Intelligent Autonomous System Project1 Report

Gilberto E. Ruiz (ger83)

February 14, 2024

Introduction:

For the very first project in the Intelligent Autonomous course, we were given the task of training a model to detect orange cones in images and find the relative world coordinates of the cone from images. The purpose of this report is to explain my thought process and progress on how I was able to implement algorithms that learn the color model of the cone, segment that target color, and localize the target object from images. In addition to that, I will be explaining my revision of my original code that was submitted on February 8, 2024 that explains how I improved my code from using the built in GMM library to making my own and also explaining the differences and the results I got from both.

Labeling the dataset:

The first step we have to take to create a learning algorithm to detect traffic cones is to download and have access to data images. Fortunately, the professor provided us with twenty five images all with different layouts and locations that include a traffic cone. There's also variations in the images where some cones are far away, super close, underlight and shiny or covered by a shadow and dark, there's also images of the cone right next to red and orange objects that aren't traffic cones.

With the data collected in my iPython notebook, specifically on cell two, I organized all the images to display that I had accessed and could edit them.

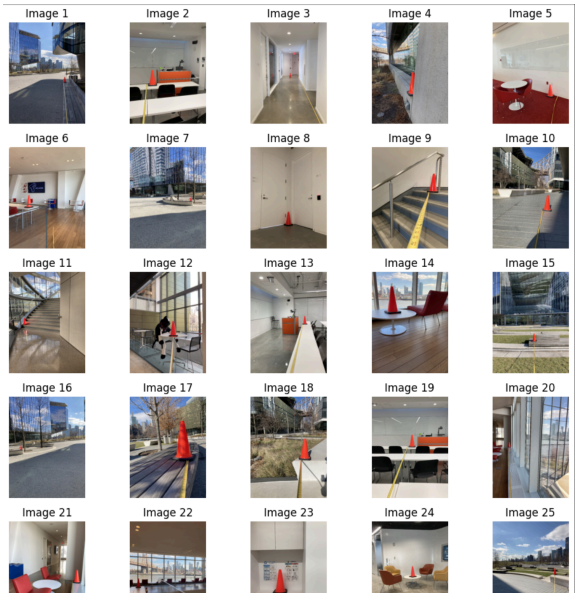

Figure 1: Subplot of all images printed

The next step after loading the image that I took was to create a python script that utilizes the RolyPoly library as a way to label each image. The reason I made it in a separate python script called “labeling.py” instead of continuing the work on the iPython notebook is because the notebook had technical difficulties getting the RolyPoly library properly working, so I just created a side script that can crop and take parts of the image and save it in a separate folder. For my implementation of the labeling aspect, I made the code run through all twenty five images twice. The first time it asks to draw a rectangle inside a traffic cone as a way to grab that specific color and store it in the folder called “SegmentedConeImages” and once the user draws the rectangle by clicking four times, they are asked to press “q” on the keyboard to do the same thing again but on a non cone object.

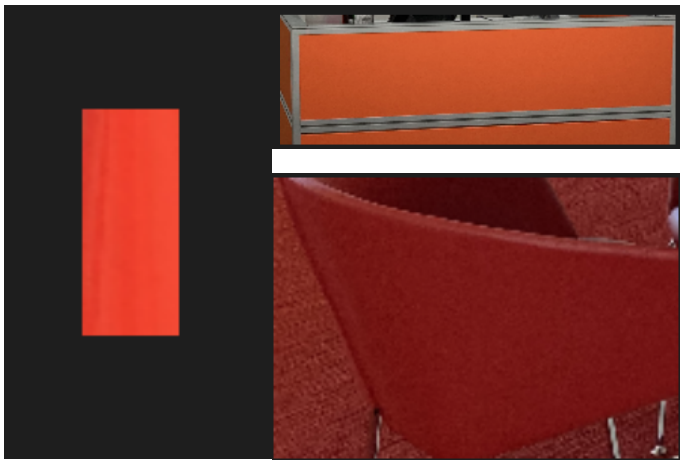

Figure 2: Example of labeled cone objects (left) and non cone objects (right)

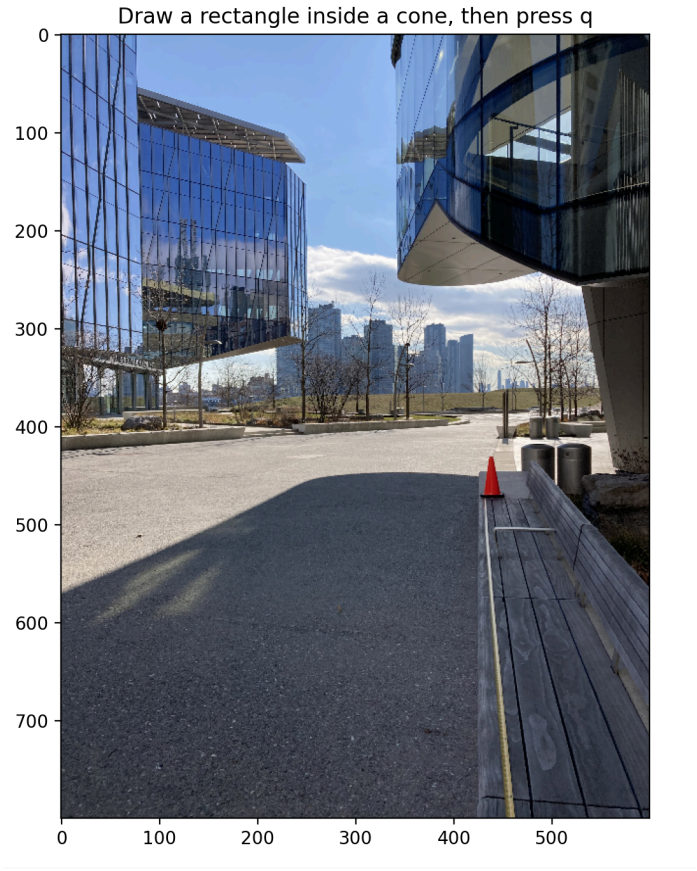

Figure 3: How the code asks to draw rectangle after running script

The cone objects were more or less the same just in different shades/intensities depending on how the cone was in the image, but for the non cone objects, I made sure to grab huge chunks of it.

Processing the Data:

After labeling the images and storing them to their respective folders, I had some trouble in getting the next steps in creating the learning algorithms. At first, I was using the K-Means Distribution Method to try and analyze the data, plot them in a color space and from there plot a Gaussian filter exactly on the area representing the cone objects and from there create the Segmentation of the traffic cones. But that process proved to be very difficult and time consuming to me and all I was able to accomplish was displaying the data in a Color Space.

Essentially, the code would read every image in the “SegmentedConeImages” folder and the “SegmentedImages” folder, and concatenate them into a numpy array. Once they were in arrays, I used a total of two clusters to calculate the centroids and labels and apply K-Means Clustering. Then all that was left to do was normalize the colors and plot them into a scatter plot that quite successfully plotted all my data in an organized way.

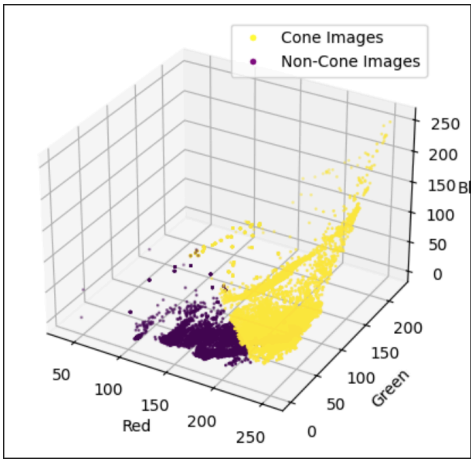

Figure 4: Data plotted in a 3D color space

Once the scatter plot was created, I was unsure on how to continue to calculate the probabilities, and algorithm to determine if an unknown pixel of an image had a cone object or not.

Creating the first version of the GMM model:

After much debugging and trying to figure out how to finalize the K-means distribution method and time constraints, I decided to pivot and restarted. This time I went with a Gaussian Mixture Model to identify if there are cones in images. The algorithm essentially began with loading the folder path to all the labeled cone/noncone objects and concatenating them into a list. From this point in time, I was having trouble finding out how to calculate the mean, covariance, weights, labels and identify if the image pixel was a cone by fitting it to a GMM model because of the many errors I would obtain by how I was reshaping the data. Because of the complications and errors I would constantly get, I decided to unfortunately use the built in GMM library and revisit that section at a later time.

With the built-in GMM model known as sklearn.mixture, I wrapped the ‘GaussianMixture’ class and fitted the model to the data representing pixel values of the cones. This process is done to ‘k’ number of components or the number of underlying Gaussian distributions to fit.

The ‘is_cone’ function then takes a single pixel’s RGB value and predicts the log likelihood of it being in the distribution of cone pixels using the fitted GMM, and then compares the probability to a threshold of “2e-06” to see if the pixel indeed is part of a cone.

After determining the probability, the “classify_image” function applies the previous process to an entire image and creates a mask where all the pixel’s probabilities to be a cone are being calculated. If the pixel is that of a cone then the code would turn it into a light blue color and if it isn’t then it would just turn the pixel to black.

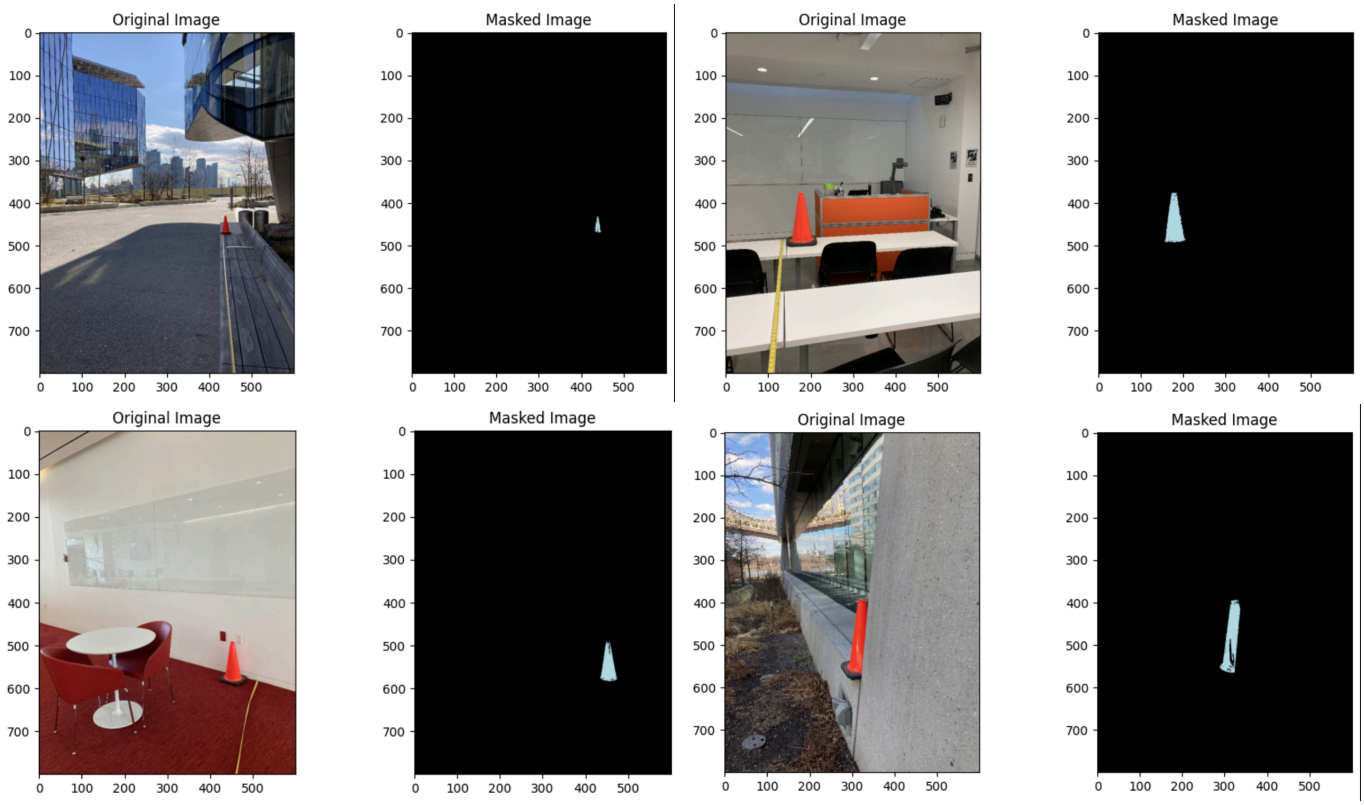

Figure 5: Results of masking using sklearn.mixture

Overall, the built in library did a really good job at determining if the pixel is a cone or not but has a few faults. In some images you can tell that it clearly should’ve been a cone but it marked it as a non cone object and that is because in the original picture, it's the part that's shiny due to

the sunlight. As mentioned before, I used this library in order to submit a working code before a due date but for the final report I revisited this part and created my own GMM Model.

Bounding Boxes and Getting Coords of Cones with sklearn:

Once the GMM model has successfully masked the images and determined whether the pixel is or isn't a cone object, the next step is to create a bounding box around that cone that's been detected and calculate the distance to the cone. The function "add_bounding_boxes_and_coords" takes the images along with all the data acquired from the GMM model as inputs and processes it to draw rectangles. Each bounding box is determined by the position and size of the detected object within the image. The distance of each cone from the camera is then estimated based on the apparent width of the cone in pixels and known physical dimensions of the actual cone (7.5 inches wide) and it's all adjusted by the focal length of the camera.

In order for the code to accomplish this, it copies the original image to preserve it and later draws the box on it and sorts the detected objects by area in descending order to focus on the larger detections/matches. For each detected object, the code draws a green rectangle and displays in text the distance of the object.

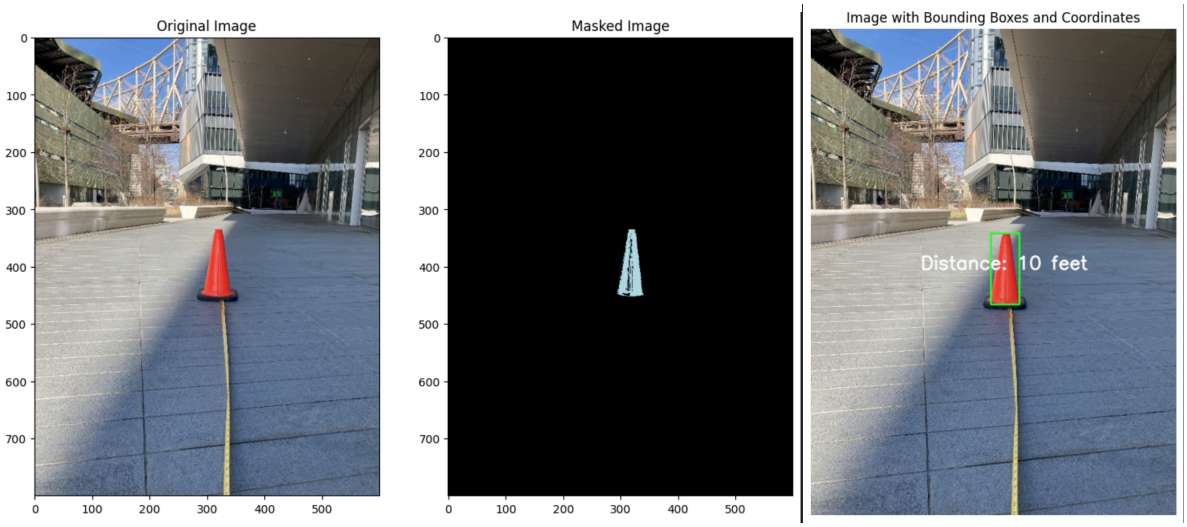

Figure 6: Successful Bounding and Distance Calculation

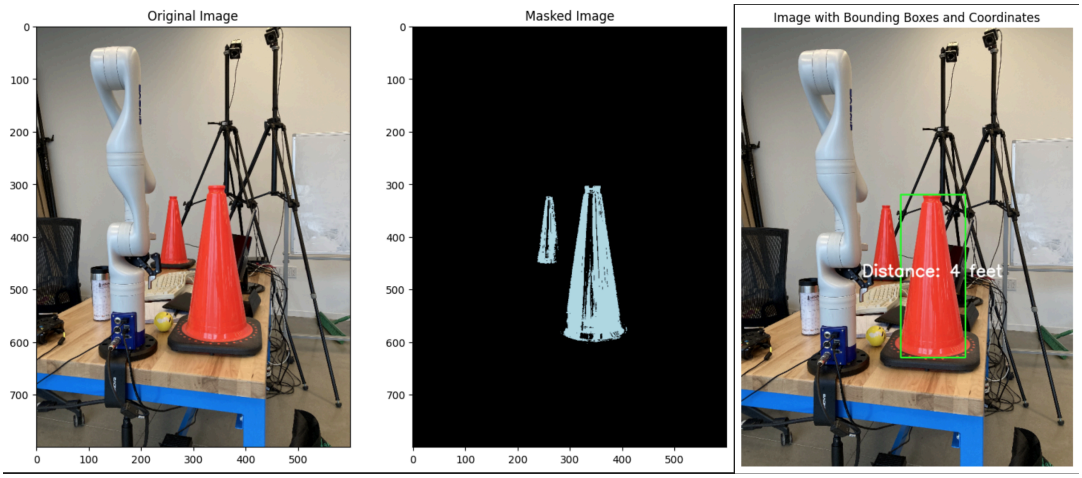

Figure 7: Unsuccessful Bounding and Distance Calculation

From the results, the code did a really good job at drawing the bounding box right on the cone and calculating the correct distance. The only time it failed was when a second cone was introduced. From the results, the code was successful in masking the two cones, but when it came to bounding the cone and calculating the distance for both cones, it fell short. This is most likely because I set a break feature after the text is printed because when I ran the code for the very first time, the code would print random distances all throughout the image and I decided to use the break to "filter" out the unnecessary misdetections. Like the cell that calculated the GMM model, I was not satisfied with the result of this bounding code so after the initial submission of the code, I also revisited this part of the code to figure out and fix it to be able to bound and calculate the distance for two cones.

Creating the second version of the GMM model:

As previously mentioned, I wasn't satisfied by using the built in library to create a GMM model, which is why I revisited it and recreated the model from scratch. With my implementation, the algorithm begins by converting the image data into a HSV color space. The reason for this is because it gets rid of some errors you might get by the original image's luminance. After that, I created a custom GMM class that creates the model's initialization, fitting, and prediction logic. The process to fit the data involves using the expectation maximization (EM) algorithm, where the function labeled "fit" initializes the model's parameters

(means, weights, and covariances). The EM steps are then computed with the E-step which gets the data points' cluster probabilities and the M-step which updates the model based on these probabilities. The code then computes the probability density function for each data point using the Multivariate Normal PDF method which is very important to assess the likelihood of the data within the clusters. After that, the code provides the log likelihoods and converts them into probabilities that predict whether the pixel is likely to be a cone object or not. Which leads to the code applying the predictive capability to actual images and masks the cone pixels. After masking the images, I displayed the results like in the previous version by printing the original image and masked image side to side as a way to better visualize the accuracy of the GMM model.

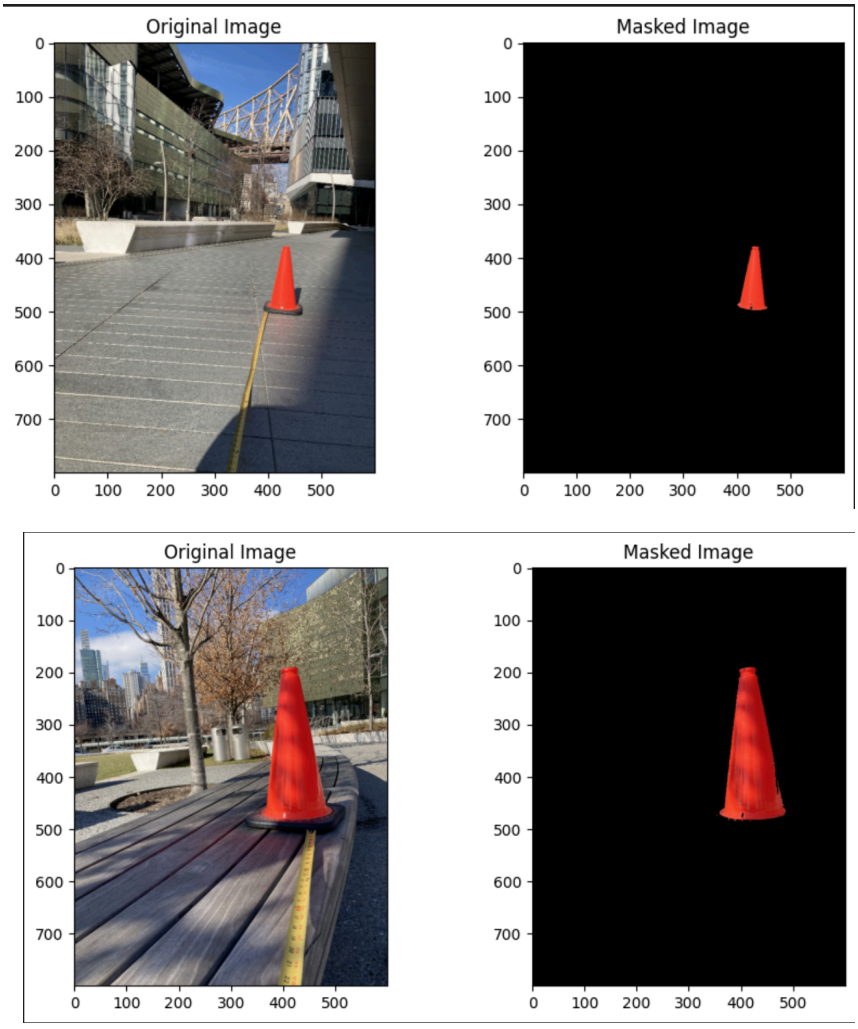

Figure 8: Results of my own GMM model

Bounding Boxes and Getting Coords of Cones with my own GMM model:

Just as I revisited the GMM model, I also improved my implementation of applying bounding boxes and calculating the coordinates of the cone. Like the previous version, it has a "add_bounding_boxes_and_Coords' method which is responsible for drawing the boxes and annotating the images with distance measurements. Essentially, it takes the image, a binary mask that indicates the locations of the cones, a focal length in pixels, the name of the current image file, and a path to a text file called "results.txt". In the function, connected component analysis is performed on the mask to identify individual cones and for each detection, if its area is significant enough (above a threshold), a bounding box is drawn and the distance to the cone is calculated using a simple pinhole camera model. The distance that's been calculated is then written on the image and on the output text file mentioned earlier. To be more precise, it records the coordinates of the base of the cone, the calculated distance, and labels it with the specific image filename. The code then executes the function on the five test image sets that the professor provided us and in the original twenty five images of the dataset. Each original image is then displayed side to side with their bounded and labeled version. In terms of how it's better than the previous version, due to it performing the connected component analysis, it was able to bound and calculate the distance of two cones in an image, which the previous version could not. But on the other hand, due to the threshold, it would sometimes bound a box on objects that weren't cones due to their specific color being too similar and passing through the threshold.

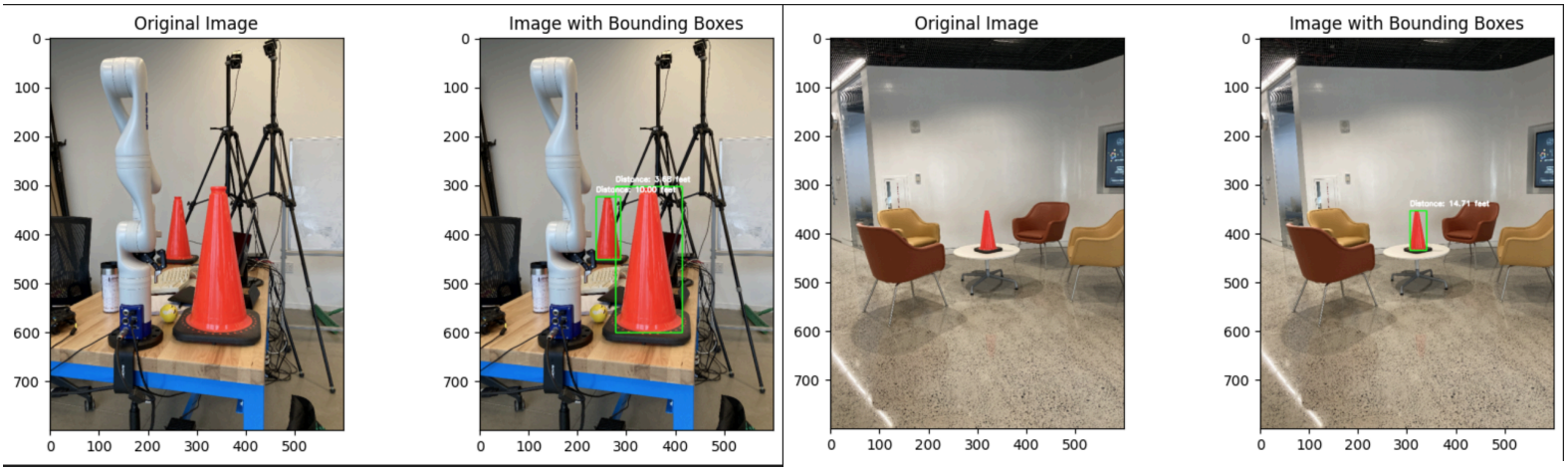

Figure 9: Successful Bounding and Calculation of Distances

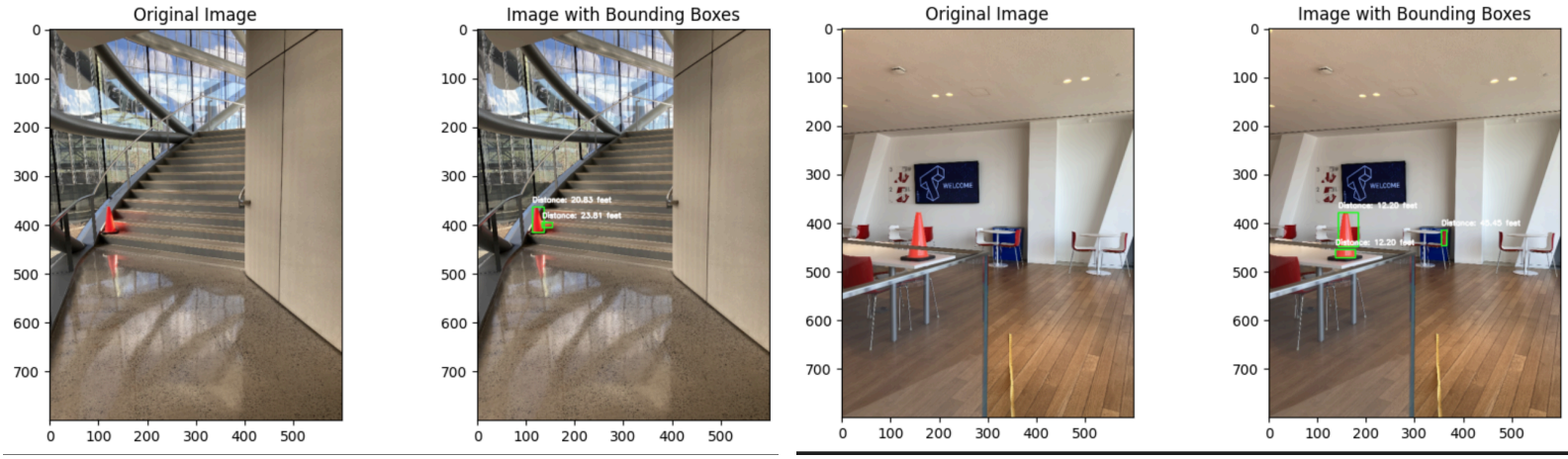

Figure 10: Unsuccessful Bounding and Calculation of Distances

```
ImageName:test_1.png, Down: 522.00, Right: 227.50, Distance: 13.51
ImageName:test_2.png, Down: 536.00, Right: 475.00, Distance: 4.46
ImageName:test_3.png, Down: 453.00, Right: 319.50, Distance: 9.80
ImageName:test_4.png, Down: 419.00, Right: 300.50, Distance: 29.41
ImageName:test_5.png, Down: 601.00, Right: 346.00, Distance: 3.68
ImageName:test_5.png, Down: 452.00, Right: 263.00, Distance: 10.00
ImageName:train_10_dist20.png, Down: 418.00, Right: 490.00, Distance: 25.00
ImageName:train_12_dist8.png, Down: 498.00, Right: 429.00, Distance: 8.93
ImageName:train_13_dist6.png, Down: 491.00, Right: 427.50, Distance: 6.67
ImageName:train_14_dist16.png, Down: 548.00, Right: 327.50, Distance: 20.00
ImageName:train_15_dist3.png, Down: 483.00, Right: 422.50, Distance: 3.94
ImageName:train_16_dist24.png, Down: 470.00, Right: 439.50, Distance: 33.33
ImageName:train_17_dist8.png, Down: 500.00, Right: 342.50, Distance: 10.64
ImageName:train_18_dist5.png, Down: 562.00, Right: 234.00, Distance: 6.58
ImageName:train_19_dist15.png, Down: 411.00, Right: 427.00, Distance: 17.86
ImageName:train_1_dist30.png, Down: 431.00, Right: 313.00, Distance: 41.67
ImageName:train_20_dist8.png, Down: 495.00, Right: 180.00, Distance: 8.93
ImageName:train_21_dist10.png, Down: 394.00, Right: 317.50, Distance: 12.82
ImageName:train_22_dist7.png, Down: 349.00, Right: 481.50, Distance: 8.20
ImageName:train_24_dist12.png, Down: 435.00, Right: 323.00, Distance: 14.71
ImageName:train_25_dist19.png, Down: 416.00, Right: 125.00, Distance: 20.83
ImageName:train_25_dist19.png, Down: 406.00, Right: 143.50, Distance: 23.81
ImageName:train_26_dist30.png, Down: 424.00, Right: 481.50, Distance: 38.46
ImageName:train_27_dist30.png, Down: 417.00, Right: 97.50, Distance: 38.46
ImageName:train_29_dist5.png, Down: 566.00, Right: 313.00, Distance: 10.42
ImageName:train_2_dist10.png, Down: 449.00, Right: 163.50, Distance: 12.20
ImageName:train_2_dist10.png, Down: 447.00, Right: 360.50, Distance: 45.45
ImageName:train_2_dist10.png, Down: 472.00, Right: 157.50, Distance: 12.20
ImageName:train_31_dist10.png, Down: 612.00, Right: 298.00, Distance: 14.71
ImageName:train_4_dist5.png, Down: 316.00, Right: 165.00, Distance: 6.76
ImageName:train_5_dist25.png, Down: 398.00, Right: 133.50, Distance: 33.33
ImageName:train_6_dist10.png, Down: 452.00, Right: 279.50, Distance: 9.43
ImageName:train_6_dist10.png, Down: 481.00, Right: 534.00, Distance: 10.42
ImageName:train_6_dist10.png, Down: 617.00, Right: 109.00, Distance: 3.21
ImageName:train_6_dist10.png, Down: 640.00, Right: 268.50, Distance: 3.70
ImageName:train_6_dist10.png, Down: 679.00, Right: 13.50, Distance: 18.52
ImageName:train_6_dist10.png, Down: 800.00, Right: 155.00, Distance: 3.29
ImageName:train_7_dist10.png, Down: 585.00, Right: 455.00, Distance: 11.36

```

Figure 11: results.txt after running Revised Bounding Boxes and Distance Calculation function

Conclusion:

This project was very interesting to say the least, it definitely challenged me to give it my all in order to get it to work. From deciding to use the HSV color space in order to separate the image brightness from color information to make the cone detection run smoother than the first

version. To, determining the threshold of 0.5 by model validation and testing various values to find a balance that minimizes both false positives and false negatives. To the decision of utilizing two components for the number of GMM (cone and non-cone objects) to model the distribution of cone colors and background colors within the images which resulted in a simple model. To also display just how my GMM model performed, I recorded the log likelihood after defining the GMM class and plotted it in a learning curve. From the result below in *Figure 12*, it shows how after an initial spike, which was probably caused by the starting point of the algorithm, the log likelihood quickly converges to a stable value. The plateau in the plot shows that the model's parameters have stabilized and that the model has reached convergence.

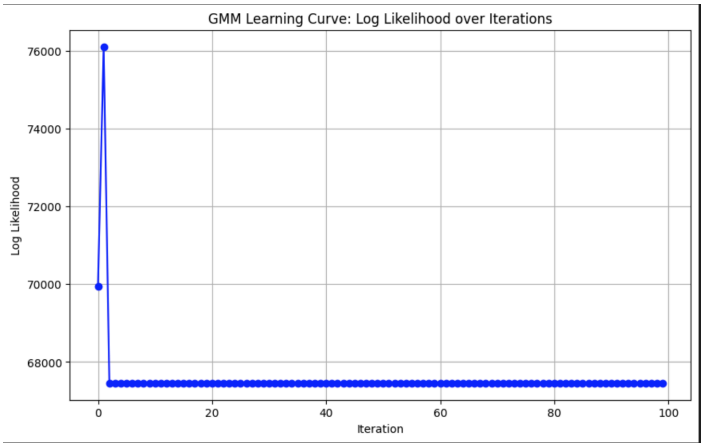

Figure 12: GMM Learning Curve Plot

And lastly, to determine the distance of the cone in the images by applying principles from projective geometry and camera optics. In other words, using the apparent size of the cone in the image, in terms of pixels, and estimating its distance from the camera based on the known actual size of the cone with the following formula:

$$Distance = ((Actual\ Width\ of\ Cone\ x\ Focal\ Length)/Apparent\ Width\ in\ Image)$$

In conclusion, this report has demonstrated the overall process, struggles and breakthroughs I went through applying the Gaussian Mixture Model for image segmentation and object detection which provides a very useful framework for estimating the real-world distances of objects from a camera, with potential applications in various fields such as autonomous vehicles, robotics, and traffic monitoring systems.