William Schulmeister
Yashashri Pendse
Sarah Bang
Brian Chin

Report

CMSC 426 Project 1: Color Segmentation Using GMM

Table of Contents

Color Space 1

Single Gaussian 2
General Methodology 2
Shortcomings 2
Output 3

GMM 7
Constants 7
Train 7
Test 7
Plot 7
Distance 8
Shortcomings 9
Output 9

Color Space

We used the RGB spectrum as that represented what the human eye saw most accurately. It also offered a degree of simplicity for implementation and testing. While there were more complex color spaces available (that could've fixed some issues that came up further at the cost of slightly more space), we stuck with RBG as the default.

Single Gaussian

General Methodology

For training, we used roipoly to outline the ball in each picture. This choice was made largely because of the TA's encouragement, but it also offered security in the training sample. The program would then compile a list of "orange" pixels which was simply a list of all the pixels in this polygonal space. From there, it would simply calculate the empirical mean and covariance and save them to a file called *singleGaussModel.mat*. In the general case, this model could just be loaded to save time for predicting.

To predict, we selected a threshold of .0000004 as it showed strong results in our training. We also selected a prior of .5, implicitly selecting two "colors" for probability: orange and not orange. Once the probability was found, the predicted binary image contained pixels that were black if under the threshold and white if over.

Shortcomings

On a basic level, the single gaussian (SG) is very dependent on user input. Even a slightly off training sample could skew the results dramatically. There was a very delicate balance finding a good threshold as the carpet as well as some background features ended up quite similar to the ball.

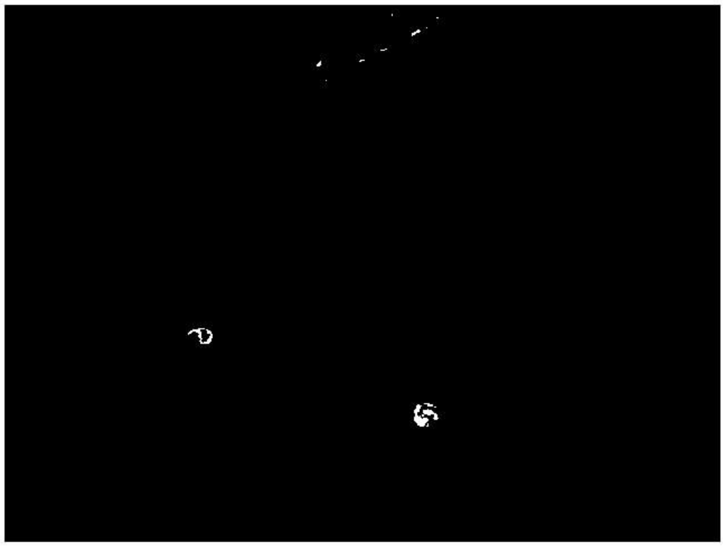
Additionally, single gaussian struggles with lighting issues. Since it only can make one empirical mean, it can struggle with the darker oranges from the shadows and the highlights from the light. We adjusted our threshold to be slightly more lenient to account for these issues, which in turn resulted in decent output, but the shortcomings became evident in the first 3 test images. Other images with similarly different lighting schemes would also give it significant problems.

Output

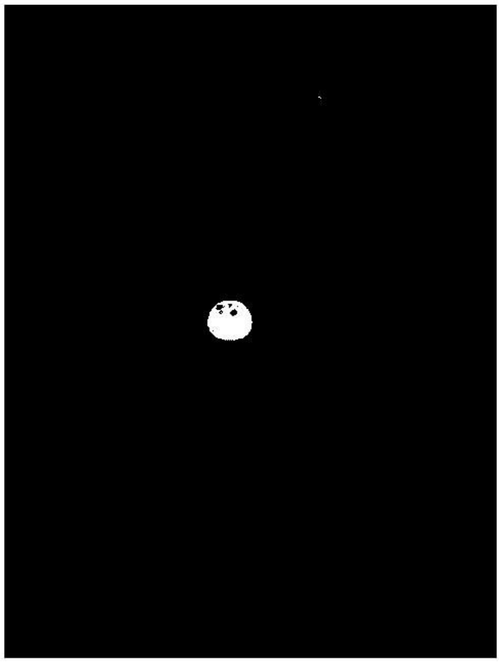
Test Image 1

Test Image 2

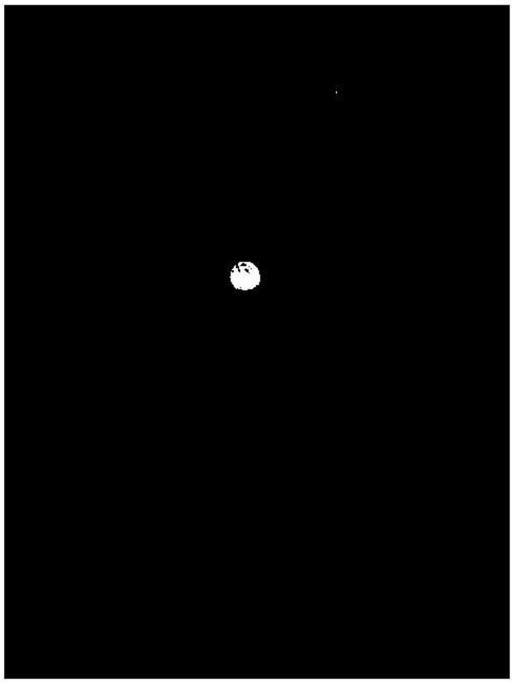
Test Image 3



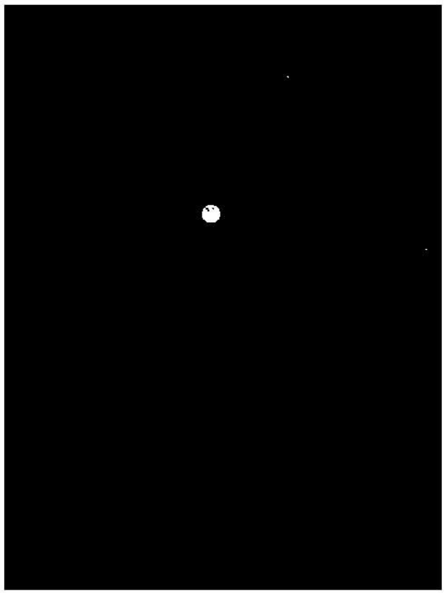
Test Image 4



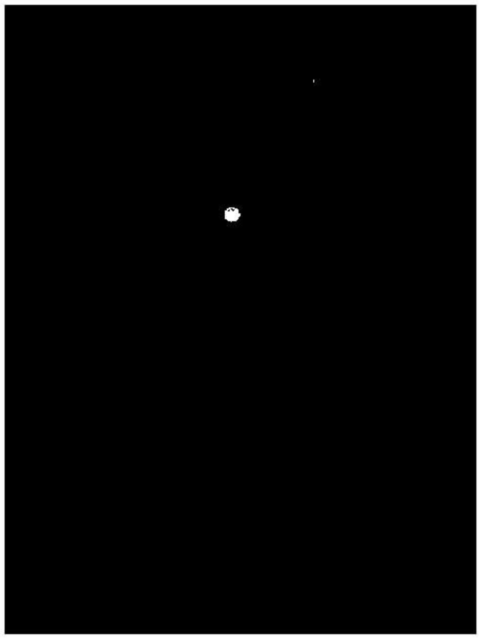
Test Image 5



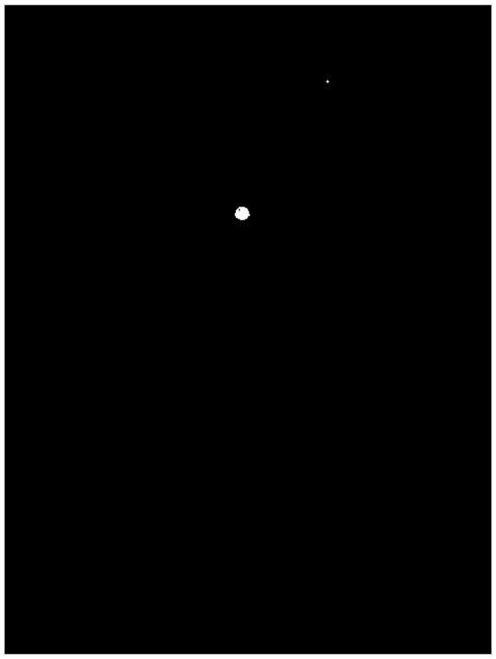
Test Image 6



Test Image 7



Test Image 8



GMM

Constants

We selected a K of 5 for our GMM. The general idea was to model the varied tones of the ball without losing too much speed. The 5 are darkest orange, less dark orange, neutral orange, lighter orange, and lightest orange (although we didn't have control over the algorithm approaching it the same way). The idea was to mitigate the lighting shortcomings of SG.

We selected a threshold of .0000001, slightly more lenient than the one used for SG. We chose it largely because of its performance on the testing set, but also because we felt that the larger K would be more resistant to lighting and thus less prone to false positives.

We selected maxIter to be 1000 to ensure that the algorithm was given every chance possible to converge. In every case observed, however, this number was not reached. Our convergence criteria, .0000001, ensured that the algorithm ran about 250 iterations before finding a good convergence.

We selected a prior of .5 for the same reason as SG.

Train

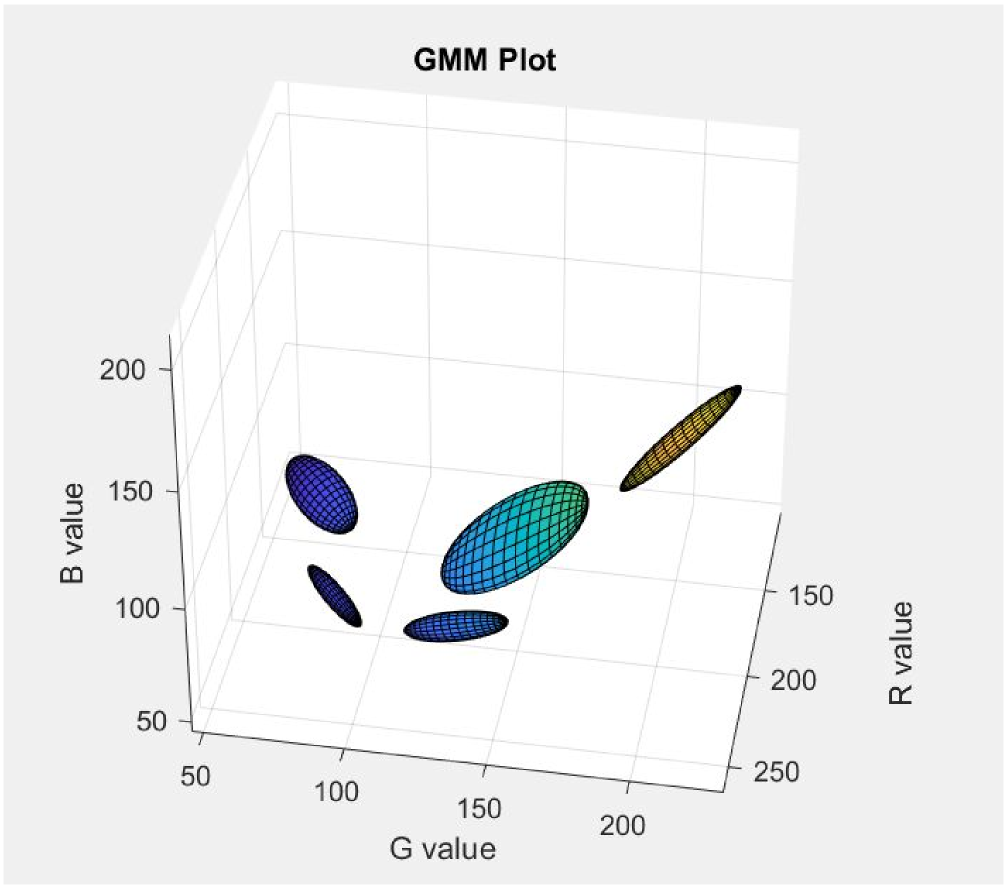
The start of training and gathering of orange pixels is done the same way as single gaussian. We used the iterative procedure of the Maximum Likelihood Estimation to approximate where the ball was. While mu and pi were initialized to random values using MATLAB's rand function, covariance was initialized to the covariance of the orange pixels. Although not truly random like in the project description, this was random enough for its use and guaranteed a square, PSD matrix. We first implemented the estimation step and then alternated until convergence with our previously defined threshold. Results are saved in *GMMmodel.mat*. Unlike our SG however, we used MATLAB's mvnpdf function to find likelihood. The effect is the same.

Test

Predicting on test data follows the standard procedure outlined in the specs. Like SG, the predicted image is binary with orange as white and everything else as black. Each image is added to a cluster modeled by a cell array to allow variable sizes and returned.

Plot

For plotting GMM, we went through each gaussian and called the mu and sigma (covariance matrix) values corresponding to that gaussian and plotted the gaussian ellipsoids. This was done to help us visualize the color space as well as give context for the results. In our GMM plot, if we clicked different places in the plot, the different RGB values that corresponded to orange were revealed. The model we used for all our results is displayed below:



Distance

Distance was implemented by first saving the predicted images from a GMM run (K=5, threshold = .0000001) on the train images into *distanceCluster.mat*. Those binary images were then manipulated using bwmorph with the 'majority' option to attempt to take out any outliers and solidify the balls. Once this was done, regionprops was used to find the largest circle of each image whose radius could be used to describe an area. A degree 4 model could then be fit (MATLAB's polyfit) from the areas found to the distances (present in the file names of the training images).

GMM ultimately fed the predicted images of the test set into a similar algorithm. Like the "training" above, they were manipulated and circles were found. The area of these circles was then plugged into the coefficients to estimate a distance. The output is as follows:

Measuring Depth:

```
{'Test Image 1 is an estimated 366.3909 units away.'}

{'Test Image 2 is an estimated 334.3631 units away.'}

{'Test Image 3 is an estimated 362.3071 units away.'}

{'Test Image 4 is an estimated 86.6215 units away.'}

{'Test Image 5 is an estimated 123.2115 units away.'}

{'Test Image 6 is an estimated 183.2808 units away.'}

{'Test Image 7 is an estimated 246.3518 units away.'}

{'Test Image 8 is an estimated 263.2882 units away.'}
```

The distance was ultimately at mercy to the quality of the GMM model, so the shortcomings of the distances from test images 2-3 are explained by the shortcomings of that algorithm. Image 1's behavior is expected as there is no actual orange ball to predict. The last 5 seem to be very accurate.

Shortcomings

It is difficult to assess the quality of GMM. The results from test images 4-8 were noticeably better than SG, as the highlights and shadows were more accurately predicted. For some images, the ball was almost entirely recognized and image 1 was appropriately without a ball. The algorithm still struggled with test images 2-3, however. We believe that this was to do with the lighting and image quality differences in these two images. Whether the color space or lack of processing to combat this, GMM was not able to recognize the ball with a different angle. At the end of the day, GMM is only K single gaussians. It is not meaningfully more resistant to these problems, only K-1 more colors.

GMM also suffers from the same implications of user input that SG did.

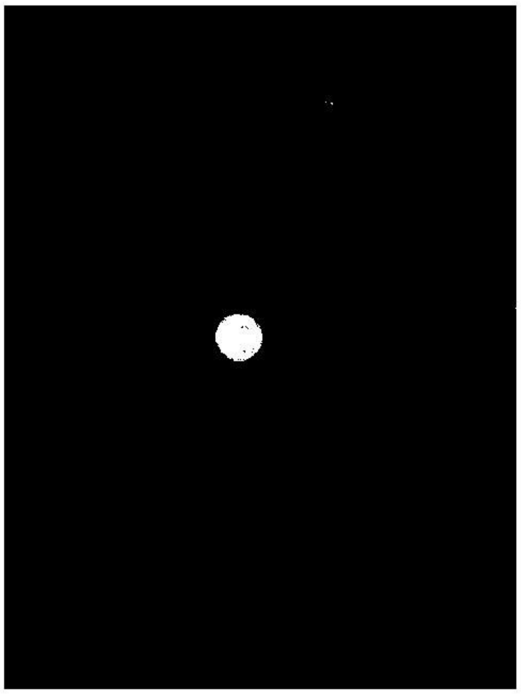
Output

Test Image 1 is an estimated 366.3909 units away

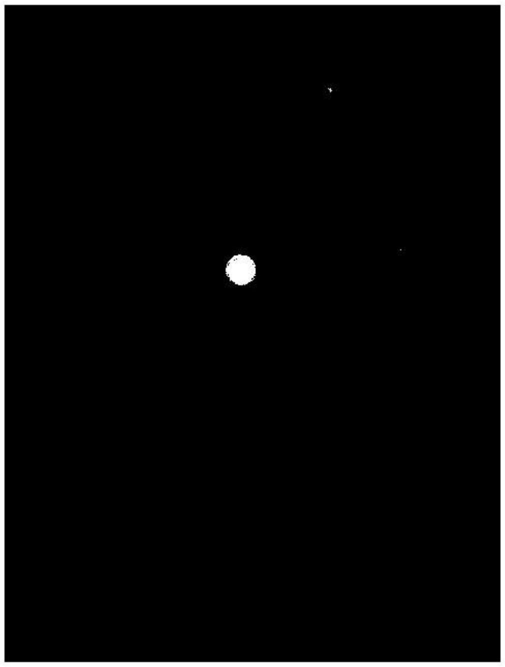
Test Image 2 is an estimated 334.3631 units away

Test Image 3 is an estimated 362.3071 units away

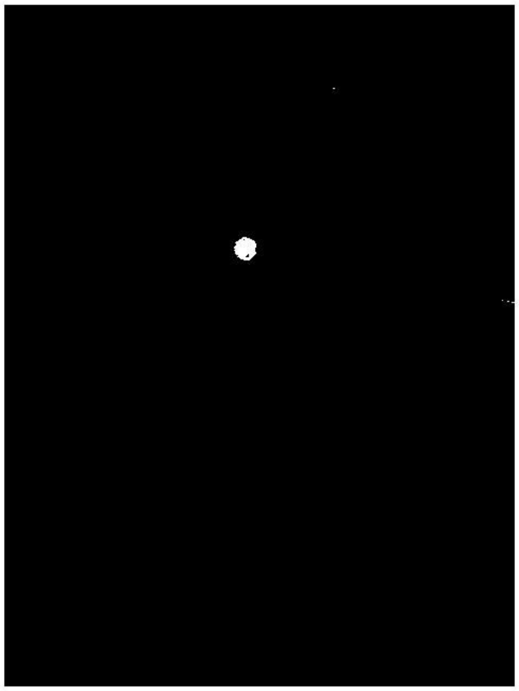
Test Image 4 is an estimated 86.6215 units away



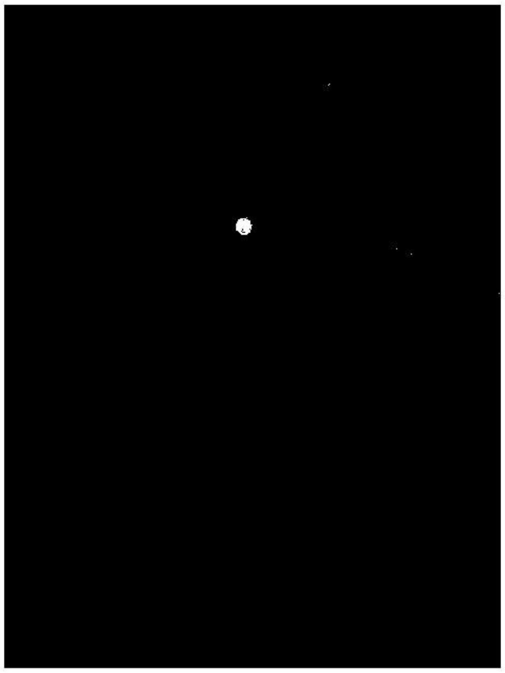
Test Image 5 is an estimated 123.2115 units away



Test Image 6 is an estimated 183.2808 units away



Test Image 7 is an estimated 246.3518 units away



Test Image 8 is an estimated 263.2882 units away