Project 3 report: Realtime Object Detection

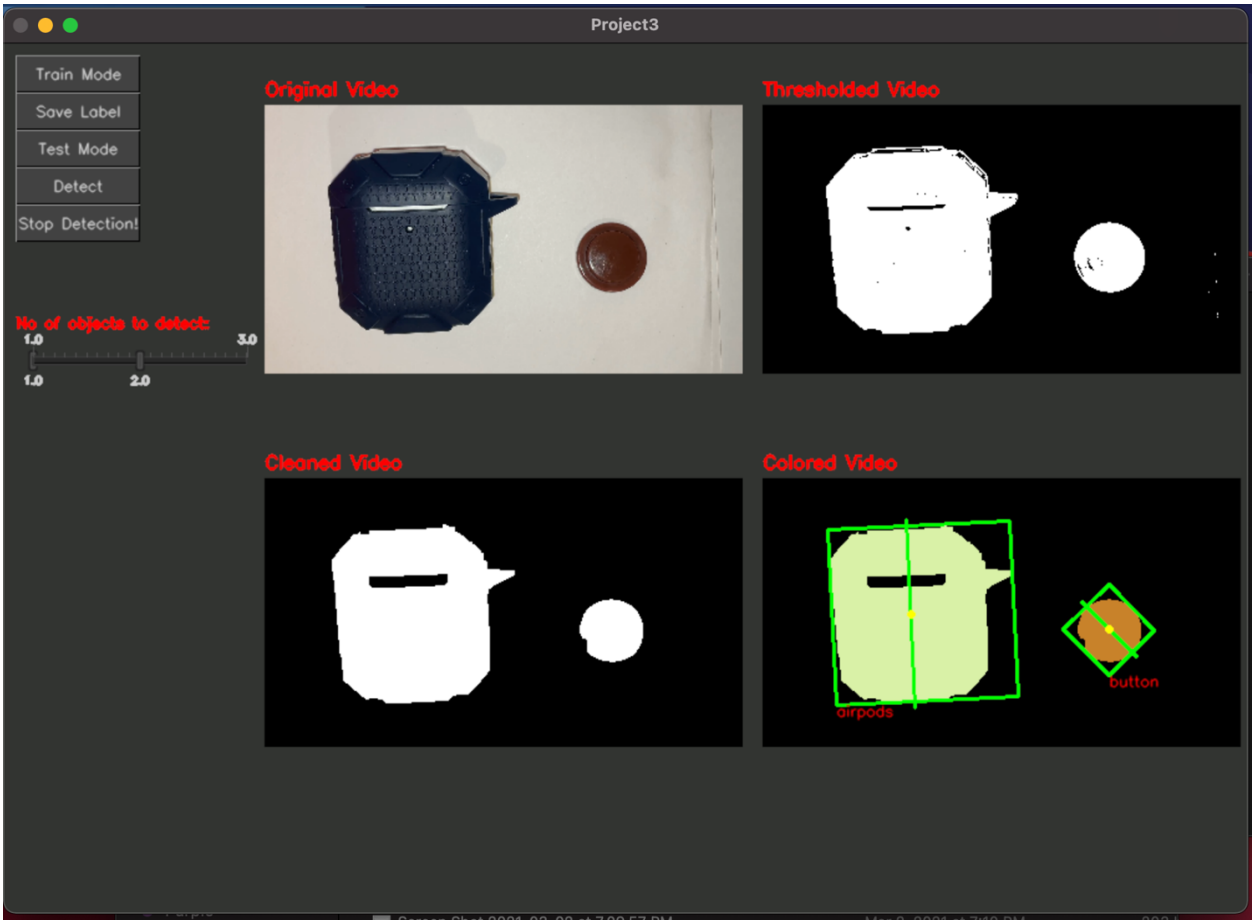
Introduction: As part of this project, we built a vision system to detect 2D objects in Realtime. We implemented multiple features for an image to build a feature vector. And used the feature vector for image classification.

Learnings and reflections:

1. In this project we learned a lot about image processing for binary images, how thresholding works, and how such simple features like bounding box fill percentage and height-width ratio be enough to classify objects in certain scenarios.
2. We learned about the practical considerations to keep in mind while doing all this. We learned about the importance of a proper setup, the importance of cleaning-up, etc.
3. We learned a lot about the tiny issues that don't come up in theory but in practice, and mention those in observations throughout our report (see point 4 in the extensions, for example).
4. We learned to build different features of an image, and use them collectively to identify objects in an image. Also we build a classification system where, we could train the system for multiple objects and then it detects those objects.

Extensions:

1. We designed a simple interactive Graphical User interface to work with this application.
   GUI:



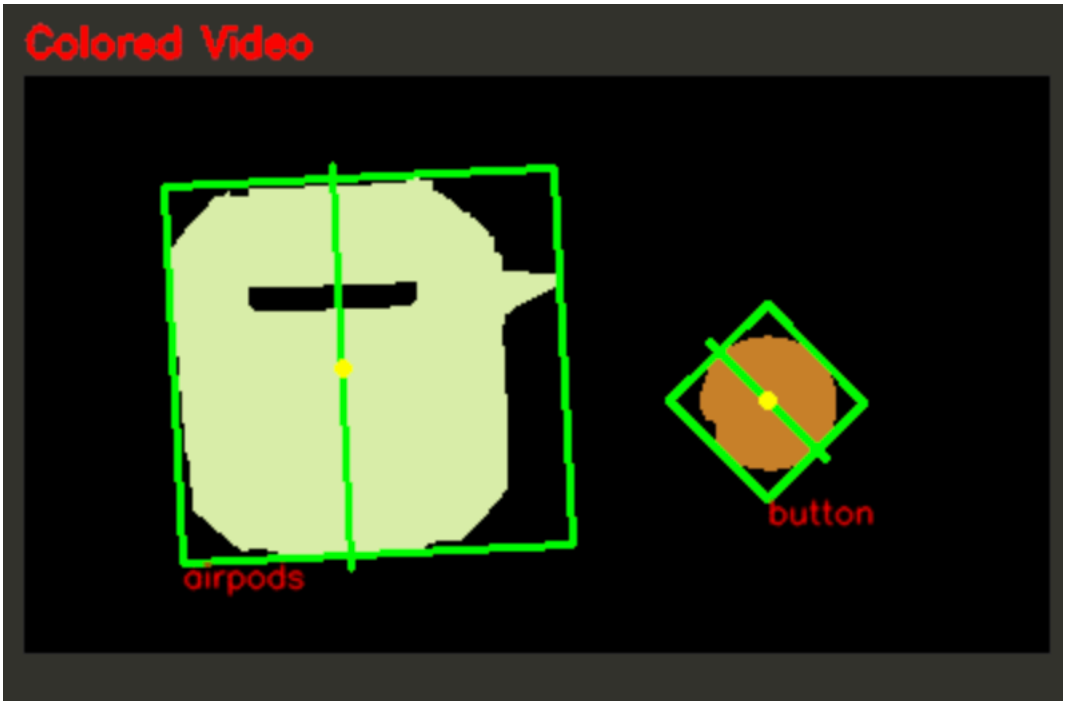
2. We built extensive database with capability to store and detect multiple objects.
3. Implemented grassfire transform from scratch to use for cleaning up the image. We also observed that it was much faster than regular iterative erosion and dilation.
4. Our system can detect multiple objects simultaneously, and basically do everything it could do with a single object in the frame with as many objects as the user desires, apart from training on all of them at once. It will color the regions in different colors, detect the object, etc.
5. Implemented a connected component (2-pass) algorithm from scratch. Also implemented the Union Find data structure from scratch, which works in amortized linear time.
6. Implemented more features – bucketed histograms of pixel densities over the primary as well as the secondary axes. We observed that these features were working well mostly. They were translation and scale invariant, but not completely orientation

invariant. Our reasoning for the same is as follows – let's say we've placed a hammer with it's hammering end on the right side, so in that case the histogram would have higher weight on the right extreme of the primary axis, but if we rotate the hammer 180 degrees then the histogram formed would have higher weight on the left side of the primary axis. And we can easily see that both these histograms are completely different from each other.

One fix we thought might work in this case is to not only calculate the distance between histogram of the image and the histograms in the db but also the distance between the reversed histogram and the db histograms and take the minimum of the two.

But we thought that the added complexity of this is unnecessary given our current object set as the features from task 3 are already working pretty well.

7. Display bounding box, primary axis and the centroid of the object on the video.



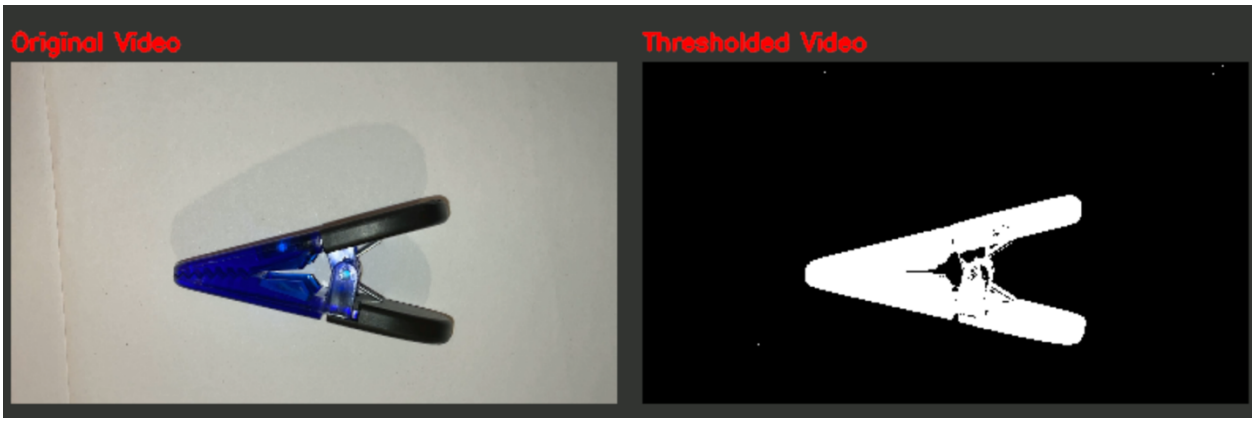
Demo Video: https://youtu.be/qoUKXvvBtTc

GUI Functions:

Train Mode – Initialize the database for storing label and feature vector.
Save Label – Save the label and feature vector in database.
Test Mode – File the database and reopen in read mode for object detection.
Detect – Start detecting the objects in realtime.
No of objects trackbar, set number of objects to be detected simultaneously.

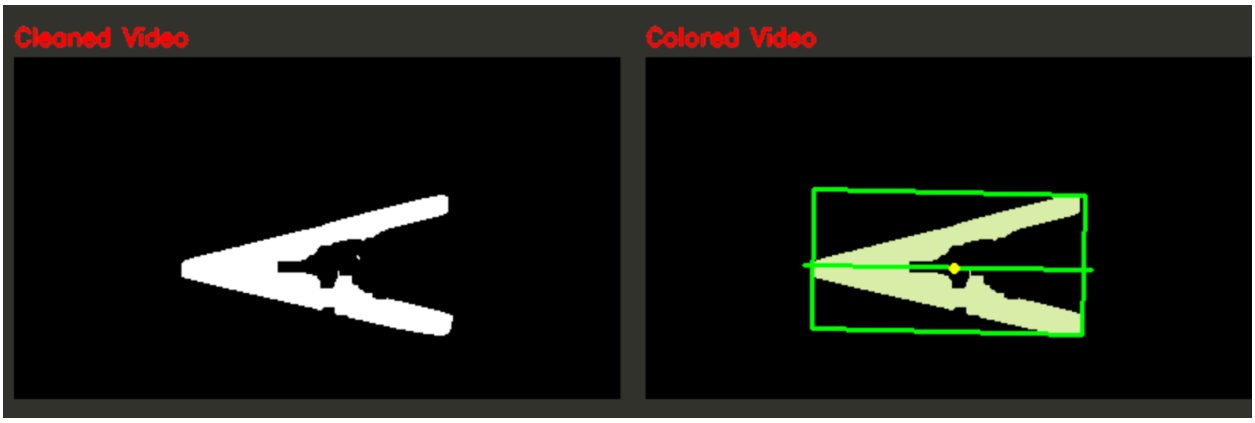
Task 1:

For thresholding an image (a frame of the video) we first convert the image to greyscale and then threshold it at a value T*255, any pixel with value above this threshold will be the background and the rest would be the foreground. The value T is selected by manual hit and trial but in general we found that any value of T in the range **0.4** to **0.6** worked fine.



Task 2:
Before segmenting the image we cleaned it using the standard iterative erosion and dilation one after the other to be able to perform opening as well as closing, the way we implemented it, the user can use 4 or 8 connectivity for either and perform as many iterations of them as needed.

Then we can segment the image into regions and display it in separate colors.



Task 3:

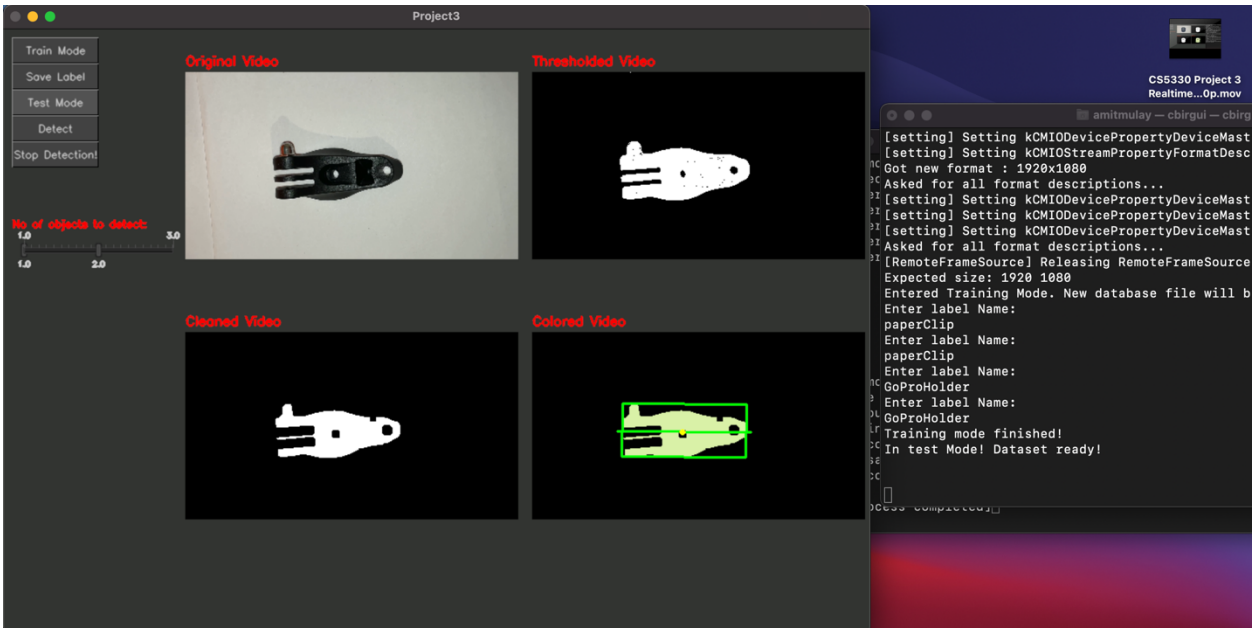
We calculate these three features primarily –

1. μ22α
2. bounding box height-width ratio
3. bounding box fill percentage (as a ratio)

Our observation was that just these three features (translation, scale and orientation invariant) were more than enough to be able to classify all our objects with 100% accuracy. We also observed that μ22α worked fine most of the time but not always, it's values varied a little when the object was placed such that it's primary axis was not horizontal or vertical.

Task 4, 5, 6:

Training samples are stored in the dataset. We used scaled Euclidean distance to measure similarity of the image feature vectors. For K-Nearest neighbors, we collected 2 samples of each object and run the classifier. And both the objects were detected correctly.



Task 7:
For this evaluation we trained the system for two objects and captured 5 images of each class. We selected mouse and airpods.

Confusion Matrix:

	Mouse	**Airpods**
Mouse	**4**	**1**
Airpods	**5**	**0**

Task 8:
Demo Video: https://youtu.be/qoUKXvvBtTc

Acknowledgement of the resources used

1. StackOverflow
2. Professor's lecture and lecture notes
3. C++ reference
4. OpenCV documentation
5. Email exchange with the professor