Project 3: Real-time 2-D Object Recognition

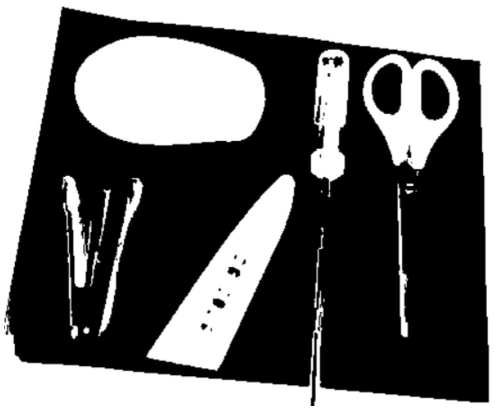
By Avnish Patel

Summary

The aim of the project is about implementing 2D object recognition. The camera will identify a specified set of objects placed on a white surface in a translation, scale and rotation invariant manner. It will recognize single objects and draw bounding box based on the regions. The images will be taken from a video sequence using my phone camera connected to my laptop webcam by IP address using an external application named iriun.

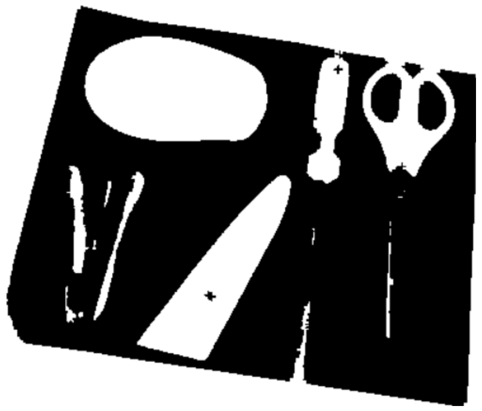
Required Image 1 – **Threshold the input video**

The image will be first converted into a grayscale image and then an arbitrary value (suitable for my set of images) i.e., 130 is used. So, any pixel with a value less than or equal to 130 will be the foreground pixel else it will become background.



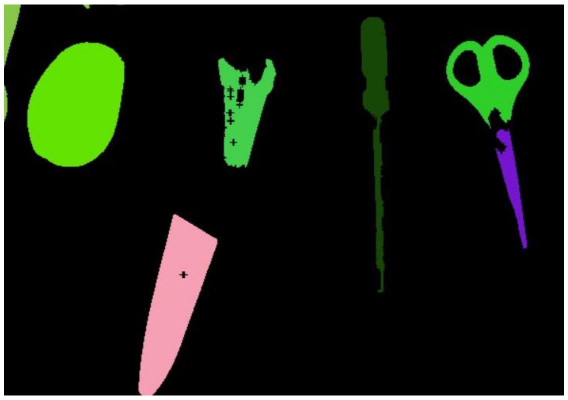
The image shows different objects placed like scissors, knife, screw driver and mouse.

Required Image 2 – **Clean up the binary image**



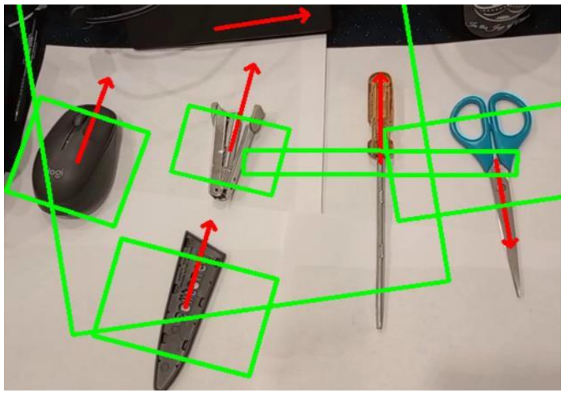
The small holes are filled using morphological operations. First dilation is applied followed by erosion. This can be seen in the above image using the '+' sign where the operation is done.

Required Image 3 – **Segment the image into regions**



Using connected component analysis by connectedcomponentwithstats function of OpenCV, region segmentation is done. The output regions are labeled, and their statistics and centroids are calculated. Then sorting of the regions based on the area (here taken 50) is done and assigns a unique color to top N (here N=9) regions. The region with an area below the threshold, here it is assumed as unwanted noise, is ignored and ultimately the above output image is obtained

Required Image 4 – **Compute features for each major region**

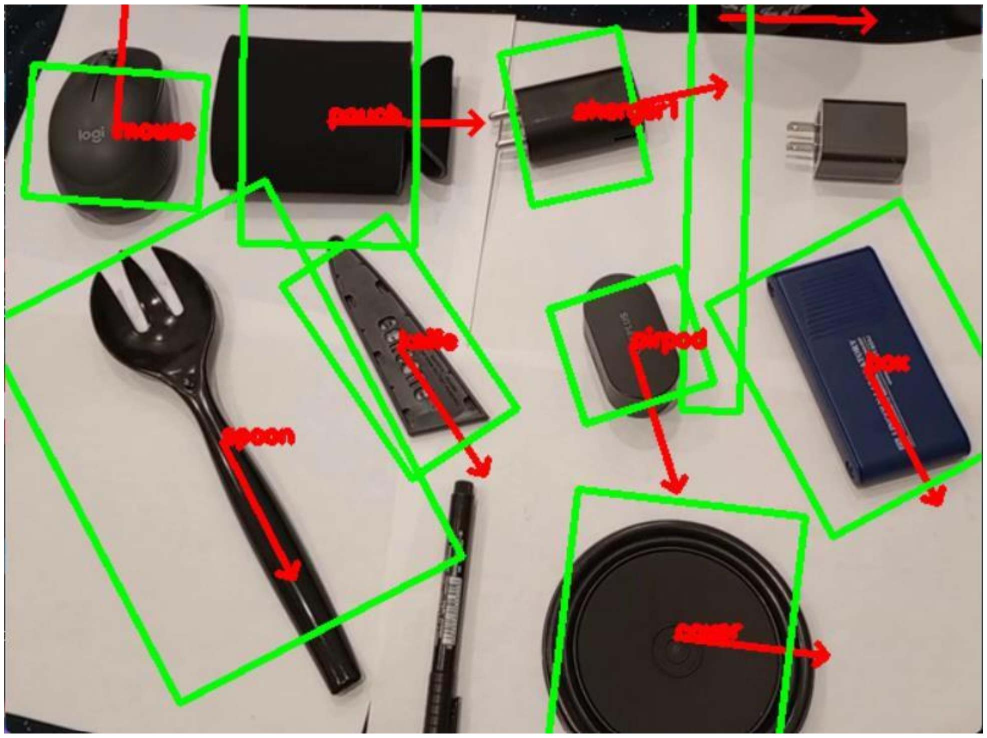


The bounding boxes are drawn on an individual object. Its moments are also calculated with its direction represented by an arrow. We can notice in the above image that an extra bounding box is there, leading to some inaccuracies of the model. It maybe detecting the white paper boundaries and representing it.

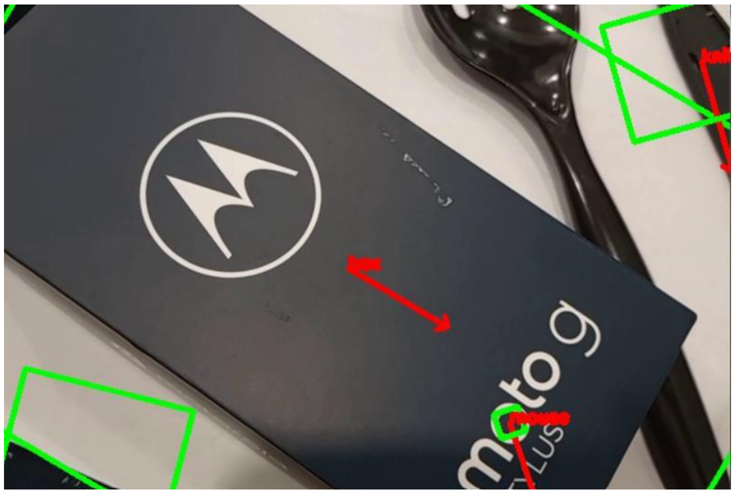
Required Image 5 – **Collect training data**

In my model, firstly all the feature vectors, attached with its respective labels (bounding boxes and the moments) are stored in a file. Overall, the system has two modes – training and testing mode. In the training mode, the model enables us to collect the feature vectors of known objects and store them. Here, when I press the key 'N', the model will prompt me to write a name/label and then store the feature vector for the current object with its label into the above file. This training would then be used for testing the dataset.

Required Image 6 – **Classify new images**



Based on the above trained classifier, we can now write labels and draw boxes with their moments and also classify unknown objects. The outputs of the image are quite satisfying as it classifies mostly the images correctly from the training set.

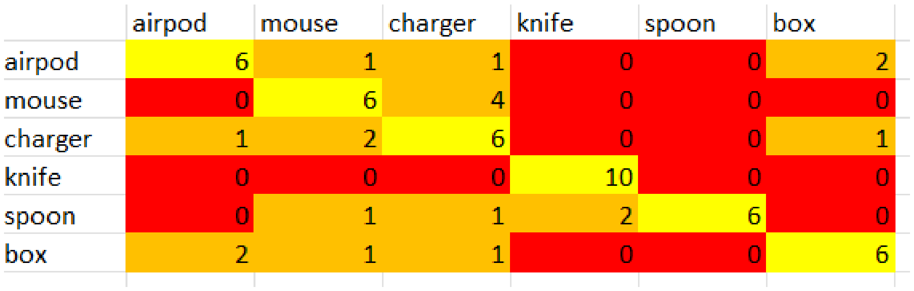


The above image is a completely new image, which is not included in the classifier. The model is able to detect and classify according to the nearest Euclidean distance. For this image, the model has performed well, seeing that it recognizes it as box. Rectangular shapes are handled well. A disadvantage, which can be seen is the ‘o’ word being classified as mouse. This represents that the model does not distinguish between images or words, it just classifies all.

7 - **Implement a different classifier**

As a different classifier, K Nearest Neighbors (KNN) is implemented. Firstly, the distance between new feature vectors and all the feature vectors is calculated using Euclidean. After sorting the distances in ascending order, first K class names are obtained. The class names are counted and the class name that appears the most is returned as the predicted class for the new feature vector.

Required Image 8 – **Evaluate the performance of your system**



	airpod	mouse	charger	knife	spoon	box
airpod	6	1	1	0	0	2
mouse	0	6	4	0	0	0
charger	1	2	6	0	0	1
knife	0	0	0	10	0	0
spoon	0	1	1	2	6	0
box	2	1	1	0	0	6

The confusion matrix represents the comparisons for the classified labels vs the true labels for total 10 times of each image in different positions and angles. For example, airpod was classified 6 times as airpod but 1 times as mouse, 1 times as charger and 2 times as box. This was to be expected since the case of an airpod is like the box that was given to it.

Some confusing results are also revealed. Such as spoon being classified as knife 2 times. It may be due to the end edges of the spoon which may bear a similar design with that of knife. The most accurate was knife as it bore a unique structure, which did not match with any of the given images irrespective of its rotation and orientation.

Image 8 – Extensions

1) Haarcascade People classifier

For real time body detection, Haarcascade classifier is implemented. Firstly, loading the pretrained classifier is done and from a video, different objects are classified.

Each frame is converted into grayscale and detectMultiScale function is used to detect full bodies in the frame. The rectangles are drawn for each detected body.

The video is attached depicting the algorithm.

2) Recognize 6-7 objects at the same time. This image was shown in the above task 6

3) Detecting unknown objects and showing its moment direction as seen in the above task 6

Reflection

This project gave me a lot of insights in the concepts of object detection and the difficulties and challenges one can face during working on them. This also gave me the opportunity to further read research papers and content on Deep Neural networks and other algorithms which would classify better than the traditional methods used in this project. I also understood the math and concept behind the region segmentation algorithm and successfully was able to train a KNN classifier to detect objects.

Acknowledgements

OpenCV Tutorials https://docs.opencv.org/4.5.1/index.html

Iriun Webcam: https://iriun.com/

Haarcascade video: www.pexels.com

OpenCV Haarcascade XML file: https://github.com/opencv/opencv