CS5330: Pattern Recognition and Computer Vision

Project 3: Real-time 2-D Object Recognition

Name: Krishna Prasath Senthil Kumaran

Email: senthilkumaran.k@northeastern.edu

Aim of the Project:

This project aims to create a system for 2D object recognition that can identify a specific group of objects when placed on a white surface. We are using ***"Nearest neighbour and K-NN classifiers"*** to classify the group of objects and a distance metric among ***"Eucledian or Manhattan"*** is passed along with the command line argument, where it calculates the distance between two feature vectors. After classifying the objects, the program asks for character input from the user, where each character has a value, and the value is labelled on each object in the live video feed. Then, if we quit the program by pressing the letter ***'q'*** the features of the object are written to a ***.csv*** file and are used for comparison.

Task 1: Threshold the input video

The frame of the video is converted into grayscale using ***cvt color***, an in-built function of OpenCV. Then a threshold is set, if the pixel value is equal to or less than the threshold it is set as foreground and the rest of the pixel as background. This task helps us to transform the frame into a binary image. Fig.1 shows the output of thresholding.

Fig.1 Thresholding of a notebook and tape.

Task 2: Clean up the binary image

The binary image that we get from thresholding is cleaned up using morphological closing. A 25x25 rectangular binary image is used to patch the holes that appear on the binary image because of

thresholding. Then it is eroded to bring the image back to its original form. Fig. 2 shows the output of morphological closing.

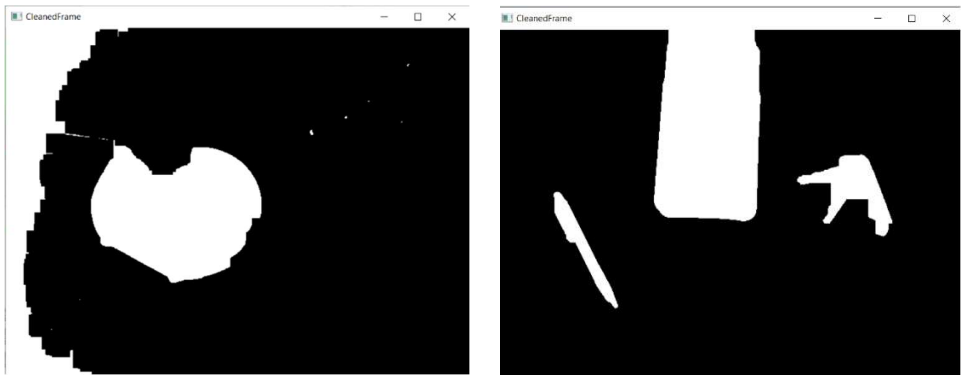

Fig.2 Cleaned frame random objects.

(Left: tape, Right: pen, calculator, keys)

Task 3: Segment the image into regions

In task 3, we use another OpenCV's in-built function called ***"connectedComponentsWithStats"*** which helps us extract the image's three largest regions. Overall, we try to identify three objects simultaneously. We pass the image, the labelled regions that we get from cleaning, features of the region in a Mat and the centroids of each object as the input for this function. After this, all the processed regions with a colour to show it has been segmented and the image is returned.

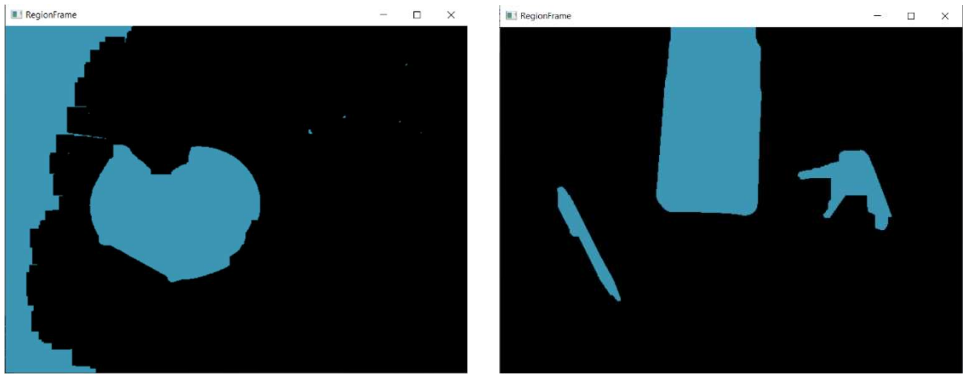

Fig.3 Top 3 matches for multi-histogram matching.

Task 4: Compute features for each major region

In task 4, we draw a boundary box around the segmented region of each object. The boundary box is calculated based on the moments and HuMoments. Moments calculates the region, area, moment, and central moments. But moments don't help us achieve the target so we use HuMoments to calculate seven central moments. As we are calculating multiple objects we are

showing them in one image. Fig.4 shows the region being extracted during the training and drawing of the boundary box and line based on the object's angle.

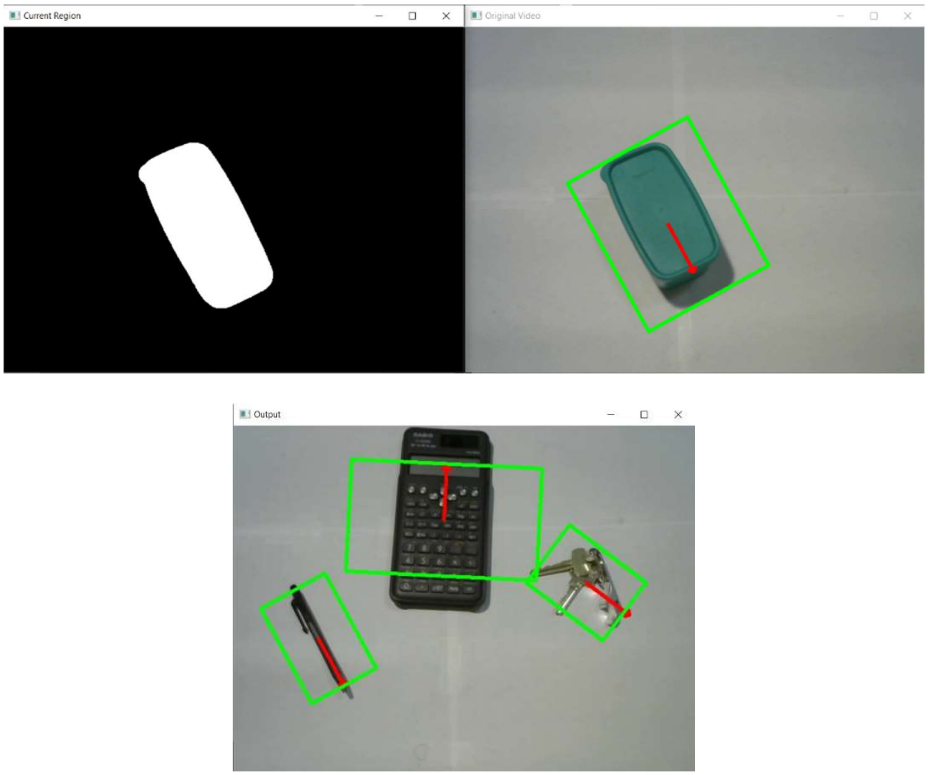

Fig.4 After drawing a line and the boundary box.

Task 5: Collect training data

In task 5, we collect the features of the objects and store them in a ***.csv*** file. To do that we press the letter ***"t"*** as the user input which makes the system start training. In training, the algorithm shows the region one by one of all three objects, and the algorithm asks for user input where we input the character. Each character has a value, which is the name of each object. For each iteration, the algorithm extracts the features of objects under a class name, and it is stored in a ***.csv*** file.

Task 6: Classify new images

In task 6, we pass an argument in the command line where we specify the location and the name of the ***.csv*** file and we specify the classifier that we use, either ***'n'*** or ***'knn'*** and then a distance metric. For distance metric, we have ***'Eucledian distance'*** and ***'Manhattan',*** to specify either one of these distance metrics we enter the letter ***'e'*** or ***'m'*** in the command line. Then based on these command line inputs the system is trained based on the specified parameter and the output is stored

in a database. Fig.5 shows the implementation of the nearest neighbour classifier along with the euclidean distance metric.

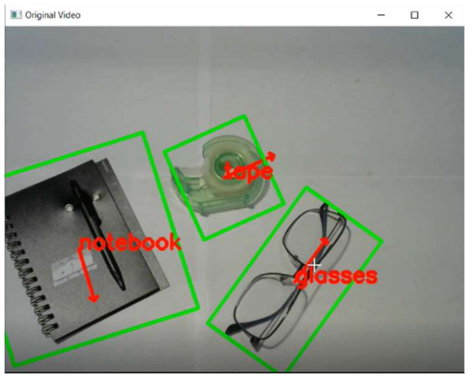

Fig.5 Implementation of nearest neighbour with Euclidean distance metric

Task 7: Implement a different classifier

We have implemented the ***'K-NN classifier'*** as another classifier. This classifier can be used by passing ***'knn'*** and the distance metric in the command line as input. Then we train the system by pressing ***'t',*** where the region is shown five times and the feature of the object is extracted five times and is stored in a ***.csv*** file. Fig.6 shows an example implementation of K-NN where the region was shown three times and the feature of the calculator is extracted when it is placed in different orientations. After three iterations, the label or the name of the object is displayed with its boundary box in real time.

Fig.5 Implementation of K-NN

Task 8: Evaluate the performance of your system

To evaluate the performance, we try to identify the object 5 times. Objects with similar features and area were confused and labelled wrong. Fig.6 is the confusion matrix that we get after trying to identify each object 5 times.

	pen	headphone	eyeglass	wrench	calculator	box	key	notebook	belt	credit card	tape	bottle
pen	1									2		
headphone		3										
eyeglass			2							1		
wrench				3								
calculator				1	1			1				
box			1			1		1				
key	1					1	1					
notebook								3				
belt									3			
credit card			1				1			1		
tape	1										2	
bottle												3

Fig.6 Confusion Matrix after three iterations.

Task 9: Capture a demo of your system working

1) Demo video for Nearest Neighbour classifier: nearest_neighbour_clip.mp4
2) Demo video for K-NN classifier: knn_clip.mp4

Extension 1: Added more than 10 objects to the database using K-NN

Objects were placed one at a time in the white space and K-NN was performed for different orientation for every iteration. Fig.7 shows the training of each object

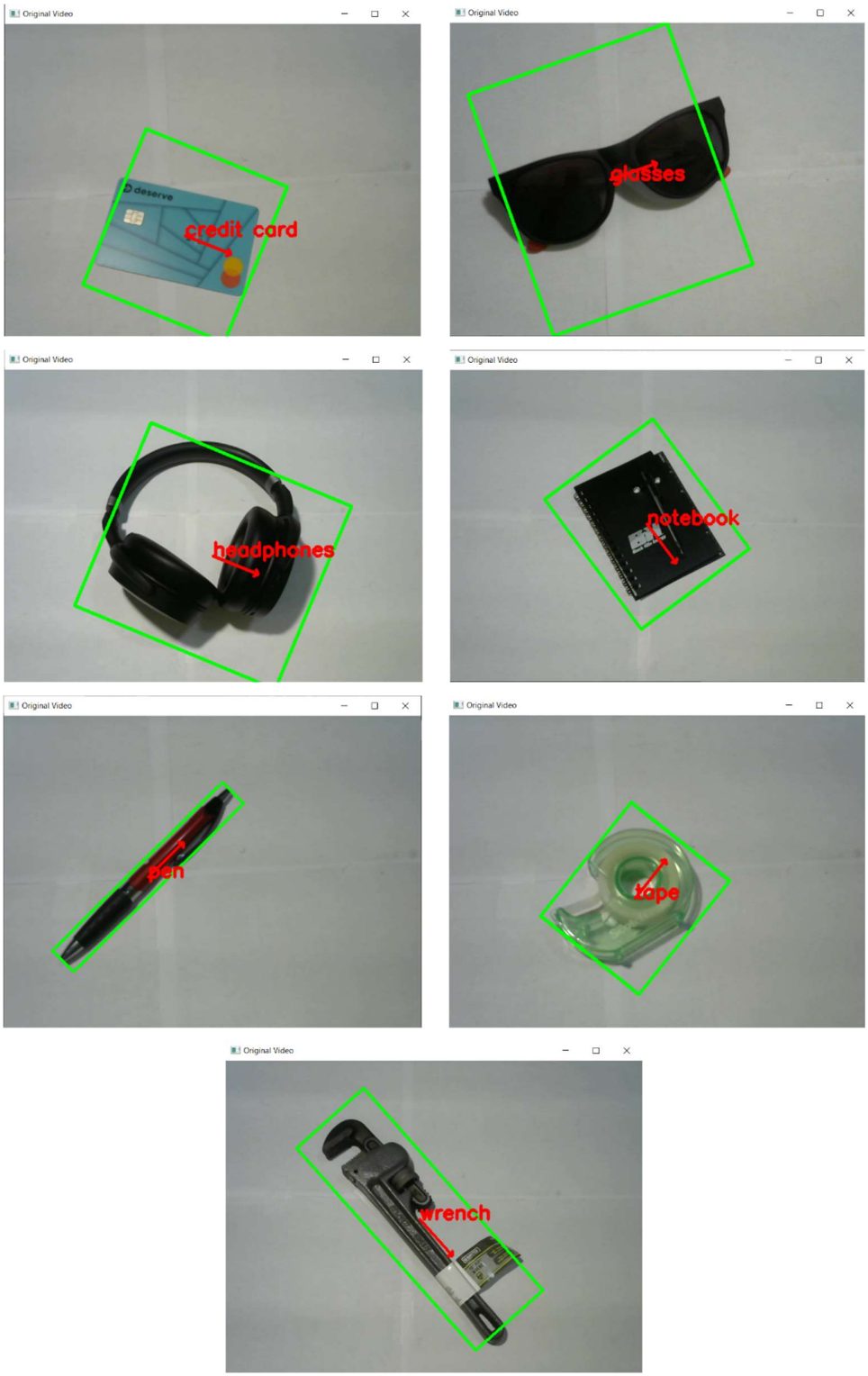

Fig.7 Training for each object using K-NN classifier.

Extension 2: Added Manhattan distance metric to both Nearest Neighbour and K-NN

As an add-on feature, I have added ***"Manhattan distance"*** along with Euclidean distance. This option can be accessed through command line by entering the letter ***'m'***. Instead of Euclidean function calculating the distance for the features, Manhattan will be used. Fig.8 illustrates the two images where both Euclidean and Manhattan distances are used in Nearest Neighbour classifier.

Manhattan distance was not fully successful as threshold was not neutral for Euclidean or Manhattan.

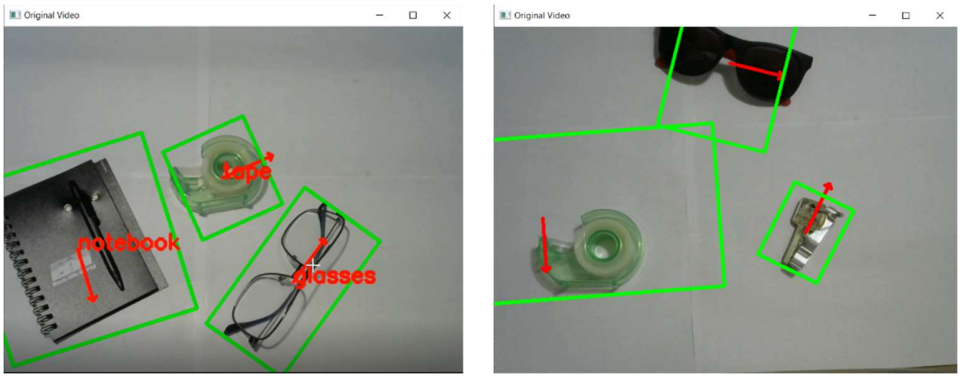

Fig.8 Left: Euclidean distance Right: Manhattan distance.

Extension 3: Identifying multiple objects.

Based on the camera frame size, we were able to place only three objects. It was possible to identify all three objects. We mention the number of regions and the system checks for that many regions and then we train each region one at a time and label them. Fig.9 shows the objects that were identified and labelled.

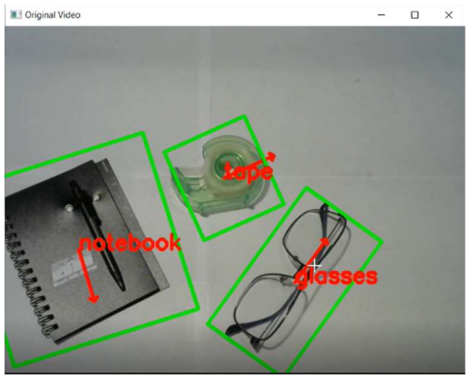

Fig.9 Three objects were identified simultaneously.

Acknowledgement of resources or people consulted for this project:

- OpenCV Tutorials: https://docs.opencv.org/4.5.1/index.html
- Manhattan distance reference: https://iq.opengenus.org/manhattan-distance/