Object Detection and Color Segmentation using GMMs

(ECE5242) Intelligent Autonomous Systems, Cornell Tech, Spring 2024

Yoseph Yan
Master's student, Electrical and Computer Engineering
Cornell Tech
ypy4@cornell.edu

1 INTRODUCTION

This project explores object detection and localization of orange cones. The challenge involves recognizing the unique color characteristics of orange cones and determining their relative world coordinates. Through the implementation of image processing techniques and machine learning algorithms, this project aims to develop a solution to identifying and locating orange cones in digital images.

2 PROBLEM STATEMENT

The main objective of this project is to train a model that can detect orange cones in images, determine their relative world coordinates, and estimate their distance from the camera. This comes with challenges such as the segmentation of the target color in varying lighting conditions and the relatively accurate localization of the cone within the image space. Color segmentation is so difficult because inconsistencies arise from the variability in appearance in different lighting conditions. Localization is so difficult because, to accurately determine the relative world coordinates and distance, the cone's size relative to the camera must be understood. The goal is to develop an algorithmic solution that addresses these challenges. The solution should be trained from a set of training images, and then perform accurately on test images. This project involves hand-labeling training data, selecting appropriate color spaces, training a color classifier, and implementing a distance estimation algorithm to achieve accurate detection and localization of orange cones.

3 APPROACH DESCRIPTION

The approach I took can be broken down into many parts. First, I hand labeled the training images using roipoly. Although I only used one class (OrangeCone), I annotated 2 classes just in case I wanted to use a second class. Because of this, I initialized two classes: OrangeCone and

OtherClass. I then looped through the images in my training images folder and drew polygons on the orange cones and then anything that was red. I then used the get_mask function to get a binary mask for the regions of interest. This is a 2D numpy array that is the size of the image. Its elements are True if the region of interest is inside the polygon, and false otherwise. I then save these binary masks as a .npy file for future use. I decided to use the RGB color space just because that is what was given to me. If my classifier did not work well, I would have tried other color spaces like HSV.

I then loaded the training images again, this time to get the pixels that correspond to the orange cone. To do this, I loaded the binary masks that I saved earlier and applied them to the training images. This ensures that I am only getting the pixels that correspond to the orange cone. I then added all the orange pixels to an array to concatenate it. After I retrieved all the orange pixels corresponding to the orange cone, I trained a gaussian mixture model (GMM). I used the expectation-maximization algorithm to estimate the GMM's parameters. I first call the get_gmm_parameters function specifying the number of clusters. I then initialized the mean and covariances for each of the RGB values. I then call the e_step function to perform the expectation step. This step consists of calculating z and g. I then call the m_step function to perform the maximization step. This step consists of maximizing the expectations calculated in the last step with respect to the model parameters. This repeats until it hits the termination criterion or if it reaches the maximum amount of iterations. The model has now been trained and the parameters (mean, covariances, and weights) are saved into a .npy file. To run my model, I just load the parameters I saved previously and calculate the probability. If the probability of it being orange is True, it creates a binary image to show which parts of the image it detects an orange cone at. I chose my thresholds through trial and error. I decided on my threshold because that is what gave me the best results when I tested my algorithm. The equations I used for the GMM and EM algorithm are on the next page.

Finally, I dilated the binary image to solidify the pixels that are detected. I then use the label function to label each of the connected regions in the image. I then use the bbox function to get the top left and bottom right coordinates of the orange cone. I then use the rectangle function to draw a rectangle around the orange cone. I calculated the distance between the camera and the orange cone through shape statistics. Since the orange cones are 7.5 inches wide and 17 inches long, I estimated the pixel dimensions to be 425 pixels wide and 963 pixels long through trial and error. I then used the coordinates of the bounding box and the formula provided in class to divide the width and length of the cone in pixels by the measured width and length of the cone in the image in pixels. I then got the average of both to calculate the distance from the camera lens in feet. I then display the segmented image, bounding box, and distance to the cone.

$$p(\mathbf{x}) = \sum_{k=1}^{K} w_k g_k(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

$$g_k(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \Sigma_k{}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right\}$$

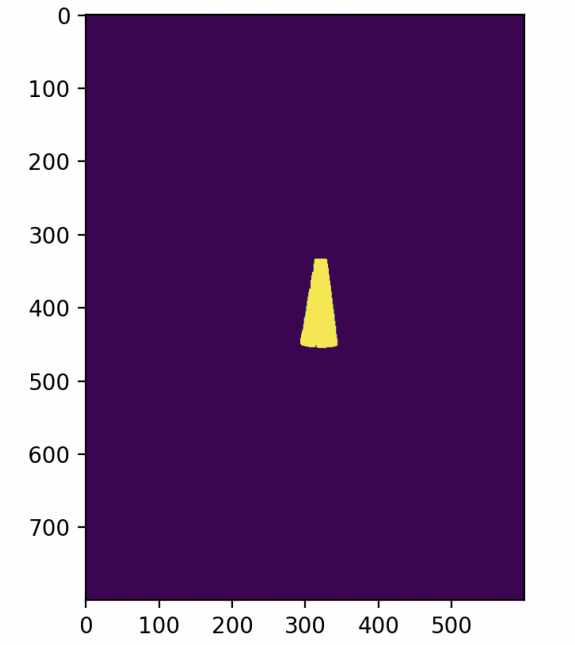
$$z_k^i = \frac{g_k(\mathbf{x}_i|\boldsymbol{\mu}_k, \Sigma_k)}{g_1(\mathbf{x}_i|\boldsymbol{\mu}_1, \Sigma_1) + g_2(\mathbf{x}_i|\boldsymbol{\mu}_2, \Sigma_2)}$$

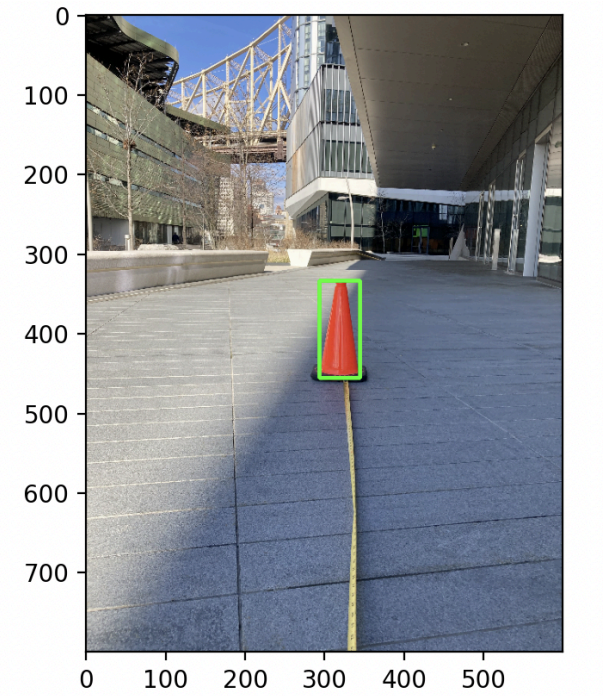
$$\hat{\boldsymbol{\mu}}_k = \frac{1}{z_k}\sum_{i=1}^{N} z_k^i \mathbf{x}_i$$

$$\hat{\Sigma}_k = \frac{1}{z_k}\sum_{i=1}^{N} z_k^i (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^\top$$

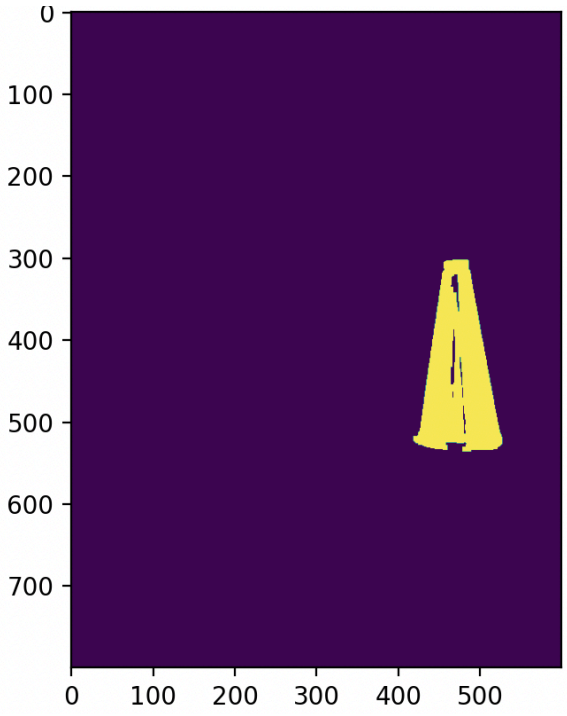
4 RESULTS DISCUSSION

Running my model on the 5 test images, I was able to correctly identify and locate the orange cones for all of the images. I have attached images of my test results below. For test image 4, I was able to correctly identify the orange cone, but I also incorrectly identified the exit sign above it. A solution to this problem would be to train another class so my model can differentiate between the colors orange and red. Besides this, I would say my model was highly successful and is able to detect and locate orange cones with high accuracy in both locating the cone and estimating the distance of the cone..

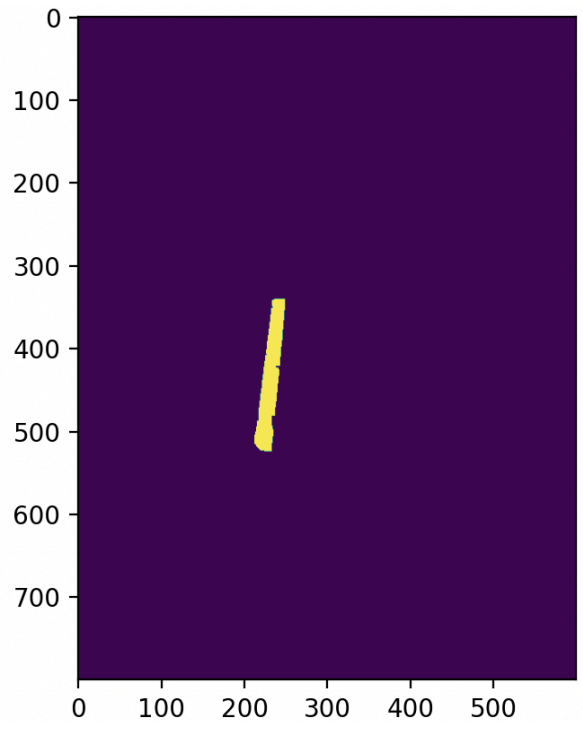


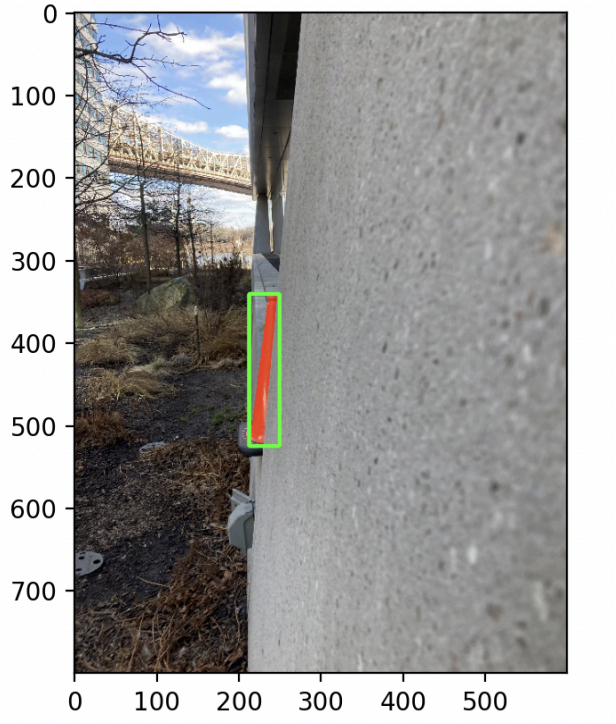



```
ImageName: test_3.png, Down: 456, Right: 318.5, Distance: 8.113387978142077
```

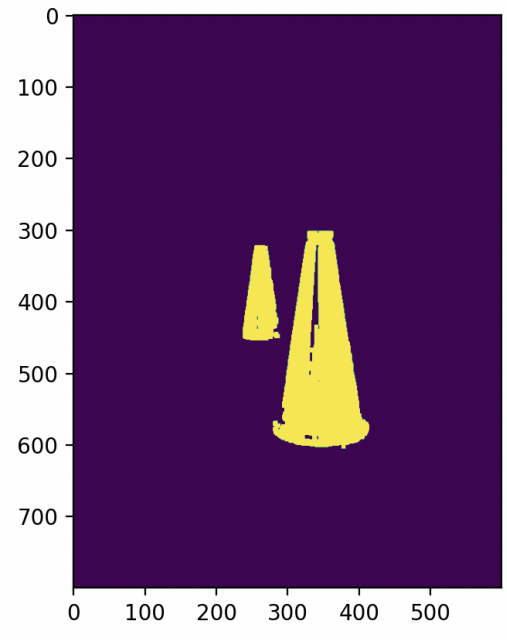


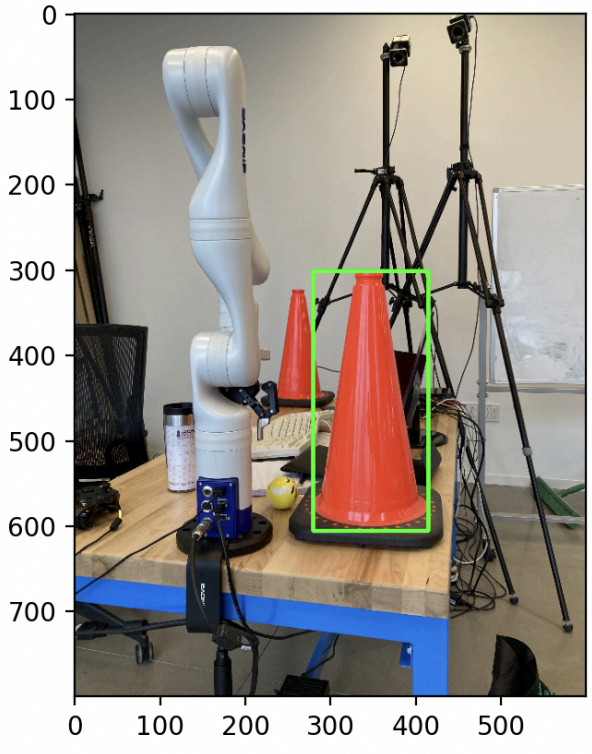

```
ImageName: test_2.png, Down: 537, Right: 472.5, Distance: 4.007233592095977
```

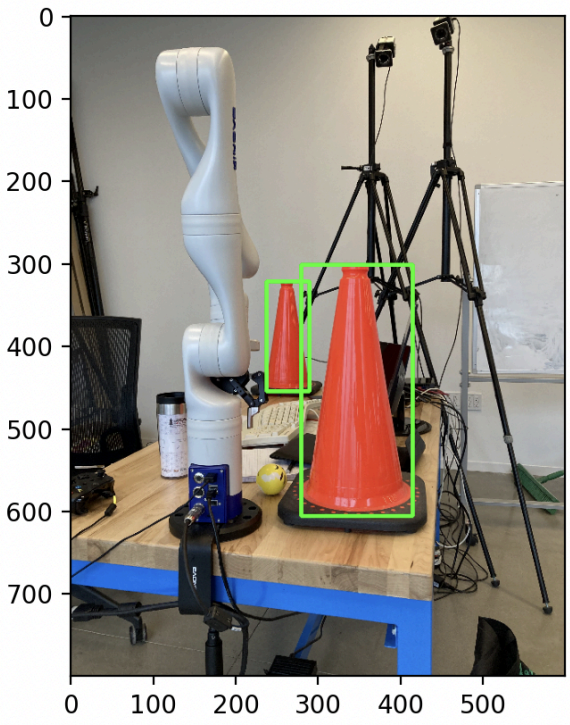




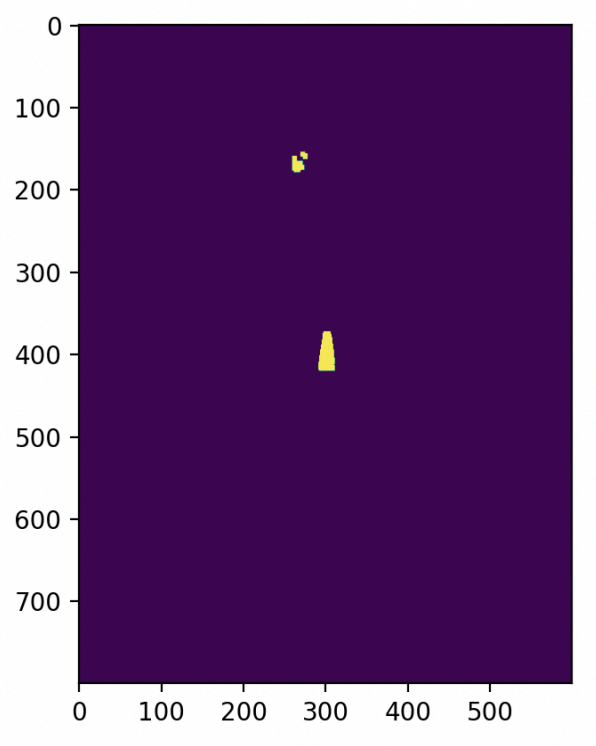

```
ImageName: test_1.png, Down: 525, Right: 230.5, Distance: 8.360091069330199
```

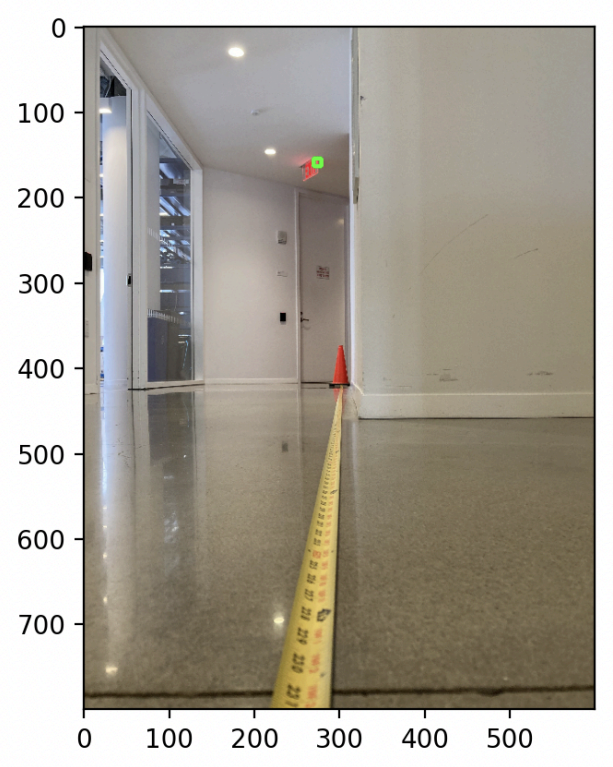


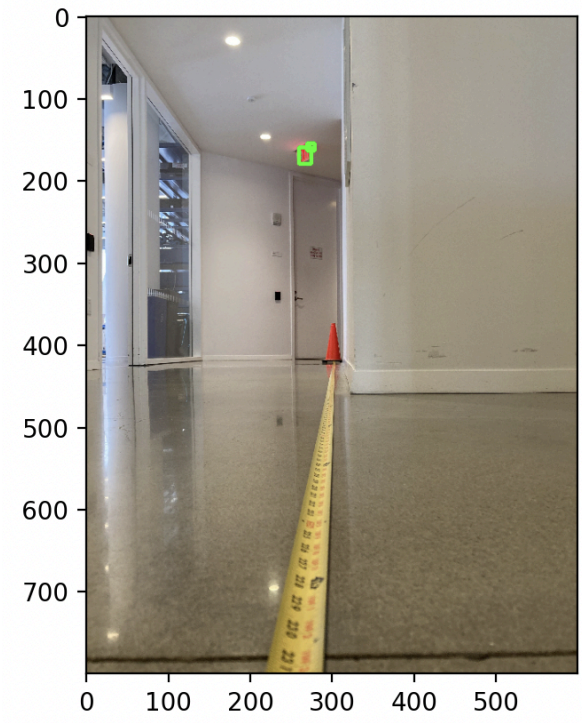


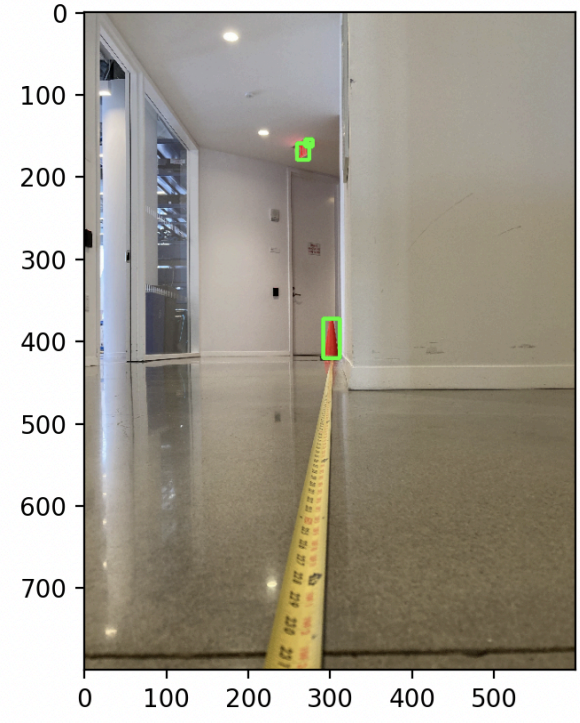



```
ImageName: test_5.png, Down: 606, Right: 346.5, Distance: 3.1579556530214425
ImageName: test_5.png, Down: 455, Right: 263.0, Distance: 7.706839213418161
```










```
ImageName: test_4.png, Down: 164, Right: 273.5, Distance: 77.11111111111111
ImageName: test_4.png, Down: 180, Right: 266.5, Distance: 38.24166666666667
ImageName: test_4.png, Down: 421, Right: 301.0, Distance: 20.65625
```

5 CONCLUSION AND FUTURE WORK

This project aimed to detect and locate orange cones in digital images. Through the development and implementation of my algorithm, I was able to successfully detect and locate orange cones in images. The results obtained from the test images demonstrate that the gaussian mixture model is able to accurately identify orange cones.

There is a lot of work that can be done to better improve my color segmentation algorithm. I could explore other color spaces like HSV or YCR_CB and determine if those color spaces can more accurately detect the orange cones. I could also use more components to train my model to not detect objects that are close in color to the orange cones (like red). In conclusion, I was able to successfully implement a gaussian mixture model through hand labeling training images, training a model, and calculating relative world coordinates and distance through shape statistics.

6 REFERENCES

https://brilliant.org/wiki/gaussian-mixture-model/
Lecture Notes
https://github.com/jdoepfert/roipoly.py
https://scikit-image.org/docs/stable/api/skimage.measure.html
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html