Project 3: Real Time Object 3-D Recognition

Atharva Pandkar

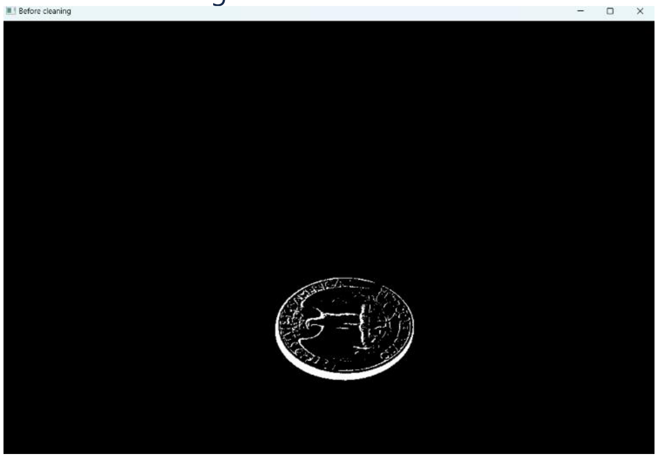
Brief description

The purpose of this project was to explore the field of real-time 2D object recognition by developing a system that can identify specific objects placed on a white surface, regardless of their size, orientation, or position, using a camera positioned overhead. The project involved several steps, including converting the image to a binary format through thresholding, removing unwanted noise, and identifying the object's region of interest. We then computed feature vectors of the objects and used them to train our system. Finally, we developed a recognition system that uses two different classifiers to identify objects in single images or video sequences in real-time. The system is capable of outputting an image with identified objects for single images and real-time video sequences.

Task 1: Threshold the input video

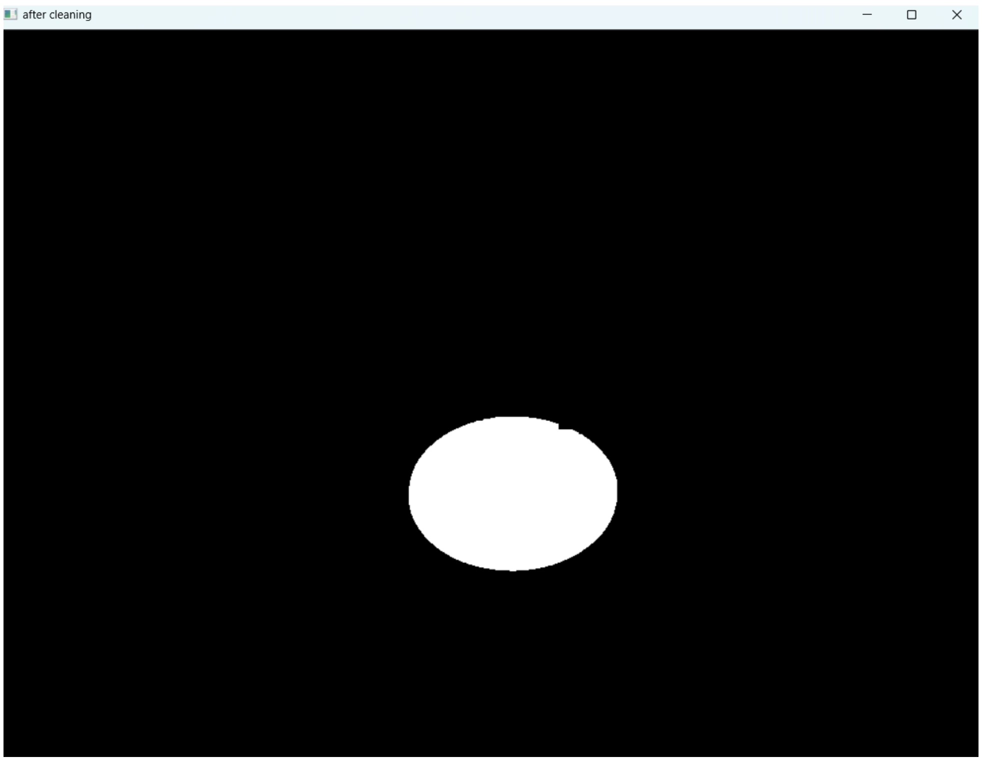
I have implemented a thresholding algorithm from scratch. This function apply thresholds on input image and return the result. The function takes in the RGB image, converts it to grayscale and then creates a binary image using threshold. The background will be black after the threshold application.

Thresholded Images:



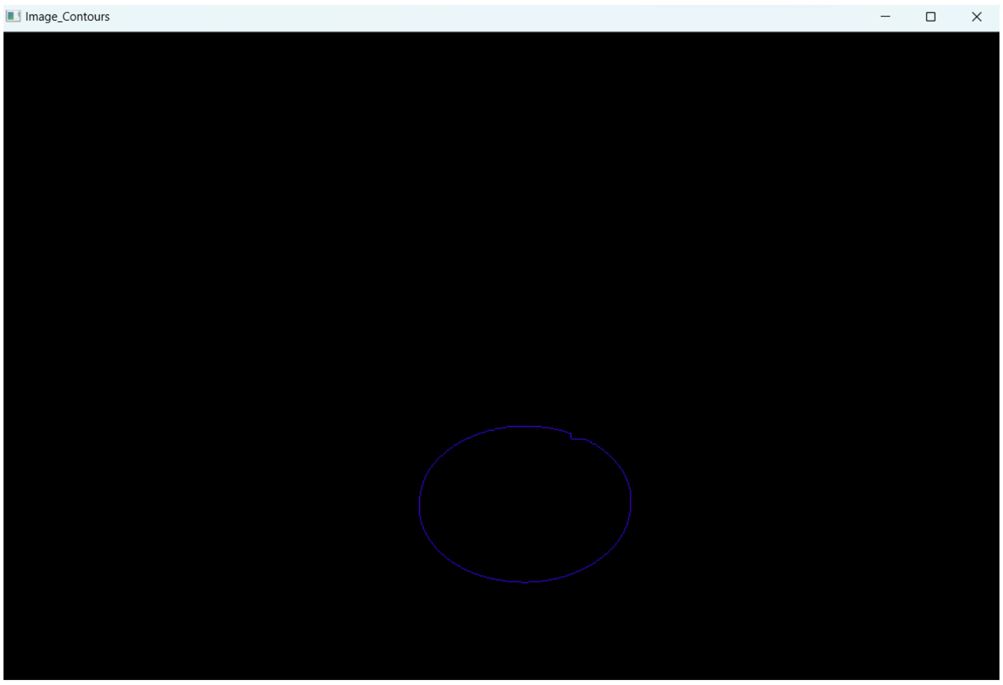
Task 2: Clean up the binary image

After obtaining a binary image through thresholding, it is common to observe holes and gaps in the object area and salt and pepper noise in the background. To address this, morphological operators such as erosion and dilation are often applied. Additionally, the morphologyEx function, which utilizes erosion and dilation as basic operations, can be used to perform more advanced morphological transformations. The output of morphologyEx is typically an outline of the object, achieved by taking the difference between the dilation and erosion of the image.



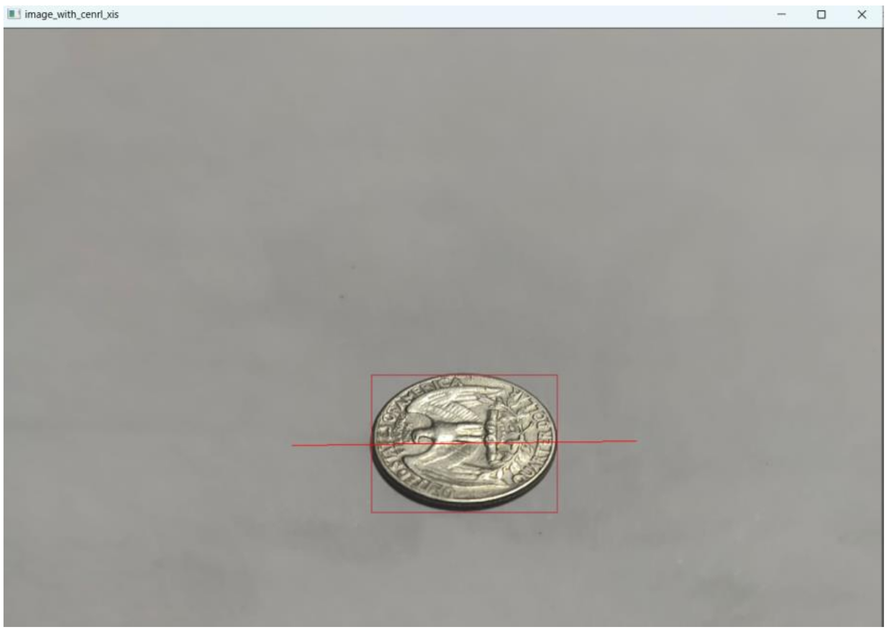
Task 3: Segment the image into regions

Although the binary images obtained from previous processing steps have been cleaned up, there may still be areas in the foreground that are not relevant to our analysis. To isolate and label the regions of interest, image segmentation is necessary. The connected components with stats function was used on the cleaned binary image to segment it into regions, and the resulting statistics were used to create bounding boxes that could be displayed on the image



Task 4: Compute features for each major region

To extract useful information from the segmented image, the findContours function provided by OpenCV was utilized to obtain the contours of the objects. Then, the minAreaRect function was applied to compute the rectangle area, extract its points, and draw an oriented bounding box using these 4 points. Subsequently, important features that are translation, rotation, scale, and reflection invariant were computed and stored in a database. These features include the percentage of bounding box filled, the aspect ratio, and the Hu moments.



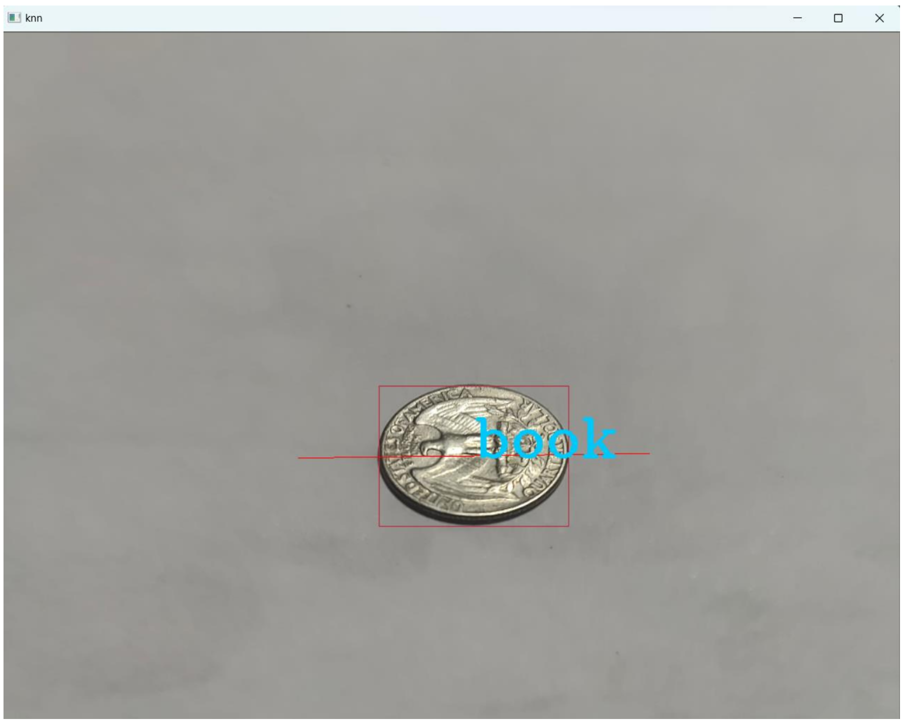
Task 5: Collect training data

To collect the features of all the objects in an image and store them in a database, a separate file named training.cpp was created. The user inputs the path of the images, and the system computes all the features of that image and stores them in the features_database.csv file. To accomplish this, csv_util header files provided by the professor for Project 2 were used.

For real-time training, the skeleton code provided for capturing video in Project 1 was utilized. The extracted features are then stored in the features_database.csv file, which now serves as the training data for object classification in images.

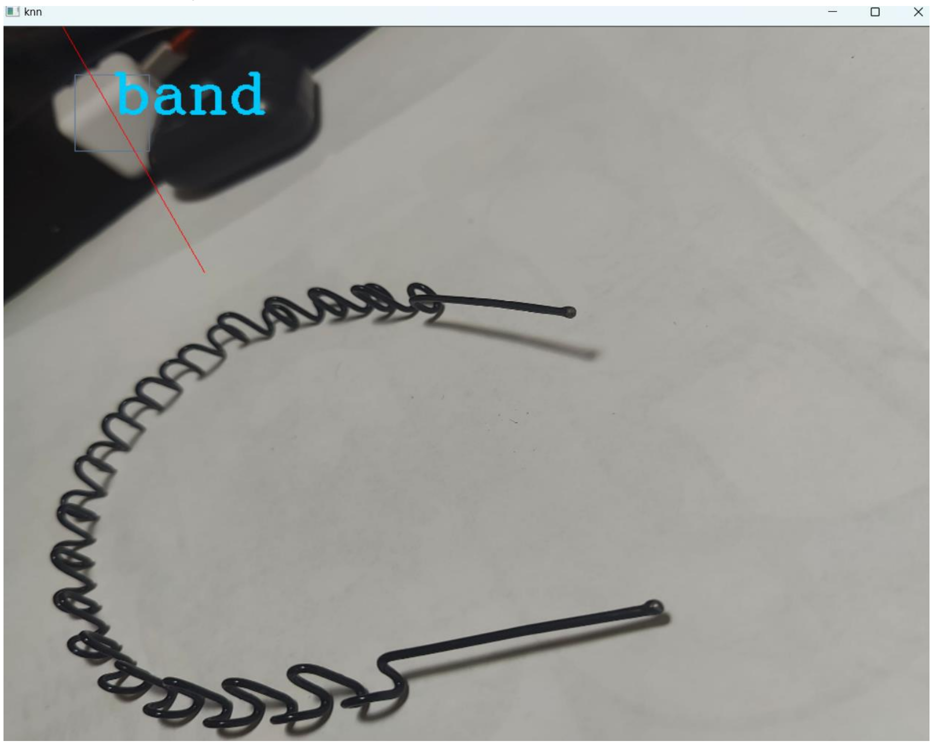
Task 6: Classify new images

To classify new objects, the distance between their feature vectors and all feature vectors stored in the features_database.csv file is calculated using a scaled Euclidean distance metric. The distances are then sorted, and the object is assigned a label based on the feature vector with the least distance



Task 7 Use a different classification

To achieve more accurate results, a KNN classifier was implemented. This function calculates the distances from all instances of each object, selects the top k distances, adds them to obtain the final distance from that object class, and then assigns the object the label of the object class with the least distance.



Task 8 evaluate your data

So the training aspect didn't go as planned. We have huge error margins. Displayed via confusion matrix.

	Band	Book	Bottle	chocolate	coin	fork	hammer	headphones	lighter	Mouse
Band	3					1	1			
Book		4		1						
Bottle			5							
chocolate		1		4						
coin					5					
Fork						5				
Hammer							5			
Headphones								2		3
Lighter									5	
Mouse								3		2

Task 9 create a working video of your file https://drive.google.com/file/d/1n9TWYZcu87OTmchJLCk05GwBgwqX1Tq_/view?usp=sharing

Reflection

This project has been a valuable learning experience for me in the field of computer vision, as it has enabled me to gain a deeper understanding of the various processes involved in object detection. I had the opportunity to experiment with different distance metrics and classifiers, which helped me understand their strengths and limitations.

The first three tasks were relatively easy and didn't take much time, but the later tasks required a lot of effort and time investment. While working on the project, I realized that the inbuilt functions provided were not entirely satisfactory, so I either discarded them or modified them to suit my needs by developing my own functions.

Overall, this project has increased my comfort level with using the OpenCV package with C++, and I'm grateful for the experience and knowledge gained.

Acknowledgment:

I would like to acknowledge Professor Maxwell's conducting coding session. I referred to the

Introduction - OpenCV Tutorial C++ (opencv-srf.com)

OpenCV: OpenCV modules

Stack Overflow - Where Developers Learn, Share, & Build Careers.