Real-time 2D Object Recognition

The aim of this assignment is to perform real-time object recognition using shape features such as moments which are scale, translation, and rotation invariant. To do this we perform various tasks as follows-

Task 1 – Threshold the input video (done without using OpenCV function)

To do this we plot the histogram of our image in 2D and then divide the histogram in two areas. This division is done using the mean value between the 2 maxima of intensity. The logic behind this operation is that the background will be white which will give us the first maxima. The second maxima is given by our object of interest. The mean value between them would divide the object and the background. The image is passed through gaussian filter first for better thresholding.

Fig 1. – Thresholding an RGB to binary image.

Task 2 – Cleanup of binary image

To do this cleaning up, I used OpenCV builtin function – floodfill which is an algorithm that fills a connected region in an image with a specified color. It starts from a seed point and spreads out in all directions until it reaches the boundary of the region, which is determined by a user-defined condition which is the seed point and the boundary condition. This algorithm is used as it is fast and fills us the holes most efficiently. It does not perform dilation or erosion, but fills in each pixel if its within the boundaries of the object. Below are the flood-filled images for the objects thresholded above.

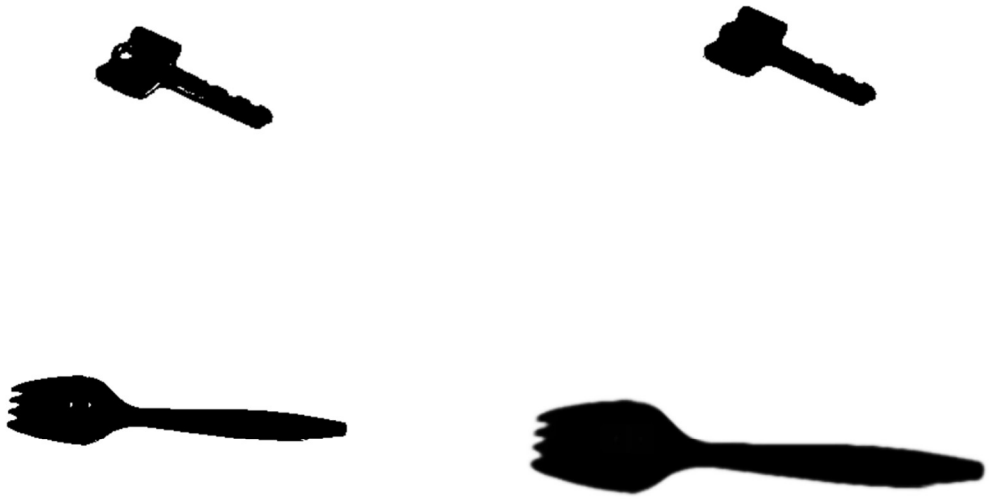


Fig 2. – Cleaning up of binary image

Task 3 – Image segmentation

We perform segmentation using the OpenCV function int cv::connectedComponentsWithStats(InputArray image, OutputArray labels, OutputArray stats, OutputArray centroids). This function performs 4 connected component analysis and outputs the labels for each component, stats for each component which includes its area, bounding box height, width, etc. The centroids returns the centroid pixel location for the component. The function type is int and it returns the number of components identified in the image. Below is the implementation for a not well denoised image with different segments colored differently. The above images did not have other components and were clean, hence below is a lighter image with a blob on top left. The color is generated randomly for each component.

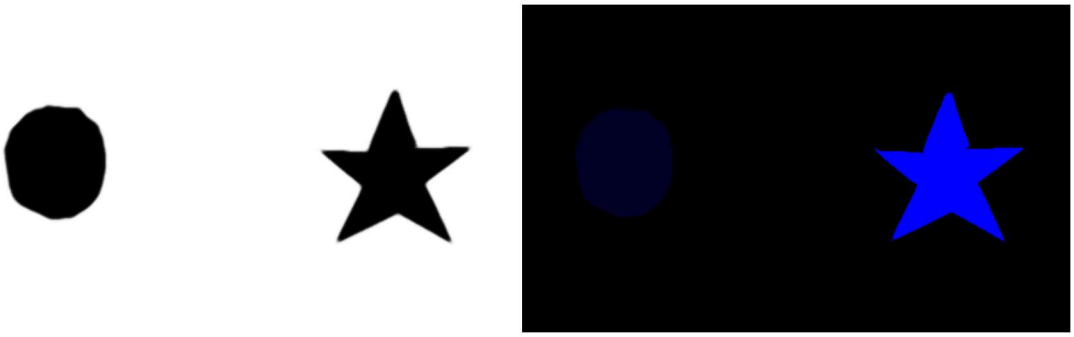


Fig 3. – highlighting different components of image (circle is dark blue left difficult to see)

Below are the blobs for task1,2.

Fig 4. – highlighting different components of images, circle is dark dark blue with a light blue on top left

Task 4 – Computing features for each major region

For major region the features I extracted were, height, base of the bounding box, area of region, % filled in bounding box. For comparison, I am using Hu-moments which are 7 scale, rotation, and translation invariant features for a region. Hu moments are derived from the central moments of an image, which are mathematical functions that describe the distribution of pixel intensities in an image. The central moments are then normalized to eliminate any scale and rotation effects, resulting in a set of seven Hu moments. These seven Hu moments are calculated using a combination of mathematical functions, including the raw moments, central moments, and normalized central moments of an image. The resulting values provide a unique set of descriptors that can be used to distinguish between different shapes and objects in an image.

To find the axis of least moment we calculate nu11, nu02, and nu20 for our region and we calculate the angle as –

$$\text{alpha} = \tfrac{1}{2} * (\text{atan}(2 * \text{nu11} / \text{nu20} - \text{nu02}))$$

For finding oriented bounding box, we find the vertices of rotated rectangle with minimum area of our object. Below are the photos with minimum moment axis and oriented bounded box which is required.

Fig 5. – Axis with least moment in red and the oriented bounded box for our objects

Task 5 – Collecting training dataset

To collect dataset, I used images as it was easier to visualize if the converted image was properly segmented. I used hu-moments (7 invariant moments) as features, these are calculated as –

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y)$$

Where x(bar) and y(bar) are centroid, hence these are translation invariant, the other 7 moments calculated are, which are rotation, translation, and scale invariant.

$$M_1 = (\mu_{20} + \mu_{02})$$

$$M_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2)$$

$$M_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{30})^2$$

$$M_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$$

$$M_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})((\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2) + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})(3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2)$$

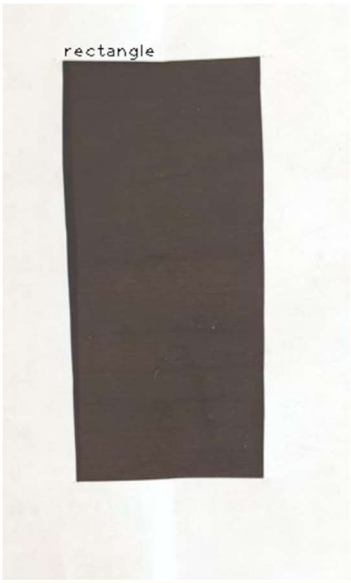
$$M_6 = (\mu_{20} - \mu_{02})((\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2) + 4\mu_{11}(\mu_{30} + 3\mu_{12})(\mu_{21} + \mu_{03})$$

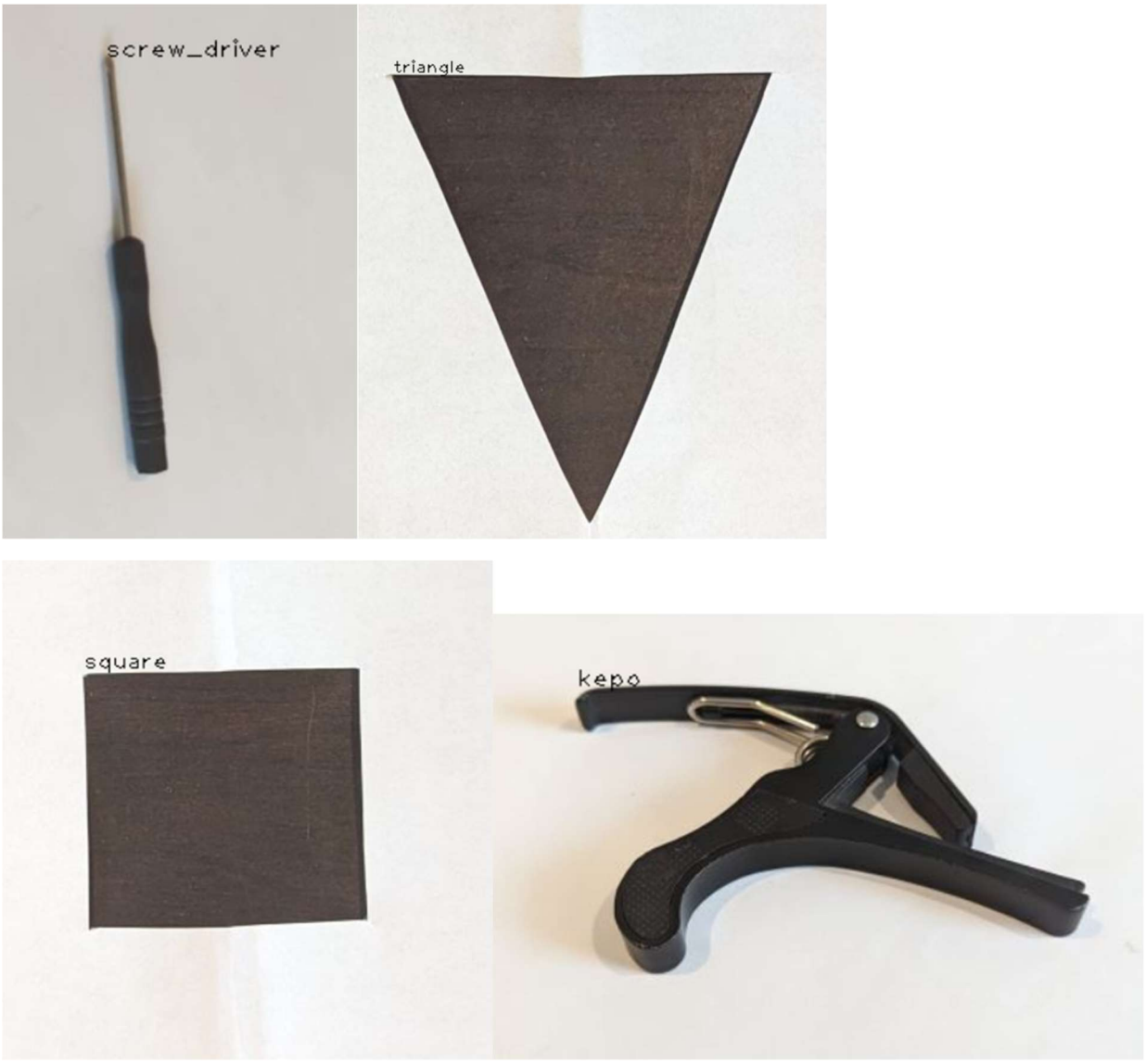
$$M_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})((\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2) - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03})(3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2)$$

The magnitude of these moment decreases with increasing index, as for M7 it is very low because of multiplication of powers of numbers between (-1,1). Hence, for comparison and better understanding I am converting them to log with base 10 values. One additional parameter calculated for better understanding is the %filled area but since this was calculated with vertical bounding box, it was not rotation invariant and hence, I did not use it for final comparison. I update the standard deviation value for the features in a csv after every time we add a new label. For recognition, I used least Euclidian distance between the features/std-dev (normalization).

Task 6 – Classify new images

Below are the classified objects using the method described above. These frames are different than when we saved these images in the database, which proves our scale, rotation, and translation invariant properties of features.





10th object (star) is shown below in KNN classifier.

Fig 6. – Classified objects with labels indicated.

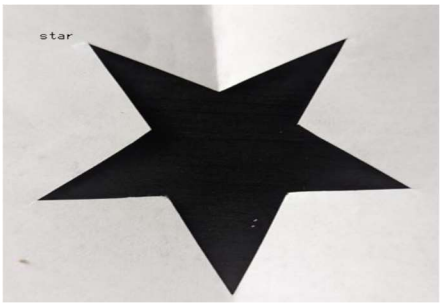
Task 7 – Implement a different Classifier

I used Knn classifier as a different classifier which calculates the least distance from a group of similar labels. My algorithm calculates the Euclidian features for each label and then normalizes it by the number of similar labels in the database. Below is the example of my database for Knn filtering.

key	1	13343	190	186	-2.88548	-6.02715	-8.90279	-9.25553	-18.3347	0.377561
key	1	3838	80	140	-2.79499	-5.77121	-8.64181	-8.94444	-17.738	0.342679
key	1	13342	190	185	-2.88566	-6.02807	-8.90204	-9.255	-18.3335	0.379573
key	1	3842	93	133	-2.79701	-5.77482	-8.6514	-8.95452	-17.7579	0.310615
star	1	47791	365	359	-3.08861	-8.24543	-11.0872	-13.0709	-25.1587	0.364719
star	1	47834	375	355	-3.08864	-8.24621	-11.0874	-13.086	-25.1911	0.359316
star	1	47796	365	359	-3.08861	-8.24662	-11.0875	-13.0686	-25.1552	0.364758
star	1	47838	377	352	-3.0887	-8.242	-11.0835	-13.066	-25.151	0.360487
spoon	1	24038	423	315	-2.47814	-4.98823	-8.05294	-8.12808	-16.2186	0.180405
spoon	1	24083	476	202	-2.4797	-4.99149	-8.0565	-8.13177	-16.2259	0.250468

Fig 7. – Classified objects with labels indicated.

It can be observed that each similar labels had some difference in its parameters because of differences while clicking the image. The output for our feature star in Knn is –



Task 8 – Evaluate your performance

The confusion matrix created for image matching for all objects is shown below –

	A	B	C	D	E	F	G	H
1	Confusion matrix							
2	Object	image 1	image 2			for correctly recognized		
3	kepo					for incorrectly recognized		
4	key							
5	lighter							
6	pen							
7	rectangle							
8	screw_driver							
9	spoon							
10	square							
11	star							
12	triangle							
13	pick							
14								
15								

The accuracy of our algorithm is around – Correctly identified /(Objects * 2)%

Accuracy ~ %70.

Task 9 – Drive link

https://drive.google.com/file/d/1jS_IP5HvIEO_itJx1hZyXOvzj9AWQEgv/view?usp=sharing

Reflection –

This was one of the toughest coding assignment I have done, maybe because I'm from mechanical. I took a lot of time and learned the importance of 30 mins policy which the professor had mentioned in the first lecture. I spent almost 2 days for debugging and could not submit the assignment solving the issue. But, at the end, after we get the labels on the objects which I had worked on for so long, all efforts are worth it. Fun assignment for sure. Could be better with an extended deadline.

References –

I used professor's lectures, OpenCV documentation, and stack overflow for references.