**ECE 5242 Intelligent Autonomous Systems**

**Project 1: Color Segmentation**

**Alana Crognale arc232**

**Feb 11, 2020**

## Introduction

Color Segmentation aims to use machine learning algorithms to be able to detect where regions of particular colors, shapes, and/or objects appear in new sets of images. In this project, our goal is to detect the location and relative distance from the camera of red barrels in images with varied scenes, lighting invariances, etc. We do this by training a model to learn the classes (colors), segment the desired target class, and pinpoint the barrel object to analyze its location and size details from a set of training images, such that we can identify the center location of and distance from any identified red barrels on new test images.

## Methodology

Labelling data:

For an initial design, I first decided to use two classes – one corresponding to the red barrel and one corresponding to everything else. With more time, adding more classes such as one for blue skies, green grass, or white floors and walls may have improved performance. I also decided to initially maintain RGB color space for the most intuitive analysis and visualization purposes, and since *most* of the images had similar lighting invariances; however, with more time, experimenting with other color spaces or normalizing the color space would likely improve performance especially in more varied test images. Using Python's roipoly, the red barrel in each image within the training dataset was hand-drawn to provide initial data labels. For simplicity and time-purposes, I only took into account labelling one barrel per image, ignoring multiple barrels per image. Then, a binary mask of each image was obtained after converting the image to grayscale, leaving a 900x1200 array with 0s corresponding to pixels with no barrel labeled and 1s corresponding to pixels with a barrel labeled. Then, each mask was aligned to its corresponding image in RGB space in order to extract the pixel's RGB values associated with each of the two classes. After each of the training images were labelled, all "red barrel" pixels were concatenated into one array and all "non red barrel" pixels were concatenated into another array to comprise the complete training dataset.

Learning algorithm:

Firstly, I separated all labeled training data above into temporary test data and temporary training data to test the algorithm described below (prior to running the optimized algorithm on actual test data, all training data was unseparated) by considering every third image to be part of the temporary test dataset for ease in verification. From this new training dataset, I obtained the mean and covariance matrix of both classes for initial parameters, as well as the probability of the red barrel class by dividing the number of entries in the red barrel class by the total number of entries (and likewise one minus this quantity for the probability of the non red barrel class). To start, I used a single (multivariate) gaussian model such that for each of the 900x1200 pixel locations in each training image, the following was performed: calculate two probabilities (the probability of the pixel's particular RGB value given the class using the multivariate gaussian formula defined by each class's corresponding mean and covariance matrix),

one for each class. Then, using Bayes' rule and the above calculated prior probability, we found the probability that this particular pixel belongs to the red class. This probability was compared to a threshold value of 0.8, so that if the probability was above or equal to 0.8, this pixel was classified as belonging to the red barrel class, and if the probability was below 0.8, this pixel was classified as belonging to the non red barrel class. This threshold value was determined by trial and error to see which produced stable performance, although I found that values ranging from 0.8 to 0.9 performed fairly similarly. With more time, a Gaussian Mixture Model (GMM) model would have been implemented with the Expectation-Maximization (EM) algorithm to maximize the log-likelihood and update/optimize the mean and covariance parameters with each iteration, rather than using the same initially calculated mean and covariance parameters throughout.

Region Analysis:

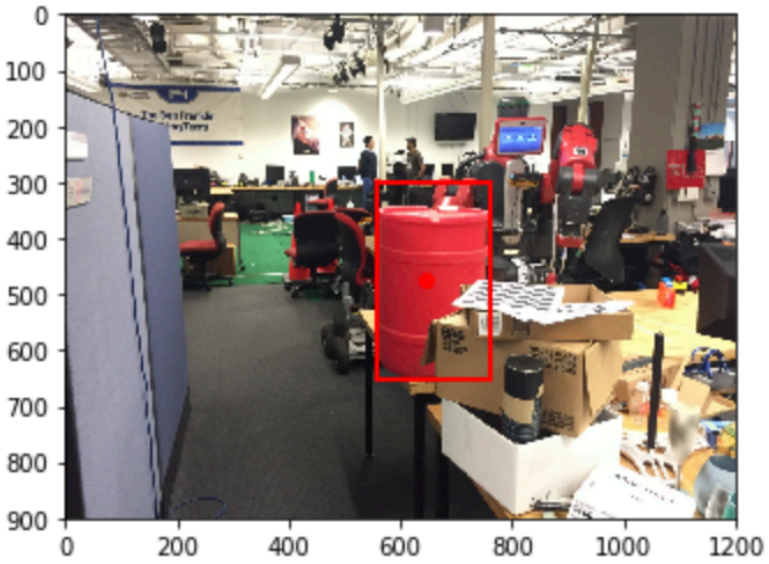
Python's skimage was used on the newly classified image pixel data to identify and display the centroid locations of each identified barrel. First, a binary matrix was made similar to the binary masks from labeled training data such that every pixel classified as a red barrel was labeled 1 and every other pixel was labeled 0. Using skimage.measure.label, connected regions of were labeled, i.e. every cluster of neighboring pixels labeled 1 was grouped together as a region. Then, skimage.measure.regionprops was used on each of the regions to obtain region properties, including area and centroid location. Due in part to the more pared down learning algorithm in combination with lighting invariance not taken into account, essentially any pixel in the red color range ended up being classified as part of the red barrel class and would have been labeled as such, so to filter out the random artifacts that were not actually red barrels, I identified only the region with the largest area as being the image's true red barrel. This method thus assumes that each image will have one and only one red barrel, and that the red barrel in the image is the largest (by area) region of red pixels. Alternatively, I attempted an area threshold so that any region with an area larger than a specified amount would be classified as a barrel to mitigate the issue of more than one barrel being present in an image, however, I found that there were many regions of random artifacts which had larger red areas in some images than the smaller red barrels in other images, so I ended up staying with my initial implementation. Once the red barrel region was identified for each image, its centroid location was located using regionprops. To calculate distance, I used the nature of equal triangle symmetry in the following formula: distance = focal length * actual barrel height / image barrel height. We were given the actual barrel height, to find the image barrel height I took a ratio of the number of pixels of the barrel region's height divided by the total number of pixels in the image's height (900 in this case), and a focal length was tuned through trial and error, ultimately landing on a value of 1.22, to match the provided true distance labels. Finally, the distance and centroid location was displayed for each image, with a red rectangle drawn surrounding the red barrel region and a red dot drawn on its centroid.

## Results

Below are each of the 10 test images labeled in red boxes where barrels were identified with a red dot to represent the centroid location:

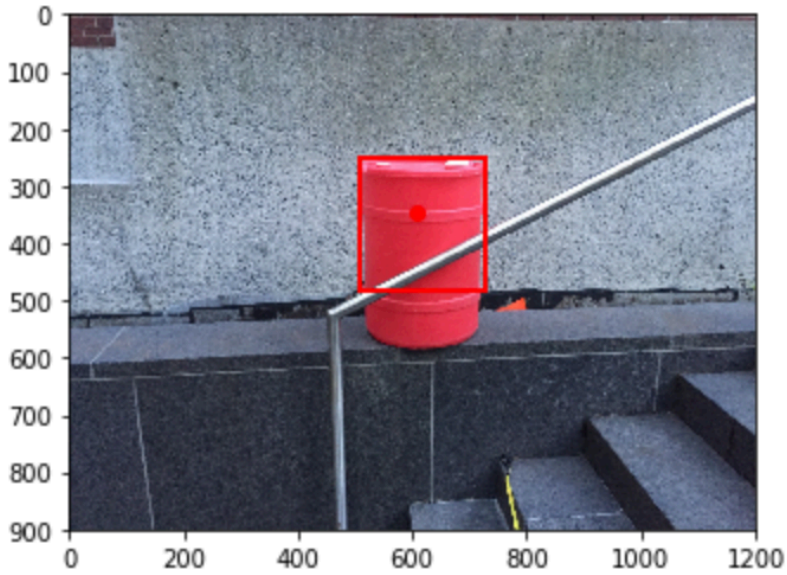
---

```
center coordinates:  (472.61965259833477, 643.113312278687)
distance in meters:  2
```



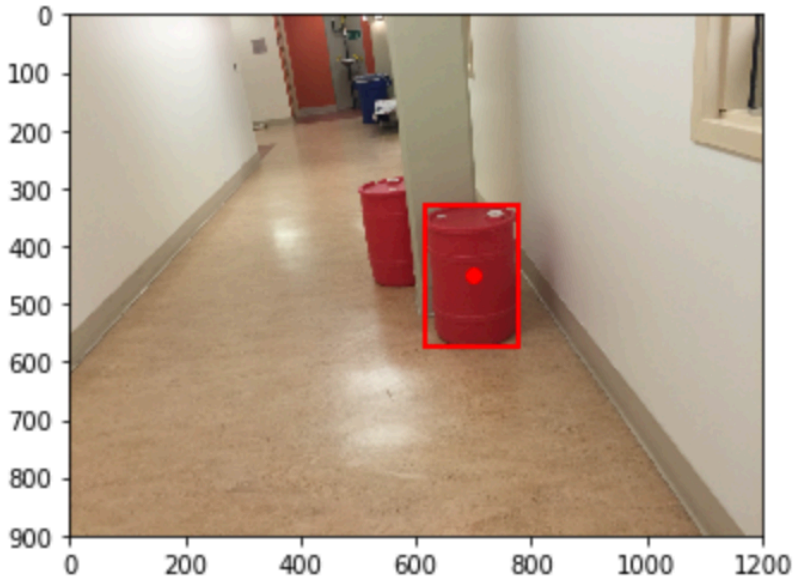

**001.png**

```
center coordinates:  (347.5916657479881, 607.6708190938155)
distance in meters:  3
```



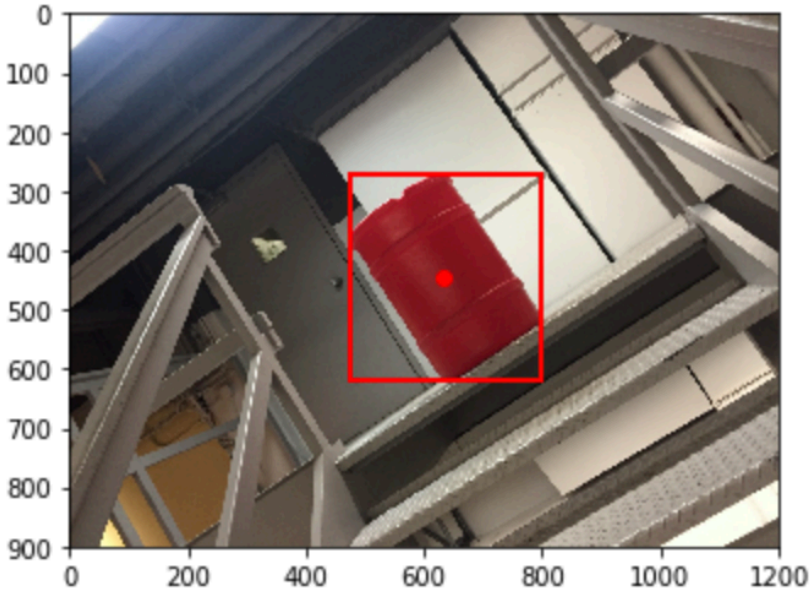

**002.png**

```
center coordinates:  (448.15797070940965, 698.6551890055033)
distance in meters:  3
```



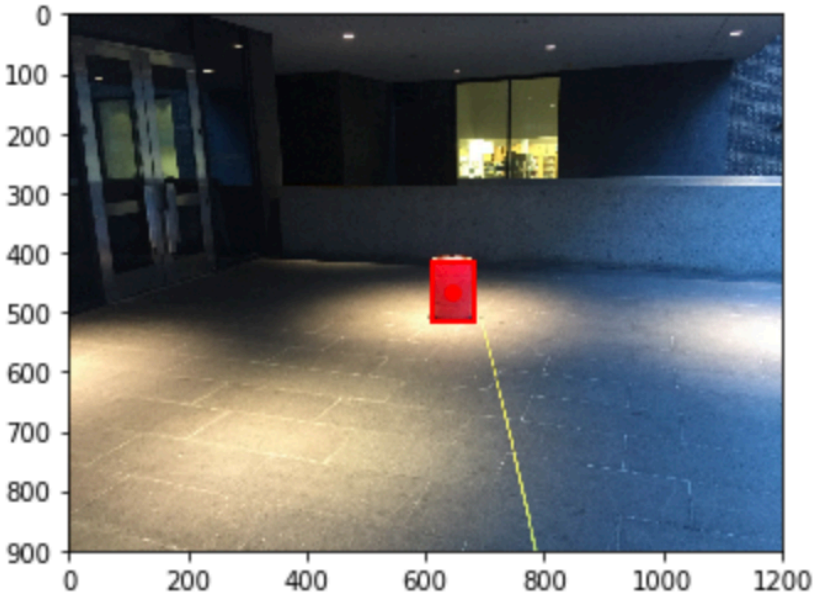

**003.png**

```
center coordinates:  (443.4729383784498, 633.8127239995375)
distance in meters:  2
```



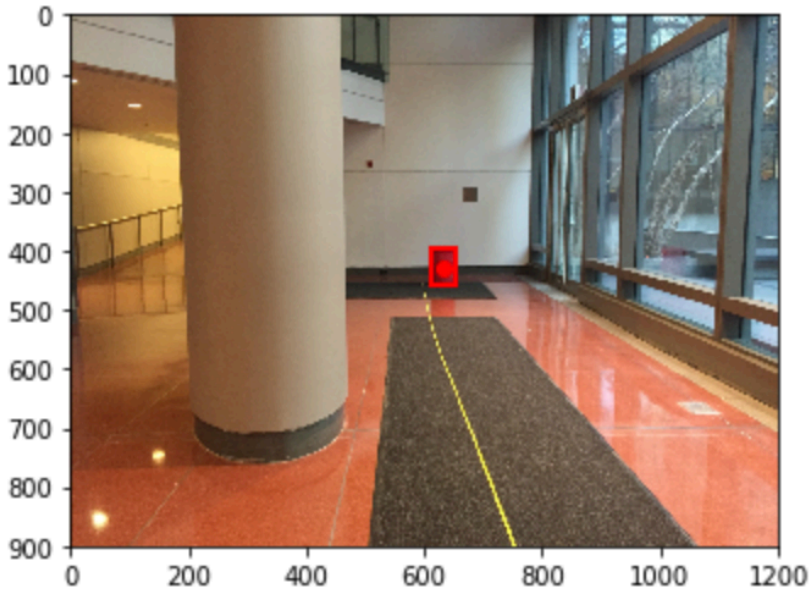

**004.png**

```
center coordinates:  (464.7023378333882, 645.3814619690484)
distance in meters:  6
```



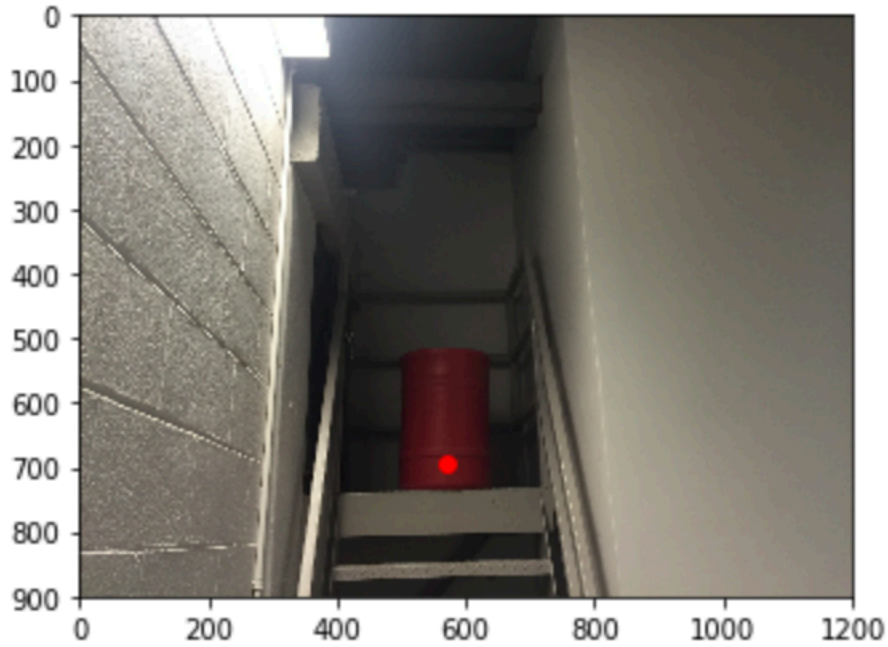

**005.png**

```
center coordinates:  (426.4380017841213, 632.5793933987511)
distance in meters:  10
```



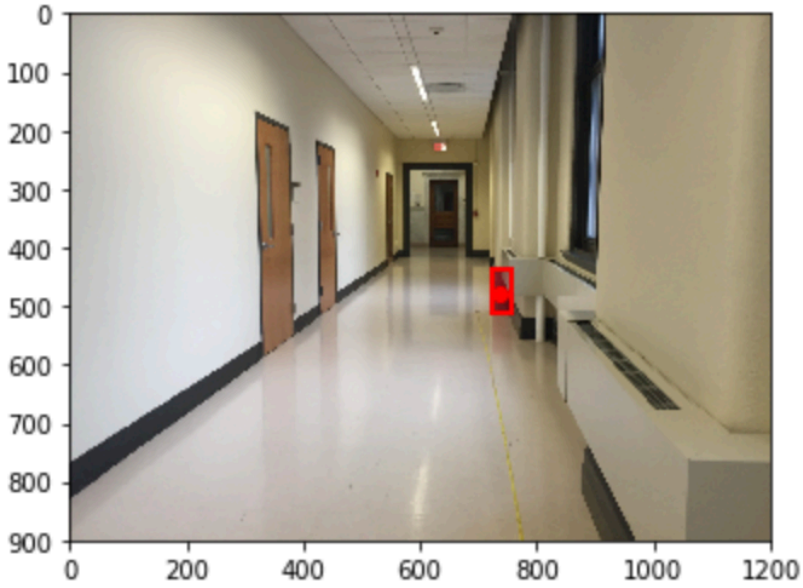

**006.png**

```
center coordinates:  (692.0, 569.0)
distance in meters:  638
```



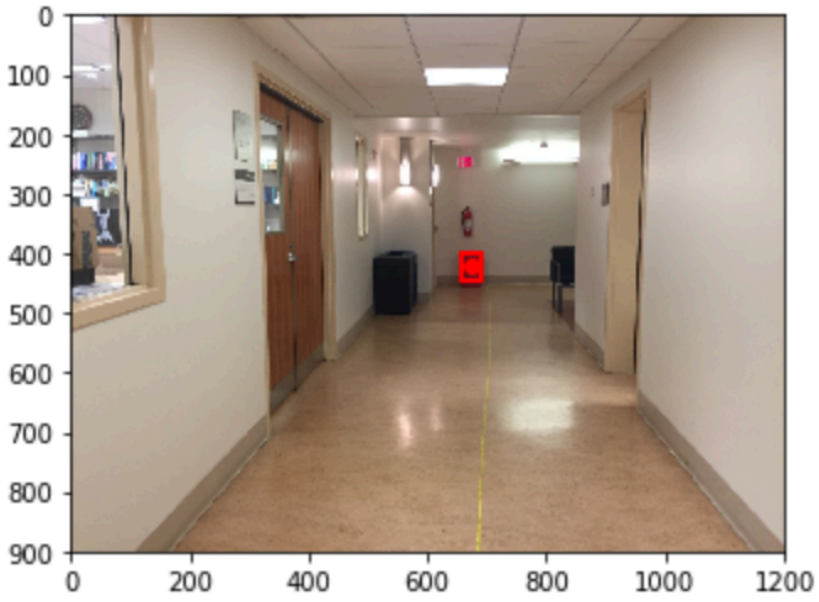

**007.png**

```
center coordinates:  (476.9969456322541, 737.0690287110568)
distance in meters:  8
```



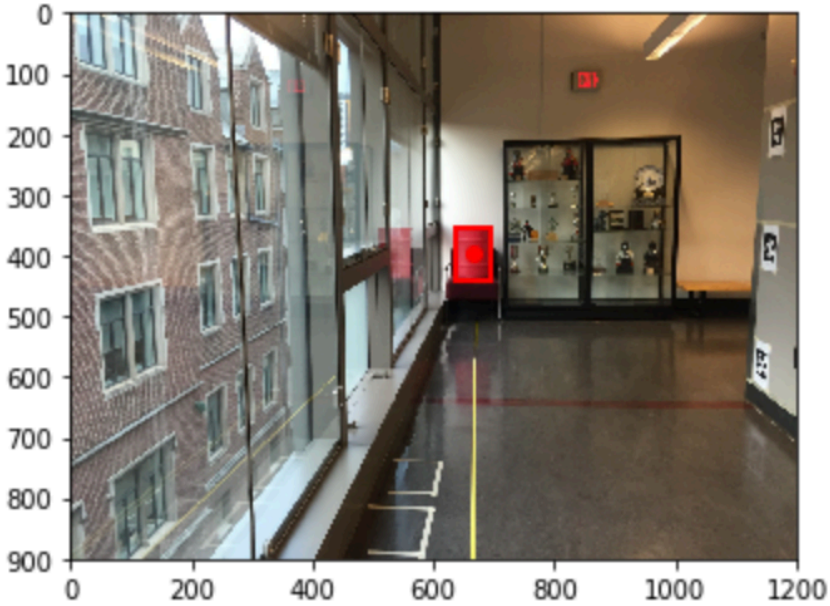

**008.png**

```
center coordinates:  (419.6147032772365, 673.6820194862711)
distance in meters:  14
```




**009.png**

```
center coordinates:  (395.4580550918197, 664.3430717863105)
distance in meters:  7
```




**010.png**

| | CentroidY | CentroidX | Distance (m) |
|---|---|---|---|
| **001.png** | 472.62 | 643.11 | 2 |
| **002.png** | 347.59 | 607.67 | 3 |
| **003.png** | 448.16 | 698.66 | 3 |
| **004.png** | 443.47 | 633.81 | 2 |
| **005.png** | 464.70 | 645.38 | 6 |
| **006.png** | 426.44 | 632.58 | 10 |
| **007.png** | 692.0 | 569.0 | 638 |
| **008.png** | 477.00 | 737.07 | 8 |
| **009.png** | 419.61 | 673.68 | 14 |
| **010.png** | 395.46 | 664.34 | 7 |

**Fig 1. Summary of results**

## Discussion

In identifying a single barrel in a set of test images, I found that the algorithm's identified centroid locations performed accurately, while the distance measurements were sometimes off due to discrepancies in identifying the correct precise barrel region height. For example, in image 001.png, the algorithm identified a slightly larger region than the true barrel region, such that the algorithm thought the barrel was then closer than it actually was. Likewise, for image 002.png, the algorithm identified only a subregion of the barrel, so the distance measurement calculated was smaller than the true distance. For images with clear lighting and one barrel, the algorithm performed quite well, whereas for darker images like 007.png, the algorithm could not identify the correct barrel region height (and thus the distance) because RGB space was used with no normalization to account for lighting invariance. To correct these types of discrepancies and improve the model given more time, I would have experimented with different types of color spaces, implemented a GMM model, EM parameter learning algorithm, and region analysis which took into account a smart area threshold and/or the cylindrical shape characteristics of the barrel to be able to identify more than one (or no) barrels. Lastly, for improved efficiency in time to run code, I would have created a lookup table of precalculated gaussian probability distributions for corresponding RGB values so that the algorithm could quickly identify this value when analyzing the test images, as opposed to calculate new gaussians for every pixel.