Project 3: Real-Time 2D Object Recognition

Samara Holmes
Northeastern University

I. Introduction

The objective of this project is to develop code to allow a computer to identify a specified set of objects placed on a white surface. The computer should be able to recognize single objects placed in the video frame regardless of translation, scale, and rotation. It should also be able to recognize these objects in real-time.

II. Methodology

For this project, we have the following tasks:

1) Threshold the input video
2) Clean up the binary image using a morphological filter
3) Segment the image into regions
4) Compute features for each major region
5) Collect training data
6) Classify new images
7) Evaluate the performance of the system
8) Video demonstration
9) Implement NN matching, comparing scaled Euclidean, and cosine distance

III. Results

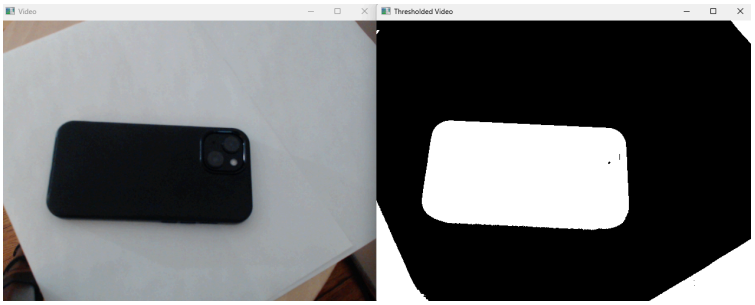


Figure 1. Thresholding examples using pliers, Xbox controller, and phone.

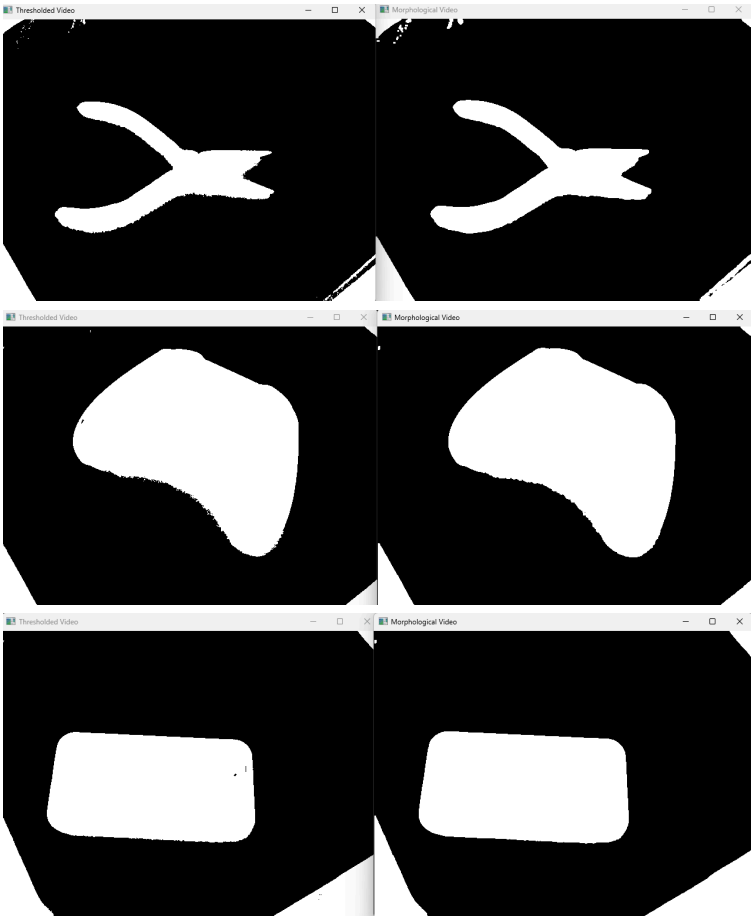
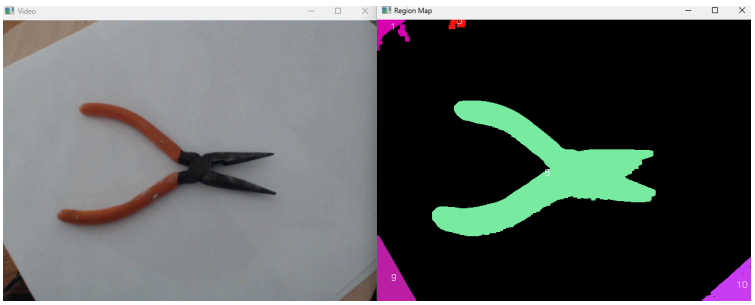


Figure 2. Morphological filter (using dilation) examples using pliers, Xbox controller, and phone.



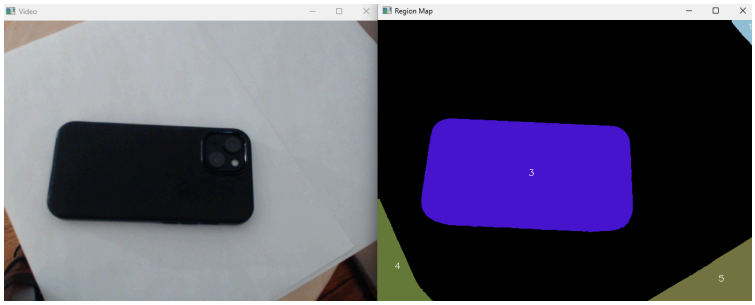


Figure 3. Region maps using pliers and a phone.

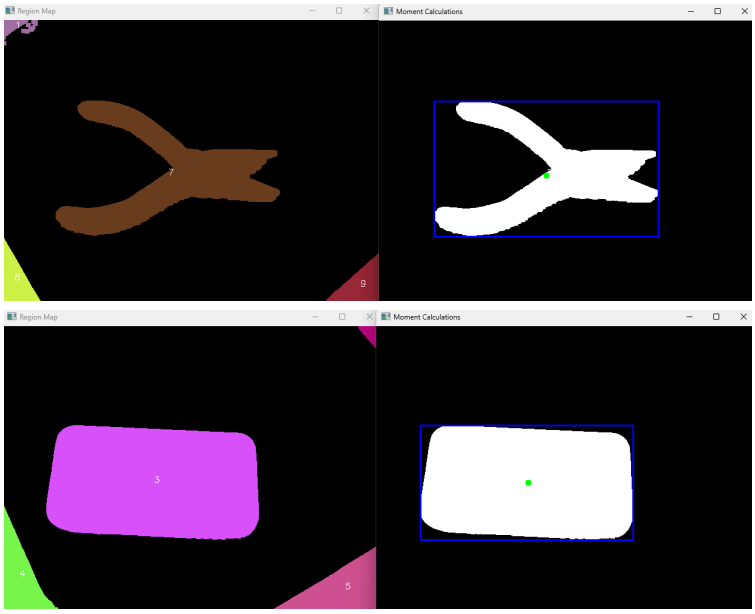
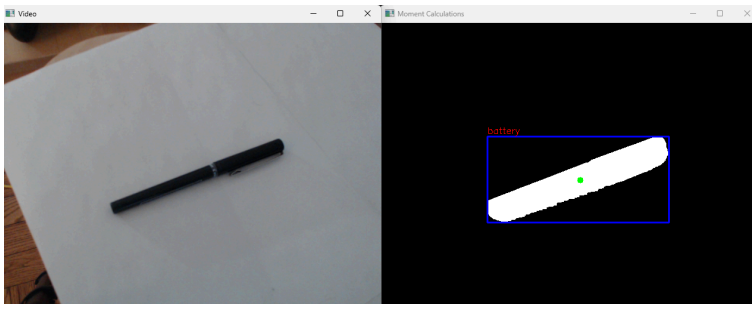
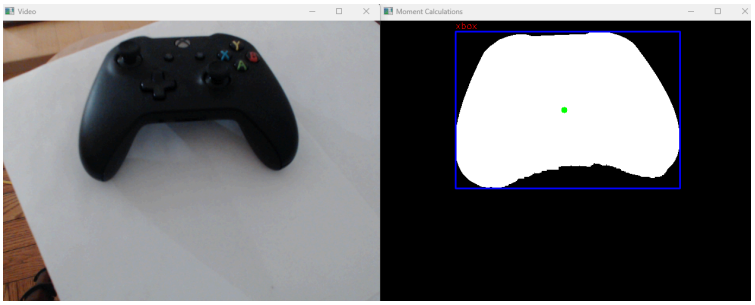


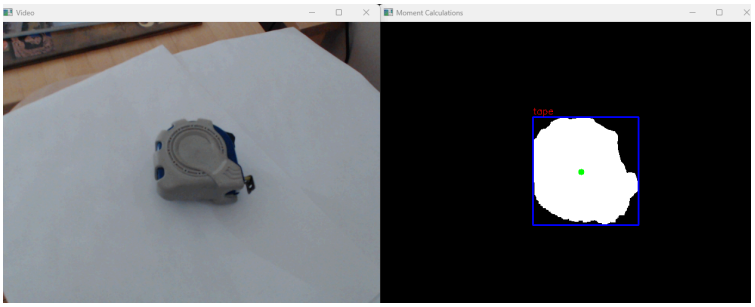
Figure 4. Region map showing the axis of the least central moment and the oriented bounding box of region 2 of an orange and a phone.

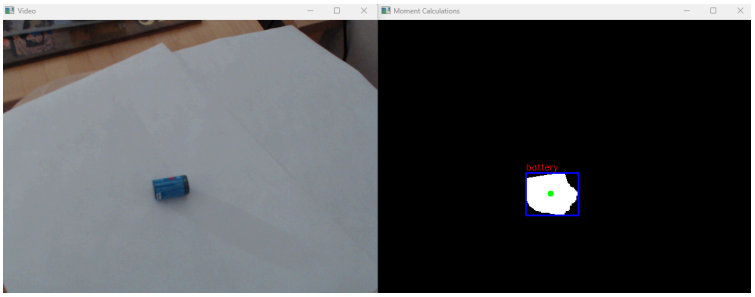
Table 1. Feature vector for objects

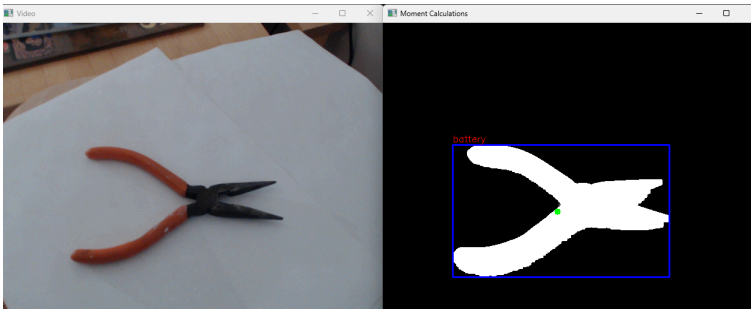
Features	Pliers	Phone
Centroid	(134.279, 10.0704)	(254.505, 262.169)
Orientation	-0.41657	0.0467859
Percent Filled	0.00235677	0.198988
Bounding Box Ratio	0.5625	0.535014

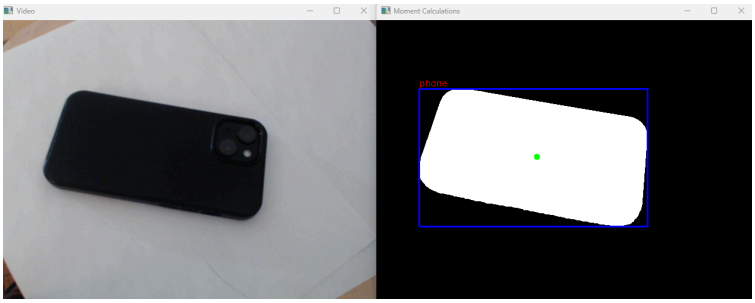












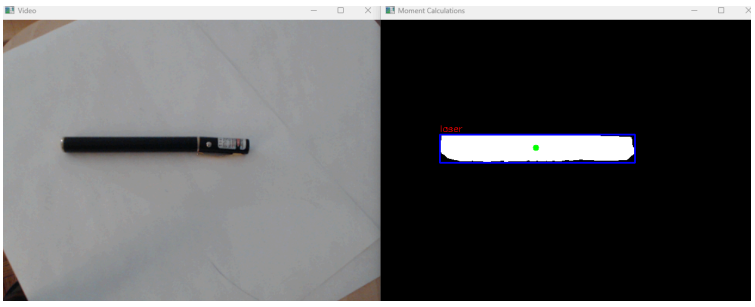


Figure 5. Result image of each category of object

Table 2. Confusion matrix results showing true labels versus classified labels (x = true, y = predicted)

	phone	orange	xbox	pliers	battery	rubiks
phone	3	0	0	0	0	0
orange	0	2	0	1	0	0
xbox	0	0	2	0	0	0
pliers	0	0	0	0	0	0
battery	0	0	0	2	2	0
rubiks	0	1	1	0	1	3

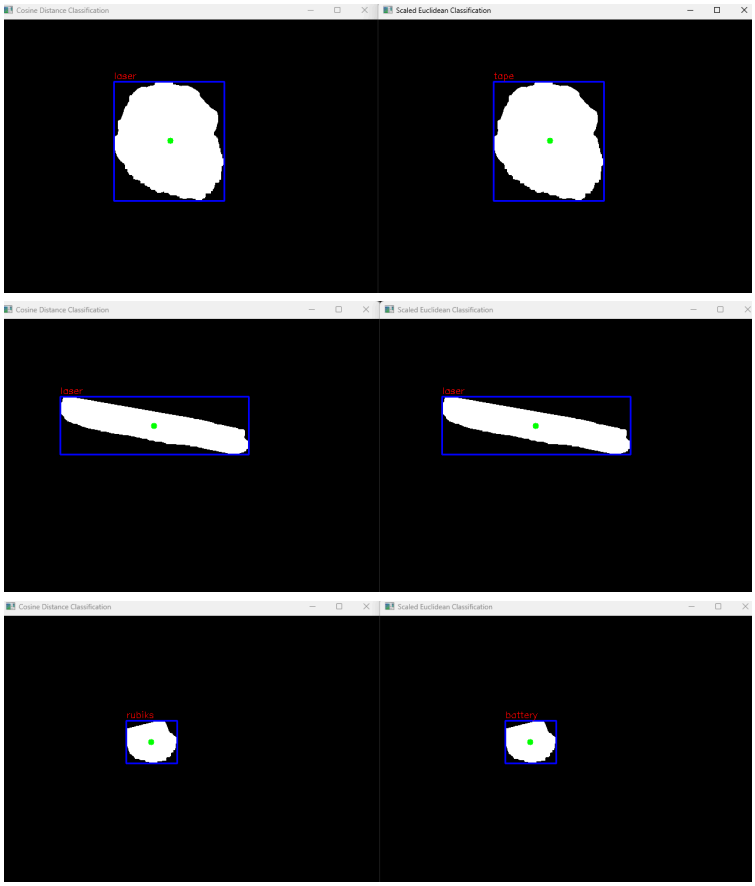


Figure 6. A comparison of the scaled Euclidean and cosine distance metrics

IV. Discussion

For the first task, in Figure 1, we were tasked with thresholding the input video. To do this, I first converted the frame to greyscale, then I applied binary thresholding using the OpenCV function. One thing to note, that I didn't realize until the later steps, was that I was thresholding using cv::THRESH_BINARY, but I should've been using cv::THRESH_BINARY_INV.

For the second task, to clean up the binary image, I created a morphological filtering function to dilate the binary thresholded frame. To help remove the noise, I also applied the 5x5 blur filter on the image before dilation. There wasn't too much cleaning up necessary, but this also varied as my lighting changed. For example, if I shined a light directly above the pliers, the outline of them in the thresholding output would be in greater detail.

For the third task, to segment the image into regions, we need to run a connected component analysis using the cv::connectedComponetsWithStats() function. We take the thresholded and dilated image and then analyze tit o find the number of components. Using this, we can store the size and labels for each component and color them. For debugging purposes, I also labeled the regions on the returned image.

For the fourth task, to compute the features for each major region, the function starts by converting the input region map to greyscale and performing connected components analysis to label and extract region statistics. A binary mask for the specified region is created, and non-zero points are extracted. Spatial moments are then calculated to find the centroid and orientation of the region. The function uses cv::boundingRect() to calculate the bounding box, which is used to determine the percent filled and the height-to-width ratio. The feature vector includes the centroid, orientation, percent filled, and bounding box ratio and they are printed to the console for reference.

To collect training data, the user needs to type 'n'. This will then prompt the user for a name/label and then store the feature vector for the current object along with its label in the file. See below for a sample from feature_vector.csv.

```
2,254.505,262.169,0.0467859,0.198988,0.535014,phone
2,361.448,288.81,1.30724,0.107415,1.2,orange
2,338.142,195.287,0.48869,0.283249,0.920513,xbox controller
2,337.773,211.396,1.0197,0.0911165,1.0582,tape measure
2,295.557,221.529,-0.0962001,0.0557129,0.235821,laser pointer
2,134.279,10.0704,-0.41657,0.00235677,0.5625,pliers
2,277.732,223.394,1.29508,0.129401,1.10132,rubiks cube
2,289.673,302.712,-0.0788509,0.0117513,0.604938,battery
2,604.451,45.7251,0.99787,0.0219173,1.29897,pen
```

Figure 7. Feature_vector.csv sample

For the sixth task, to classify a new feature vector using known objects, we label the unknown object according to the closest matching feature vector in the object database. This process is known as nearest-neighbor recognition. For this task, we use the scaled Euclidean distance metric. Figure 5 shows the objects with detected labels based on data from the feature vector database.

What was interesting is that there was some misclassification. The pliers and pen were detected as batteries. The pen being confused as a battery is reasonable because of their similar shape, however, the pliers being misclassified isn't expected. The tape measure was correctly classified, however it was being confused with an orange in some of the trials.

As shown in the figure below (Figure 8), I also changed the orientation of the pliers to see what classification it would be given, and it was given an unknown object configuration because the orientation is so different than the training orientation.

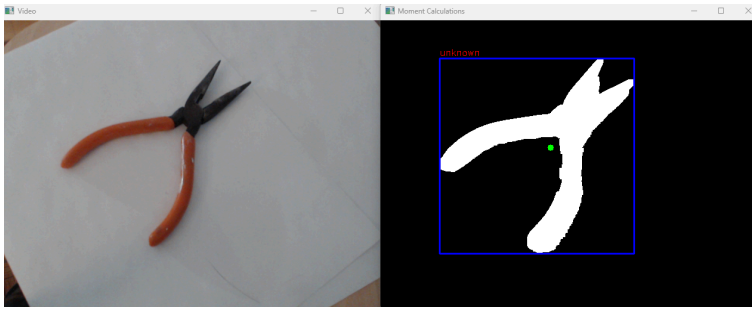


Figure 8. Unknown object classification

To evaluate the performance of the system, we generated the 6x6 confusion matrix as shown below. This demonstrates the results showing true labels versus classified labels of 3 images of each object in different positions and orientations (see Table 2).

Click here to view the demo video of the classification system.

We implemented nearest-neighbor matching with different distance metrics. Previously, in task six, we used the scaled Euclidean distance metric. In Figure 6, we compared scaled Euclidean to cosine distance. It is observed that scaled Euclidean performed better than the cosine distance metric. Cosine distance caused a misclassification for the tape measure and the battery.

As another extension task, I implemented an object recognition model, YOLOv11. To do this, I first needed to train the model in Python and export it to an ONNX format. This took 100 epochs on my laptop which only took a few minutes to process.

V. Reflection

Developing an object recognition system allowed me to learn more about image processing and feature recognition. The system identifies and classifies objects in a video stream using feature vectors and nearest-neighbor recognition. Key elements include feature extraction, data storage, nearest-neighbor recognition, and a training mode. Features like centroid, orientation, percent filled, and bounding box ratio are computed and stored in a CSV file for classification. The largest challenge I faced was computing the features for a given region, which was mainly due to needing to convert to different image formats for different computations.

VI. Appendix

In the first project, we were tasked with running DepthAnythingV2 on a given image, however, it was canceled due to many students being unable to run the library necessary to run the algorithm. Despite this, I figured out my prior issue with running the library and was able to run DepthAnythingV2 on images and video clips. Click here to view a clip of Avatar (2009) run with DepthAnythingV2. Figure 9 shows the results of running the algorithm on a photo of a puffin I took in Iceland.

Figure 9. A comparison of an image of a puffin and a depth map from DepthAnythingV2.

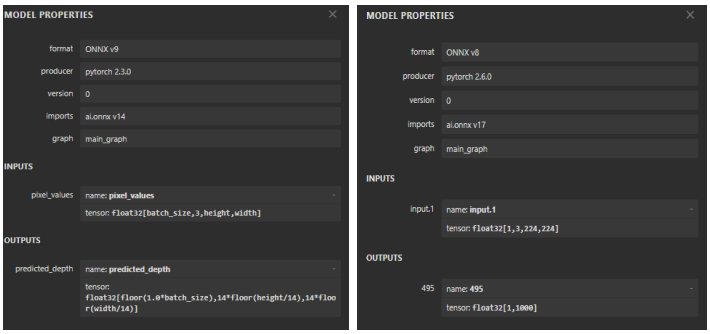


Figure 10. A comparison of model properties of the DepthAnythingV2 model and the NeuFlowV2 model from Netron.app.

A couple of extensions that I wasn't able to complete in time were:
1) using information from DepthAnything to threshold the object from the background
2) implementing NeuFlowV2 to show optical flow in real-time and thresholding against the moving object

VII. Acknowledgments

https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html

https://medium.com/@rajilini/morphological-operations-in-image-processing-using-opencv-and-c-8580de272606

https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f

https://pytorch.org/tutorials/beginner/onnx/export_simple_model_to_onnx_tutorial.html

https://netron.app/

https://github.com/ultralytics/