gesture at unsure position


fist at top-right

III Parsing and performance

The user is meant to place their fist in the centre of the image and then splay it in the top right corner of the image in order to unlock the code. The password is set as the following:

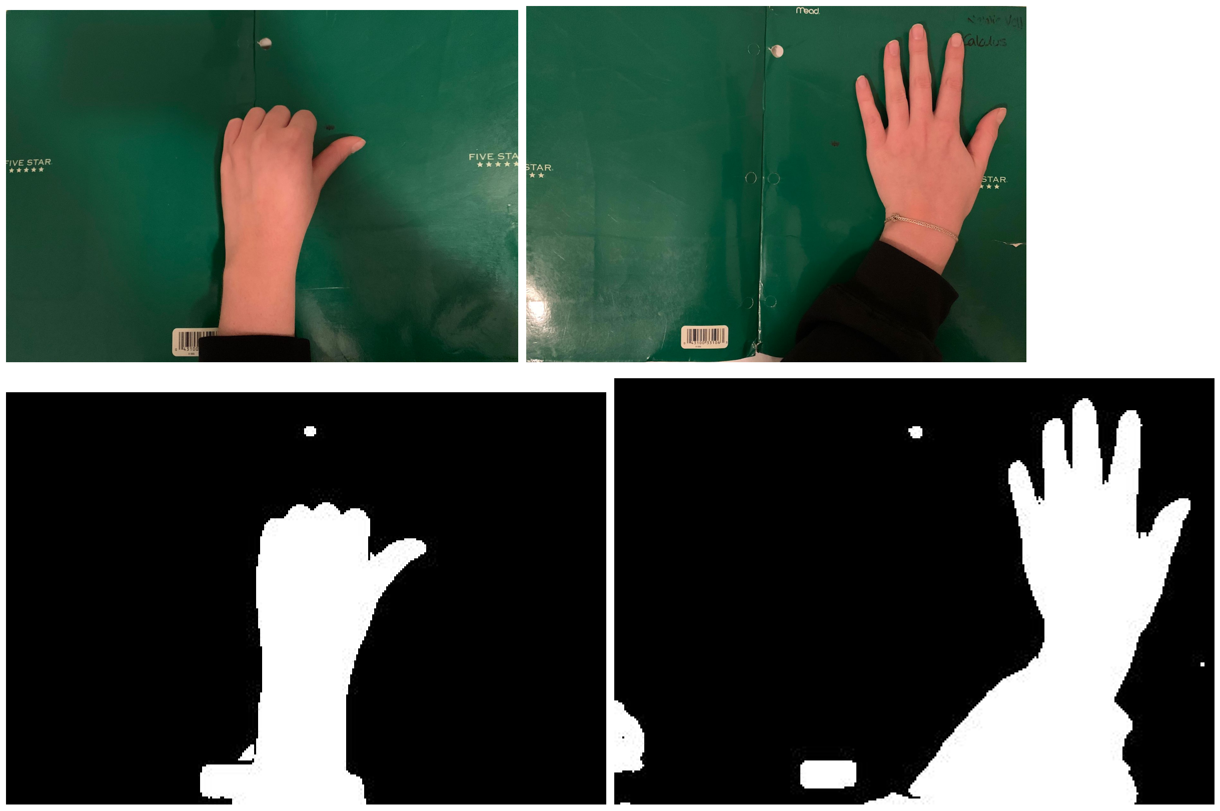
password = (("fist", "centre"), ("splay", "top-right"))

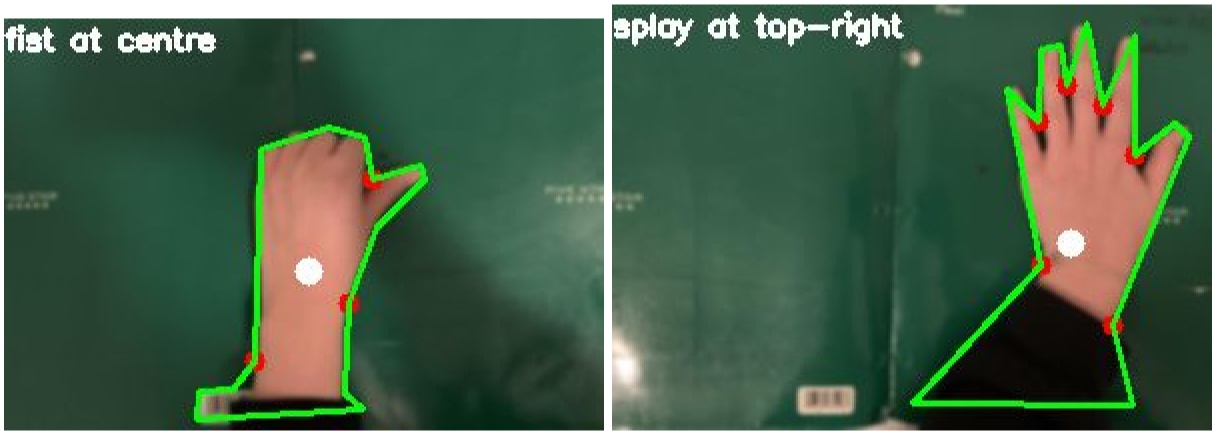
The grammar allows for two gesture-position vocabularies, and the sequence of the gesture-positions matter in order to unlock the password. Running the python file with the names of both file images tells the user whether they got secret code correct or not.

A. Successes

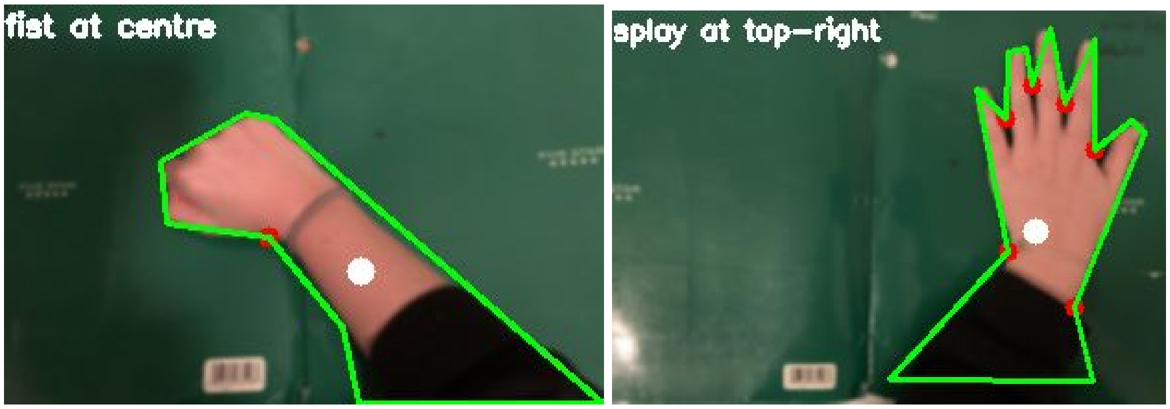
The program was very good at getting successes if the images were taken in a well lit area with an intensely green background. While thick sweaters did introduce less accuracy, they actually were not too much of an issue in detecting the gesture and location because I would remove the lowest contour points anyway in the program. This added more focus to the actual hand making the gesture rather than the lower area which has some objects (like clothing) that are not part of the hand.

1.

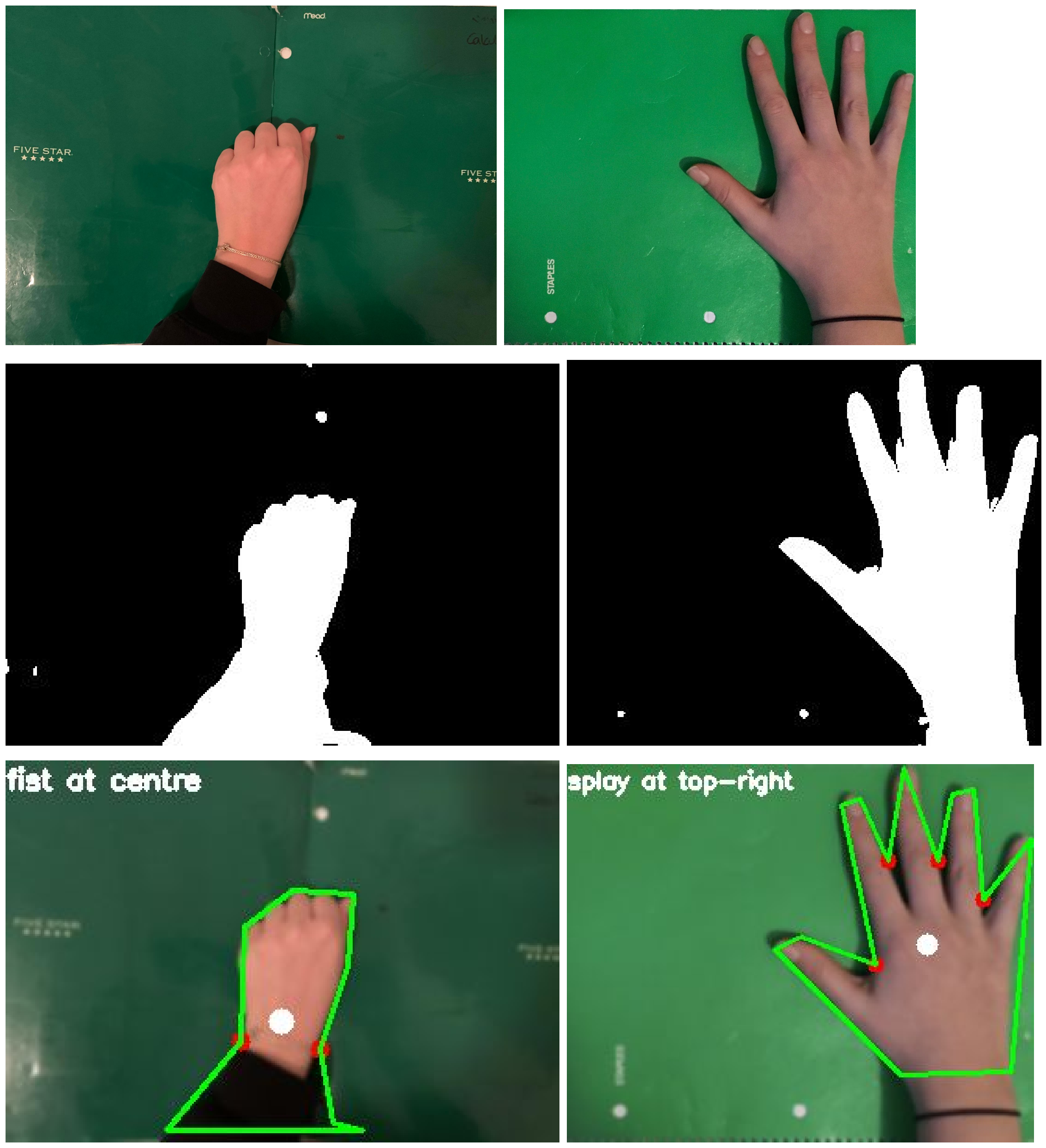



fist at centre
splay at top-right

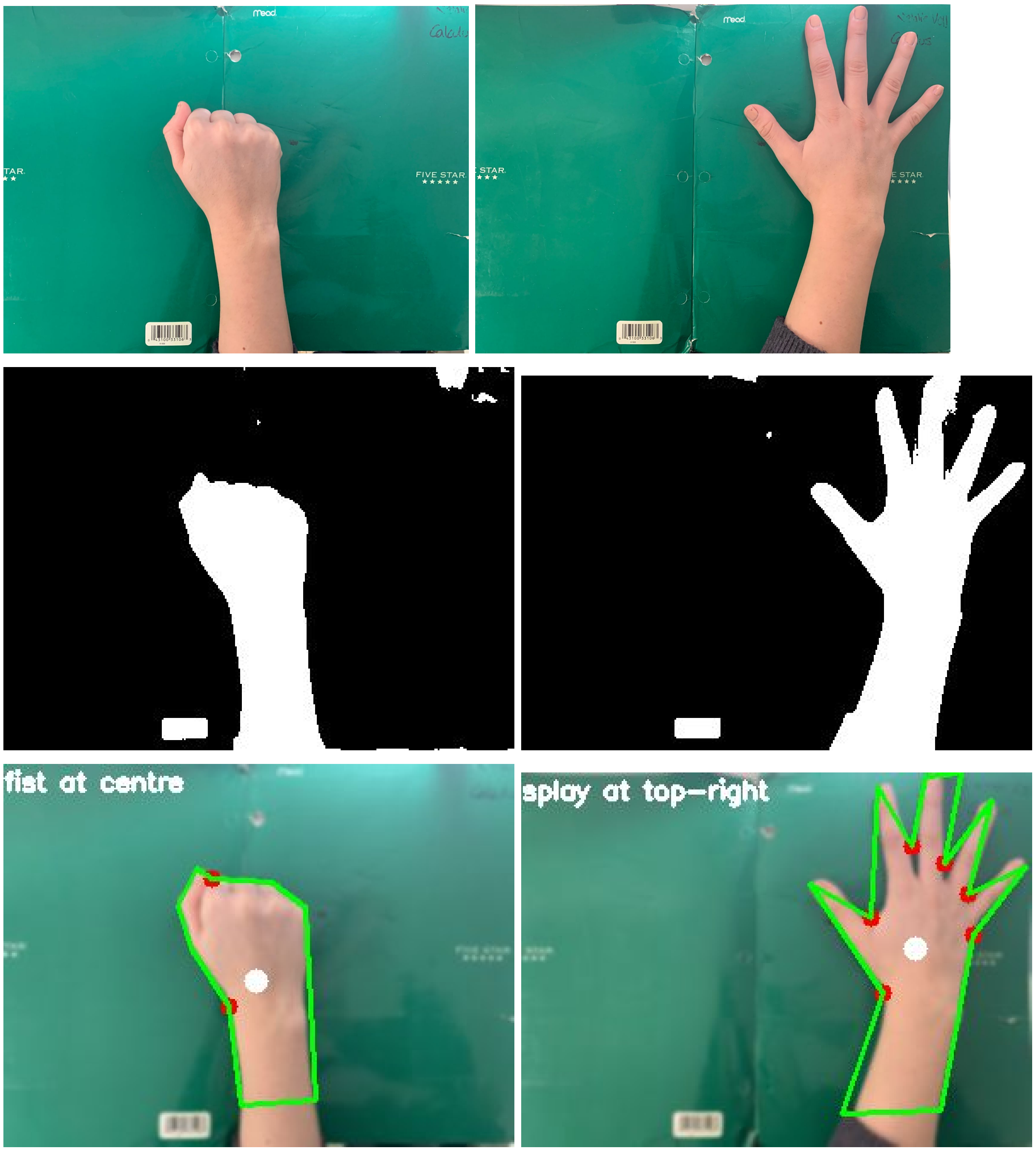
2.


fist at centre
splay at top-right

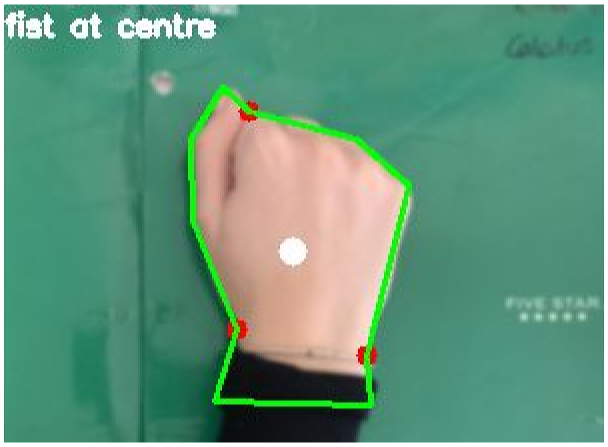
3.

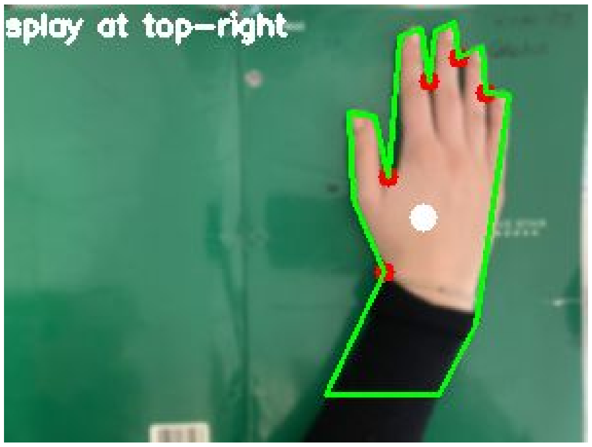


4.

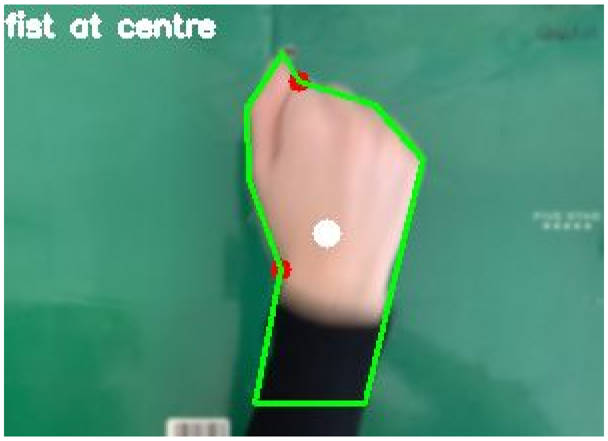


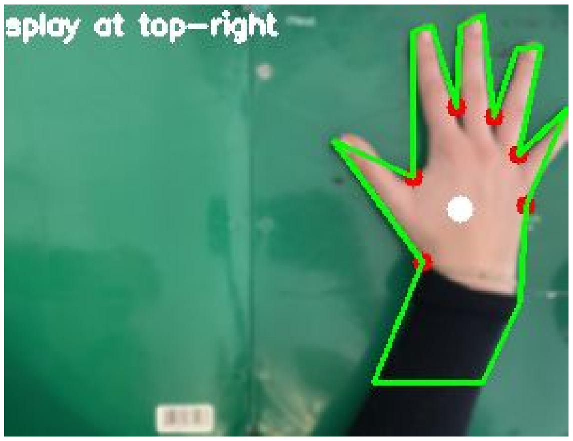
5.


fist at centre

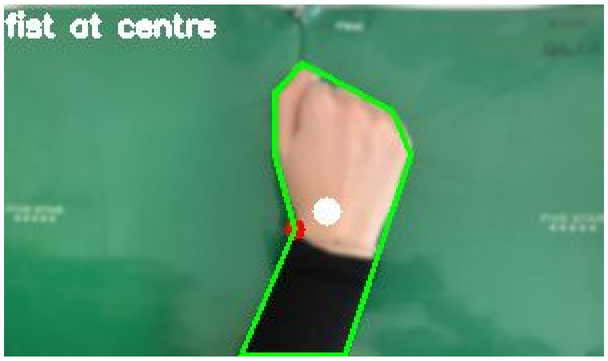

splay at top-right

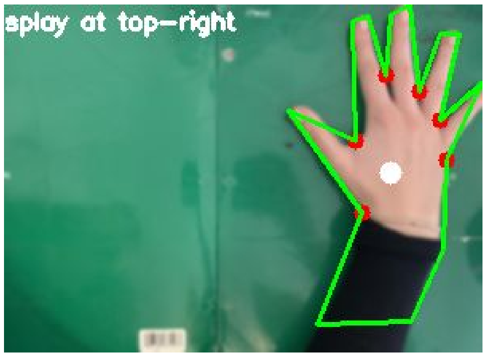
6.


fist at centre

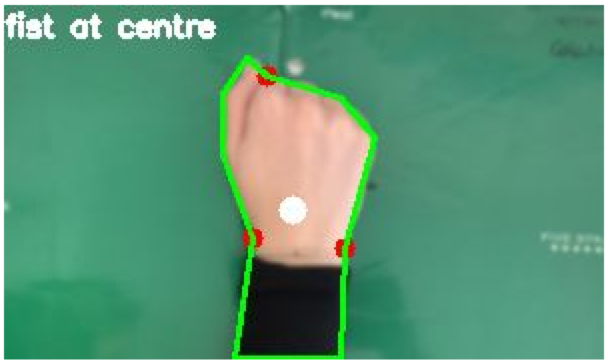

splay at top-right

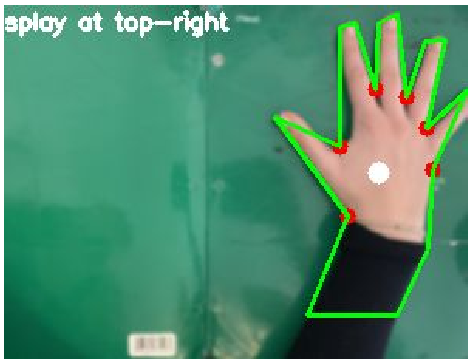
7.


fist at centre


splay at top-right

8.

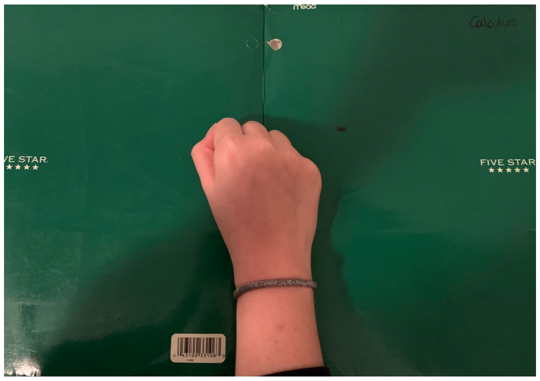


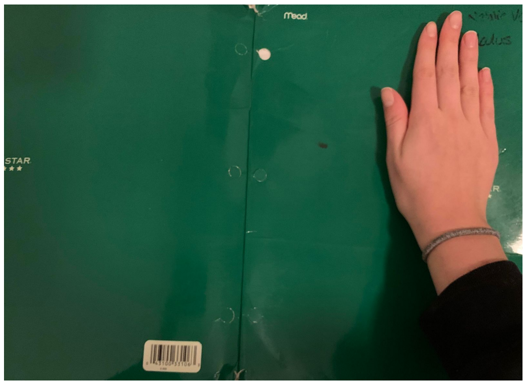


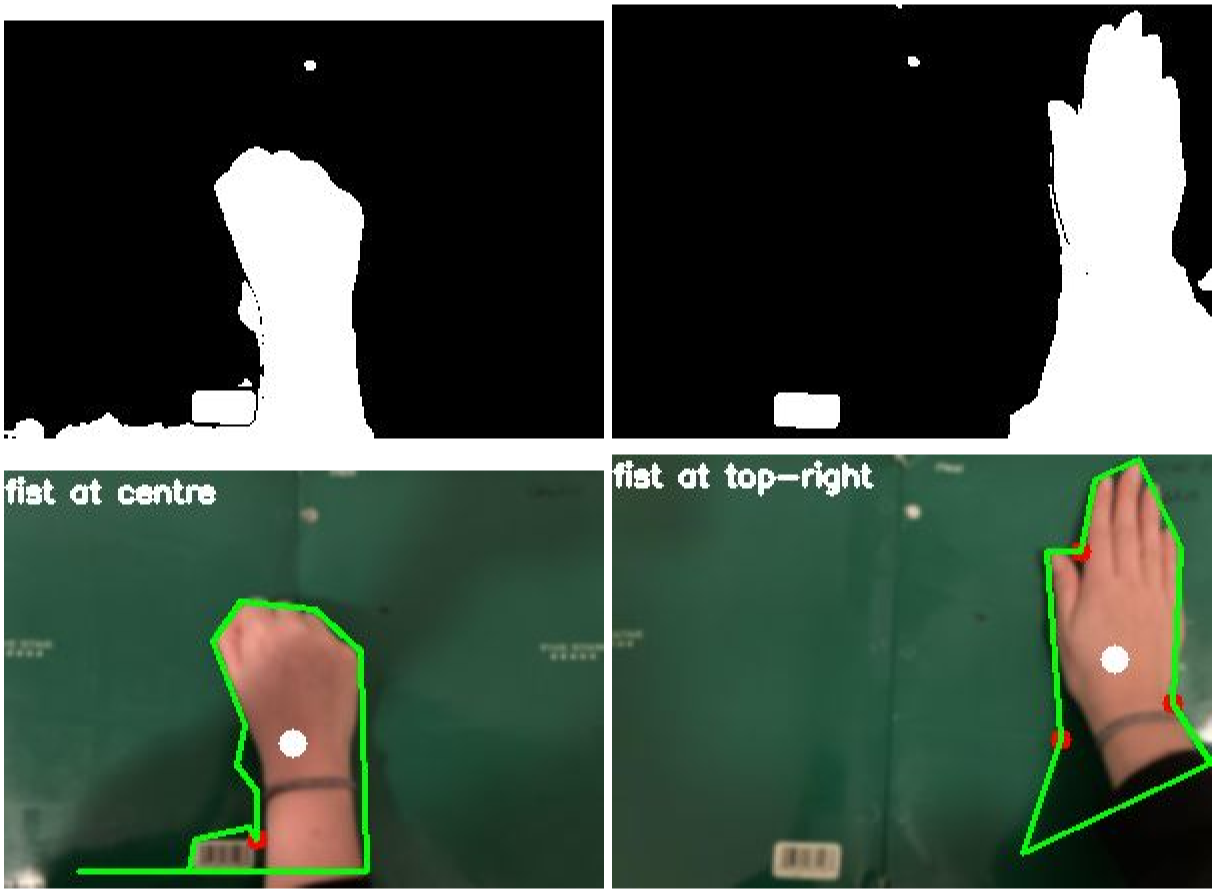
B. Failures

From the failures, I learned that the method I used to detect which defects count towards a splay can fail easily if the user does not splay their hand wide. In all the below examples, the fist was able to be detected, but the splay, if not spread out as wide, would be detected as another fist. Surprisingly, the intense shadows were not the problem here.

1.

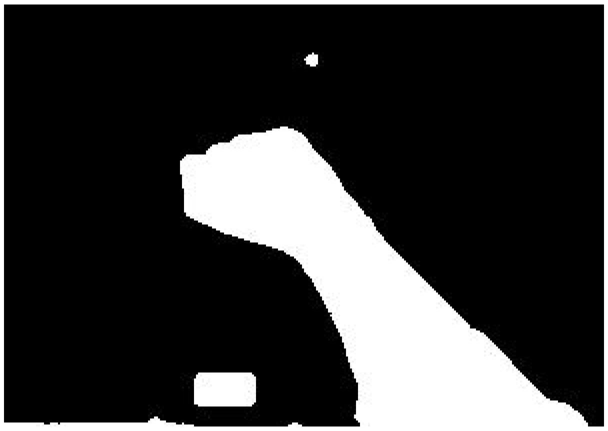


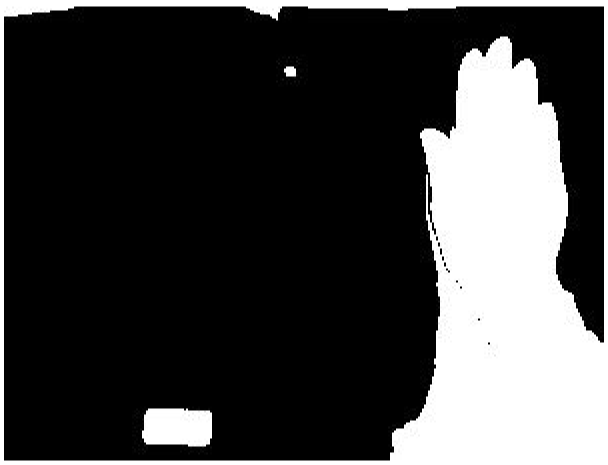


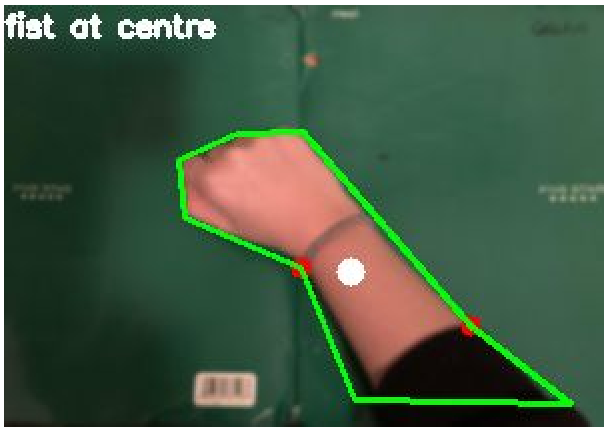

fist at centre
fist at top-right

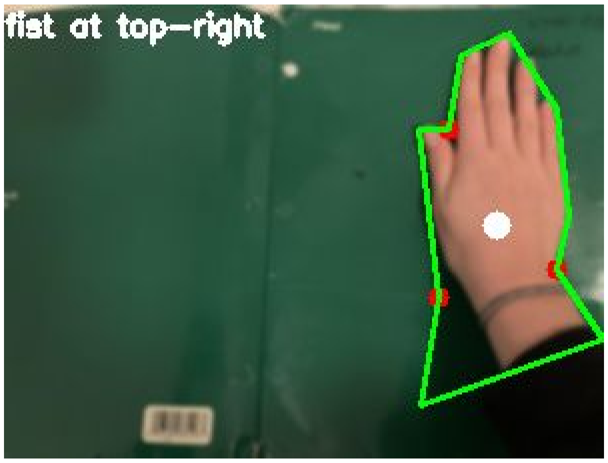
2.

Mead
Calculus
FIVE STAR
FIVE STAR

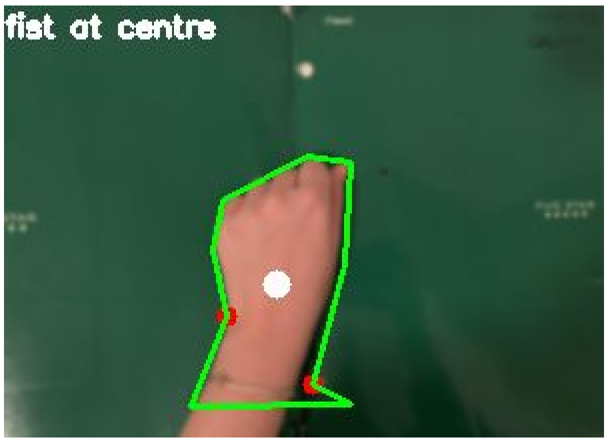


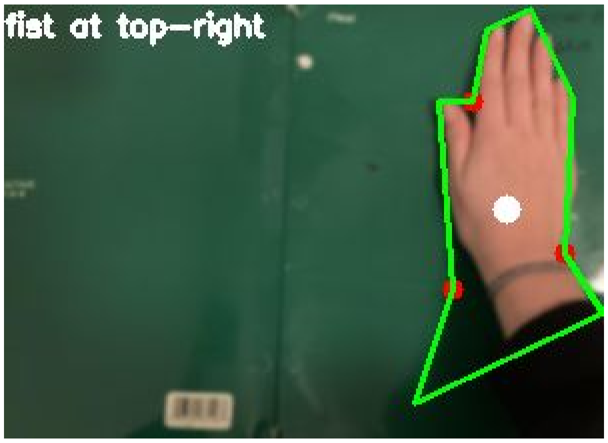



fist at centre

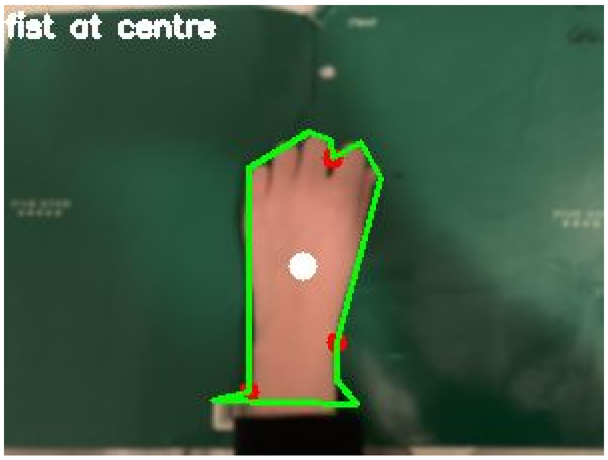

fist at top-right

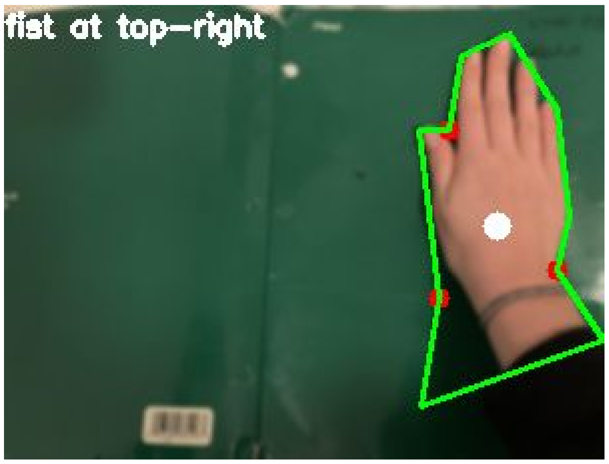
3.


fist at centre


fist at top-right

4.

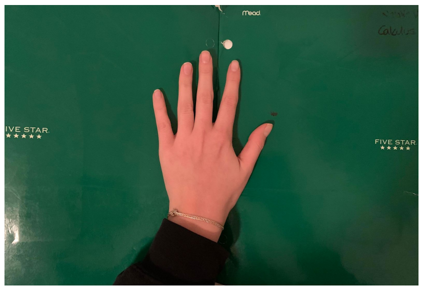


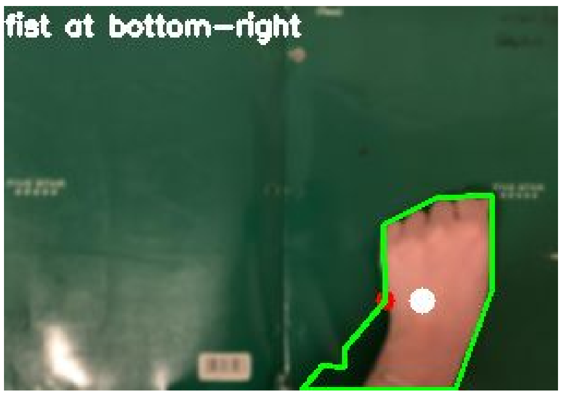


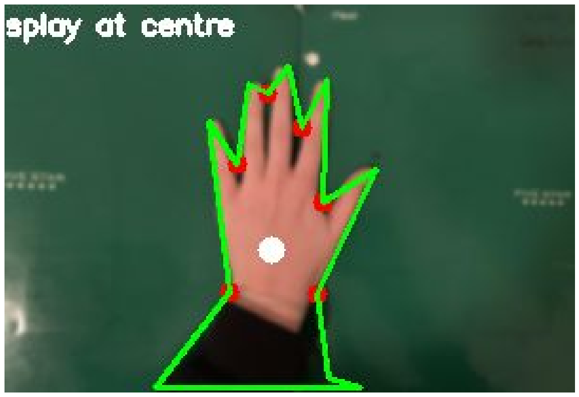
IV Data Reduction Enhancement

I extended the "where" to add relative data positioning. In order to do this, I rely on the moment point for the contour, and send that tuple to the main function of my program along with the gesture and position. The moment point helps identify where the hand is centred on the image. This data enhancement is enabled because I extended my vocabulary of the where to have centre and all four corners instead of those where positions needed only for the secret code. It would have been more enough to record a "centre" where and a "top right corner" where in order to determine if the secret code is unlocked. However, by using x and y coordinates, I can figure out relative positioning.

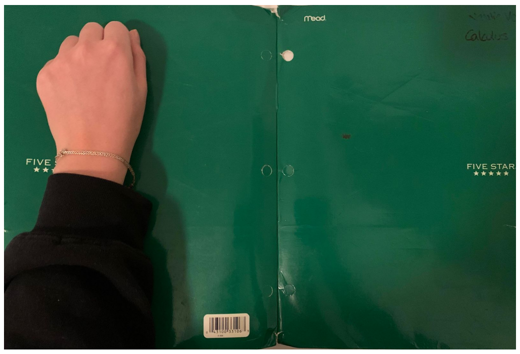
The accuracy is augmented through the contour method that attempts to remove the arm and only keep the hand based on where the extreme-north point is in the contour. However, there is error when the user does not put their hand in straight from the bottom with their fingers facing North, as this moves the moment point to the right or left and becomes more inaccurate. However, if the user follows the instructions and does not change the setup within sequences, the accuracy is quite high. Below are some examples of how the software understands relative positioning:

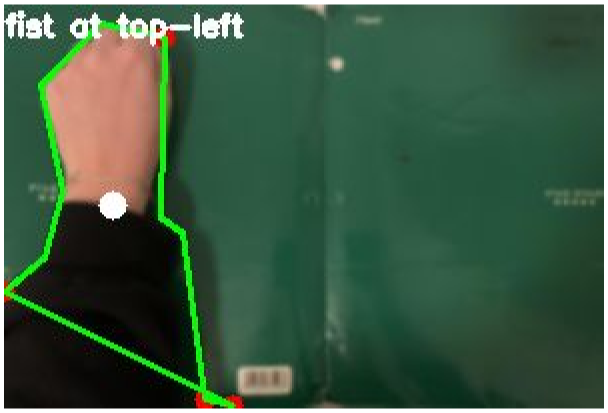


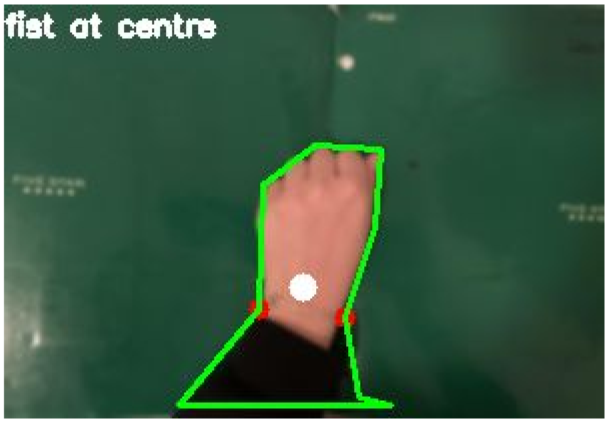




```
The second gesture is further left and further up.
Incorrect password.
```







```
The second gesture is further right and further down.
Incorrect password.
```