CS5330 Project 3

Real-time 2-D Object Recognition

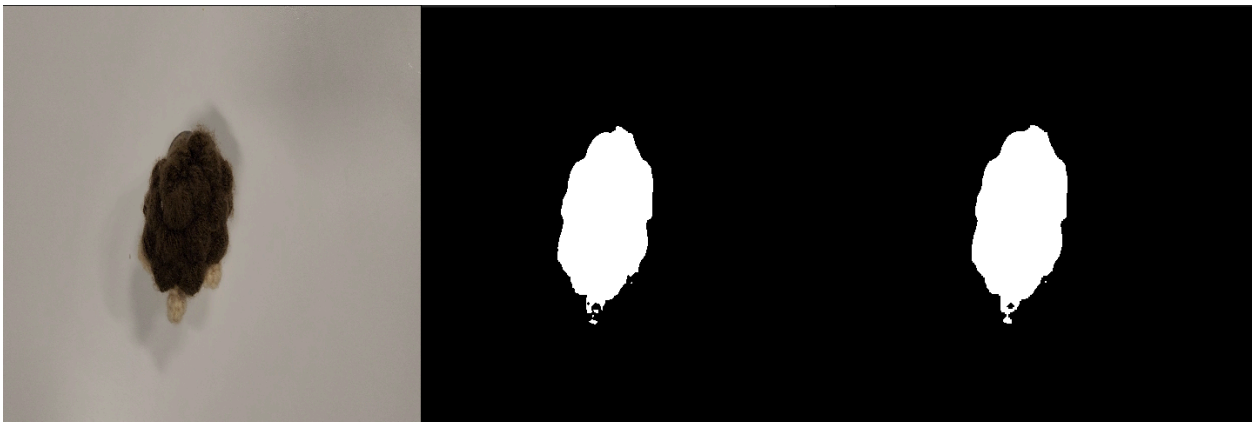
Students: Ronak Bhanushali; Ruohe Zhou

1. Description

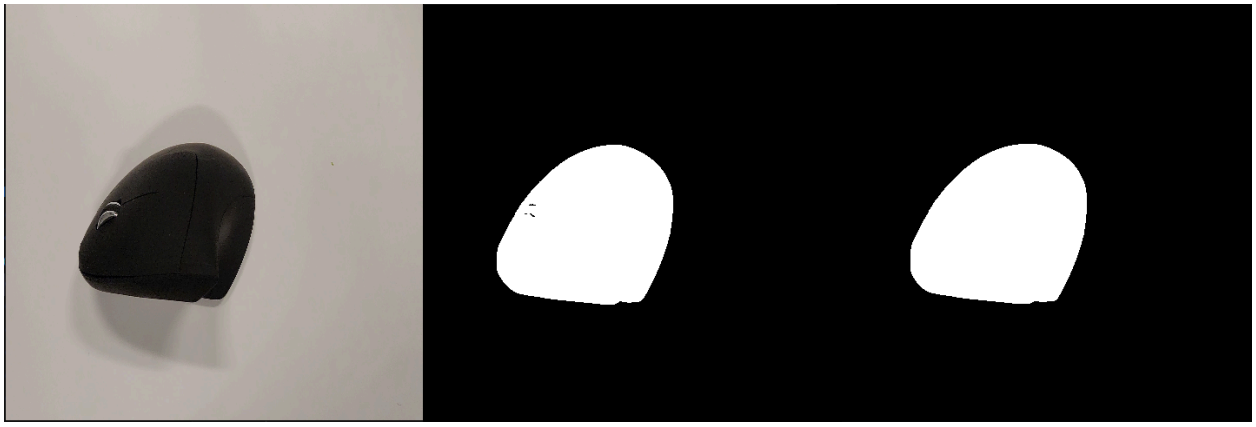
This project is focusing on applying computer vision techniques, including thresholding, morphological filtering, and segmenting video images to create an object recognition system. Features such as image moments, which are statistical properties of an image's shape distribution of intensities (area, centroid, orientation) as well as region filled were extracted to form a database for the classification of objects. This project focused on separating objects from background, segmenting and finding the regions that belong to those respective objects, extracting features that are unique to objects to build a database and finally comparing unknown objects with the database to classify the objects using nearest neighbour and scaled euclidean distance. This project also introduced us to using deep neural networks for extracting features

2. Required Images

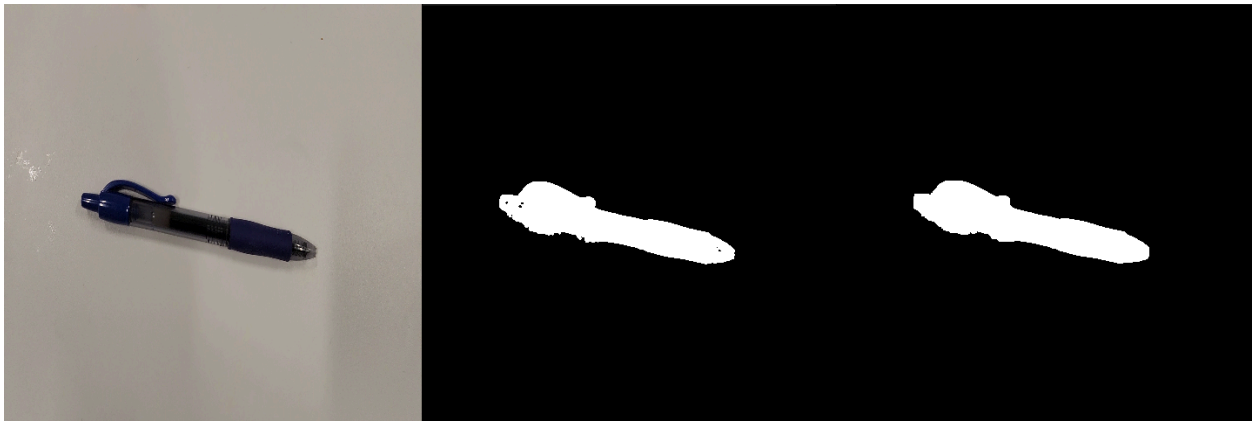
(1) Threshold Image and (2) Clean Image



Original **Threshold image** **Cleaned Image**



Original **Threshold image** **Cleaned Image**



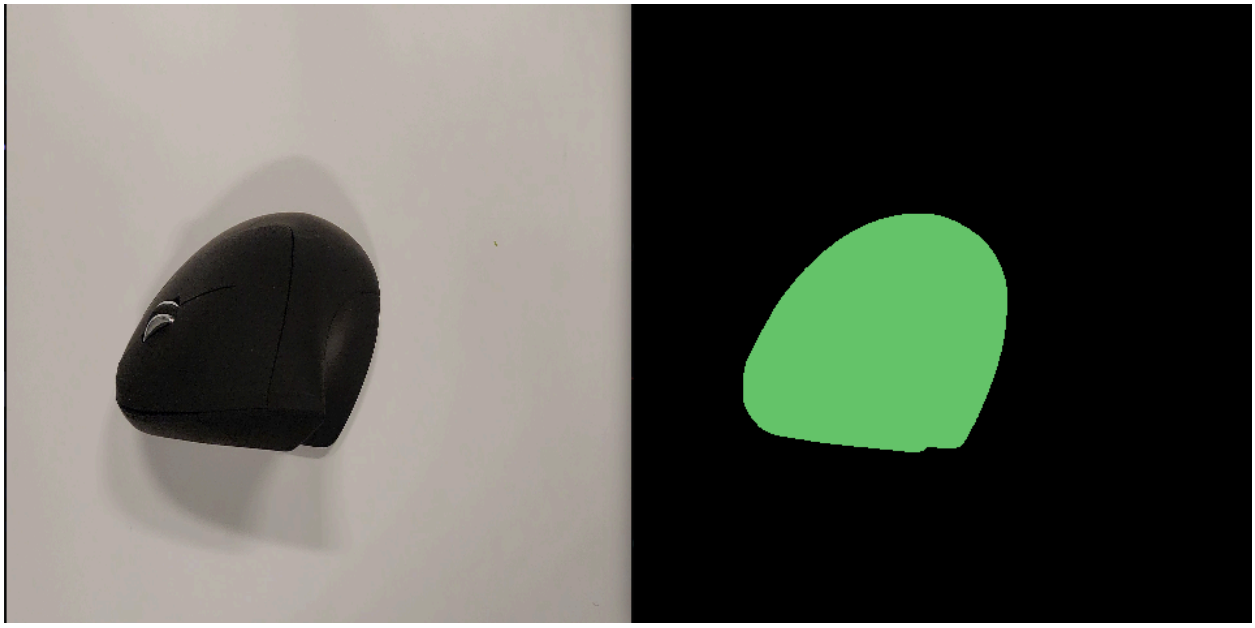
Original **Threshold image** **Cleaned Image**

After applying thresholding to create a binary image and highlighting objects of interest, morphological operations were applied to clean the image. Dilation and erosion were built from scratch to help closing small holes within objects and separating connected ones, effectively enhancing the visibility of the objects against the background. Dilation was done with 8 connectedness and erosion was done with 4 connectedness (following with rule of dilate with 8 and erode with 4 or vice versa). The effect can be seen clearly for the mouse and Pen where small holes in the thresholded image due to reflection or color of object are filled out after morphological operations

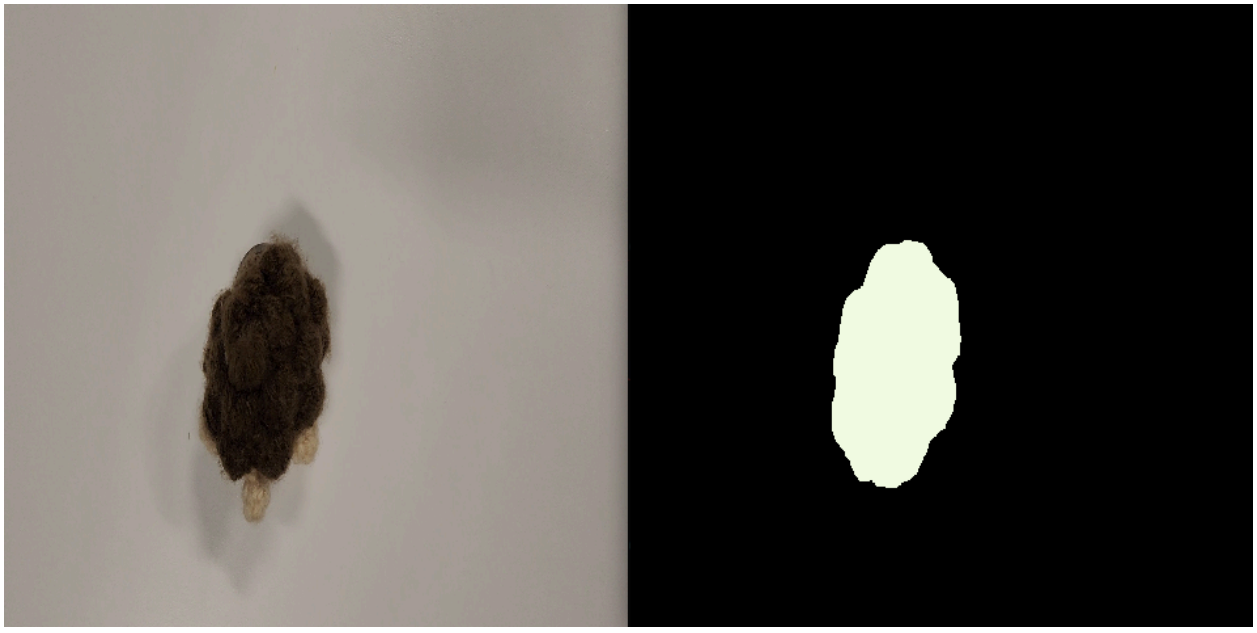
Extension 1

To better learn the functions, we wrote the thresholding, erosion and dilation codes from scratch. We also implemented the different connectedness options into the code.

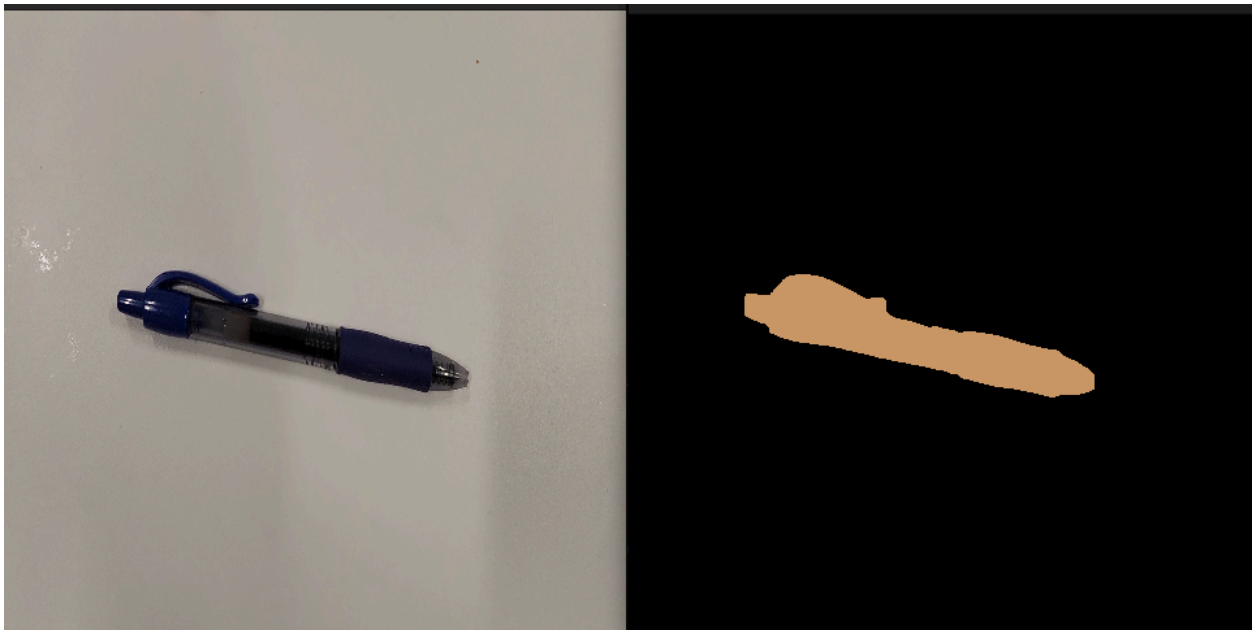
(3) Segmented Images



Original **Segmented**



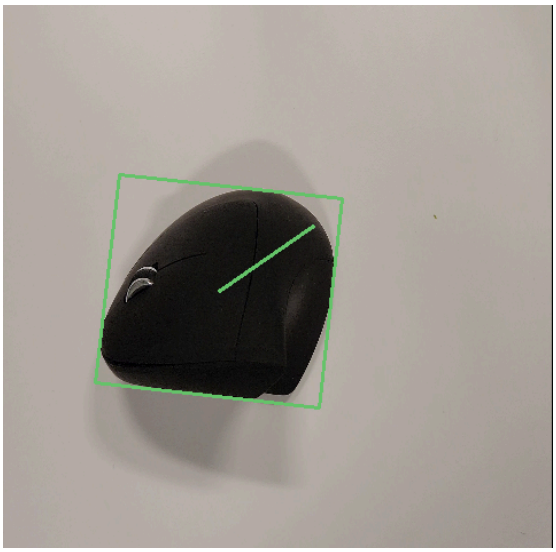
Original **Segmented**



Original **Segmented**

ConnectedComponentsWithStats method was used to label different regions and calculate their statistics and centroids. Regions that are too small were removed and only objects large enough were colored based on their centroid's proximity. To keep the color on each object stable, we keep a track of centroid and color in a custom struct. We check if the centroid of a set of labels is within 50 pixels or not. If it is within 50 pixels, we keep the same color and update the centroid, otherwise we make a new entry in the prevRegions with a new centroid and a new color

(4) Features for Regions



The bounding box is the rotated bounding box for the object. The line that we see is the line of least moment. The features we chose for objects are the values for all moments. Since we already used histograms in a previous project, this is a new and relatively unique feature to use. However, we noticed that moments vary with the area of the object i.e. magnification in our case. Hence to avoid that we use HU moments which work quite well even when scale and orientation changes.

(5) Training Data

To collect training data, user presses ‘N’ when the object is identified, and the system will prompt the user to input the name of the object. After the labels are entered, computed features (moments) of a region is mapped with the user-entered label and this data is saved into the dataset file called ‘features.csv’. We keep appending to this file so that the user can add more data later on

```
ring,0.9441,0.0871,0.0244,0.0757,0.0032,0.0223,0.0005
hairclip,0.3257,0.0022,0.0298,0.0005,0.0000,0.0000,-0.0000
lipstick,0.2853,0.0476,0.0042,0.0034,0.0000,0.0007,-0.0000
lipstick,0.9407,0.6113,0.3209,0.0292,-0.0022,-0.0197,0.0018
lipstick,0.8912,0.7669,0.0147,0.0129,0.0002,0.0111,0.0000
blackcircle,0.1876,0.0047,0.0000,0.0000,0.0000,0.0000,0.0000
blackcircle,0.1711,0.0039,0.0000,0.0000,0.0000,0.0000,-0.0000
bowl,0.2829,0.0062,0.0002,0.0015,0.0000,0.0001,0.0000
hairtie,0.9610,0.1764,0.0024,0.0001,0.0000,0.0000,-0.0000
nailcipper,0.5461,0.2669,0.0010,0.0007,0.0000,0.0003,0.0000
hairtie,0.9747,0.0703,0.0024,0.0003,0.0000,0.0001,-0.0000
nailclipper,0.6098,0.3235,0.0056,0.0007,-0.0000,-0.0003,-0.0000
bowl,0.2920,0.0087,0.0002,0.0009,0.0000,0.0001,0.0000
bowl,0.2646,0.0083,0.0001,0.0006,0.0000,0.0000,0.0000
ring,0.7243,0.0720,0.0142,0.0557,0.0008,0.0142,-0.0013
ring,0.8723,0.1534,0.0419,0.1705,0.0118,0.0662,-0.0082
blackcircle,0.1796,0.0034,0.0000,0.0000,0.0000,0.0000,-0.0000
hairclip,0.2922,0.0271,0.0129,0.0023,0.0000,0.0002,-0.0000
hairclip,0.2572,0.0017,0.0116,0.0001,0.0000,0.0000,0.0000
```

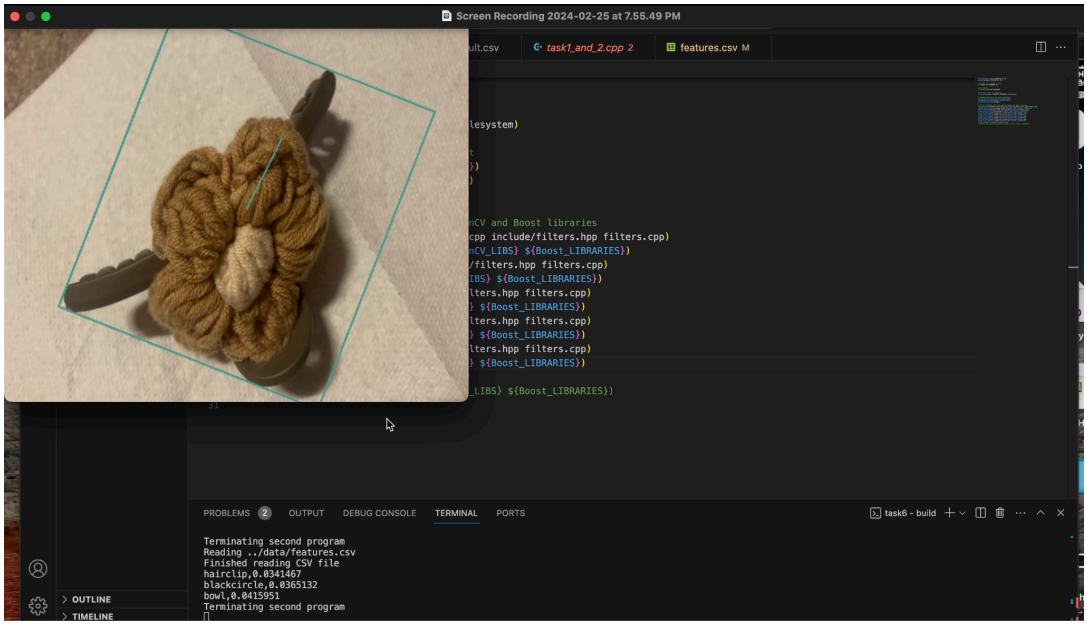
Extension 2

For this extension, we decided to add more objects to the training set. We used 7 different objects in total to make the training set. We made sure to add unique objects with similar shapes to make sure we can differentiate them. For example we added a black circle and a black ring (hairband). These have the same shape but should have very different moments since the mass of the ring is much less than the black circle

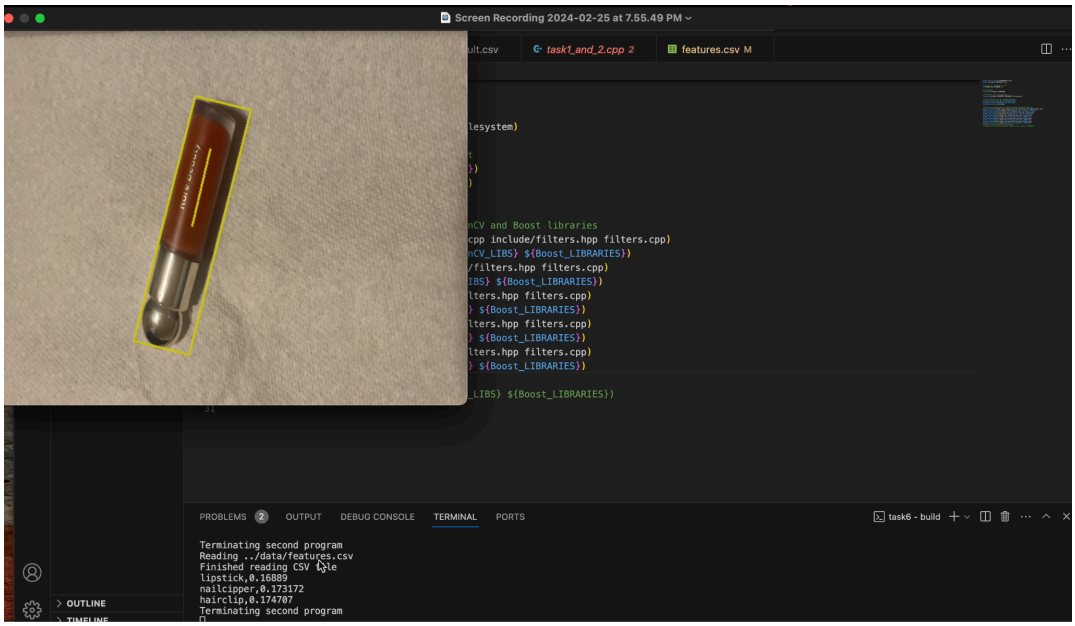
(6) Classify New Images

This object is successfully classified as a hair clip, as shown in the terminal output in this image below. The terminal prints out the top three classification with their corresponding euclidean distance/scaled euclidean distance depending on which metric we choose to use. When the user presses ‘i’, we compute the features for the object in frame and compare it to the features in our csv file. We compute the mean

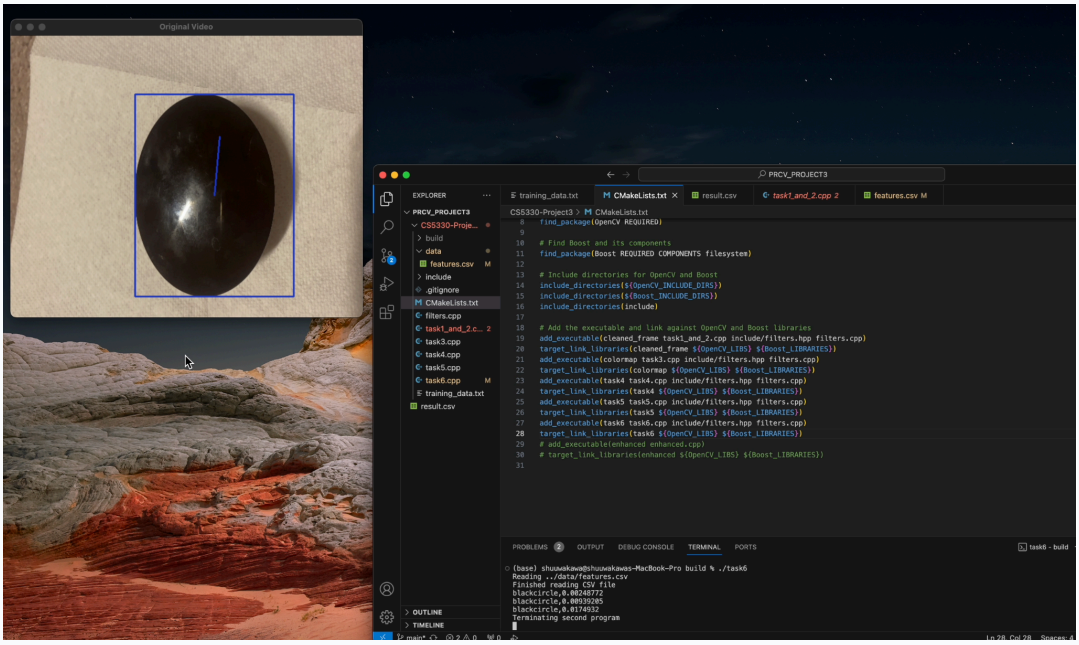
and std dev online everytime so that even if the data in features.csv changes, we don't have to recalculate and store that in a separate csv. We can use the same pipeline everytime.



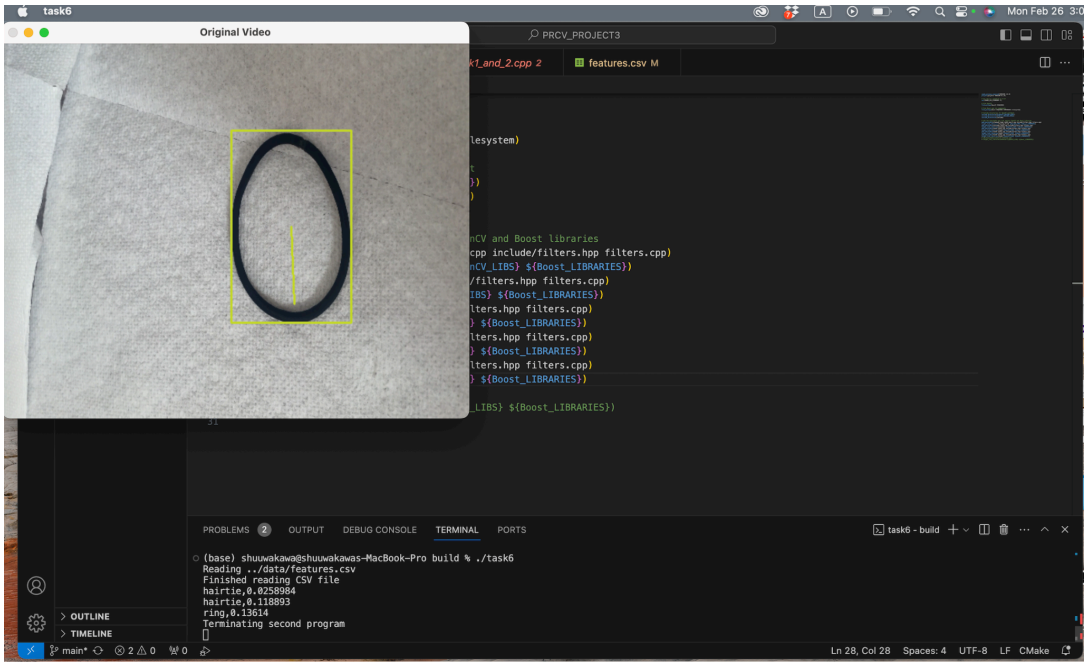
This object is successfully detected as a lip stick:



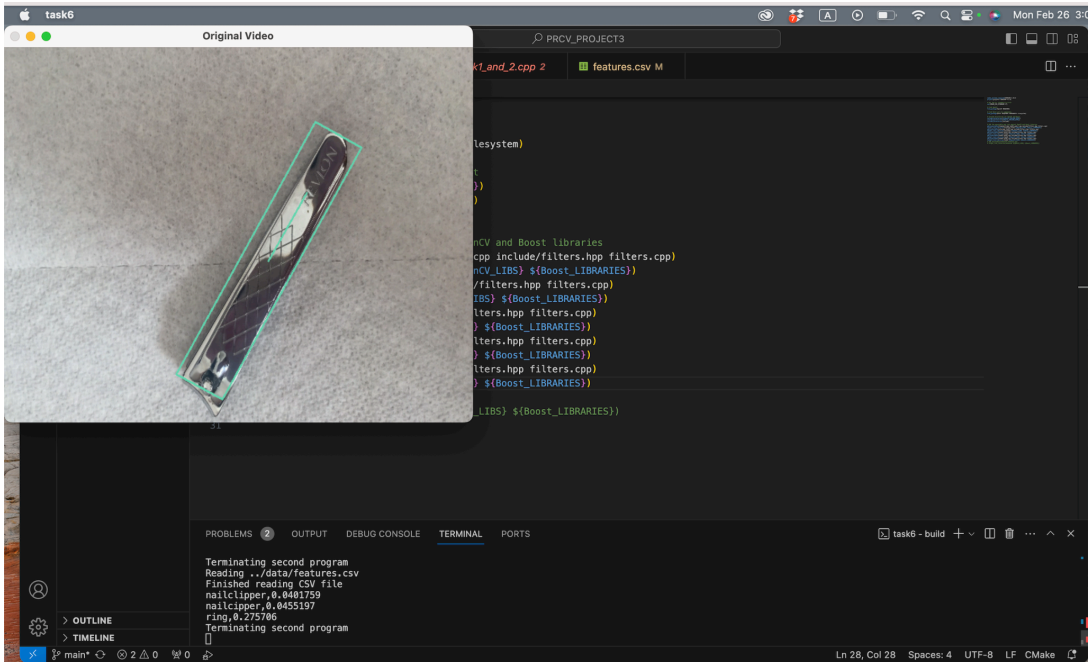
This object is successfully detected as a black circle.



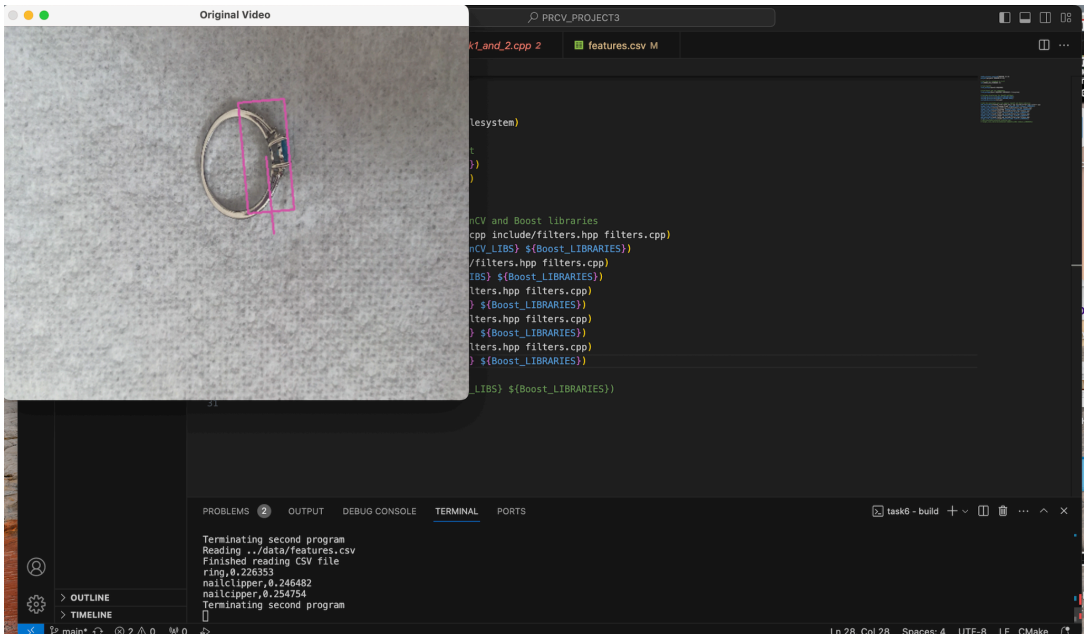
This object is successfully detected as a hair tie.



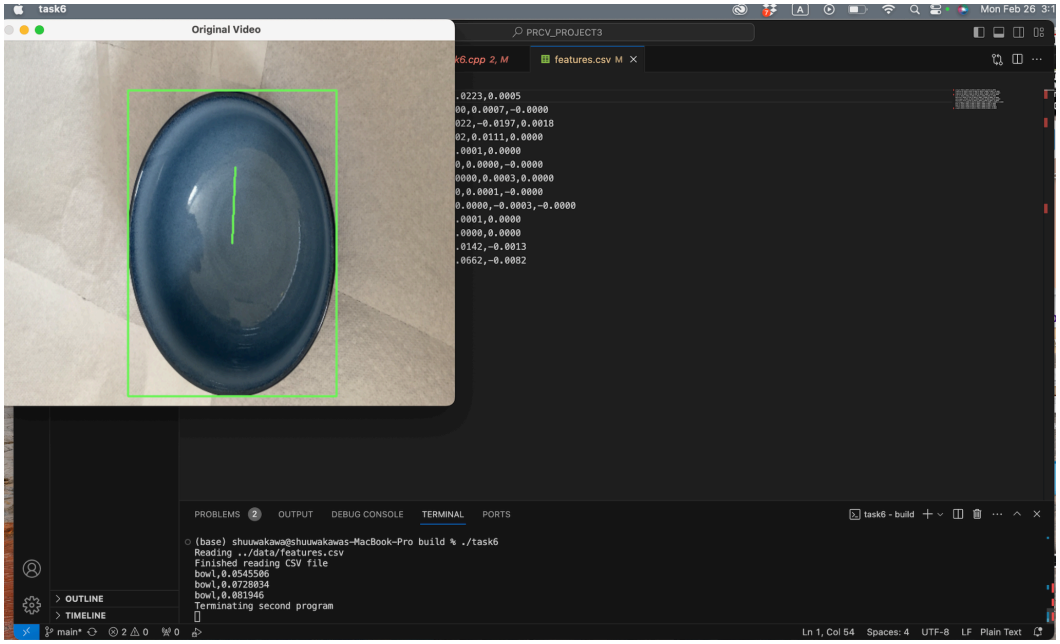
This object is successfully detected as a nail clipper:



This object is successfully detected as a ring:



This object is successfully detected as a bowl:



These are few examples of the inference running. The rest of results can be seen in the confusion matrix below

(7) Performance Evaluation (confusion matrix)

result.csv

	ring	hairtie	blackcircle	lipstick	bowl	nailclipper	hairclip
ring	3	0	0	0	0	0	0
hairtie	2	1	0	0	0	0	0
blackcircle	0	0	3	0	0	0	0
lipstick	0	0	0	3	0	0	0
bowl	0	0	0	0	3	0	0
nailclipper	0	0	0	1	0	2	0
hairclip	0	0	0	0	0	0	3

(8) Video Recording of System Demo

Video of everything running at once -
https://www.youtube.com/watch?v=slFUOzBhv34
Videos recorded with video writer of opencv-

Segmented output- https://youtu.be/xMUBubQdtx4
Features and bounding boxes - https://youtu.be/SR4mEE1rj2o
Original Video attached on gradescope
Note - Videos are different since both of us tried using different objects at our disposal

(9) DNN Based object recognition
For this task, we use mobilenetv3 large with the classification layer removed to get the feature vectors of the image. The output feature vector is of the size 1x960. We save images for the dataset on 'a' keypress (anchor) and infer on 'i' key press. Dataset is not made directly from features but from running the network. We first save the roi from the rotated rectangle that we see in the frame whenever user presses 'a'. When user presses 'i', the roi of the object in frame is saved in a different folder. Now we run the DNN which reads from both the folders and makes feature vectors and saves in two separate csv files - features_dnn.csv and features_query_dnn.csv. Then we read from features_query_dnn.csv and pass the feature vector along with the features_dnn.csv to find the right match. The advantage of this method was that we could give just the RGB ROI with some background in it and the model would still give a good feature vector because of the pretrained weights.
Extension 3-
This was the biggest exploration for this project. Assignment had a sample ONNX file and sample c++ code to run it. To learn more about how to make our own ONNX files and run them using python, we learned about pytorch and how to import models in pytorch. We made a custom network which was essentially mobilenet with final classification layer removed and global pooling applied to convert feature maps into one feature vector. Then we used the inbuilt torch function to export it to an onnx file. We used ONNXRuntime on python to run inference on the ONNX file and save the output to csv files. We have to do some preprocessing like resize and normalizing before passing it to the network. Another part of this extension was learning how to call python files from c++. We learned that we could use the system() function to execute command line prompts from c++.

3. Learnings
We learned a lot from this project. To name some things -
1. Thresholding and selecting right thresholds and applying morphological operations
2. New cv functions like Moments and HuMoments
3. Segmentation and computation of features
4. How to generate confusion matrix
5. New distance metrics like scaled euclidean distance
6. Making and running DNN and using the feature space from it
7. Calling Python executables from C++

4. Acknowledgement

- https://szeliski.org/Book/
- https://www.cs.toronto.edu/~urtasun/courses/VisualRecognition/visual_recognition.html
- Stack overflow for debugging
- OpenCV Documentation
- ONNX Runtime documentation (https://onnxruntime.ai/docs/get-started/with-python.html)