CS 5330 PATTERN RECOGNITION AND COMPUTER VISION
Project 3: Real-time 2-D Object Recognition

Ruchik Jani 002825482
Anuj Patel 002874710

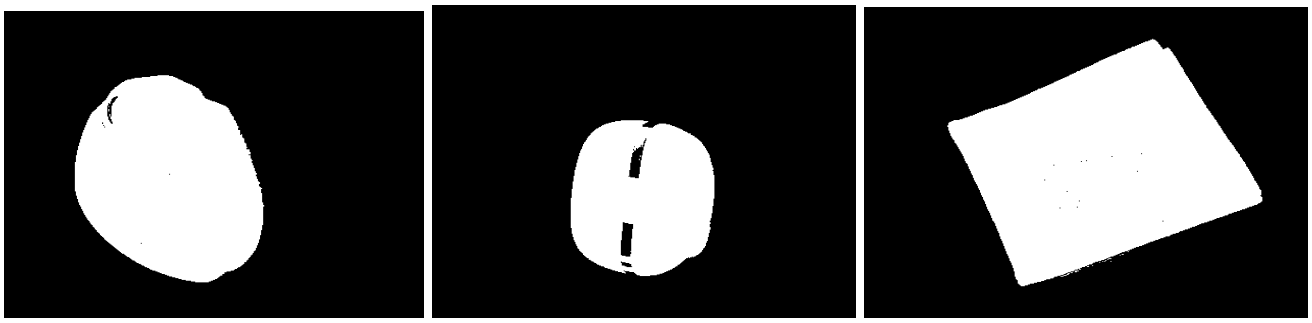
1) Project description:

The goal of this project is to create an Object Recognition (OR) system that can recognize and categorize objects in a live video feed. The first step is pre-processing the video by thresholding to separate objects from their background. This is further improved by blurring the images. The binary image so formed is then cleaned up using the shrinking morphological filter to improve its quality for subsequent processing by eliminating noise and filling in any gaps. The system then segments the cleaned image into regions representing different objects, using connected components analysis. For each significant region, it computes features such as shape, size, and orientation, crucial for distinguishing between objects. A key part of the project is collecting training data by labelling feature vectors from known objects and creating a database for future object classification. The system classifies new objects by comparing their features against this database, using methods like nearest-neighbour recognition.

2) Image descriptions for project tasks:

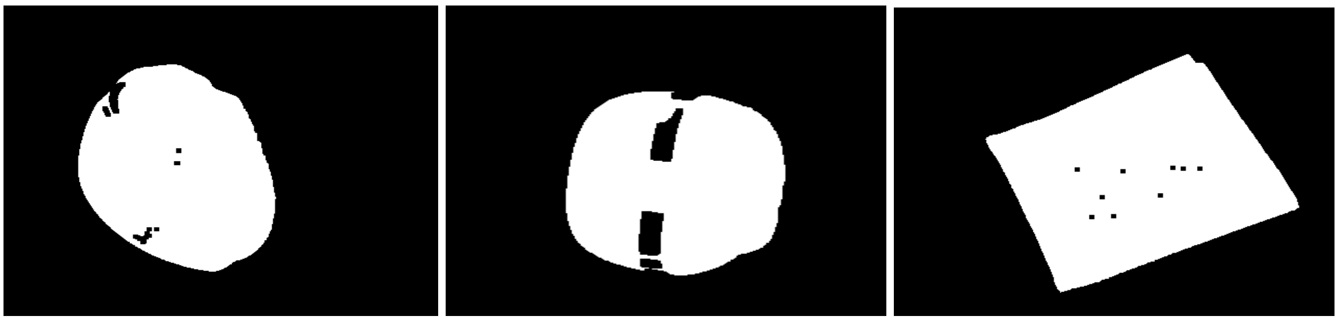
Task 1 - Threshold the input video:

Original Images


Thresholded Images

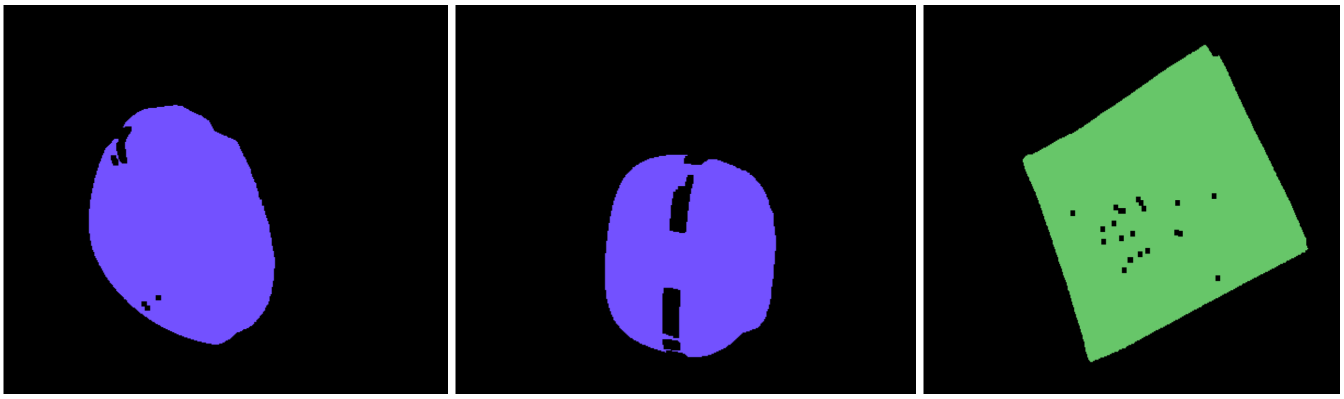
We have developed our custom thresholding function in this project. This code implements a method called dynamic thresholding using k-means clustering. It first converts the input image to grayscale if it's not already, then reshapes it into a single column to perform k-means clustering with two clusters. The mean of the cluster centers is computed as the threshold value. Next, binary thresholding is applied to the grayscale image based on this threshold value, effectively segmenting the image into two regions. The resulting binary image is then stored in the provided output matrix, and the threshold value used is printed out for reference.

Task 2 - Clean up the binary image:


Binary Images

We have used shrinking (also called erosion) on the thresholded image as it has some noise. Shrinking helps eliminate this noise by removing pixels from protrusions or thin structures. This helps in getting a more accurate shape of the object by smoothing out the boundaries of the object. We have written the code for shrinking from scratch. This code implements a custom erosion operation for binary images using a specified kernel. It iterates over each pixel in the input image, considering a local region defined by the kernel size. For each pixel, it checks if all the corresponding pixels in the local region, as defined by the kernel, are foreground pixels (255). If so, the output pixel is set to foreground; otherwise, it is set to background (0). This process effectively shrinks the boundaries of foreground objects in the image.

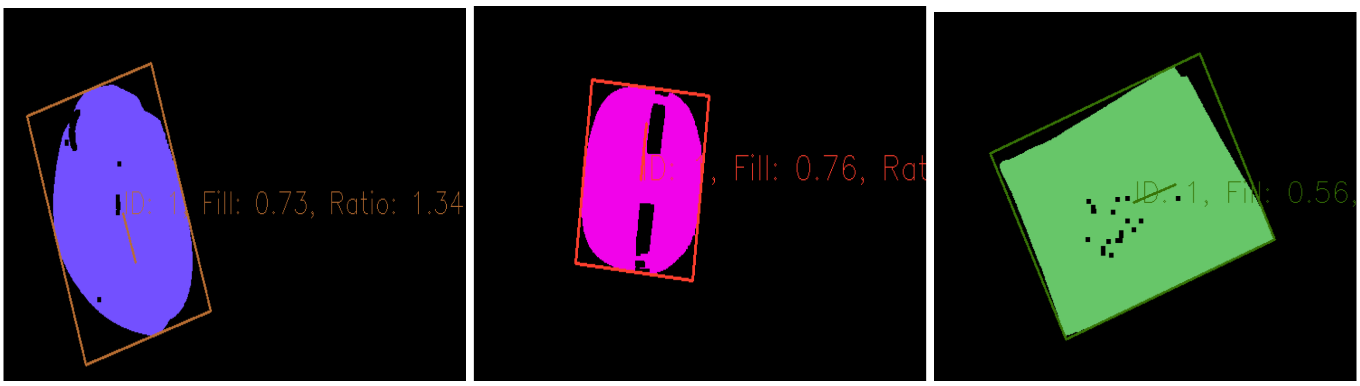
Task 3 - Segment the image into regions:



Segmented Images

For segmentation, we have configured the program to display segmented regions for 2 of the largest regions in the video frame. This was done to improve recognition accuracy. We have used the OpenCV function called 'connectedComponentsWithStats' for this task.

Task 4 - Compute features for each major region:



Segmented Images showing Region Features

For this task, we have built a function which computes the following region features: Area, Centroid, Orientation Angle, Bounding Box Ratio and Percentage Filled. Additionally, the function visualizes these features by drawing the oriented bounding box and centroid on the output image and annotating it with text displaying the region ID, fill percentage, and bounding box ratio.

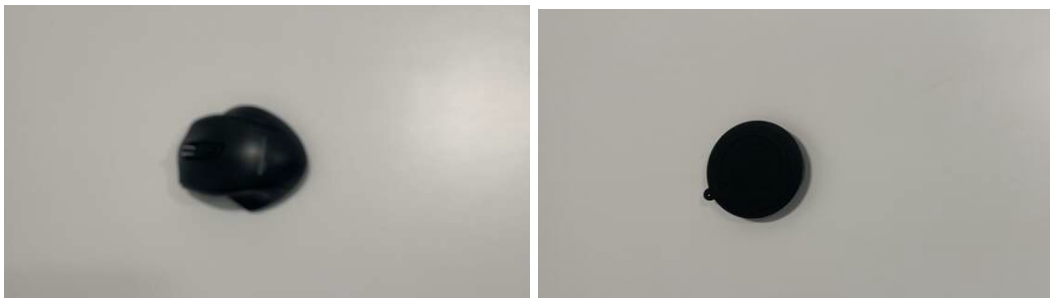
We are using an OpenCV function called moments() to compute a set of moments for the provided region of interest (ROI) in the image. Specifically, it computes central image moments, denoted by m**pq**, where **p** and **q** are the order of the moments. These moments are used in calculating the orientation angle. In this task, the function is calculating the following moments:

1. m**11**: orientation of the region.
2. m**20**: elongation or compactness of the region.
3. m**02**: shape of the region.

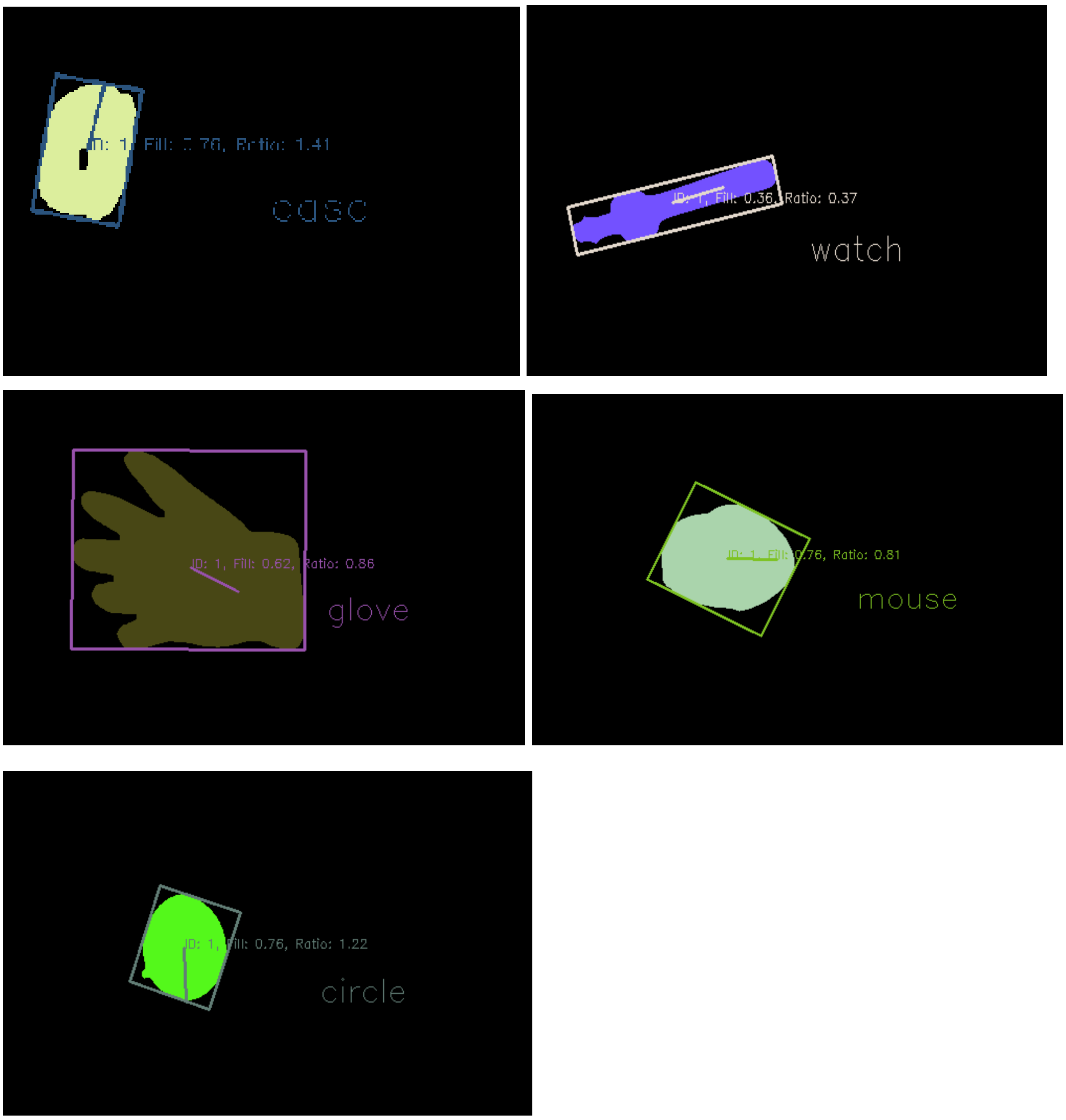
Task 5 - Collect training data:

We have collected training data from a training set which consists of images of different objects. The images of an object were captured at different orientations. We have developed a separate program for this data collection which iterates over images in a specified training set folder, loading each image and performing operations like thresholding, erosion, and segmentation to identify significant regions in the image. Region features such as area, centroid, bounding box ratio, and orientation angle are computed for each identified region and added to a feature vector which is stored in a CSV file. Additionally, it displays the original and the segmented image with region features for verification. The code also prompts the user to label the image so that the label can be stored along with other region features in the feature vector for recognition purposes.

Task 6 - Classify new images:



Original Images


ID: 1, Fill: 0.76, Ratio: 1.41
case
ID: 1, Fill: 0.36, Ratio: 0.37
watch
ID: 1, Fill: 0.62, Ratio: 0.86
glove
ID: 1, Fill: 0.76, Ratio: 0.81
mouse
ID: 1, Fill: 0.76, Ratio: 1.22
circle

Recognized Images

We used the nearest neighbour approach to classify objects w.r.t. Euclidean distance metric. Given a new object represented by its features (newObj) and a database of objects with known labels (database), the classifier function iterates through each object in the database. For each object, it calculates the distance between the features of the new object and the features of the current object in the database using the distance metric. It then identifies the object in the database closest to the new object in terms of feature similarity and returns the label of the closest object as the predicted classification for the new object.

Task 7 - Evaluate the performance of your system:

Confusion Matrix for the 5 objects:

True Label → Predicted Label ↓	**Mouse**	**Circle**	**Case**	**Watch**	**Glove**
Mouse	2		1		
Circle		2			
Case	1	1	2		
Watch				3	
Glove					3

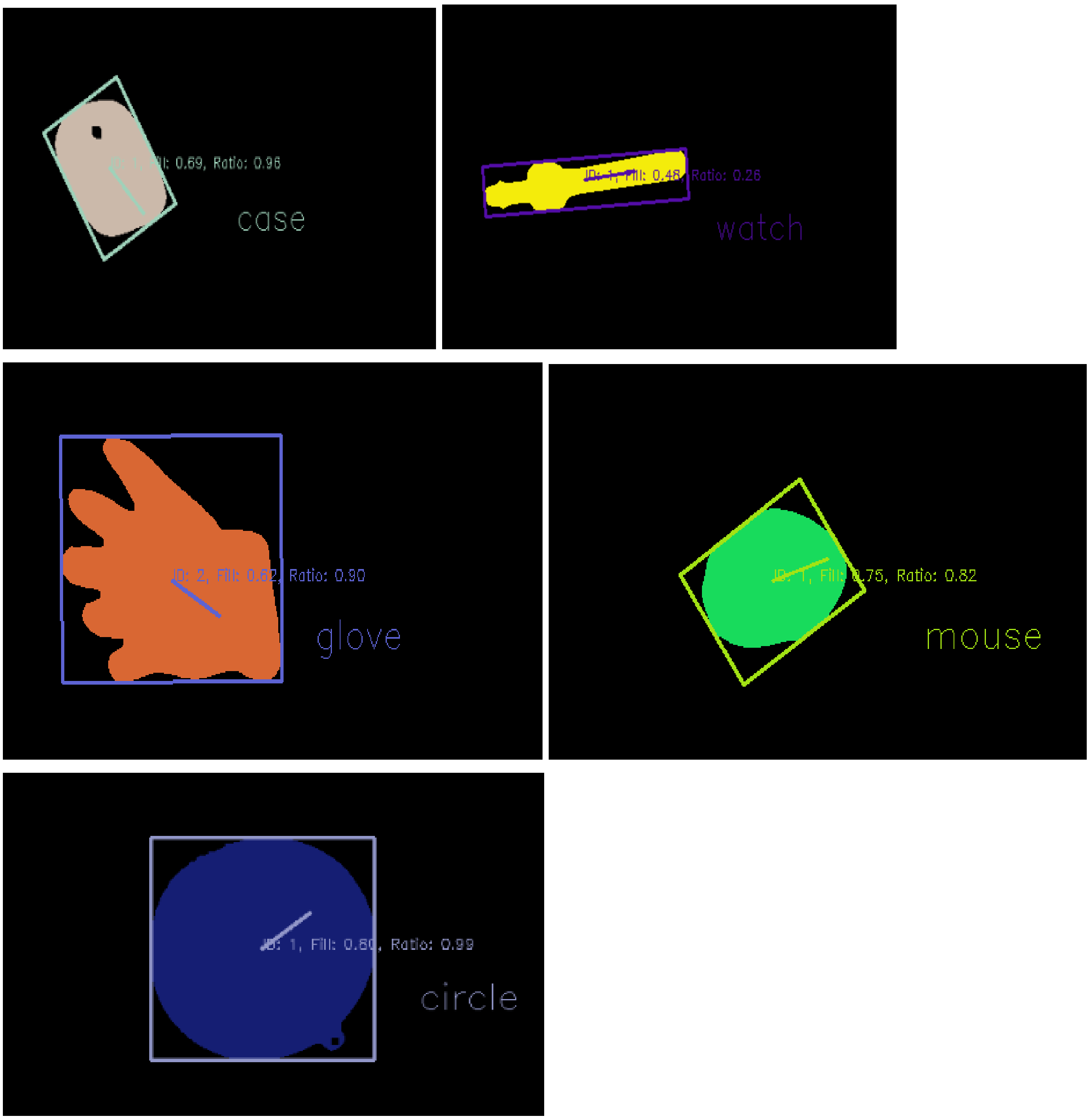
We tested our classifier on 3 different images of each object in different positions and orientations. In most of the cases, the classifier gave accurate results except for Mouse, Circle and Case. It wrongly detected mouse and circle as case, and case as mouse.

Task 8 - Capture a demo of your system working:

This is the link to the demonstration video of our system with a live video stream of the object captured from our phone. We were using the DroidCam mobile app to carry out this task. The first run of the program involves the Nearest Neighbour classifier, and in the second run, the program is executed with the K-Nearest Neighbour classifier:

https://drive.google.com/file/d/1mol3Z8MxhSMIhYUqEfuX8jcwVSrL_huF/view?usp=sharing

Task 9 - Implement a second classification method:


ID: 1, Fill: 0.69, Ratio: 0.96
case
ID: 1, Fill: 0.48, Ratio: 0.26
watch
ID: 2, Fill: 0.62, Ratio: 0.90
glove
ID: 1, Fill: 0.75, Ratio: 0.82
mouse
ID: 1, Fill: 0.80, Ratio: 0.99
circle

Recognized Images

We have selected the K-Nearest Neighbours approach as a second classifier for our system. This is because the dataset we have built is not large enough to drain a deep neural network (DNN) effectively. Also, the K-NN classifier required less computational power to operate as compared to a DNN.

Given a new object represented by its features (newObj) and a database of objects with known labels (database), the function calculates the distances between the features of the new object and all objects in the database. It then selects the k nearest neighbors based on these distances and counts the occurrences of each class label among the nearest neighbors. Finally, it returns the class label that appears most frequently among the nearest neighbors as the predicted classification for the new object. We have selected the value of K=3.

The KNN classifier is mostly accurate. However, it fails to recognize some objects properly. The Nearest Neighbour approach works well since the dataset is small and all the objects are quite different from each other.

,

3) Short reflection on learnings:

During this project, we learned a lot about how to build an object recognition system from scratch. We learned a lot by developing custom functions for thresholding and morphological filters on our own. By developing the training system, we created a database of feature vectors which was used in object recognition using different classifiers like Nearest Neighbour and K-Nearest Neighbour. We learned how to determine the accuracy of the system by building a confusion matrix. Overall, the project gave us a lot of exposure to different tasks related to a vision system and integrating them to create a single product.

4) **Acknowledgement:**

We have referred to the following sources to complete this project:

1. https://opencv.org
2. Computer Vision by Linda Shapiro and George Stockman
3. www.google.com
4. https://www.youtube.com/watch?v=oGkxJQ9SB8M