Class: CS5330 - Computer Vision
Author: Rishi Patel
Due 02/21/2025

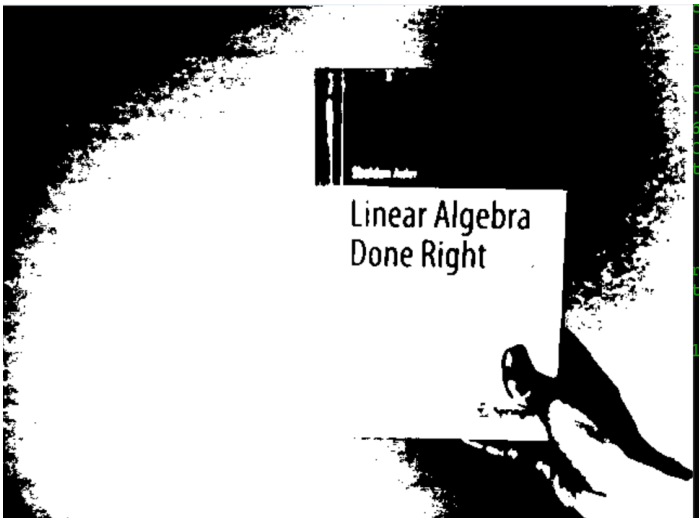
Project 3: Real-time 2-D Object Recognition

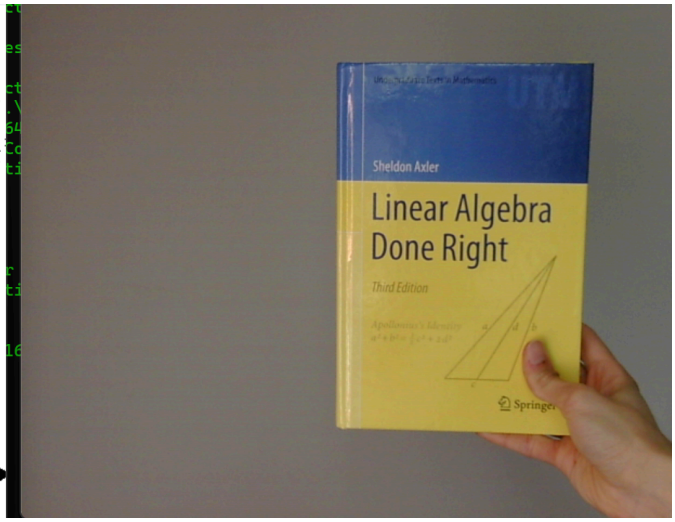
Project Description: Real-time 2-D Object Recognition in C++, The purpose of this project create a real-time object recognition program. This program is to be able to train objects on certain features, and through those features, be able to identify an object. This was done by thresholding the input video, cleaning up the binary image, segmenting the image into regions, computing the features, collecting the data, then classifying the image. At the end the performance was also evaluated through a confusion matrix and

Required Result 1: Threshold the input video

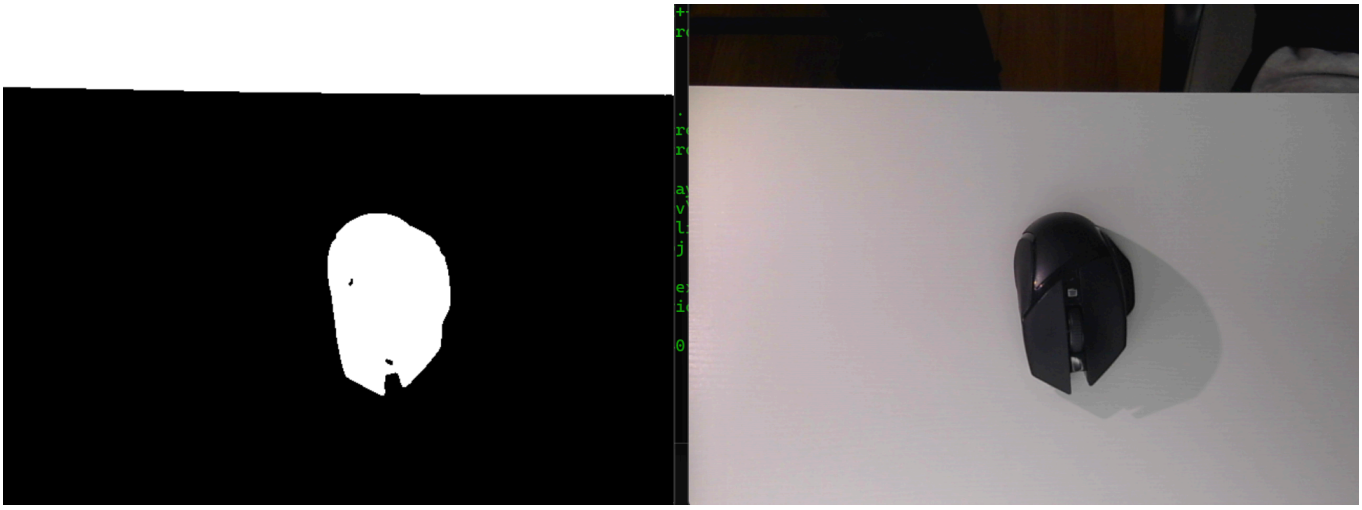
For preprocessing, I used gaussian blur from Project 1 (2 1D Matrix), turned to HSV, converted to grayscale, and adjusted grayscale value based on saturation.

Both below are not of the required images but thought I would add to the report in my initial testing, was trying to use a gray background but when the sun came down. When I used a white calculator, the colors inverted because my calculator

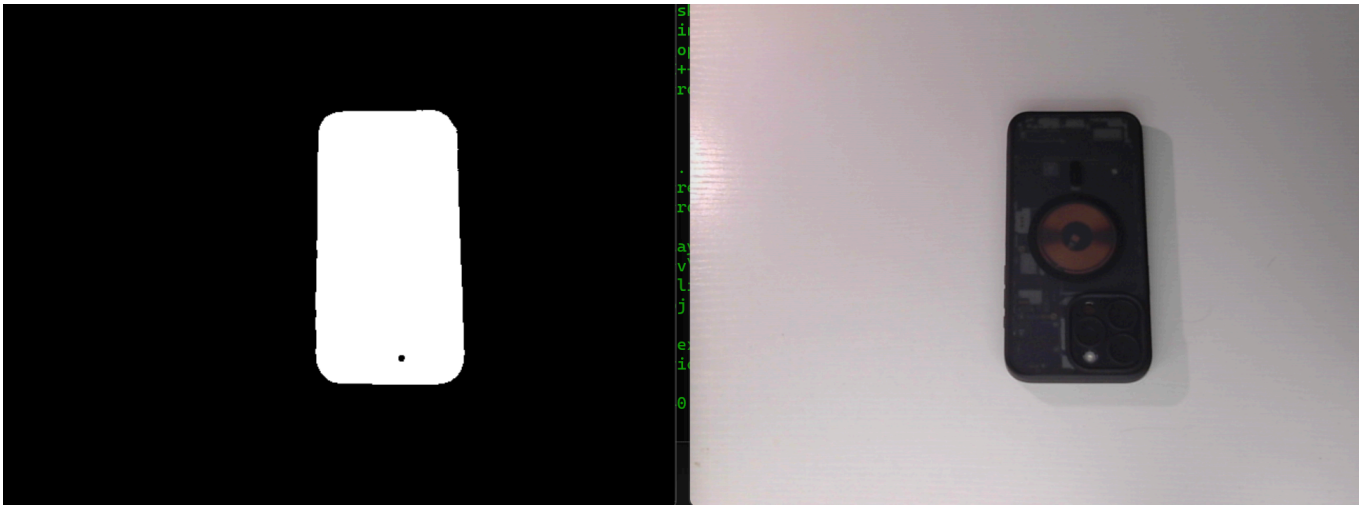




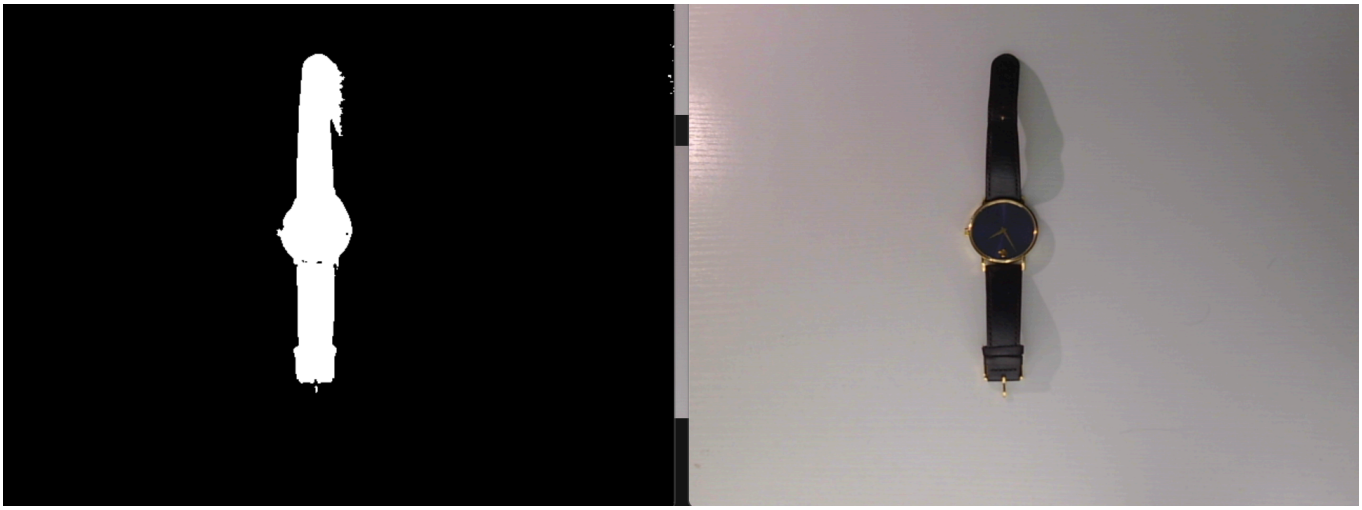
Realizing I should probably get rid of the grey background before it causes me more of a headache, I changed the camera to my desk to now get better results with a pure white background
Mouse:



Phone:

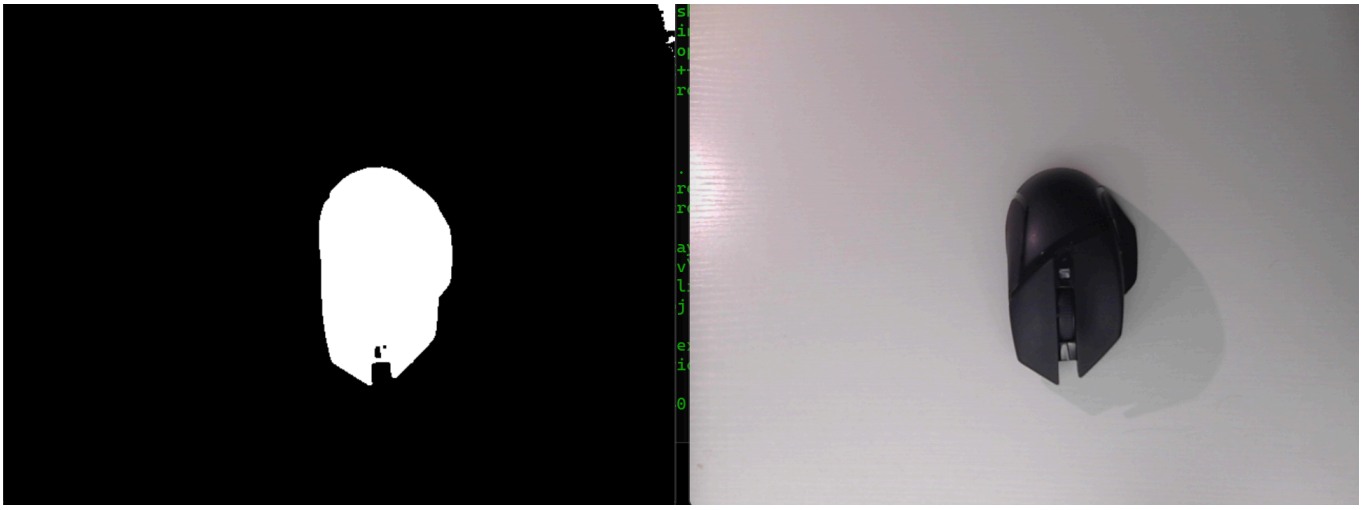


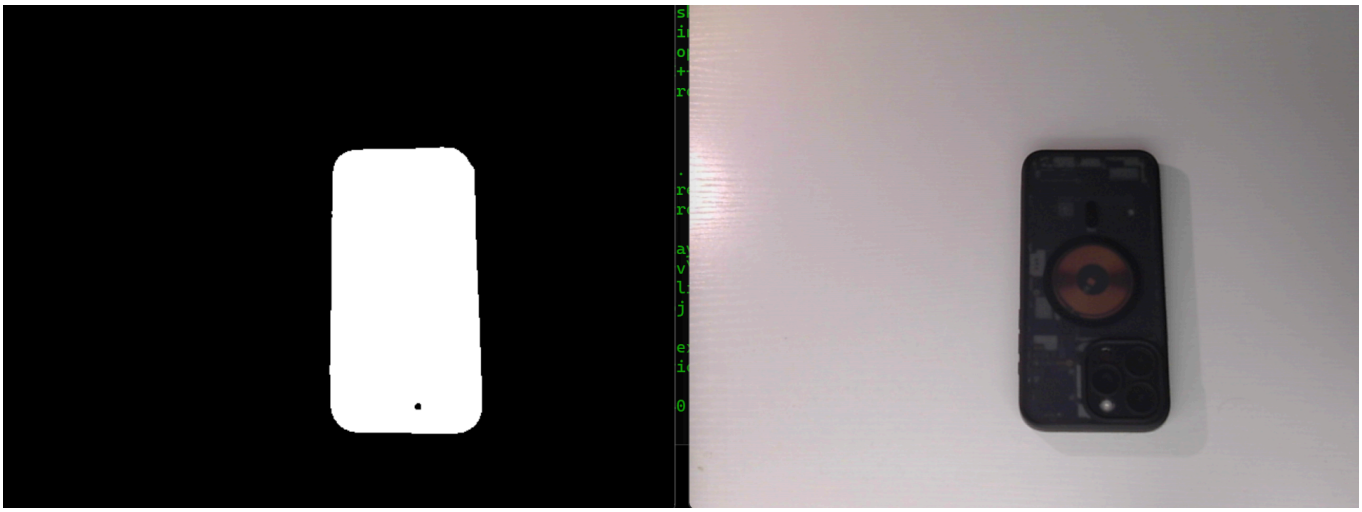
Watch:

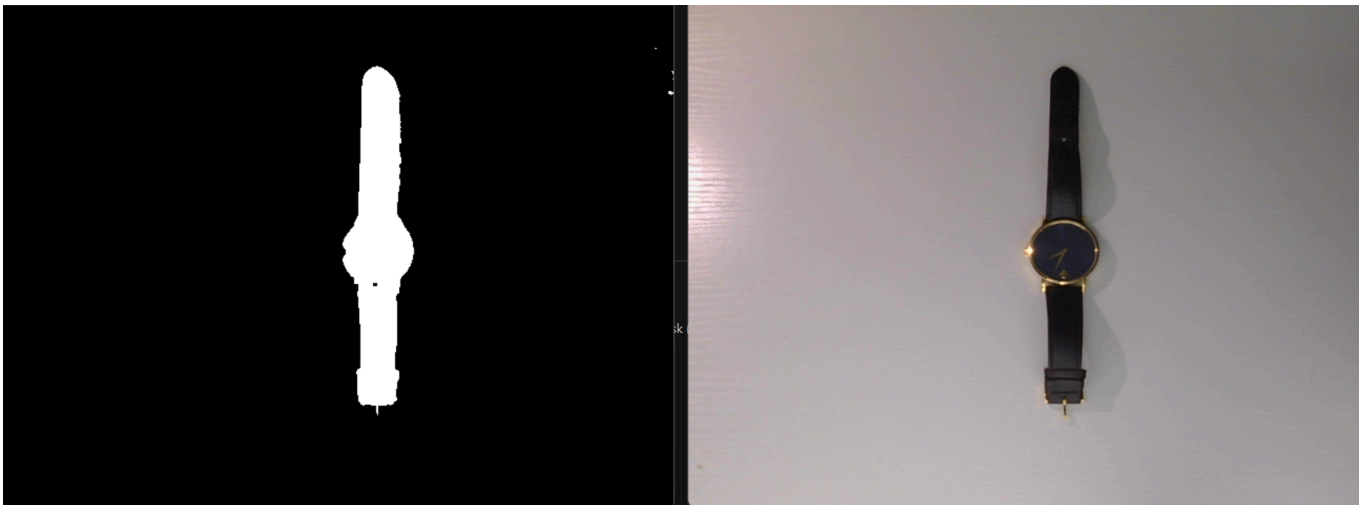


Required Result 2: Below pics are not required ones, just thought they would be good for the report. I tried a kernel size of 3, 5 and 7, below are 3 and the required result: kernel size 7

Kernel Size of 3:

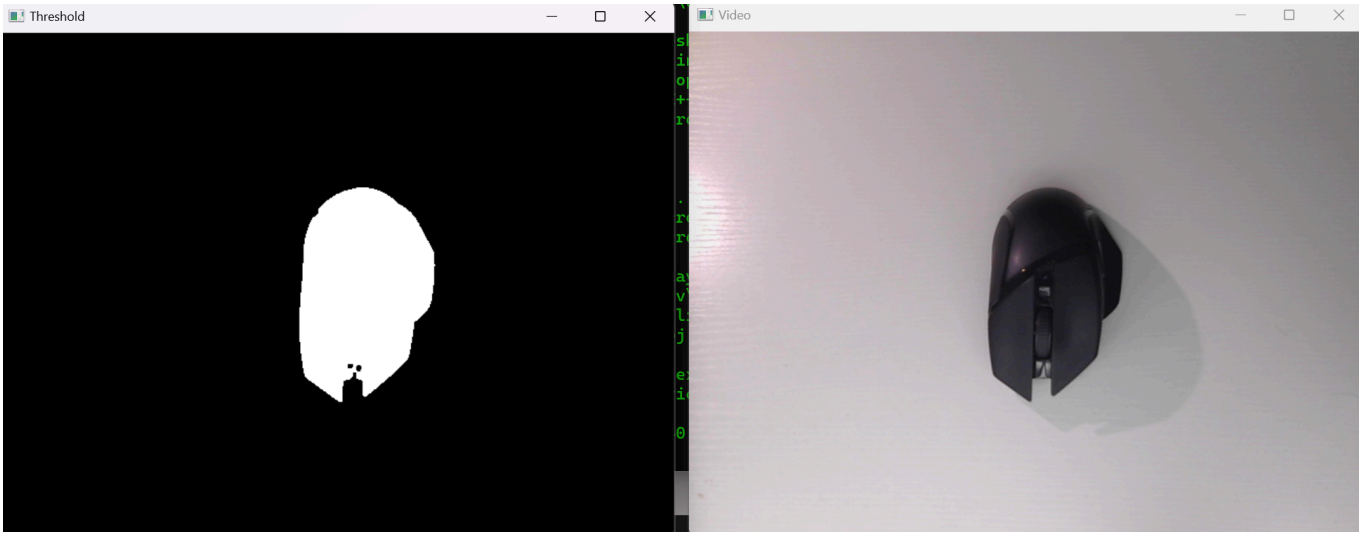


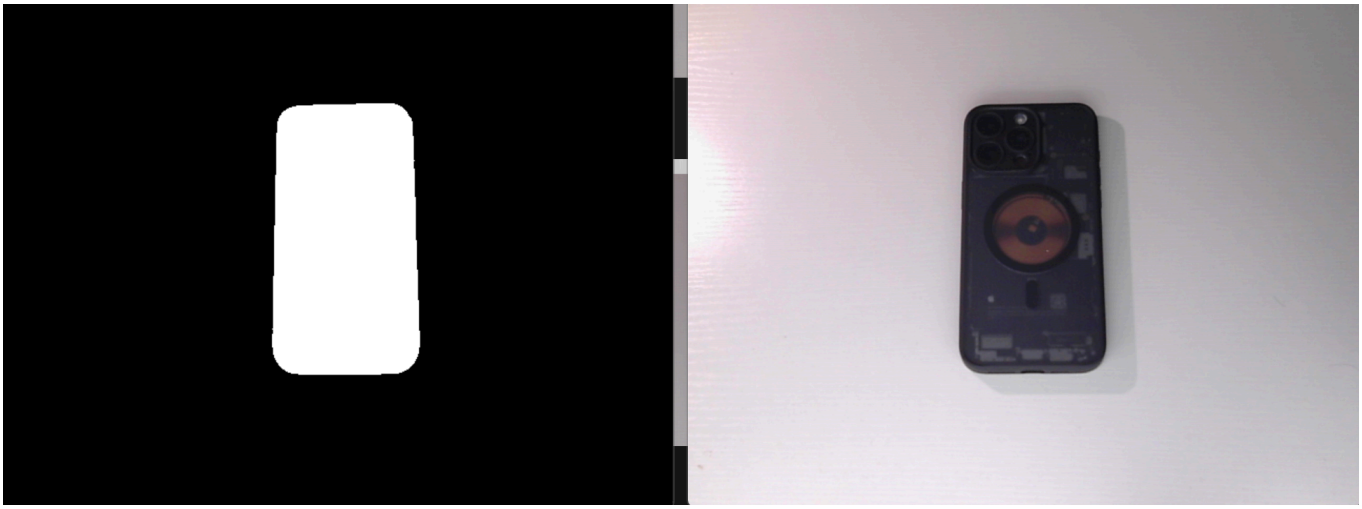


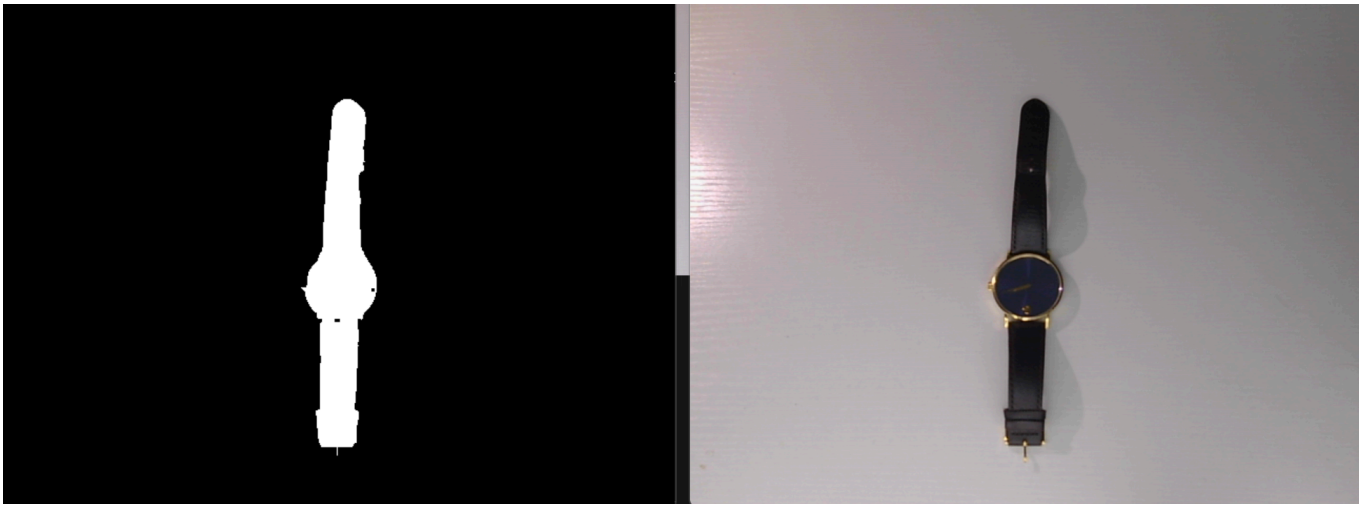


I found Kernel Size of 7 to be the best:

Required Result 2: Clean up the binary image



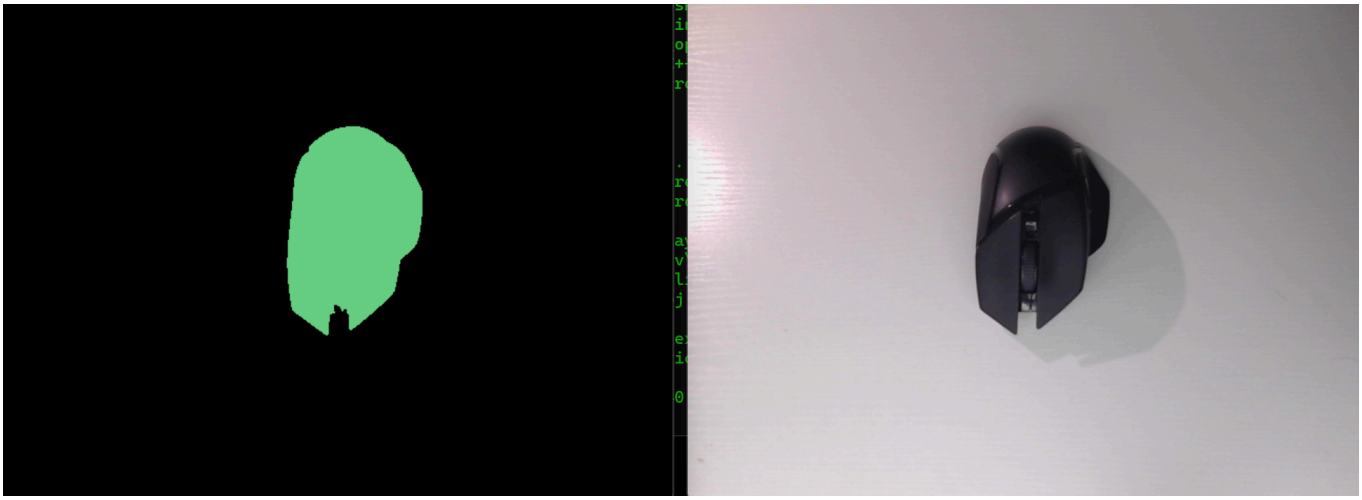


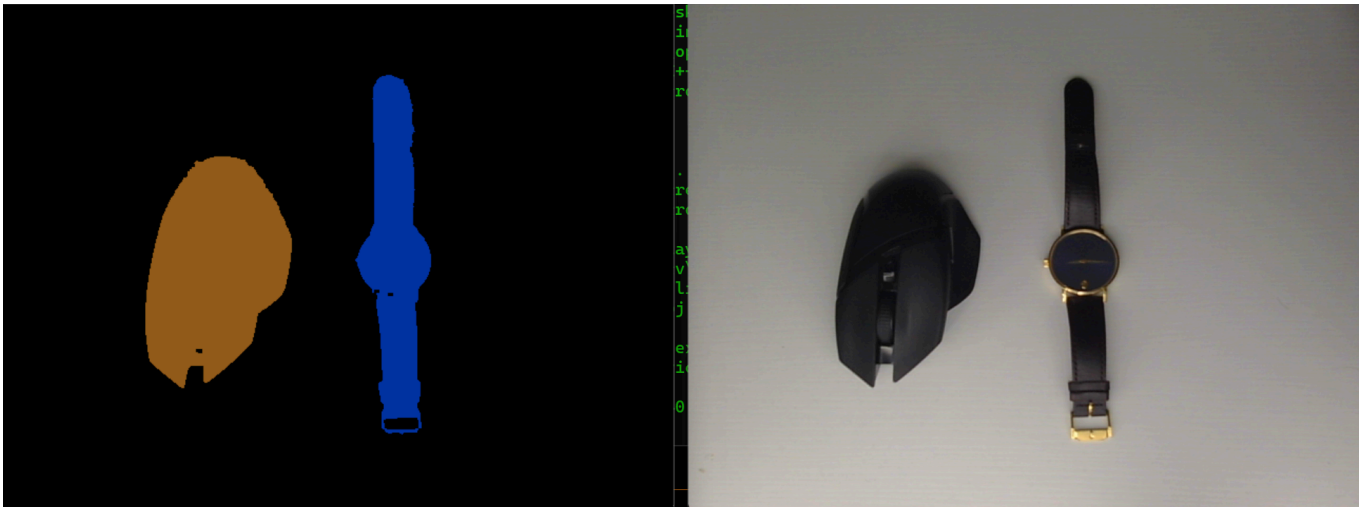


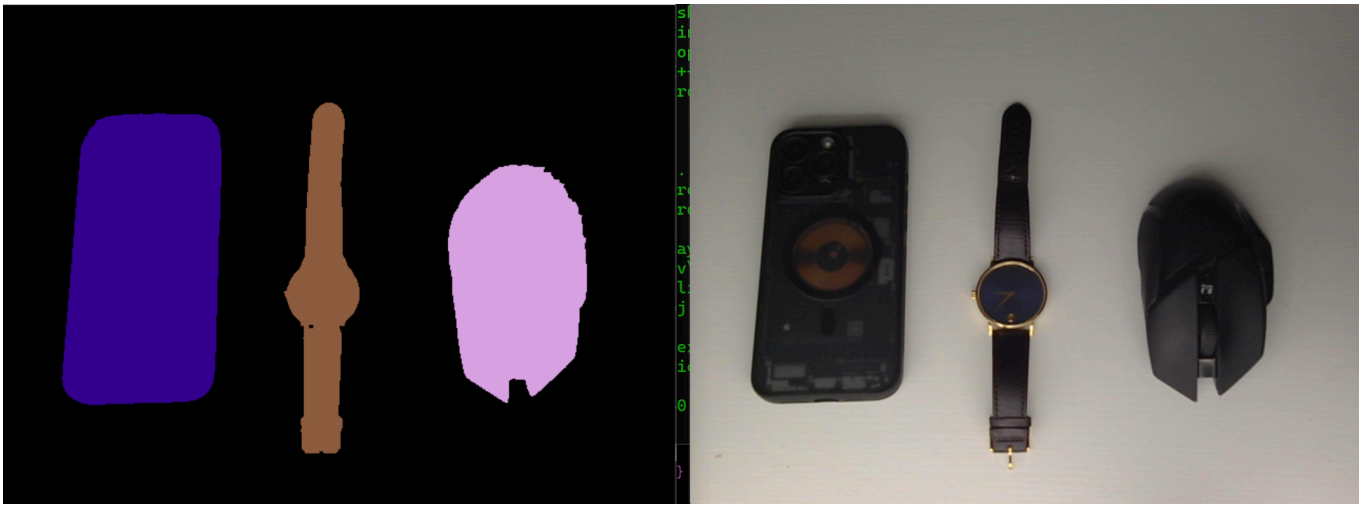
Because there wasn't much noise and really just some small holes that need to be filled, I decided to go with closing (dilating -> eroding) with a kernel size of 3 originally and moved it up to 7. I was mostly focused on closing up the hole in the iPhone camera lens which a kernel size of 7 was able to do. However I also had simple erosion, dilation, and opening cases in my morphology function in order to be able to switch to it easily and to play around with it.

Required Result 3: Find connected components

I used different seeds and techniques for each three pics but all in all they are finally tracked





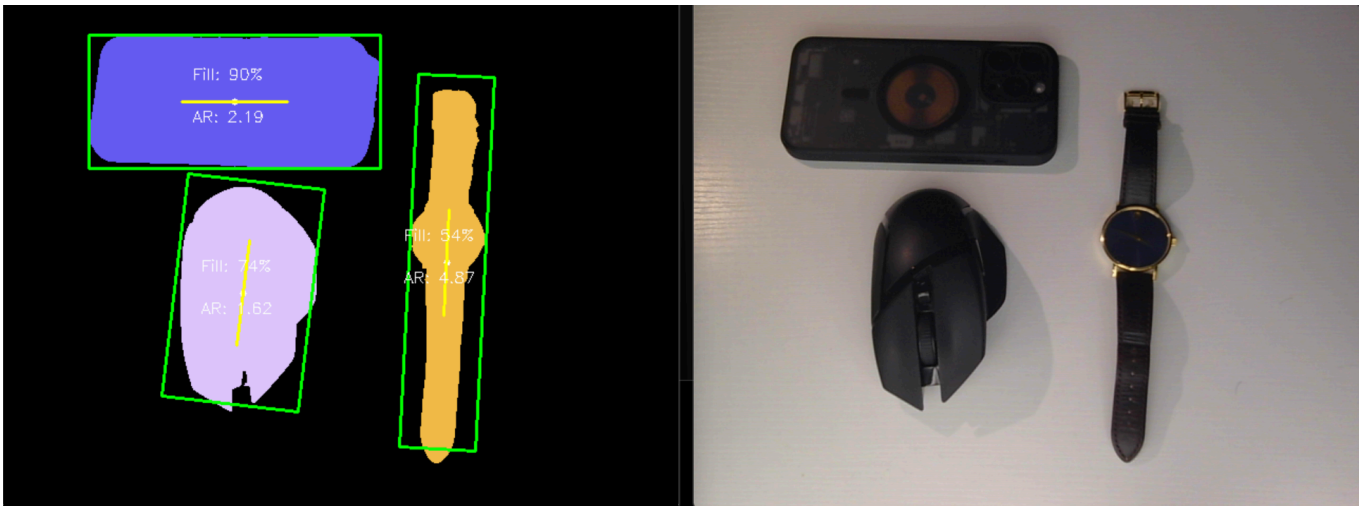


So I implemented region-based analysis rather than boundary based analysis.

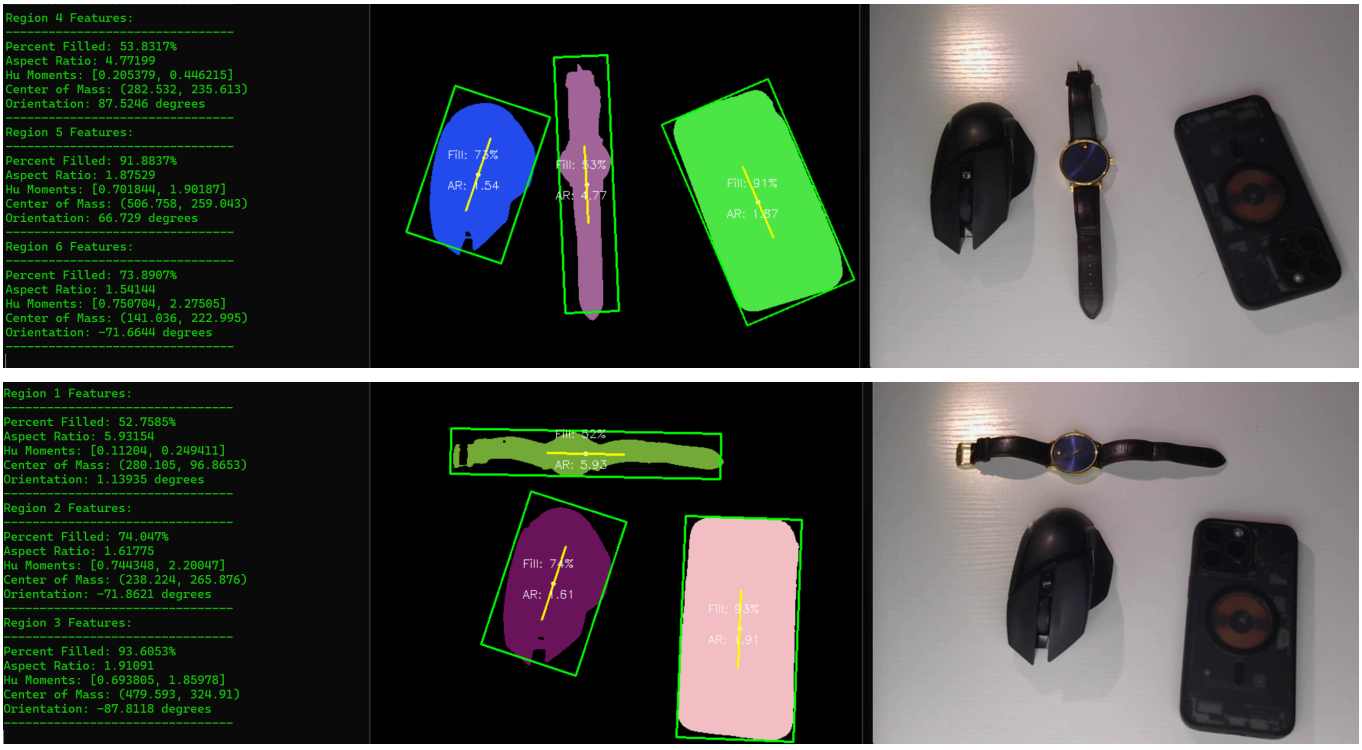
OpenCV moments calculated spatial moments with $m00$: the area of the region ; $m10, m01$: For Central moments: First order moments for centroid $mu20, mu11, mu02$: second order central moments (pixel distribution); Normalized Central moments $nu, nu11, nu02$: Central moments

normalized by area; and finally Hu Moments. I didnt personally use the normalized central moments but they are used in the Hu moments. Overall Spatial moments are used for area and centroid, central moments are used for orientation, and Hu moments as shape descriptors. The different features I calculated ended up being precent filled, aspect ratio, Hu moments, Center of mass, and orientation angle. All three can be seen in the given picture

Without features tab



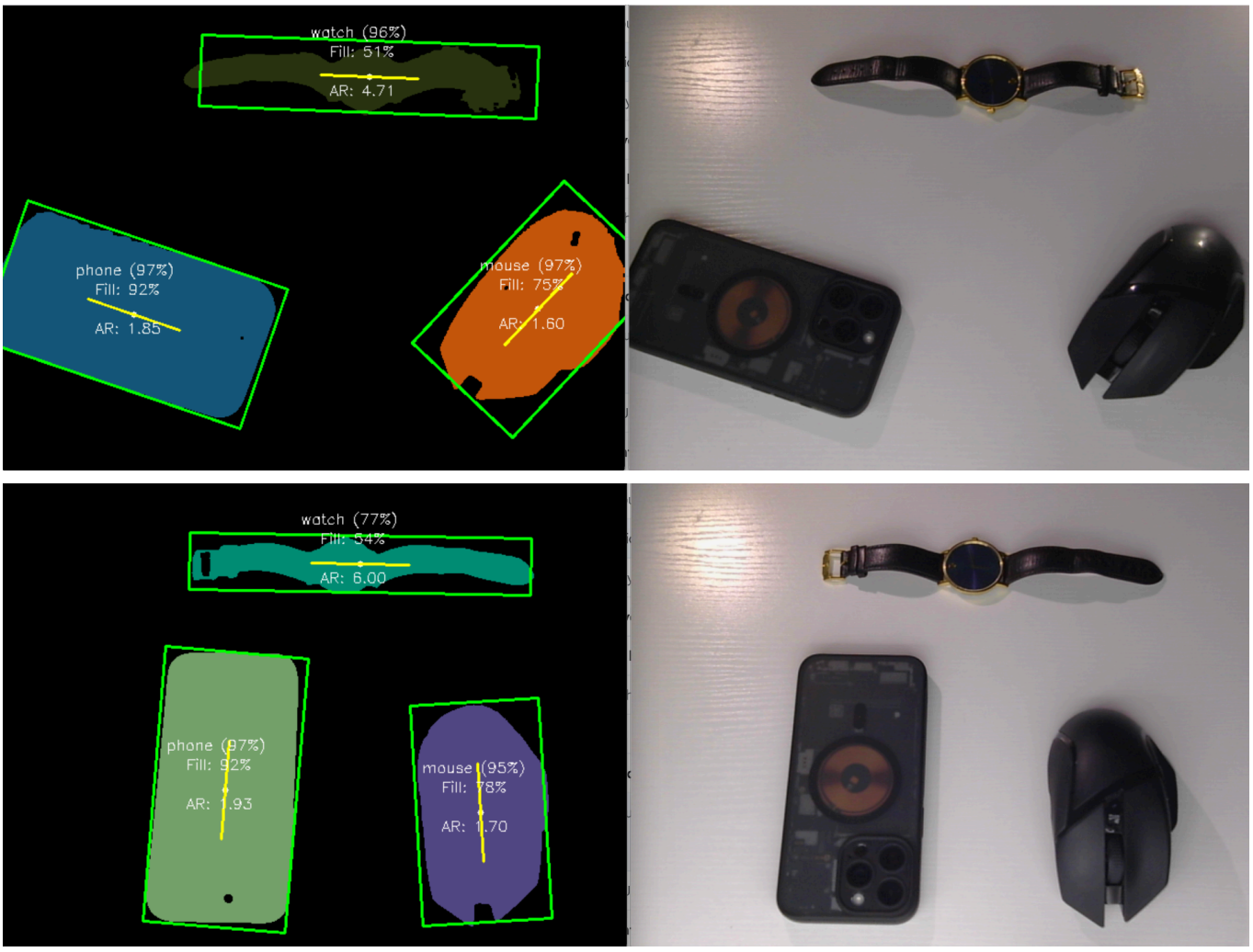
With Features:



How my Training works:|

My training logic just to simply it relies on one image being in the frame at a single time. This works for me because noise is very rare in my setup. Then when N is clicked, I simply save the features I want (precent filled, aspect ratio, Hu1&2) with a label too the object_features.csv

Now with Objects saved, I was able to classify these objects! Pretty neat:
As for the distance metric, i simply used scaled Euclidean distance where each feature difference is divided by the std of that feature.



With more images trained

Confusion Matrix:

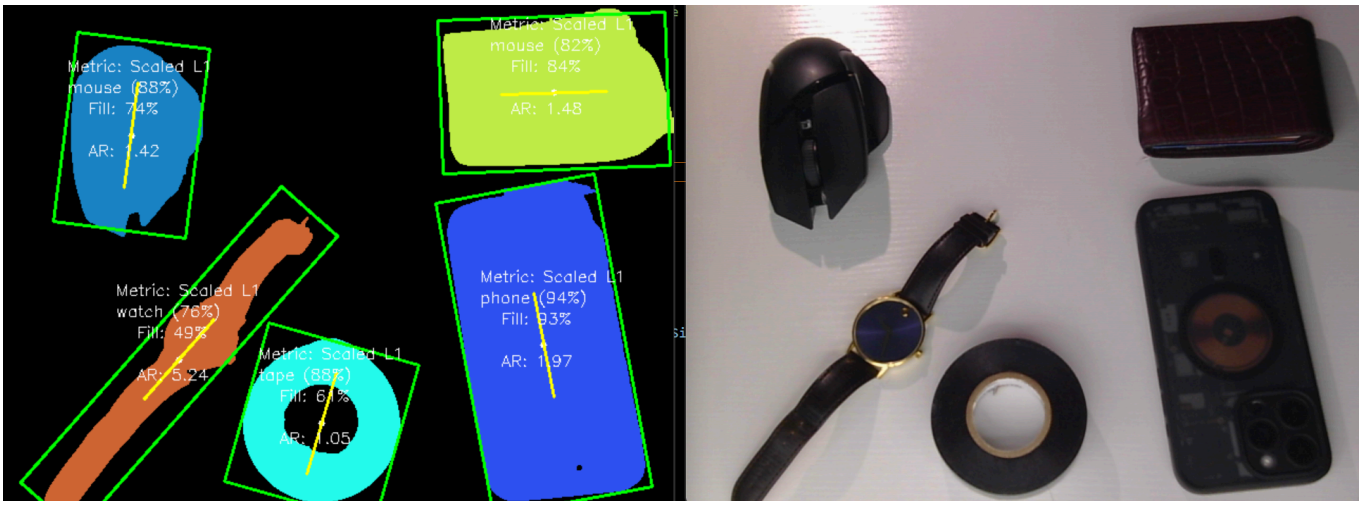
Required: Confusion Matrix Output

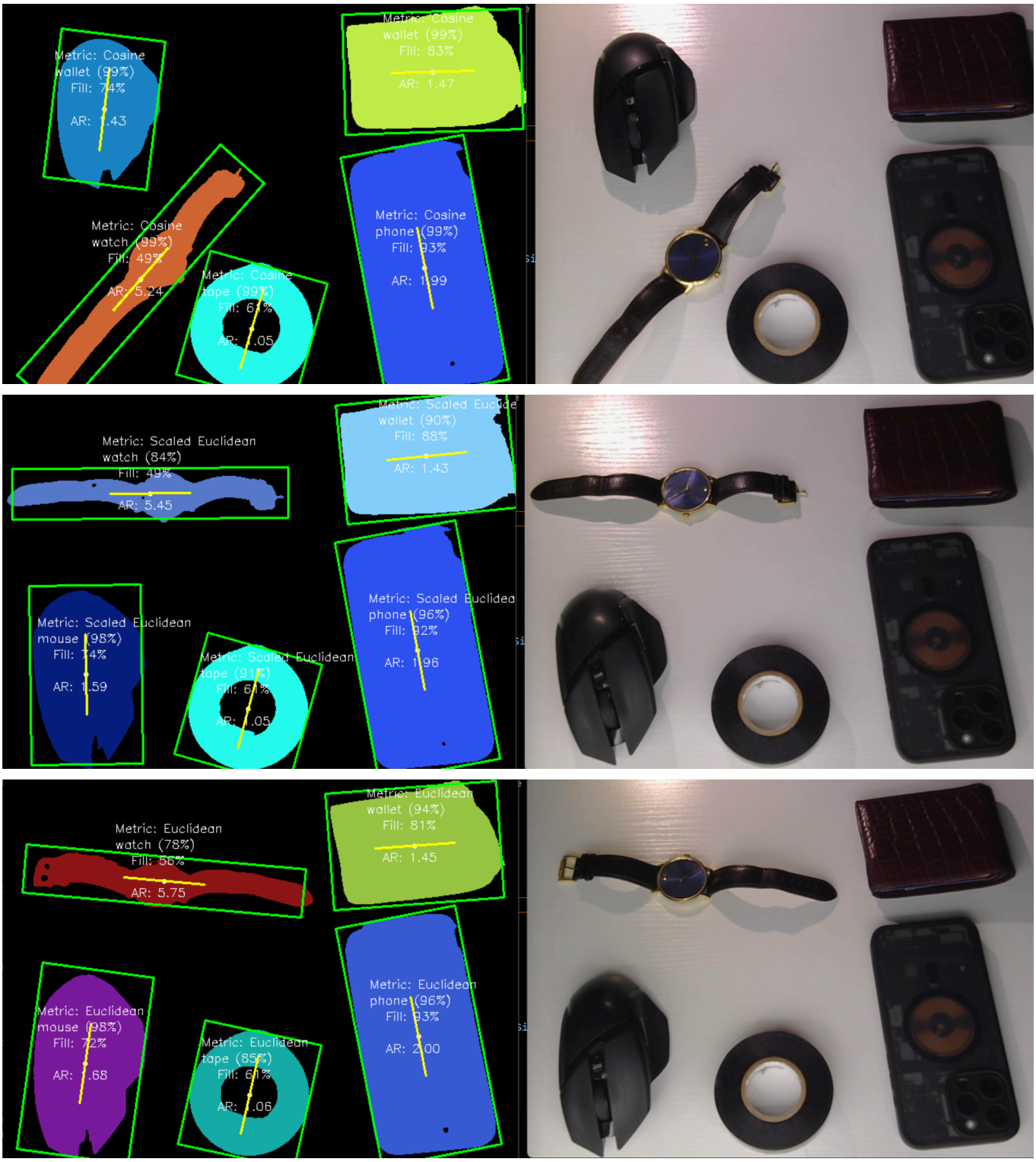
mouse	3	0	0	0	0
phone	0	3	0	0	0
watch	0	1	2	0	0
wallet	0	0	0	3	0
tape	1	0	0	0	2

Video Demo:
https://youtu.be/UCCIHovymdE

Question 9:
So I went with: Use NN matching, but compare different distance metrics. For example, compare simple Euclidean, scaled Euclidean, cosine distance, scaled L-1. I implemented this by making it so that tapping "d" would cycle through the different distances, so I recorded the results for each of the objects. Below are pictures of the results.

So very interesting results when we compare the results of the different distance measurements. Cosine distance has highest confidence but makes incorrect classifications, scaled Euclidean is a balance of confidence and accuracy, and Euclidean and scaled L1 don't do terribly as well, but they just aren't as confidence as scaled Euclidean. Overall Scaled Euclidean seems like the way to go as it offers the best "balance"





Reflection:

Overall, this was a fascinating project that highlights how straightforward object detection can be. Initially, I expected challenges in differentiating between the wallet and phone, but their distinct aspect ratios made the task easier than anticipated. Additionally, the wallet's unusual fill percentage further helped in distinguishing it. I opted for a simpler approach by using only four features while ensuring they remained rotation-invariant. Also it was pretty evident that Scaled Euclidean is the best balanced distance measurement to use. Also figured out just how

important lighting is this project and how shadows can really affect your results. Sometimes changing the lighting would result in misclassification. All in all, fun project.

AI Use: I used AI to create a nice looking README again, saved a lot of time. also used it to help with what different functions do after reading documentation. Also used it for making the Doc Comments easily as I tend to just start writing without comments, and remove useless inline instantly. Also helped me understand results when i was confused.