Project 3: Real-time 2-D Object Recognition

Name: Dev Vaibhav

Email: vaibhav.d@northeastern.edu

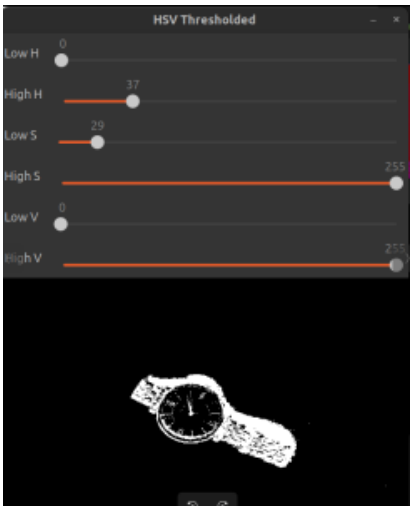
Class: CS 5330 Pattern Recognition and Computer Vision

Description: This project is about building a real-time 2D object recognition system that can recognize a set of objects in a rotation, scale, and translation-invariant manner. The goal is to have the computer identify a specified set of objects placed on a white surface detected by a camera looking straight down. The processes involved are: thresholding binary image, cleaning binary image, and then segmentation of image/video stream. Feature vectors invariant to scale and orientation are chosen viz. Hu moments (0-5, excluding 6 which is not rotation invariant), along with percentage filled area and aspect ratio of the oriented bounding box. Closest neighbor is calculated from the DB and a label is assigned to the target image.

The below setup is used to capture the live footage or train the data set.

Task 1) Threshold the input video: Thresholding means separating the image pixels into foreground and background. It can be done in various ways. I tried implementing this using 3 variations: **(ALL FROM SCRATCH, One method can be considered for Extension)**

1. I am converting live footage to HSV, making highly saturated pixels in the HSV frame darker in the original image. Then converting this modified color image to greyscale using my own function (re-used function from Project 1) and applied a threshold value on the greyscale image (found experimentally) which outputs a binary image. This approach is used for further processing.
2. I applied thresholding on the HSV image (using the cv::inRange function by placing trackbars on the GUI window. Playing with it was fun. Although this approach was dumped after finding method # 1 to be more effective, it was the first choice. The code for this part is commented in "mode.cpp" , line # 698



1. Found the histogram for the HSV image to get a better idea about the color distribution in the image to pick the threshold values intelligently. Although this approach was dumped after finding method # 1 to be more effective, it was the second choice. The code for this part is commented in "mode.cpp" , line # 681 and function is present in "filter.cpp" line # 701.

Task 2) Clean up the binary image: The binary image obtained after thresholding had very few spots and holes. **Grassfire transform is implemented from scratch** (can be used as an extension) using a single function that can compute distance transform and perform either erosion or dilation. Dilation did not seem necessary while running the trials. Hence, this part is skipped in the processing. Although, the function can still handle this.

Task 3) Segment the image into regions: I used the cv::connectedComponentsWithStats() function to identify the components in the binary image.

1. I created a mask to include only those regions whose area is greater than 600 (found by experiment)
2. These regions form the contour to segment the image.
3. Found the oriented bounding box (OBB) of each region and filtered them based on two conditions.
    a. The region's centroid should not be near the edge of the image.
    b. The oriented bounding box's corners should not be near the edge of the image.
4. This OBB can be used for extracting feature vector.

Task 4) Compute features for each major region: I wrote a function to compute the feature vector of each region. It consists of 9 elements

1. Hu moments (7 in count): 1-6 of which are scale, translation, and rotation invariant. These are log normalized.
2. Oriented bounding box percentage filled ration
3. Oriented bounding box aspect ratio

Task 5) Collect training data: The csv file can be computed from a directory of images that are already labeled by their name or new images can be captured during runtime by entering testing mode by typing "./binProcess t"

1. A bunch of images (~5 for a class) were taken task from the camera's live feed if key 'n' is pressed. The user is then asked to input the label. This label along with its feature vector is saved in the CSV file. Apart from this, the image is also saved to the disk.
2. Computed feature vectors for all labeled images: HuMoments, percentage filled ratio, and aspect ratio.
3. Append the feature vectors into a CSV file.

Task 6) Classify new images: If the minimum scaled euclidean distance of the target image with the DB is greater than a threshold (found empirically), it is considered a new image. The user is then asked to input the label, the image gets saved to the disk along with its feature vector.

Task 7) Implement a different classifier: Implemented KNN (K=3) from scratch. Unique labels are extracted from the CSV file along with their indexes on which the feature vector is stored. Since, it is a multi-class problem, for each class, the distance of the target image with the class' image is calculated and the sum of minimum K is calculated which denotes the distance to that class. The class with minimum distance is selected as the label for the target image.

Task 8) Evaluate the performance of your system: Each object is tested 5 times under the camera and tested if a good match is found or not. Most of the weight lies along the diagonal. We can see that for some objects, the major weight does not lie along the diagonal and this could be rectified by adding more instances of the object to the database, better lighting conditions, better hardware, and fine-tuning the algorithm more.

The most interesting object is the robot toy which is white in color. Thus, the algorithm found it difficult to calculate the contour (changing constantly) in the live feed. However, for a single image input, it is working fine.

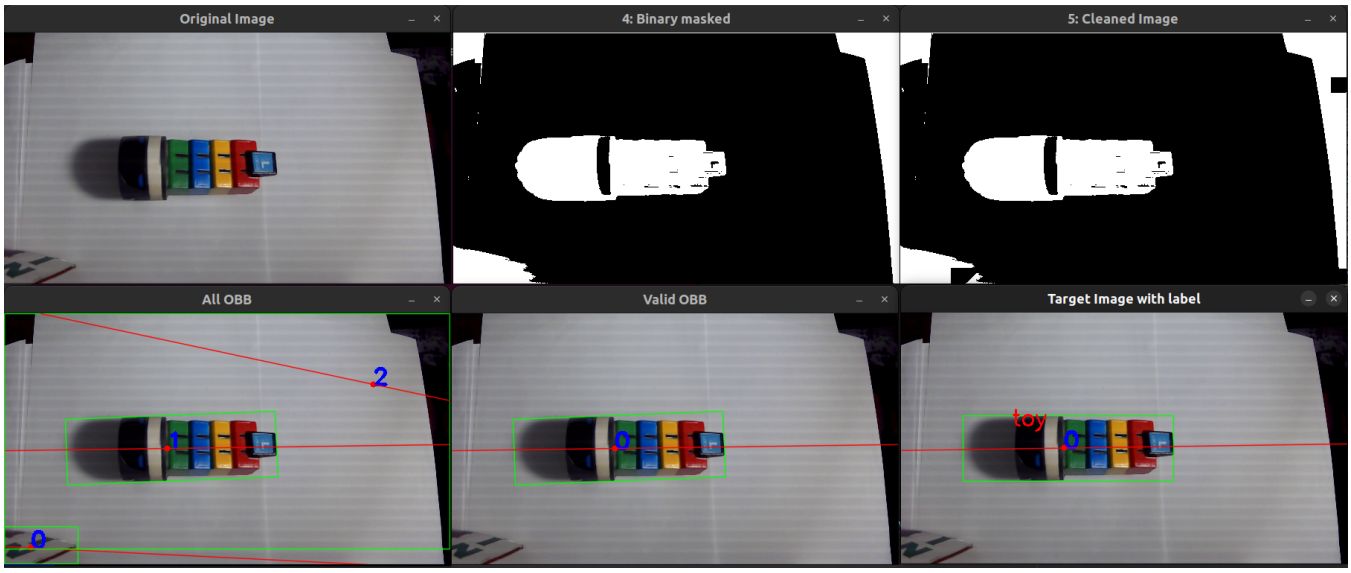
Marker Eraser also looks similar to the axe deo in terms of OBB shape and other quantities. And to the top of that, because of varying lighting conditions, results are incorrect for this object.

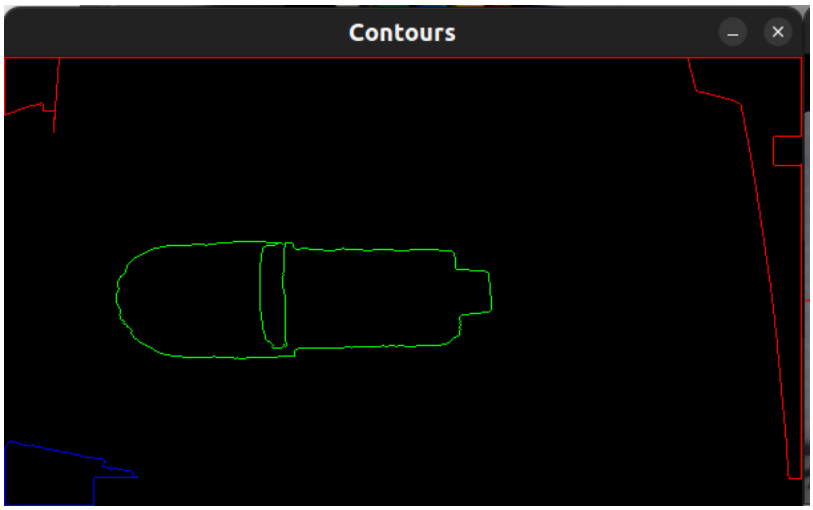
	DETECTED LABEL	mouse	paper_clip	butter_knife	marker_eraser	pen	lighter	toy	ball	statue_liberty	specs_case	fork	robot_toy	axe_deo	cloth_clip
REAL LABEL	**Number of iterations**														
mouse	5	5													
paper_clip	5		4										1		
butter_knife	5			5											
marker_eraser	5				1									4	
pen	5			1		4									
lighter	5						4				1				
toy	5							2		1	1			1	
ball	5	2							3						
statue_liberty	5									5					
specs_case	5										4			1	
fork	5											5			
robot_toy	5												2		3
axe_deo	5										2			3	
cloth_clip	5						1			2					2

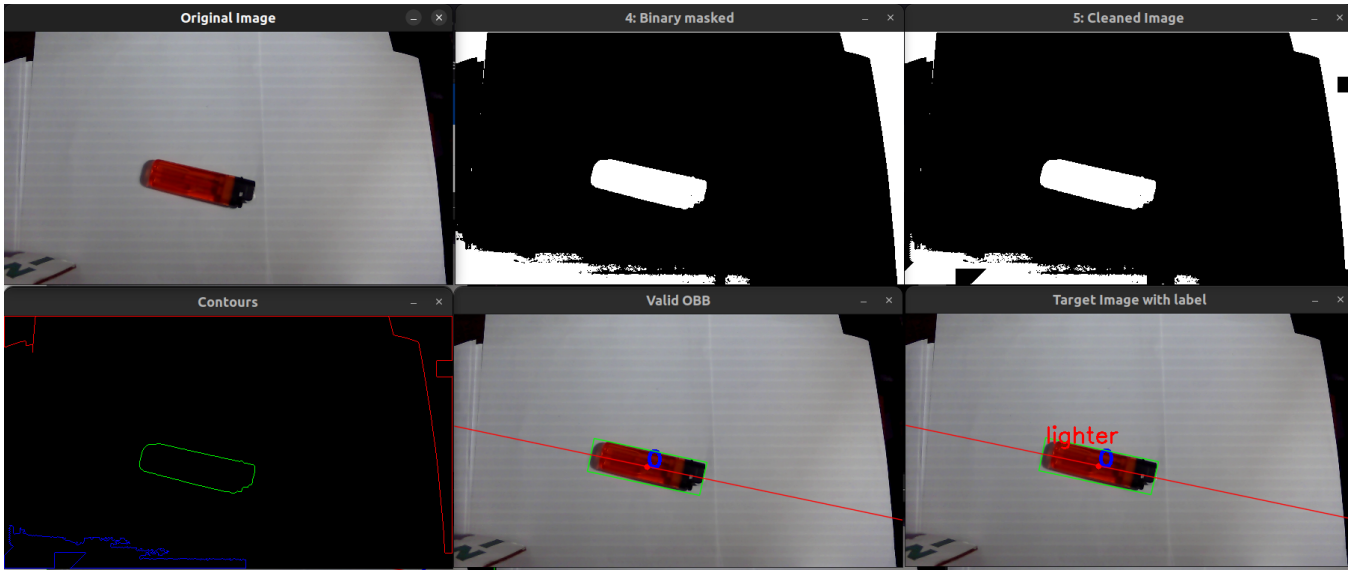
Task 9) Capture a demo of your system working: https://youtu.be/uIPqgtcVaUQ

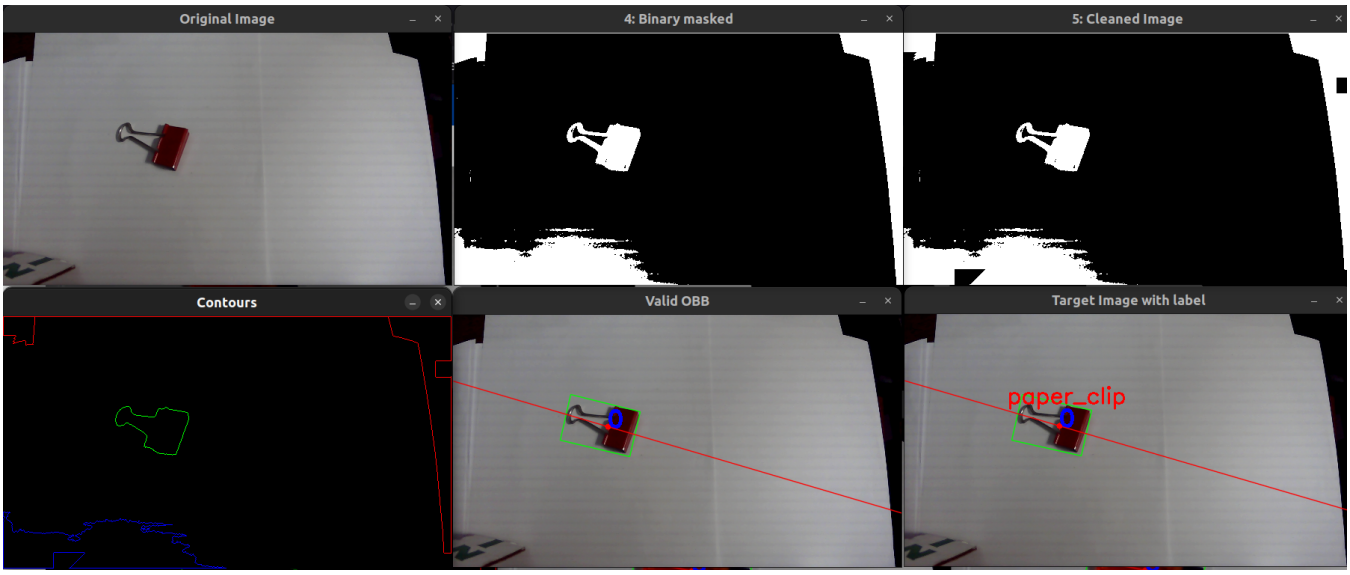
Live testing results depend on the lighting quality. If it fluctuates, then the results vary and might show incorrectly. To improve the system performance, more images can be added to the database or better hardware can be used for image capturing.

The combined result of all the tasks:


Original Image
4: Binary masked
5: Cleaned Image
All OBB
Valid OBB
Target Image with label
2
1
0
0
toy
0


Contours


Original Image
4: Binary masked
5: Cleaned Image
Contours
Valid OBB
Target Image with label
0
lighter
0

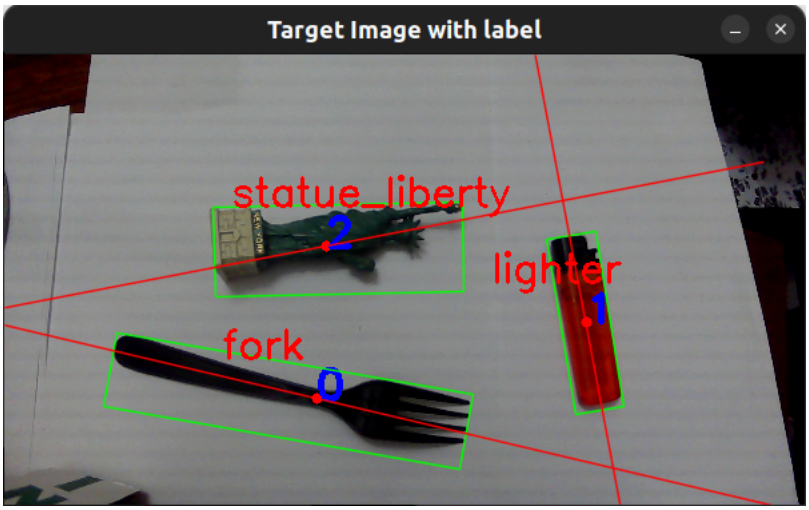


EXTENSIONS:

1. GrassFire Transform: Growing and Shrinking (from scratch):

1. Implemented a single function for grassfire transform which can computer the distance transform in either 4 or 8 connected and can perform either erosion or dilation.
2. Calculated manhattan distance of foreground from background and vice-versa
3. The function can also perform a max number of erosion and dilations.

2. Multiple object detection (from scratch) from both live video and image: Calculated feature vector of all the valid OBB in the image, found the best match from the DB and displayed it on the screen.



3. More than 10 objects: I took ~5 images of 14 objects for this task. More images can be taken to improve the accuracy of the system

4. Different modes of operation of the system: The program has three modes of operation

1. Basic training mode(b): Reads images from a directory and writes feature vectors to a CSV file
2. Object recognition mode(o): Either take image input from the user or live feed and finds the closest match in the DB
3. KNN classifier mode(k): Takes image input from the user and finds the closest match in the DB using KNN search
4. Testing mode(t): Takes live feed from the camera and is used to tune and test the system. This mode is untidy to look at.

4. Add a new object to the DB: If a new object (detected based on a threshold value found empirically) is placed in front of the camera and the user presses the 'n' key, then the program asks for a label from the user, then the feature vector and the image is saved to the CSV file and disk.

5. Comparison of different distance metrics: I computed three distance metrics from scratch without using any library and tried to compare their performance. It looks like the scaled euclidean and Manhattan L1 norm performed equally well for this dataset. Chi-Square performed poorly.

	DISTANCE METRIC	Scaled Euclidean	Manhattan L1 norm	Chi-Square
OBJECTS				
mouse		YES	YES	YES
paper_clip		YES	YES	NO
butter_knife		YES	YES	YES
marker_eraser		YES	YES	NO
pen		NO	NO	YES
lighter		YES	YES	YES
toy		YES	YES	NO
ball		YES	YES	YES
statue_liberty		YES	YES	NO
specs_case		YES	YES	NO
fork		YES	YES	NO
robot_toy		YES	YES	NO
axe_deo		YES	YES	NO
cloth_clip		YES	YES	NO
% Accuracy		**92.86**	**92.86**	**35.71**

REFLECTION:

I learned and implemented the fundamentals of 2d object recognition from an image/ live video feed. Got to know about different properties of a region such as moments, centroid, central moments, central axis angle, HuMoments, bounding box size, percentage filled, and aspect ratio of oriented bounding box. I gained a deeper understanding of different opencv functions and came to know some limitations of a few.

Acknowledgment:

I'd like to thank the resources listed below for their assistance in understanding and referencing various concepts while working on this project. I'd also like to express my gratitude to Professor Bruce for his great video explanations of these ideas. I have mentioned many more references inside the code marked with "Src: "

https://learnopencv.com/shape-matching-using-hu-moments-c-python/

https://answers.opencv.org/question/8871/how-to-calculate-humoments-for-a-contourimage-in-opencv-using-c/

https://answers.opencv.org/question/120698/drawning-labeling-components-in-a-image-opencv-c/

https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html

https://www.geeksforgeeks.org/chi-square-distance-in-python/