Project 3: Real-time 2-D Object Recognition
Pattern Recognition and Computer Vision (CS-5330)

Team Members:
Poojit Maddineni (NUID: 002856550)
Venkata Satya Naga Sai Karthik Koduru (NUID: 002842207)

Throughout the project, we learned a lot about building an Object Recognition system using computer vision techniques. Here's a brief overview of the insights gained:

Thresholding and Preprocessing:

- Thresholding is crucial for separating objects from the background, and preprocessing techniques like blurring help enhance region uniformity, improving thresholding accuracy.

Morphological Filtering:

- Morphological operations play a vital role in cleaning up binary images by removing noise and filling holes. Understanding image artifacts helps in selecting appropriate filters.

Connected Components Analysis:

- Segmenting images into distinct regions using connected components analysis is essential. Factors like region size, centrality, and boundary touch influence segmentation quality.

Feature Extraction:

- Extracting invariant features such as percentage filled and bounding box ratios is critical for object recognition. Geometric properties like moments and oriented bounding boxes provide valuable information.

Training Data Collection:

- Systems should facilitate the collection and labeling of feature vectors for known objects. Implementing a training mode simplifies this process and allows for user interaction.

Classification and Recognition:

- Classifying unknown objects using stored feature vectors and distance metrics is fundamental. Nearest-neighbor recognition using scaled Euclidean distance is a common and effective approach.

Performance Evaluation:

- Evaluating system performance on diverse object images is essential for validation. Confusion matrices help visualize classification accuracy and identify areas for improvement.

In summary, the project provided practical insights into the iterative process of developing an OR system, emphasizing the importance of image processing techniques, data representation, and algorithm selection for achieving robust and accurate object recognition capabilities.

Workspace Setup:

For the workspace setup, We opted for a simple yet effective arrangement. we utilized a white background, preferably A3 paper, as the backdrop for the workspace. This choice helps enhance contrast and visibility, making it easier to detect and analyze objects. To stream the video feed, we used the Iriun app, and we then used the app for the transmission of live video from a mobile device to a PC.

Tasks:

Task 1: Threshold the input video:

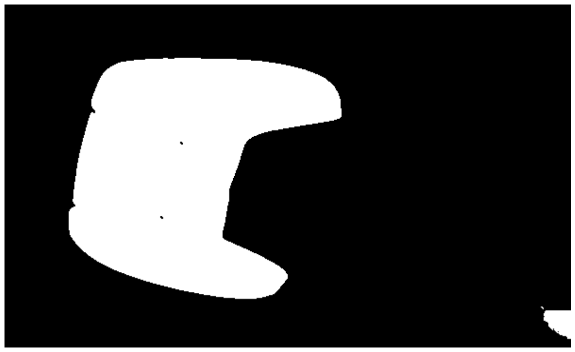
In Task 1, we have to threshold the RGB image to segment it in the next tasks. The image was preprocessed by applying a Gaussian filter to smoothen out the image and to reduce the noise in the image, then the image was converted to a grayscale image to prepare the image for converting to a binary image.

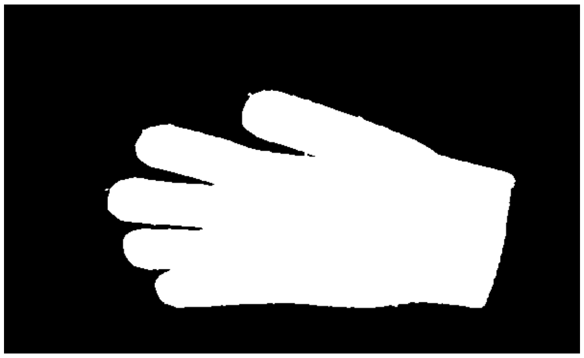
Our thresholding algorithm, implemented from scratch, involved iterating through each pixel in the image using nested loops—one to traverse rows and another for columns. We empirically determined an optimal threshold value of **120** through trial and error. This threshold value effectively distinguished objects from the background.

During thresholding, we assessed the intensity of each pixel against the predetermined threshold. Pixels with intensities greater than the threshold were set to 0, denoting foreground objects, while pixels below the threshold were set to 255, indicating the background. This process resulted in creating a binary image, where objects of interest were distinctly separated from the background, facilitating subsequent analysis and processing.

We also tried using K-means to dynamically threshold the image but we did not find a noticeable difference in the output, so we stuck to simple thresholding for lower processing time.

We chose 5 objects, a wallet, a calculator, a controller, a glove, and a bottle opener.


Thresholded Image of Controller


Thresholded Image of a glove

Thresholded Image of a Calculator

Task 2: Clean up the binary image:

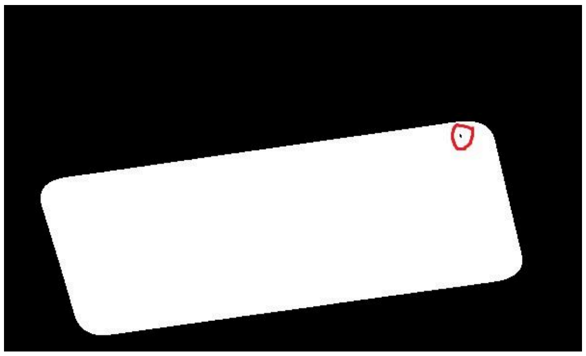
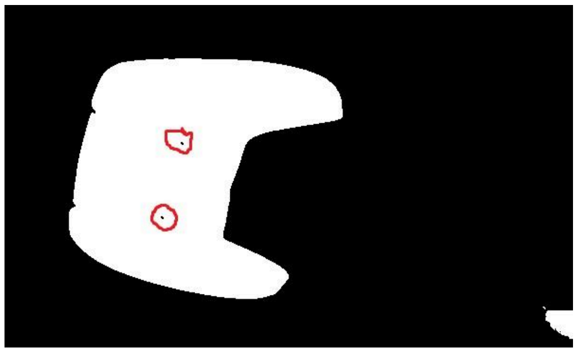
In Task 2, we employed a morphological operation known as closing to enhance the quality of the binary image. Closing involves two sequential steps: dilation followed by erosion. The reason why we chose this method is because we observed small holes in our image, so we decided to dilate, and then erode so the holes get covered and the features are returned to their actual size.

Dilation expands the regions of foreground pixels in the binary image by considering the intensity values of pixels in the neighborhood defined by the kernel size. This process helps in closing small gaps or holes within the objects, effectively filling them in and making the objects more solid

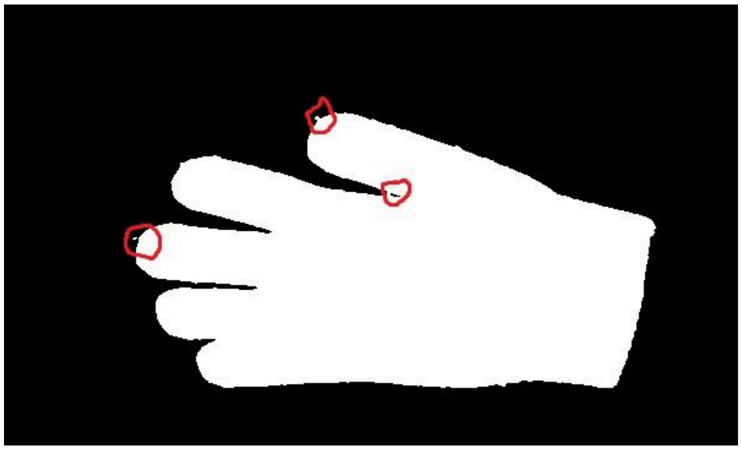
However, dilation also has the effect of increasing the size of the objects, potentially altering their original shapes. To counteract this, we perform erosion, which is the opposite of dilation. Erosion shrinks the foreground regions by finding the minimum intensity value within the neighborhood for each pixel. This step helps in restoring the objects to their approximate original size while preserving their shapes.

The combination of dilation and erosion in the closing operation is particularly useful for cleaning up binary images by closing small gaps or holes within objects without significantly altering their overall size or shape. This process ensures that the objects are accurately represented in the binary image, facilitating subsequent analysis and processing

Let us look at the binary images again to understand the need for dilation and erosion

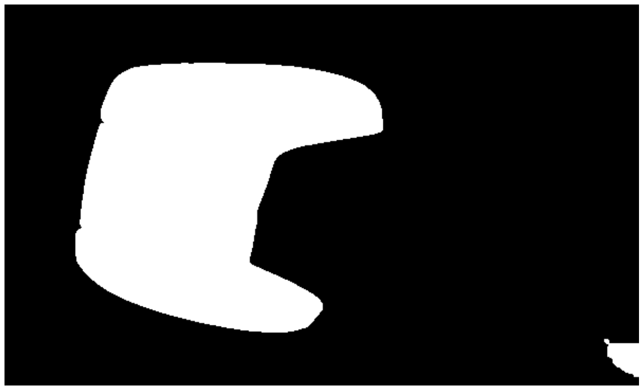



Thresholded Image of Calculator Thresholded Image of Controller

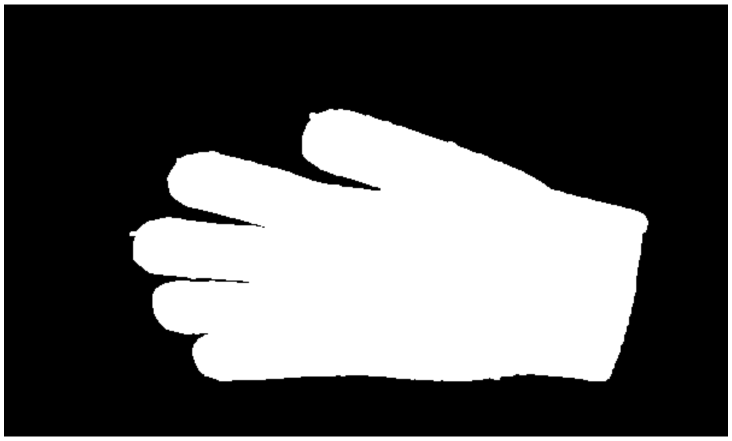


Thresholded Image of Glove

We can see in the above image inside the red circular regions, we can see these holes, to eliminate them, we dilated the image, and below are the results fo the operation.



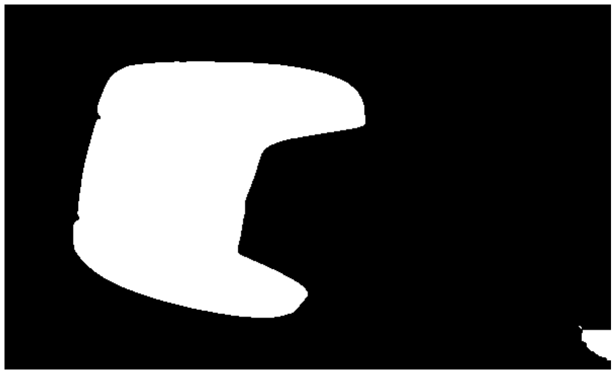
Dilated Image of Controller Dilated Image of Controller



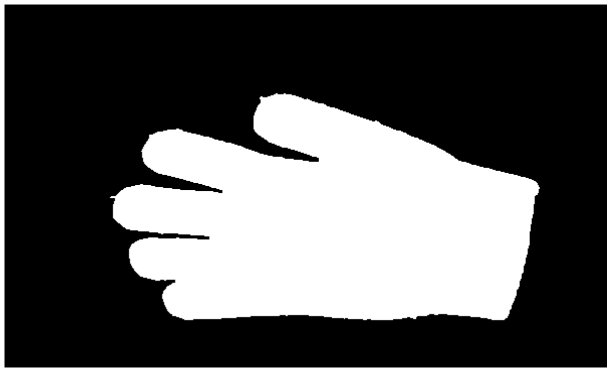
Dilated Image of Controller

Its clear that dilation removed these defects but also increased the area of the regions as a whole, to bring the regions back to their original sizes, we are performing an erosion operation, the pictures related to that are below.

Eroded Image of Controller



Eroded Image of Controller

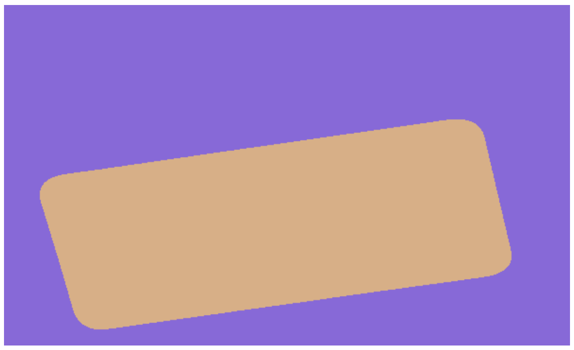


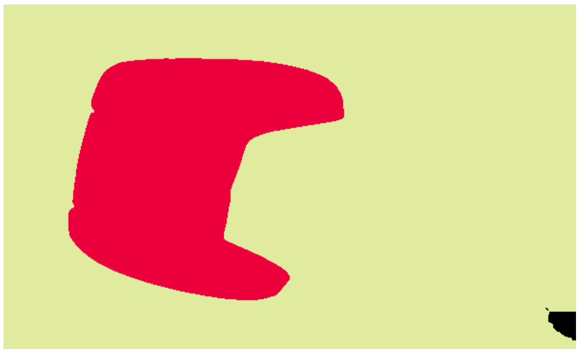
Eroded Image of Controller

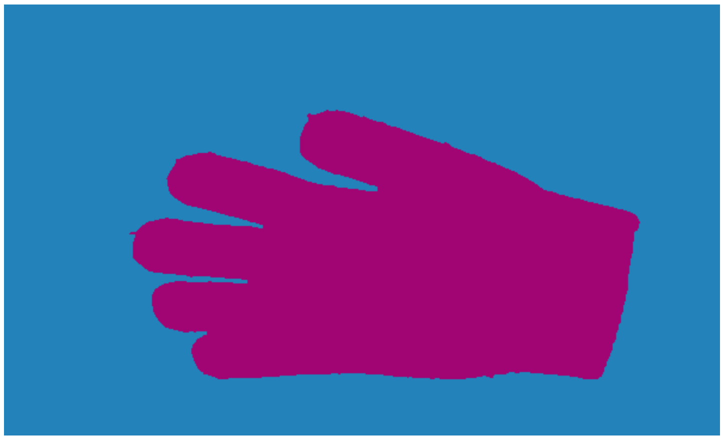
Task 3:**Segment the image into regions:**

In this step, we segmented the eroded image using the inbuilt OpenCV function, and connectedComponentsWithStats function. The function returns the centroids of the regions, the region map and the stats for the image.

After extracting that, computed the area of the the individual regions and empirically found that thresholding the region are to a minimum of 5000 pixels was best suitable for out setup. Then the individual regions were assigned a color and displayed in a window. The output images are below.

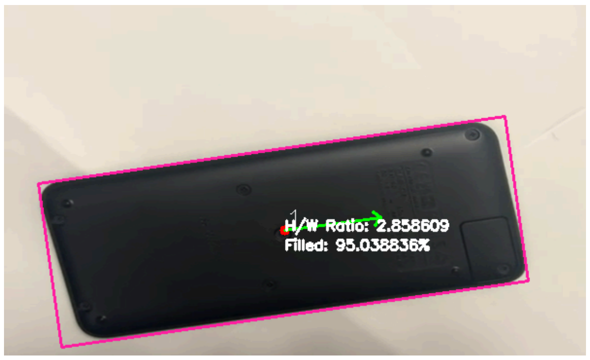

Region map with a calculator in the workspace

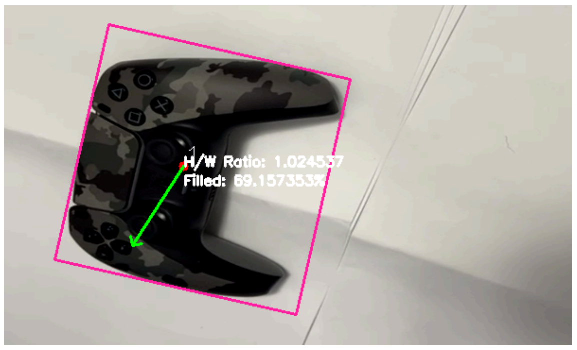

Region map with a controller in the workspace


Region map with a glove in the workspace

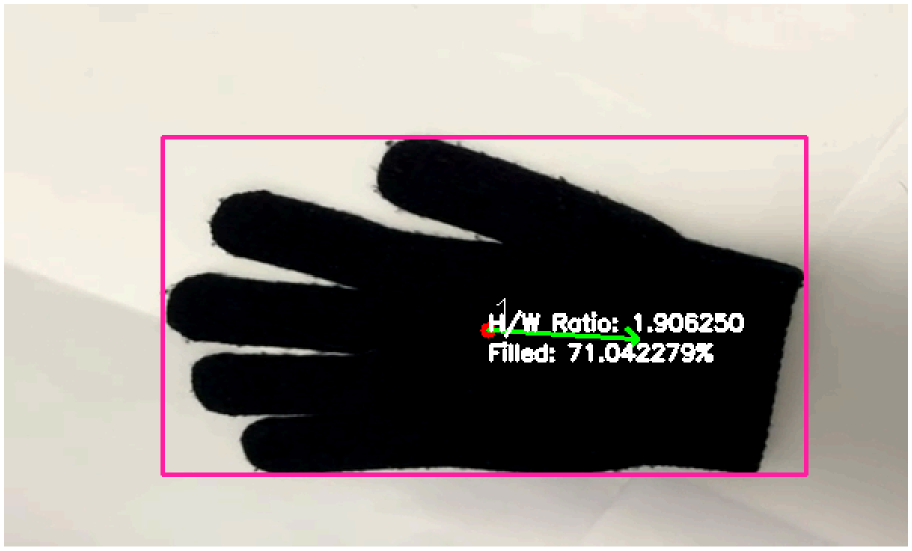
Task 4:**Compute features for each major region**

We have calculated the features for the regions by computing the eigenvalues and eigenvectors to get the axis of the least central moment. The axis can be seen in the images below with a green arrow pointing away from the centroid. Then we compute the feature vectors such as the bounding box height-to-width ratio and the percentage filled ratio create a structure to store the corresponding values of the regions along with the centroids.



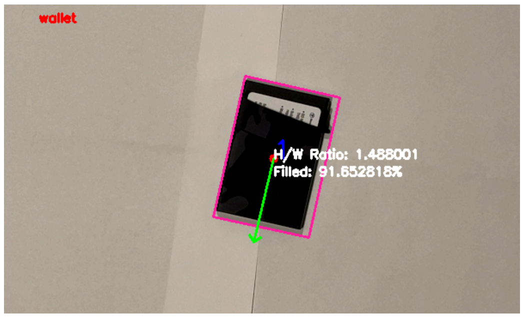


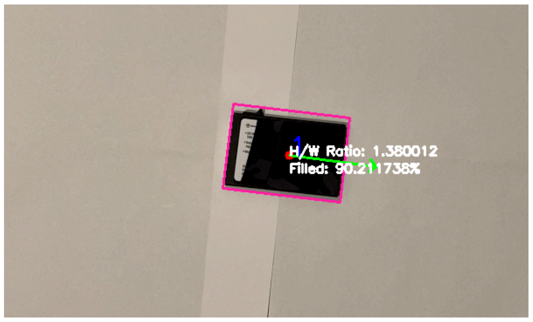
Feature Vectors of Calculator **Feature Vectors of Controller**

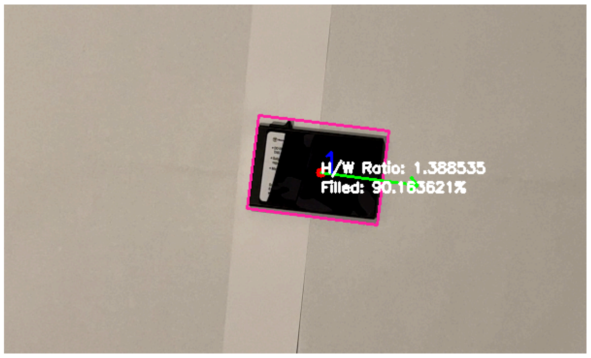


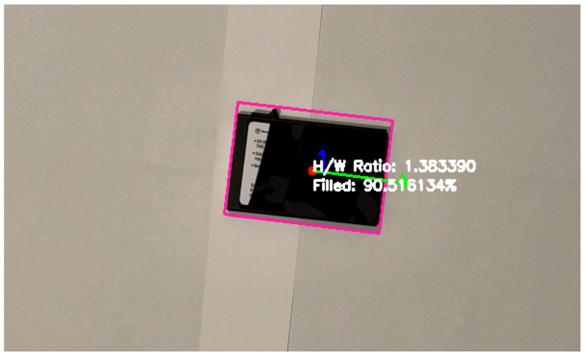
Feature vectors of gloves

The feature vectors are translation, scale, and rotation invariant. The images of them can be seen below.









In the above images, we can observe that the feature vectors remain mostly the same and don't vary when the object changes orientation, position, or distance from the camera. If there is any change, that is mostly due to any shadows formed due to the lighting conditions.

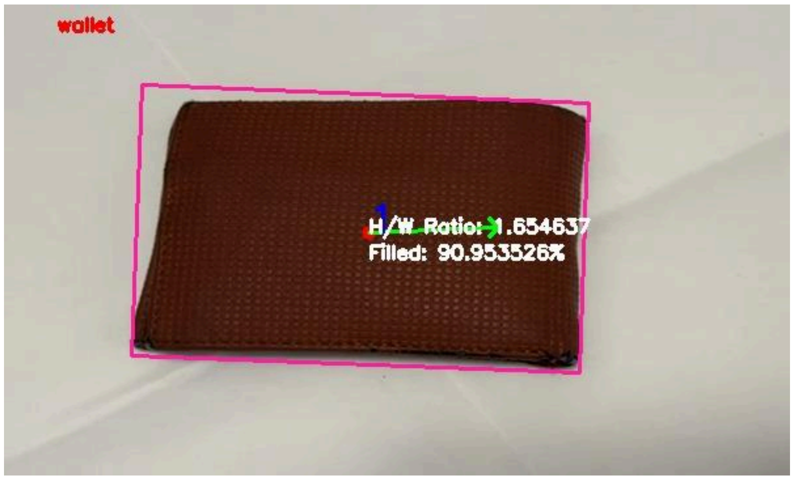
Task 5:**Collect training data**

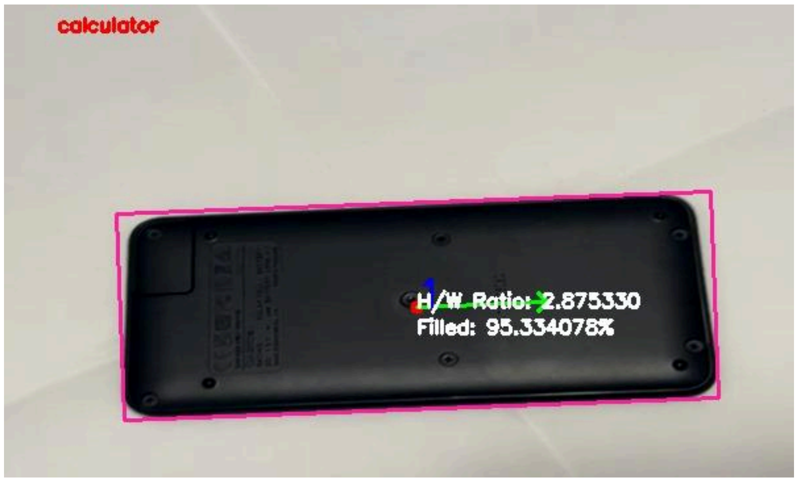
We collected training data for about 5 objects, a wallet, a calculator, a controller, a glove, and a bottle opener.

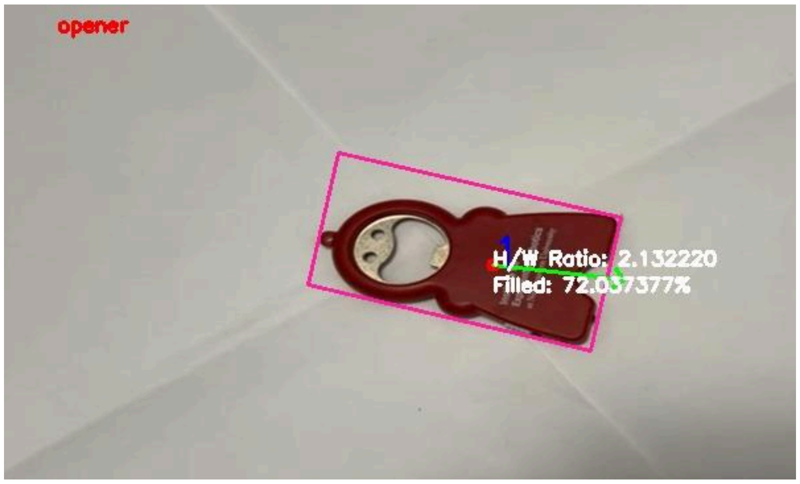
We place the object in the workspace with a white background, then we press "n" to store the feature vectors, after we press "n" the terminal asks the user to enter the label for the object. Then the object will be stored in a CSV. We repeat this until we train all our data.

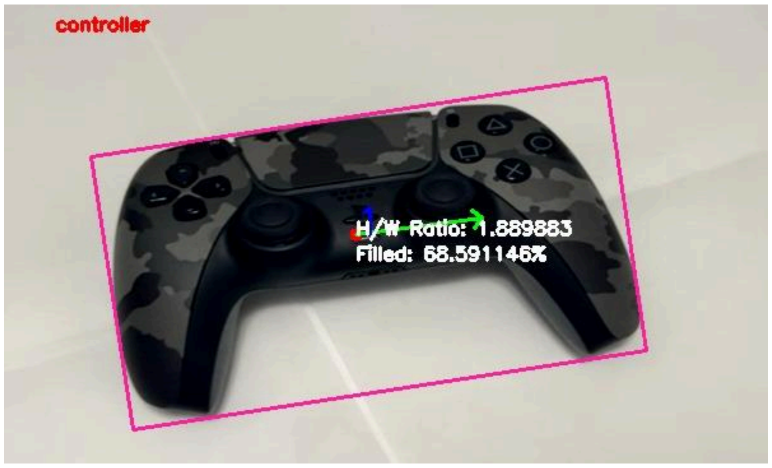
Task 6:**Classify new images**

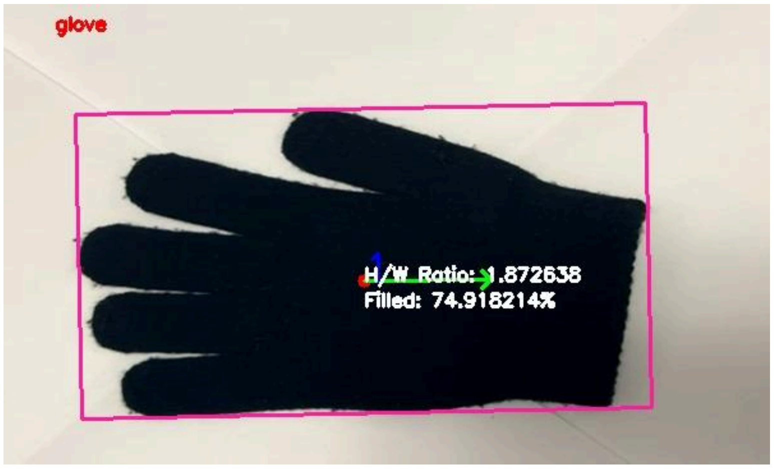
Below are the classified images:



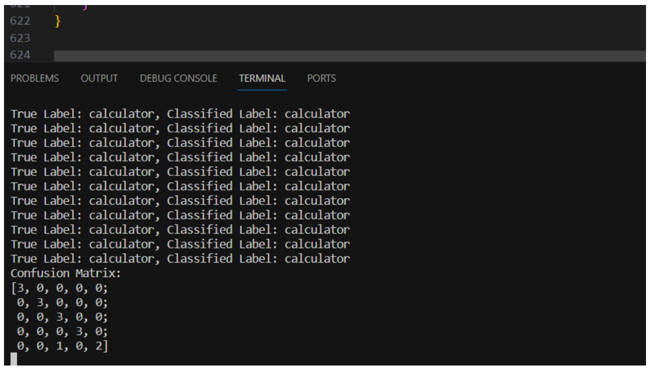








Task 7:**Evaluate the performance of your system**



```
    }

True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
True Label: calculator, Classified Label: calculator
Confusion Matrix:
[3, 0, 0, 0, 0;
 0, 3, 0, 0, 0;
 0, 0, 3, 0, 0;
 0, 0, 0, 3, 0;
 0, 0, 1, 0, 2]
```

Confusion Matrix

The program was able to classify the image pretty accurately, we had an accuracy of **93.33%** for an image data set of **5 images** with **3 samples** each.

Task 8:**Capture a demo of your system working**
The video demo can be found here:
🗀 **Project-2 Video Files**

Task 9:**Implement a second classification method**

For the second classification method, I chose to implement K-Nearest Neighbor (KNN) detection with K > 1. KNN is a simple yet effective classification algorithm that works by finding the K closest data points in the feature space and assigning the most common label among them to the query point.

Here's how I implemented KNN detection with K > 1:

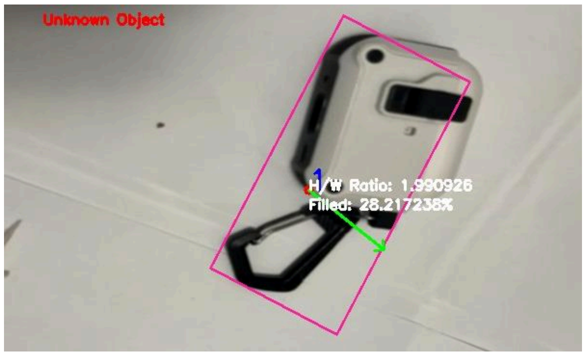
Feature Extraction:
Similar to the baseline code, I extracted feature vectors from the objects in the images. These feature vectors include characteristics such as bounding box area and height-to-width ratio.
Training Data:
I used a dataset containing labeled feature vectors to train the KNN classifier. This dataset consists of multiple examples for each object class, allowing the classifier to learn the variations within each class.
Distance Calculation:
When classifying a new object, I calculated the Euclidean distance between its feature vector and the feature vectors of all training examples.
K Nearest Neighbors:
Instead of considering only the nearest neighbor as in the baseline code, I considered the K nearest neighbors. This helps in reducing the impact of noise and outliers in the data.
Classification:
After identifying the K nearest neighbors, I assigned the label that appears most frequently among them as the predicted label for the new object.
Comparison with Baseline:

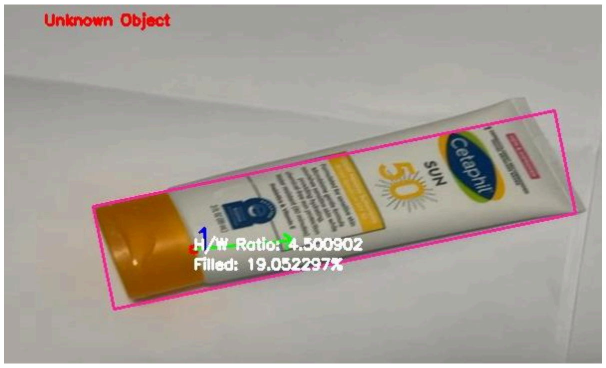
I didn't observe much difference between the performance of KNN with K > 1 and our baseline method, which could be attributed to our small dataset. Both methods performed similarly and appeared quite sensitive to changes in the environment.

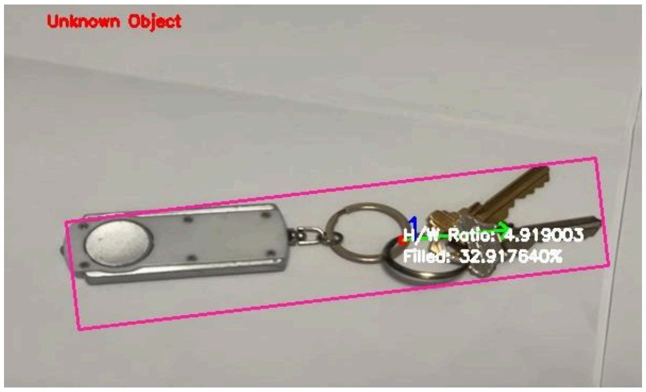
Extensions: My partner and I opted to take an extra travel day and we were both working on the extensions separately, hence you will find different objects being used for extension 3 and 5

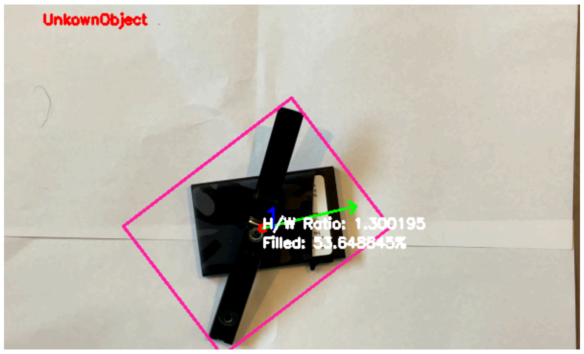
Extension-1

Detecting Unkown Object:
I basically used a threshold for the scaled Euclidean distance to see if the distance is less or more in order to compare, based on that, the object is classified and unknown.

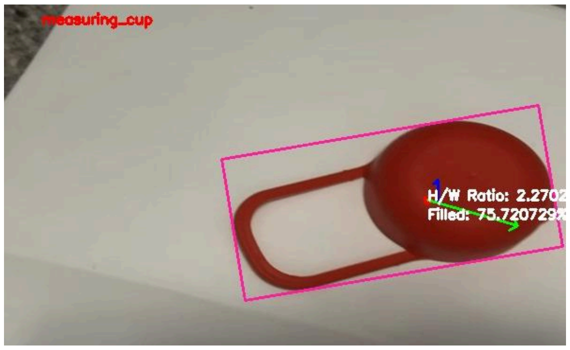


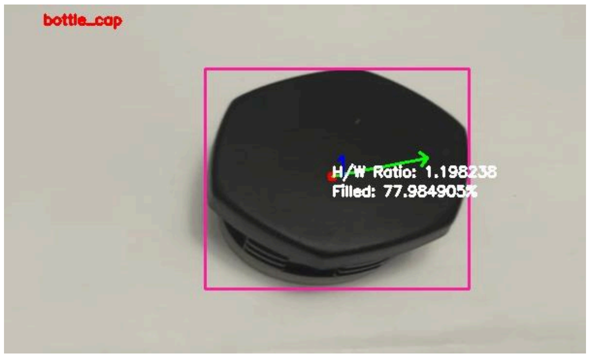


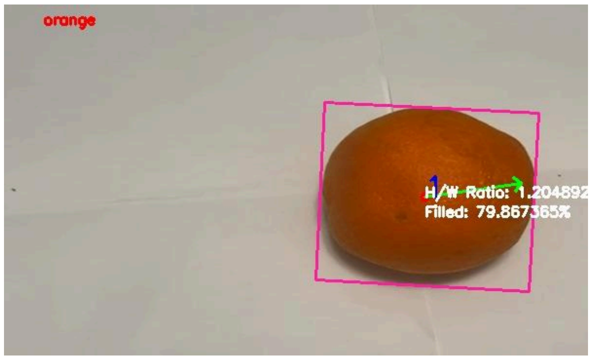




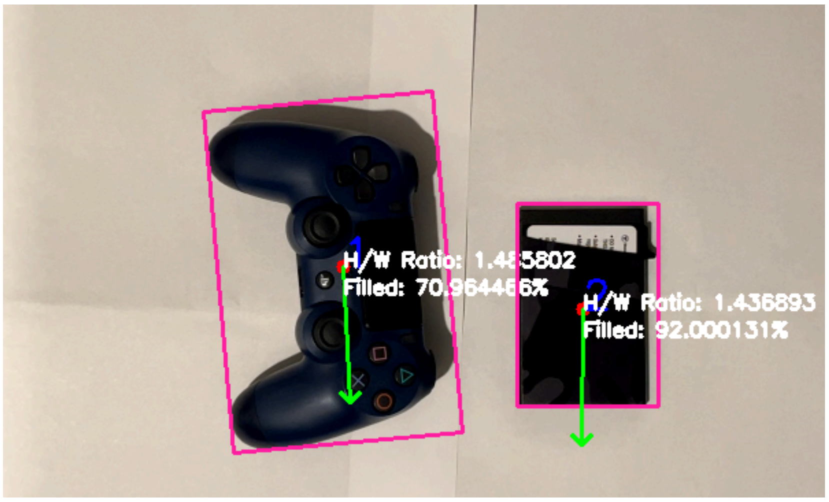
Extension -2
Extended database of objects:
We were able to classify more objects, below are the photos.







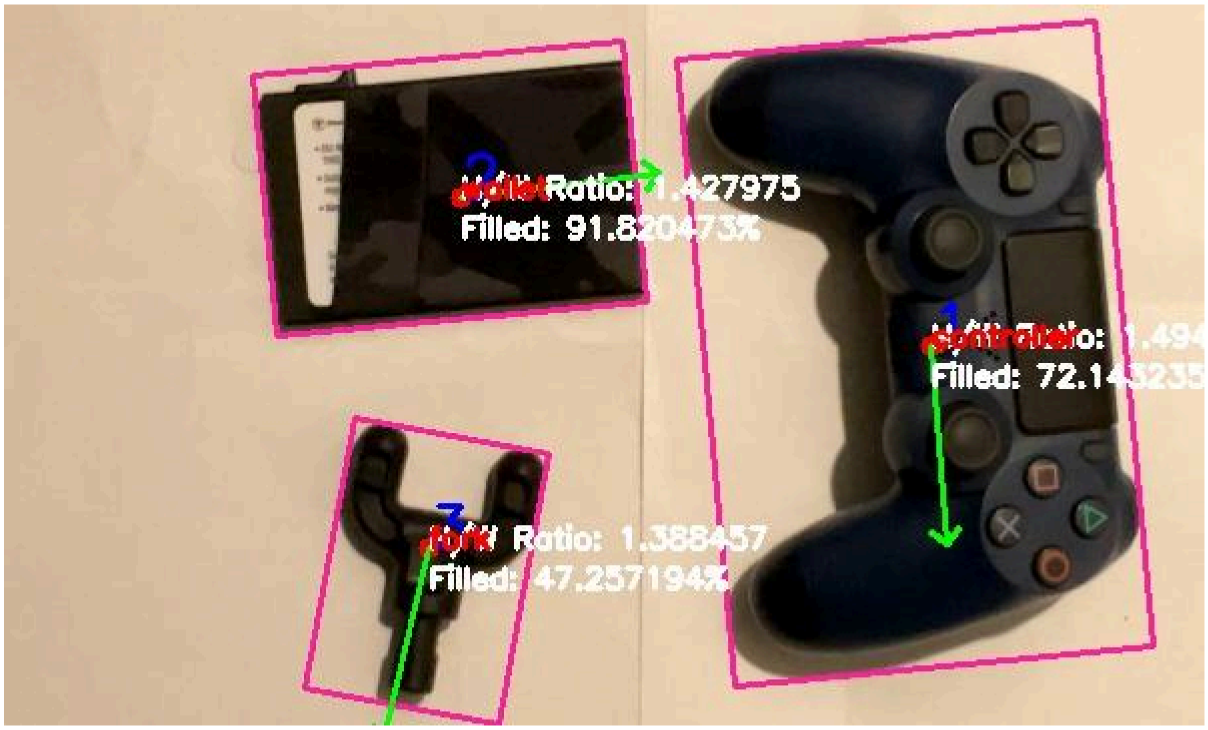
Extension -3
Simultaneously identifying features of multiple objects



Extension 4:Implemented Task -4 from scratch

Extension 5:Multiple object detection

In this extension, we have successfully performed multi-object detection. The image below illustrates accurate labeling for the detected objects, including the controller, fork, and wallet.



Acknowledgment:

We would like to express our gratitude to Professor Bruce Maxwell for his lectures. His teaching was instrumental in helping us think constructively about our solution. We also extend our thanks to Stack Overflow and to all the developers and contributors on the platform. The wealth of resources available there was invaluable during our debugging process. We appreciate the assistance of OpenAI's ChatGPT, which was useful for conducting deeper research on the general topic. Additionally, we recognize the contributions of the numerous technical writers on Medium who have shared their expertise in computer image processing through their writings.

References:

- PyImageSearch-https://pyimagesearch.com/2020/04/06/blur-and-anonymize-faces-with OpenCV-and-python
- GeeksforGeeks-https://www.geeksforgeeks.org/measure-execution-time-with-high-precision
- YouTube - https://www.youtube.com/watch?v=WXV06M1Qzlo
- YouTube - https://www.youtube.com/watch?v=8jLOx1hD3_o
- YouTube - https://www.youtube.com/watch?v=HS4c3kBEWr4
- AutomaticAddison - https://automaticaddison.com/how-the-Sobel-operator-works/
- OpenCV Doc - https://docs.opencv.org/3.4/db/d64/tutorial_js_colorspaces.html
- https://code.visualstudio.com/docs/cpp/cpp-debug
- https://www.reddit.com/r/computervision/
- https://en.wikipedia.org/wiki/Image_histogram#:~:text=In%20the%20field%20of%20computer,for%20peaks%20and%2For%20valleys.
- https://medium.com/@sasasulakshi/opencv-image-histogram-calculations-4c5e736f85e
- https://pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-images/#:~:text=The%20k%2DNN%20algorithm%20classifies,%E2%80%9Cvotes%E2%80%9D%20for%20each%20label