CMSC 426 - Project 1
Color Segmentation Using GMM

Authors:
Marc Haller
Anthony Mahshigian
Matthew Wong

1.) Single Gaussian

Our Decisions:

To find the probability distribution we need to find the likelihood of orange pixels. To find this, we need to find all the orange pixels in the given images. To do this, we used the matlab function roipoly() to create masks for each image to determine where the orange ball is. After creating the mask, the covariance and mean for the orange pixels can easily be found through calculations. We then use the probability density function of the Gaussian distribution with the calculated parameters to figure out which pixels are most likely to be orange.

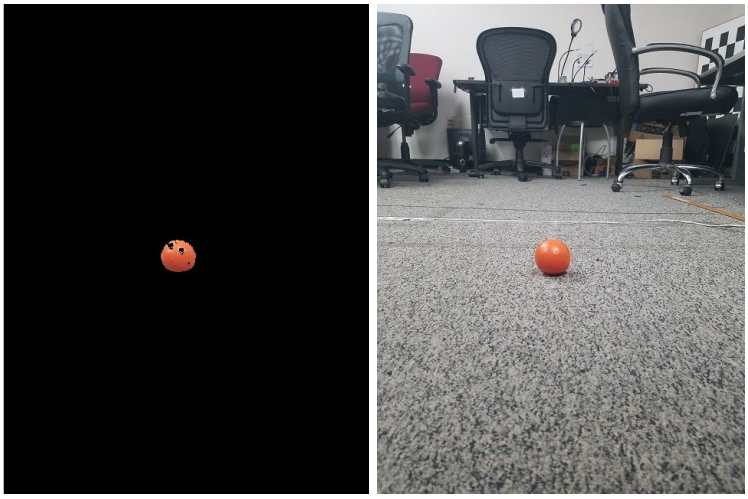
Color Space Choice:

We chose the RGB model for our color space because it is what we were most comfortable using after learning about it in lecture and was easy to use for a Single Gaussian.

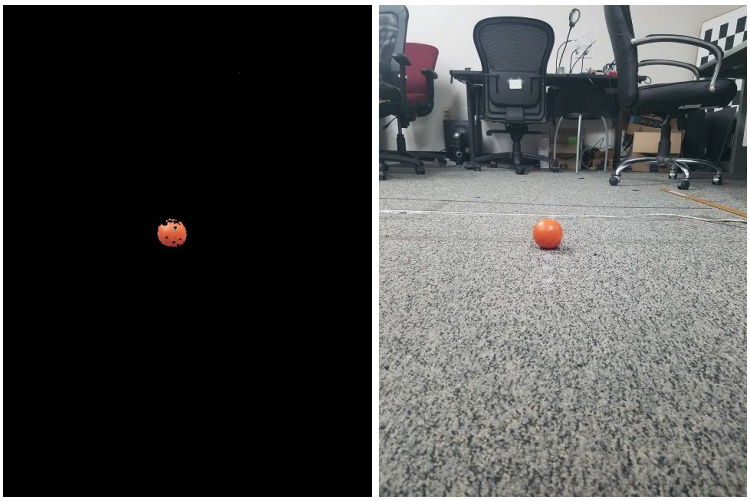
Initialization Method:

To calculate the mean in the training phase, we used the mask that was made by roipoly() to sum and average all the values togethers for the red, green, and blue channels to find the mean for an orange pixel. We used 20 out of 23 images to train and build mu and sigma. For testing, using the final 3 images, we then calculated the likelihood which was modeled after a Gaussian distribution to find all the pixels that the algorithm believed to be orange. This was then multiplied by the prior (0.5 in our case) to get the probability of the posterior. We then set a threshold for each pixel's probability to determine which pixels are read as orange.

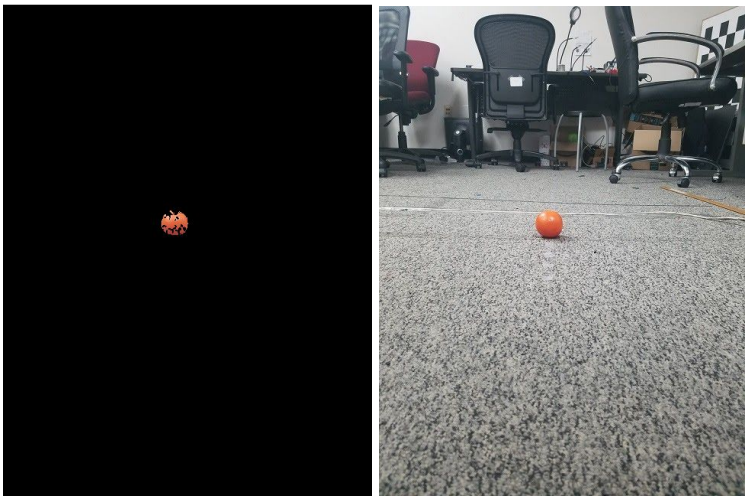
76.jpg



91.jpg

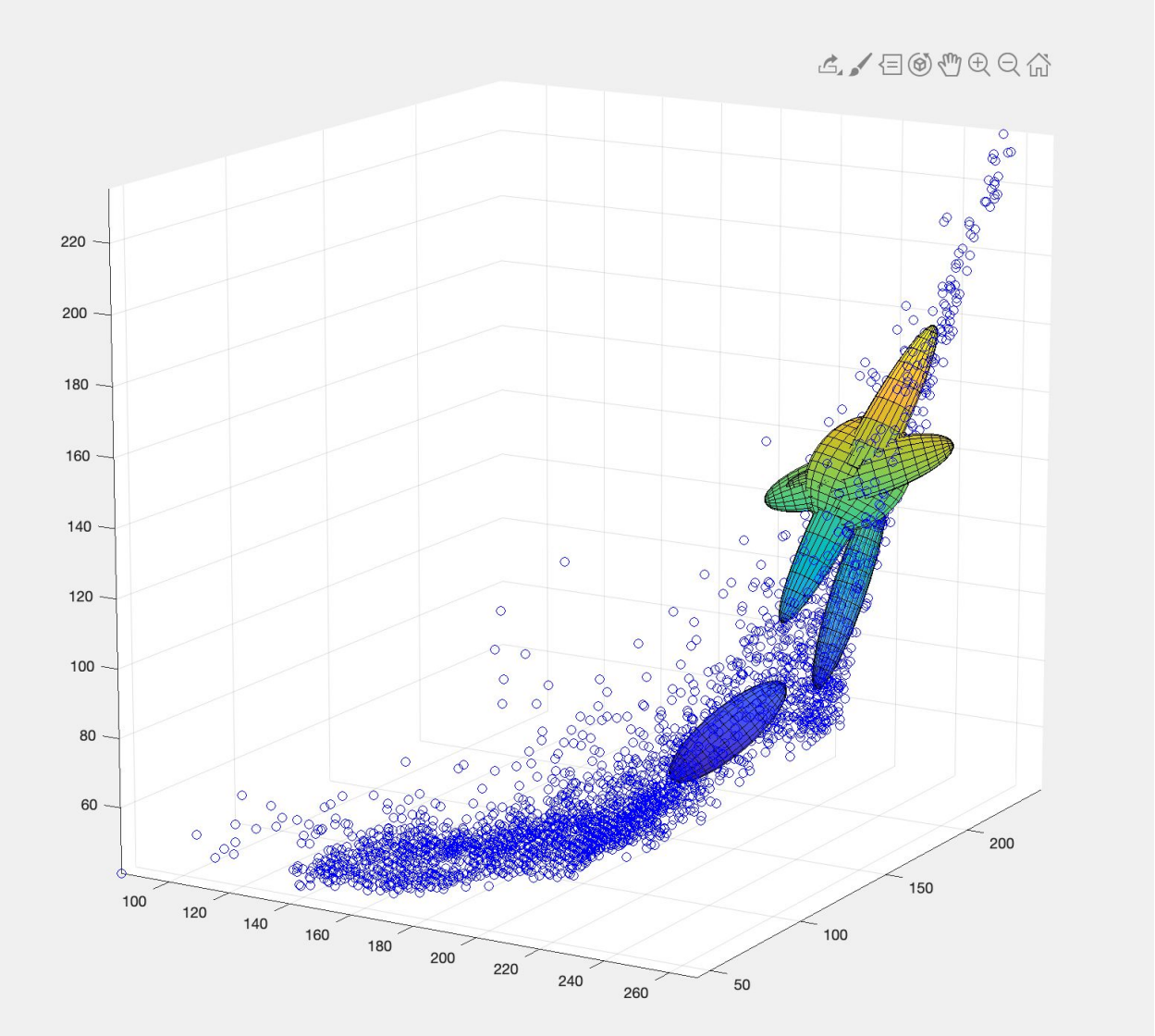


99.jpg



Pictured above is the result of the three testing images we used, with the found orange pixels on the left and the testing image on the right.

2.) Gaussian Mixture Model (GMM)



Our Decisions and Single Guassian vs Gaussian Mixture Model::

Since a single gaussian model isn't completely able to capture the color space due to changes in lighting, the color might not be able to be captured in a single ellipsoid. In order to capture the entire color space to account for changes in lighting, a mixture of ellipsoids are needed to segment the color.

With a gaussian mixture model, we now need to find multiple, up to k, ellipsoids. This can be found by using maximum likelihood estimation with the process being divided into two parts, an estimation step and a maximization step. These steps can be repeated until the function converges (in our case when two consecutive calculated means are within the tolerance of 0.000001)

Color Space Choice:

We chose RGB for our color space because after looking at different color spaces, we found that this was easiest to work with because it was most intuitive - all of us having our experience in RGB, and it was what was gone over in lecture.

Initialization Method:

To initialize the mean, we used the rand function in matlab.
Initializing the covariance is harder for GMM because the covariance matrix has to be a positive semidefinite matrix. We just initialized to the same value for the diagonal, and then it was adjusted throughout the algorithm. We ended up with vastly different Gaussians that did a pretty good job at covering the training data.

Number of Gaussians:

For our dataset, we chose the number of gaussians to be 7. We chose this number because if the number of gaussians are too small, we wouldn't be able to entirely capture the entire color space which would result in the algorithm underfitting. On the other hand, if we chose a larger number of gaussians, the algorithm might be more accurate but at the expense of the algorithm overfitting which would not be a reliable generalization since overfitting can cause a poor prediction.

Distance Calculation:

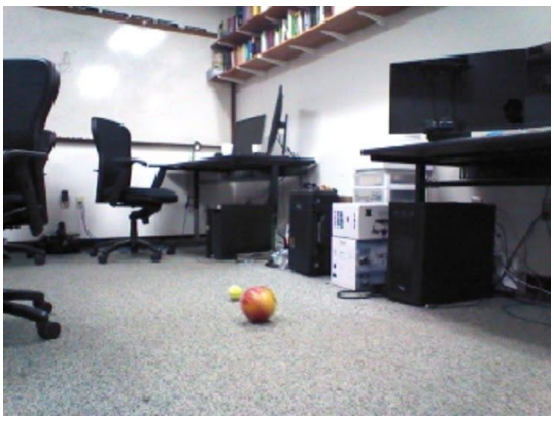
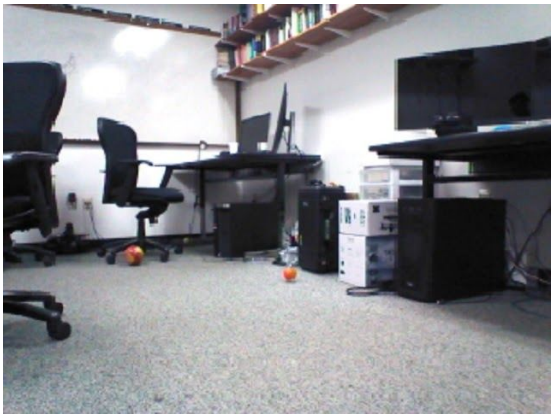
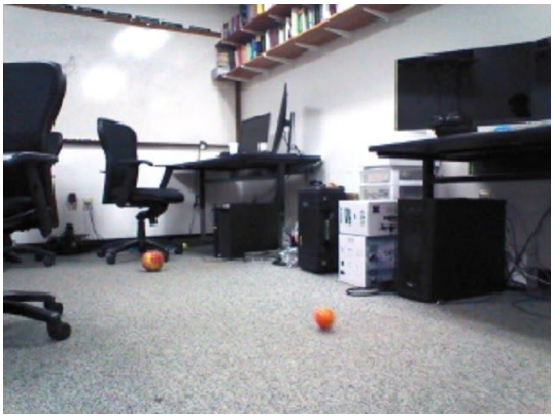
Training Phase:

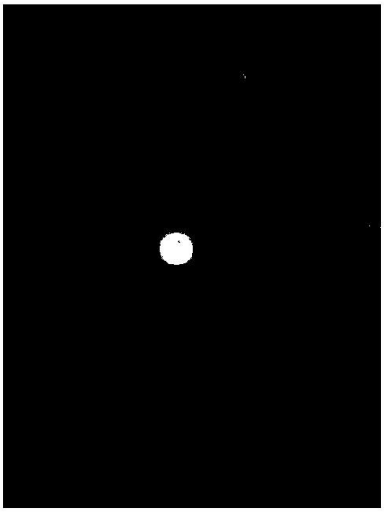
To train, we calculated the area of the orange ball using the masks that were produced by roipoly(), since the resulting picture is a binary image. We then fit a polynomial curve from the areas calculated vs the distance away from the camera given from the picture names.

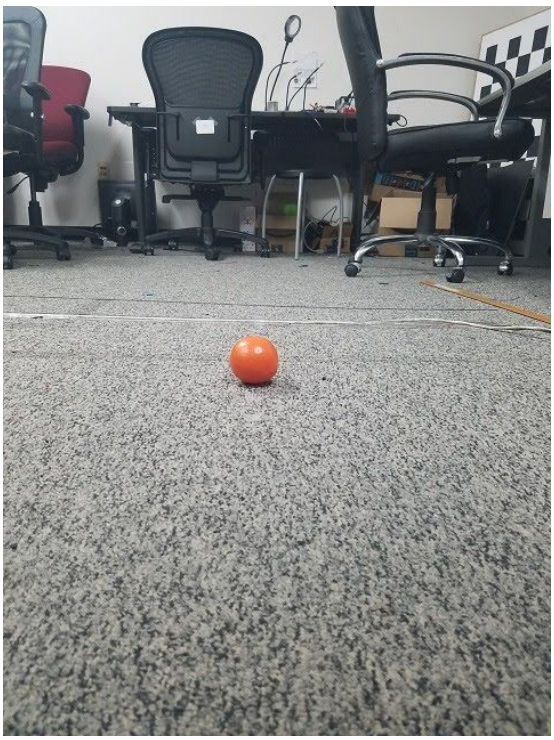
Testing Phase:

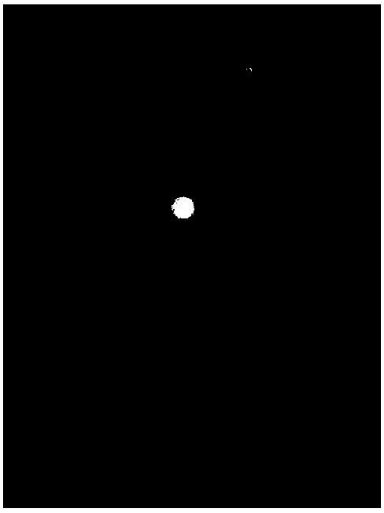
To test we first calculated the area of the orange ball. Then using the parameters of the curve that was made in training and the areas from the testing phase, we can approximate the distance of the ball given the measured area.

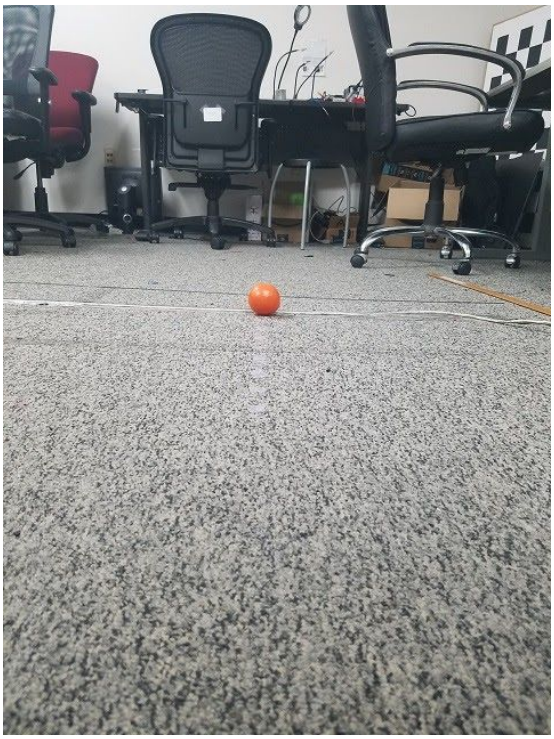
Test Image Results

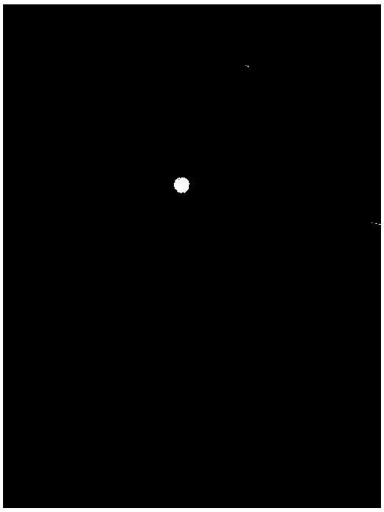
	
	
	

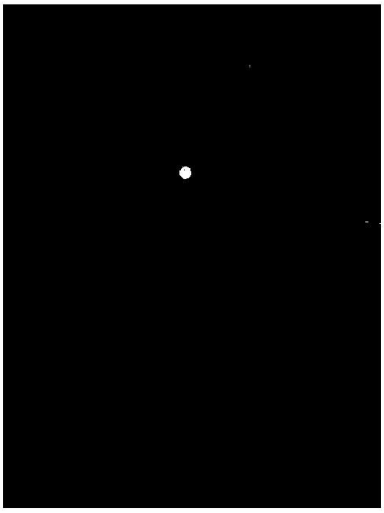


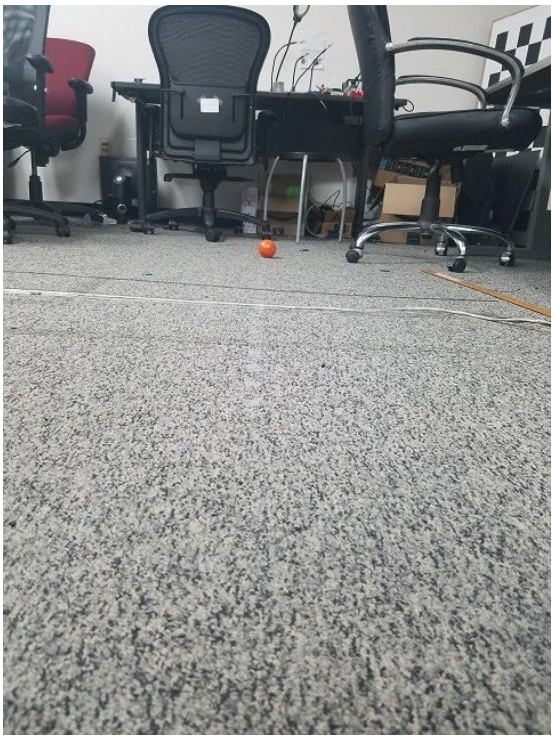


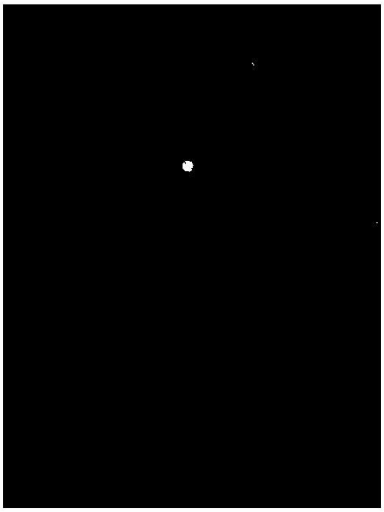


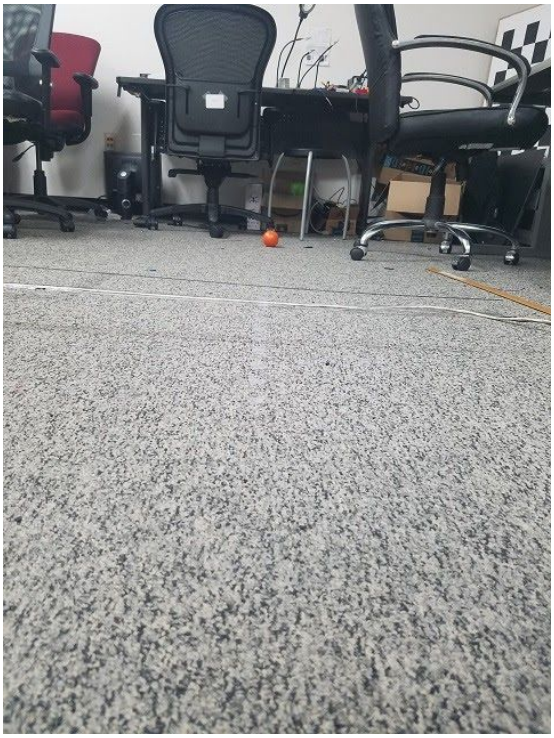




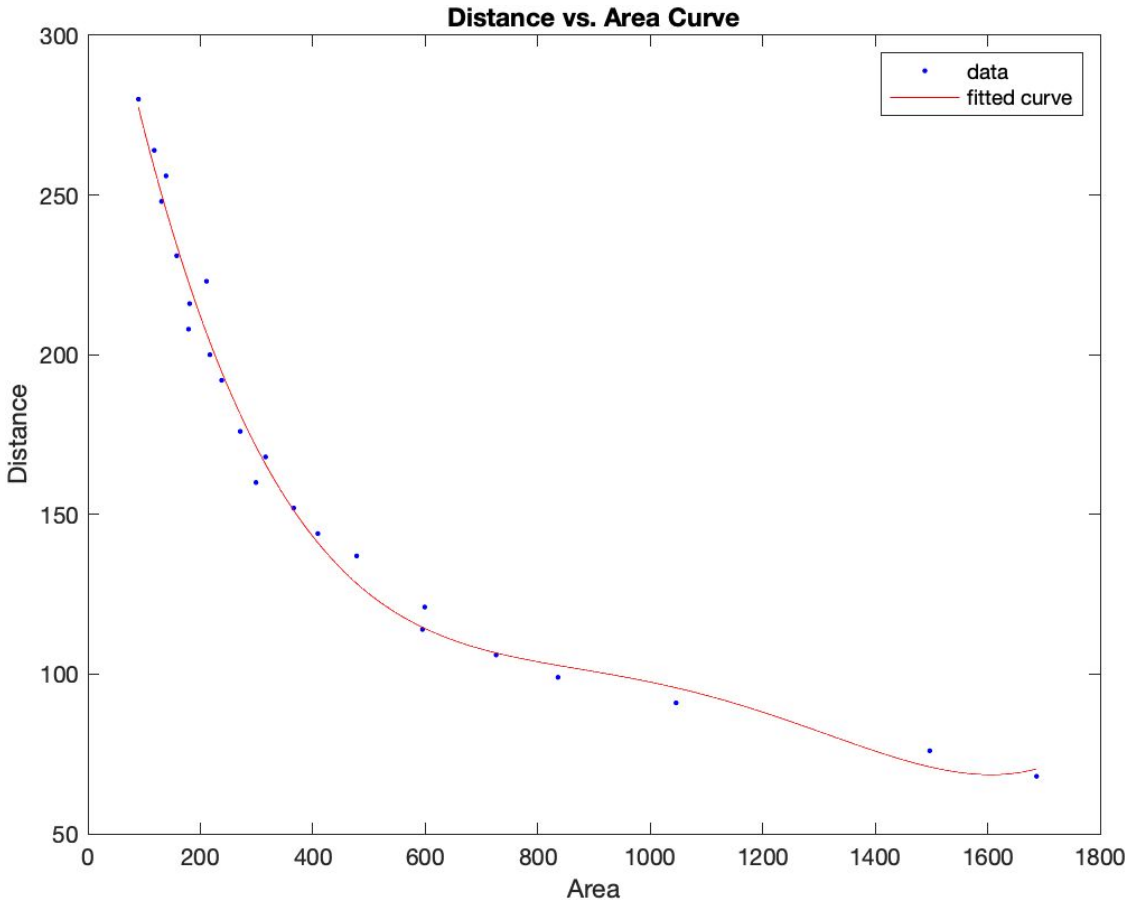








3.) Distance Estimation



This figure shows the line that we fitted with the training image data which will then be used to approximate the distances for the testing images.

Distance Estimation Results:

Image Name	Estimated Distance (cm)
1.jpg	171.924
2.jpg	245.9795
3.jpg	150.4835
4.jpg	87.9268
5.jpg	117.8466

6.jpg	196.348
7.jpg	250.2664
8.jpg	271.1413

4.) Overview

Problems We Faced:

While doing the single Gaussian model, we had issues with calculating the correct determinant of the covariance matrix. There were lots of mix ups over where to divide sigma by n and we ended up doing it more times than needed, which skewed results and had every pixel be orange. Essentially, our calculations were off.

A problem we had when doing the distance calculation was that some of the masks that we made resulted in giving us an area of ~900 pixels when the rest of the masks were ranging between [10-100] pixels.

Another challenging problem that we had was determining the threshold values for our single gaussian and gaussian mixture model.

How We Solved Our Problems:

To solve the problem with our single Gaussian, we essentially read through the code and made sure every calculation at each step was done exactly once, and we deleted repeated steps.

We solved the problem with the ball sizes by remaking our masks in roipoly() until they were a consistent size (either 59-60 KB) with our other masks files. This helped us to create an accurate fitted curve to determine the distance of the orange ball.
Strengths:

The strengths for the single gaussian is that it is not as computationally intensive as GMM.

The strengths of the GMM algorithm is that it is not sensitive to different lighting conditions.

Limitations:

One limitation we had while doing the project was using roipoly() to create the masks for our testing data. This was a limitation for us because there is always going to be some human error when creating the masks because we have to manually click and create the outline for the orange ball when trying to make our masks.

One of the limitations we found in the single gaussian algorithm was that it could not handle noise inside of the ball such as reflections of light or shadows. It would also often identify pixels in the background as orange since they would have close RGB values.

Another limitation that was imposed on us was that the performance of the algorithms heavily depended on the thresholds that we chose.