Color Segmentation Report

The report contains six parts, introduction, problem formulation, technical approach, failure discussion, tests results and brief summary.

The key points in the report are applications of multiple models to filter out objects with similar color as red barrel, DFS algorithm applied in image processing and improvement method to make the algorithm perform better.

I. Introduction

This project is aimed to train a probabilistic color model from training image data, the most common used one is Gaussian model. And then we use this kind of model to segment and detect a target of interest objects. In this project, we need to figure out all red barrels in image. The basic procedure shows below:

a. Label ROI in the image manually and store all data in certain file.

b. Construct corresponding models for each data sets respectfully.

c. Use algorithm to obtain the bounding box (contour) of detected red barrels in the image.

d. Apply image processing for each red barrel and using camera parameters to calculate four corner positions and distance information of each red barrel.

II. Problem formulation

A. Color Space

It's pretty tricky to choose suitable color space when training and testing images. In order to figure out red color much clearer, I convert image color into HSV form.

The trick when using HSV is, the value H of red color might locate in 0 ~ 10 or 170 ~ 180, so at first I thought training all red color in one GM is quite not suitable. However, after viewing the model of HSV and knowing the singularity of HSV, I found that GM is suitable to represents red, we can add 180 for each H value of pixel and combine two sides distribution data into the middle one.

In this project, I trained GMM for red color class which I think will guarantee the accuracy of model.

B. Multiple Models

Only training red color of barrel region is not enough. Since there might be other similar red colors in the image belonging to non – barrel objects, such as red chair, red wall, res bicycle and so on. If we only apply one model for detection and some other red items stay close with red barrel, the algorithm will treat this part of region as a unit one. Increasing the threshold is a possible approach to deal with above problem, but too high threshold will always filter out pixels in red barrel as well and determine suitable threshold for different images is quite difficult.

Therefore, training extra GMMs for these confusing items is necessary for separating red barrel with these confusing objects and reaching higher accuracy of detection as well.

In addition, setting corresponding weight for each GMM also matters a lot. Here are two possible approaches. First one, it's pretty fine to use a priori on the color which is uniform, that is, compute the probabilities of all GMMs and pick the max. The other one is utilizing a prior that favors some colors or models, such as set higher weight for red barrel while relative lower weight for red wall. Therefore, using Cross Validation set is a perfectly fine approach to decide weights.

C. Time Complexity in Code

Either training data or testing images, we need to deal with a quite large number of pixels as well as RGB values, therefore, using for loop in code to handle each single pixel is not ideal approach which will cost too high time complexity. However, vectorization will speed algorithm a lot and benefit the whole project.

D. Image Processing

After color segmentation, in order to detect red barrel correctly, it's necessary to handle several following situations. Locating a whole and intact red barrel shown in the image is pretty straightforward, however, when some other objects occluding in front of target red barrel which will lead the detected region to be separated into several pieces, or some certain items with quite similar red color locate near to red barrels which can cause extension of red barrel contour. Both above cases will confuse the algorithm and may make the computer ignore these target barrels.

Besides, when computing physical distance of detected red barrel, rotation of camera will cause a quite error between computed distance and the actual one since it's necessary to consider and multiply rotation matrix instead of only rectified projection matrix.

III. Technical Approach

This part shows some several technical approaches in my code to deal with mentioned problems above. And we mainly focus on decision estimation and DFS algorithm in the image processing.

A. Hand Label

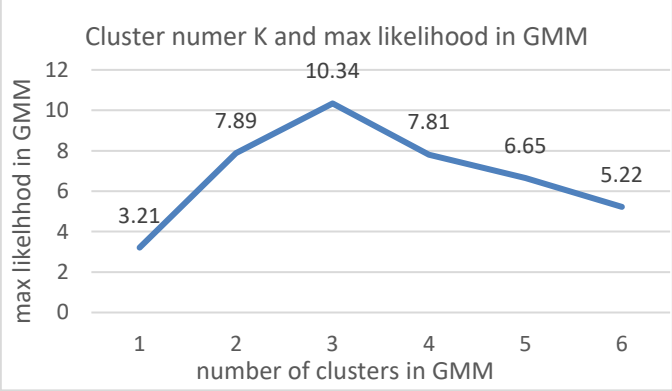
Typically, using built function in openCV to read in training image and convert color space for better vision while labeling. Also use 'roipoly' to label region of interest in the image and then store corresponding pixels within the mask for later training.

One trick is, when storing target pixels based on mask, no need to pass through all 3 layer images which will cause memory unpack problem, as the mask is same for the whole image, so one layer is enough.

B. Model Training

When training probabilistic model, vectorization is quite important. Although it's not wrong to train data stream one by one, vectorization will truly speed up process quite a lot, especially when constructing GMM. Below are two examples in GM and GMM training code using vectorization.

In this project, I choose GMM as my probabilistic model. EM algorithm is the core part of model constructing. And the suitable number of cluster in GMM also matters a lot. Typically, I use cross validation set to test max likelihood of model with different K value and choose the max one as ideal number of clusters. Below is the diagram shows the relationship between number of k in my red barrel GMM and the max likelihood of training data.



According to the curve above, I select k = 3 for my GMM of red barrel. Same procedure for other extra GMMs.

In addition, suitable data structure of training set helps a lot. For me, I store all parameters belonging to a single GM in a dictionary, the key is a string shows the type of parameter and the corresponding value is array stores mean, covariance and weight. Below is the demo data structure in my GMM code.

```
table = {}
table["r"] =  np.zeros((1, size))
table["mean"] = barrel[random.randint(0, size - 1), :]
table["covar"] = np.array([[diff[0], 0, 0], [0, diff[1], 0], [0, 0, diff[2]]])
table["wei"] = 1 / float(k)
set_GMM.append(table)
```

Besides, the initialization is also important. Instead of using K – means approach, I randomly select data in training sets as my initial mean value for each cluster. For covariance matrix, I compute the difference between the max and min value of H, S and V value, then create a diagonal matrix as my initial covariance matrix. Initial weight is just uniform, 1 / k.

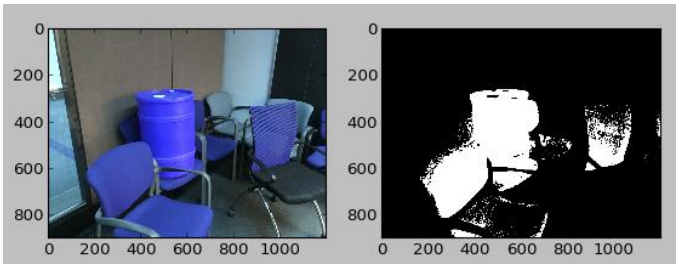
For EM operation, it's necessary to consider terminal condition. For me, I set the total iteration times, and also comparing the current max likelihood with previous one, if the difference is lower than the threshold, which means converging enough, just jump out the loop.

C. Decision Made on test image

Essentially, we decide the label of each pixel via the probability computed by Bayes Rule. The trick thing is, computing the probability of color respected to the given pixel directly is quite difficult while getting the probability of pixel with respected to the given color class is straightforward and easier.

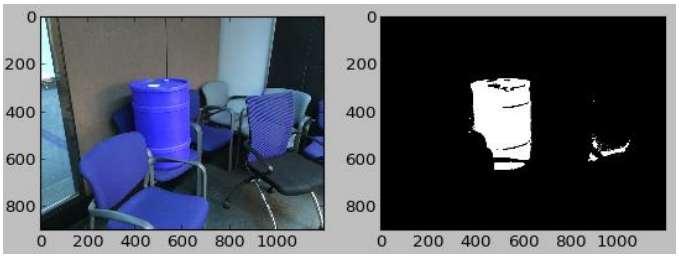
What's more, since I trained several GMMs to represent different classes, the selection of suitable corresponding weight matters a lot. Of course, using a uniform weight is not a bad idea and the result performs well, however, it's better to set a favor weight for different GMMs in order to increase the detection accuracy, and we can apply cross validation set to obtain and verify these weight factors. Finally, pixels belonging to the red barrel class must get the max likelihood value from red barrel model compared with other GMMs.

Here is a simple example to show the difference of using only one GMM to do decision estimation with applying several GMMs. When I apply only GMM of red barrel for detection, I can get below result.



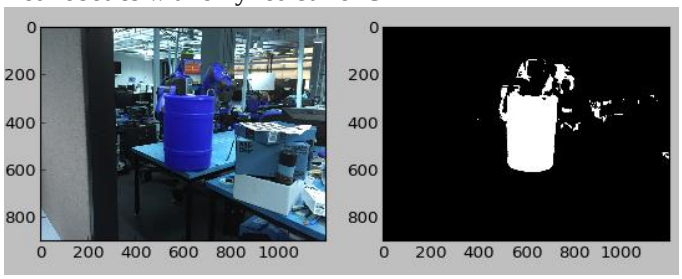
You can find that although the algorithm red barrel region, it will also detect chairs region nearby since the color of chair is similar as red barrel. Therefore, applying only one GMM is not enough unless I set higher threshold to filter those noise pixels, but higher threshold will also drop true pixels in the red barrel.

Due to this problem, I train another GMM for representing red chairs and then for each pixel, compare its probability in these two classes. And the result looks better.

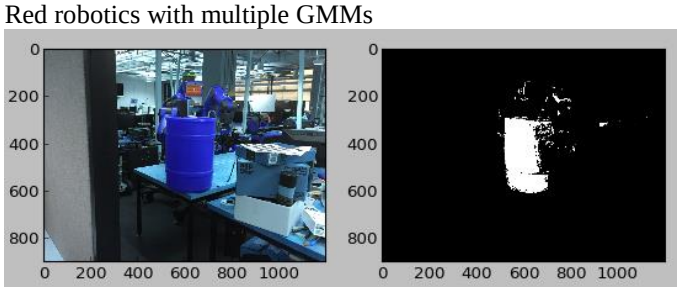


Below are other examples to show the importance of multiple GMMs in color segmentation. And I total trained four GMMs in order to filter out confusing pixels.

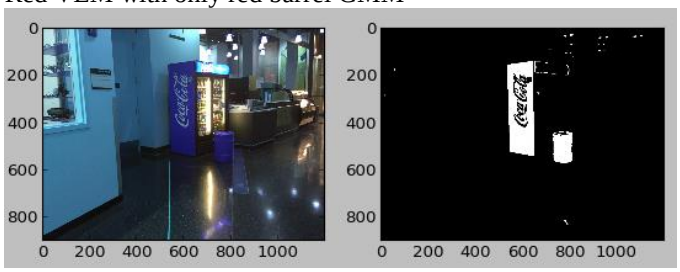
Red robotics with only red barrel GMM



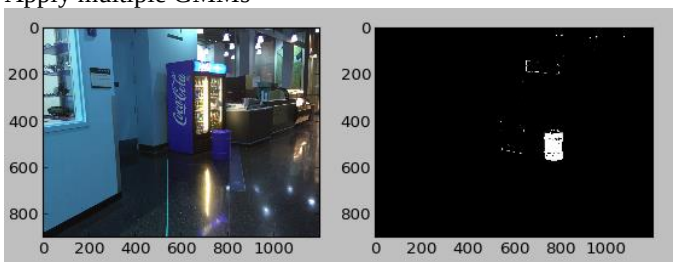
Red robotics with multiple GMMs



Red VEM with only red barrel GMM



Apply multiple GMMs



Therefore, when implementing decision estimation for each pixel, using multiple GMMs makes color segmentation better which does a lot favor in the image processing. Of course, we need to train corresponding GMMs for those objects with similar colors and you want to filter out.

D. DFS in Image Processing

Once the color regions are identified, here comes the approach to decide where the barrel is located in the images, including the physical distance and the coordinates of a bounding box for each detected barrel.

Basically, I utilize function 'regionprops' for region analysis. After dropping out regions with too small number of pixels and clearing part of noise, applying built functions to draw contours of detected region and get region information as well.

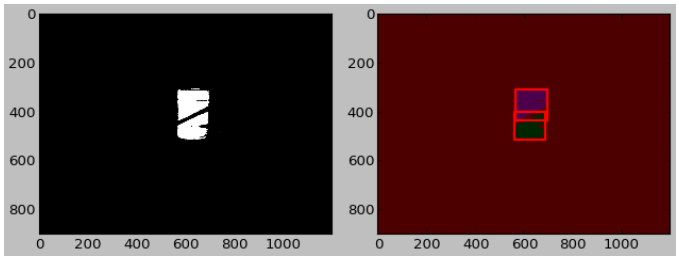
Dealing with a whole, clear and intact detected red barrel is pretty straightforward. However, in order to handle those special cases such as the red barrel is occluded by other items or objects with similar color locate close to red barrel which might cause strange contours of true red barrel and confuse the computer as well, I apply DFS algorithm in the image processing part.

The basic idea of my DFS algorithm is pretty simple and clear. I enumerate all possible combinations of detected contours, and then apply certain test algorithm to determine whether the region is red barrel or not.

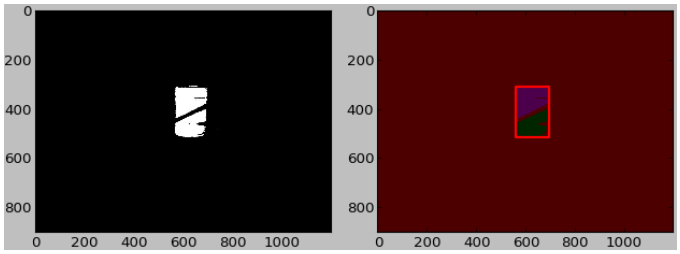
For example, say we get 4 contours before implementing DFS finding step, which means we have total $C_4^1+C_4^2+C_4^3+C_4^4=4+6+4+1=15$ possible contour combinations.

Then traverse each possible combination contours and test whether the region belongs to true red barrel. In test part, I check several properties of contour, including the ration of contour's height and width, the number of pixels filled in the empty space between object region and contour, and density of the contour.

Below are some pictures to show the function of DFS. Without DFS finding, we can get all possible contours shown below.

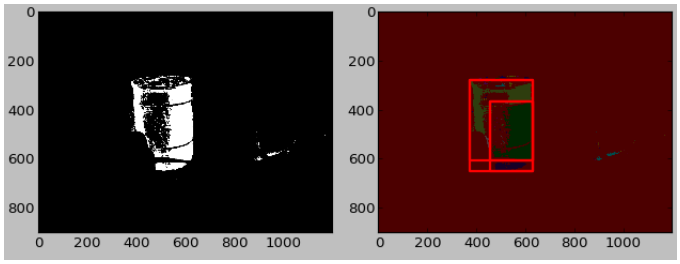


Obviously, there are total three contour candidates in the image. My DFS algorithm will test all possible combinations which is total 6 and return possible contour to represent red barrel.

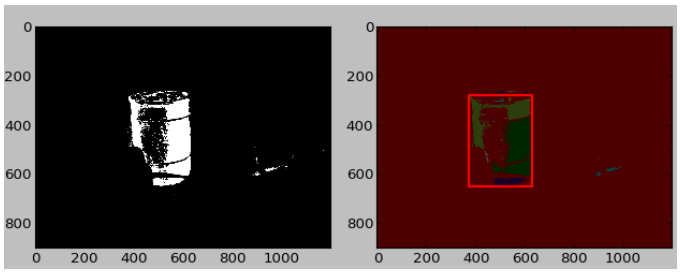


Therefore, the designed DFS algorithm is pretty good to deal with the cases that red barrels are split by other objects.

However, the approach above will bring another problem, this method cannot deal with the cases that some contours locate inside other contours. For example, after the implementation of DFS I can get the detected contour below.



You can find that the outside contour contains other two possible ones. If return directly, it will give us three detected red barrel but in fact, there exists only one barrel. In order to handle this case, I update DFS algorithm. Before returning the found result, one more step is to check all possible overlap cases and drop the overlap region with least probability to be a barrel, such as check the size ratio. Finally, it gives us the correct result.



E. *Distance Estimation*

Once find the target red barrel, we need estimate the physical distance of the target object by using computer vision knowledge. The theory formula shows below.

$$y_{im} = f_m \frac{h_{im}}{h_{ccd}} \frac{Y}{Z} \dashrightarrow Z = f_m \frac{h_{im}}{h_{ccd}} \frac{Y}{y_{im}}$$

Based on training images with distance information, we can estimate two camera parameters, focal length and height of CCD sensor.

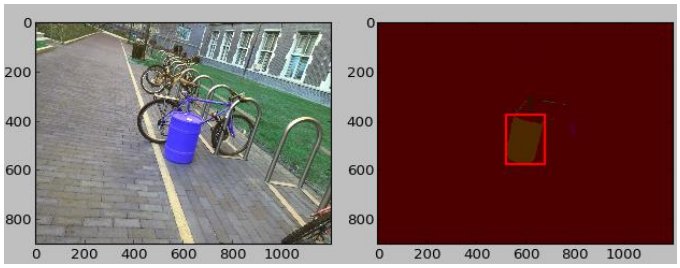
h_{im} : Represents the height of image, images in my training set is 900 pixel units.

Y : Represents the physical height of red barrel, and we have red barrel with height 57cm.

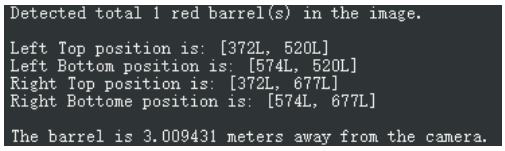
y_{im} : Represents the height of red barrel in image plane, we can get it by positions of four detected corners.

IV. Discussion on Failures

At first show one of successful detection results.



The actual physical distance is 3m, and detection parameters are below

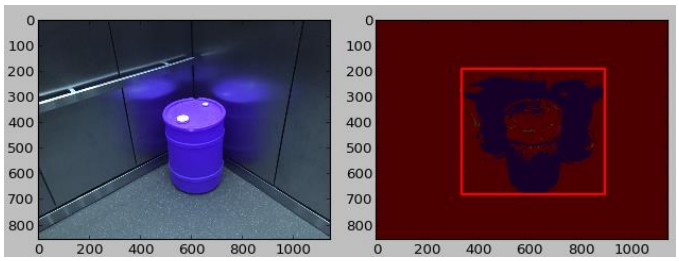


However, the algorithm meets some problems.

A. *Various GMM models*

Based on the training data, I trained total 4 GMMs, including red barrel, red chairs, red robotics and red walls. But once meet new other objects with similar red color, the program cannot distinguish it but just treat it as red barrel pixels.

The typical example is the reflection of red barrel.



Since the shortage of data of reflection pixels, the algorithm will treat all reflection region as true red barrel. Increasing threshold is not ideal since the color is similar, higher threshold will always filter out pixels in true red barrel as well. Therefore, in order to increase the accuracy of detection and estimation, more number of data and various data type for training is quite important.

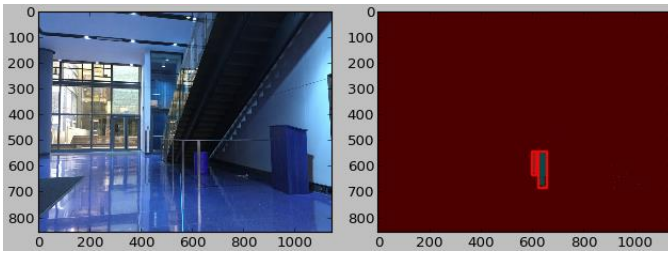
B. *Naive testing algorithm in image processing part*

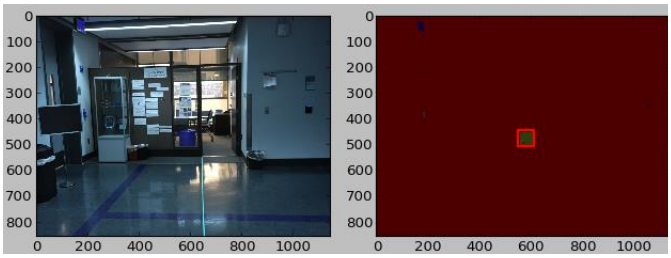
To be honest, determining whether the contour is red barrel or not might be the most difficult part in this project. We need to drop out some regions look like to red barrel which are truly not, and also cannot ignore those are truly red barrel but look strange. In my algorithm, I test the size ratio of each contour, set threshold for legal contours. And also test the number of pixels filled by contour, if the filled region is too large, I will choose to drop it.

However, the above approach is pretty naive and will cause missing red barrel in some cases.

For example, if the red barrel is rotated a lot which leads contour looks like a square rather than a rectangle. In this case, my ratio threshold may filter out this contour. Another case is, the size ratio will be changed by occluded objects.

Below figures show two detection failures.





The strange size shows above will truly mislead the algorithm and then these contours that contain true red barrel will be filtered out in my test part.

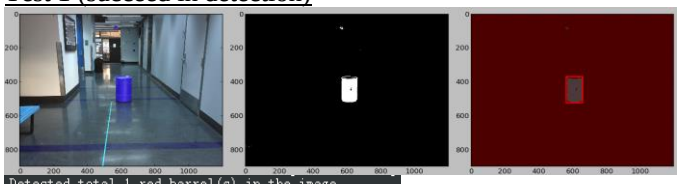
Therefore, the improvement in testing part is necessary to increase the detection accuracy. For example, instead of checking only two properties, it's pretty fine to consider more other attributes, such as the difference between centroid positions of contour and detected region, the density of pixels in the contour.

What's more, voting approach is better than the approach that once find unfit properties and then filter out directly. We can set weights of these checking properties and mark corresponding points after each property checking. Then sum up all points of the current contour. If the total score is higher than threshold, we treat it as red barrel, otherwise, just drop it. I believe via this improved approach, we can increase the accuracy of detection.

V. TEST RESULTS AND IMPROVEMENTS

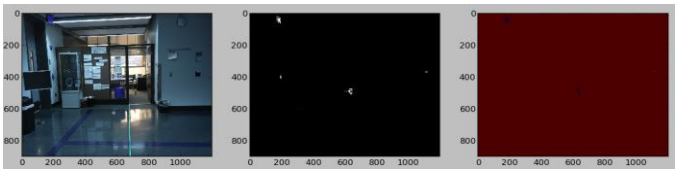
Total 10 test images, I succeed to detect 8 of them and missed 2. For location and distance estimation, 2 of them are wrong while other 6 are correct (I hope).
(The leftmost image is the original one, middle image is the color segmentation result and the detection result shows in the rightmost side)

Test 1 (succeed in detection)



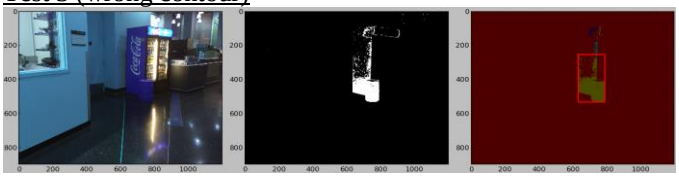
```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [369L, 563L]
Left Bottom position is:  [522L, 563L]
Right Top position is:  [369L, 661L]
Right Bottome position is:  [522L, 661L]

The barrel is 3.973235 meters away from the camera.
```

Test 2 (detection miss)

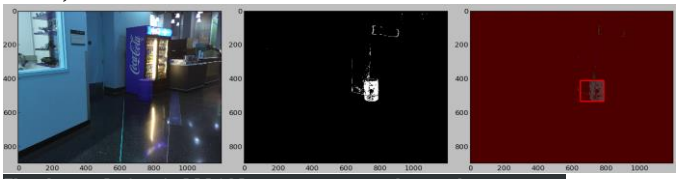


The problem happens in the color segmentation part since I set several GMMs for testing and the weight for each of them is not suitable in this case.

Test 3 (wrong contour)

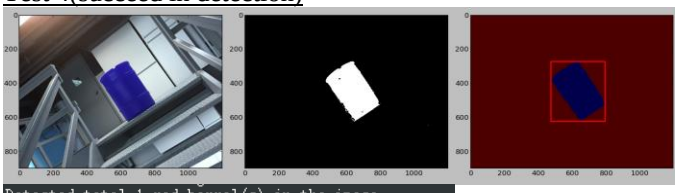


When I set a higher threshold of red barrel and weight of red VEM, GMM the result shows better.



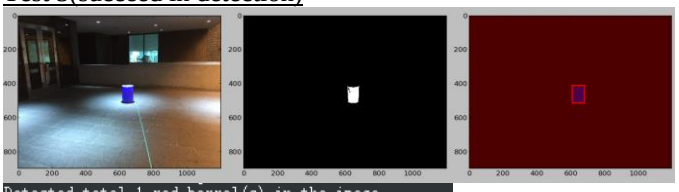
```
The barrel is 4.902460 meters away from the camera.
```

Test 4(succeed in detection)



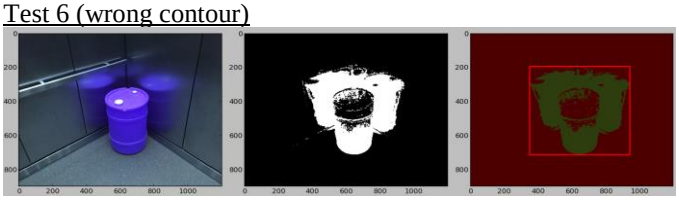
```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [271L, 475L]
Left Bottom position is:  [624L, 475L]
Right Top position is:  [271L, 796L]
Right Bottome position is:  [624L, 796L]

The barrel is 1.722110 meters away from the camera.
```

Test 5(succeed in detection)



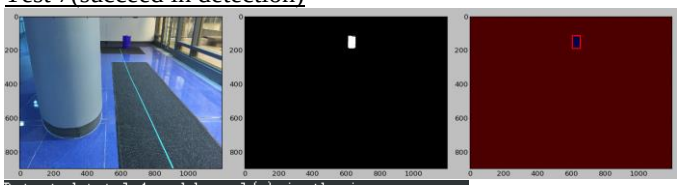
```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [409L, 609L]
Left Bottom position is:  [515L, 609L]
Right Top position is:  [409L, 681L]
Right Bottome position is:  [515L, 681L]

The barrel is 5.734953 meters away from the camera.
```

Test 6 (wrong contour)



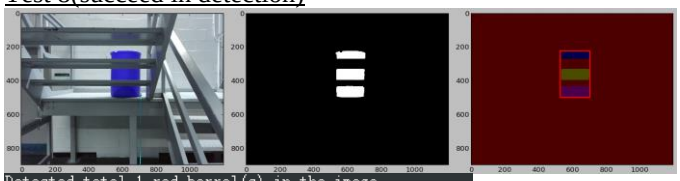
The reason is I didn't train a GMM for reflection red barrel, so the algorithm will detect all reflection pixels which leads to the extension of contour.

Test 7(succeed in detection)



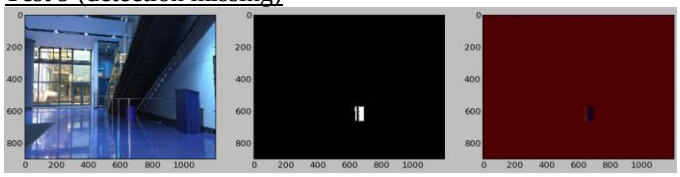
```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [112L, 608L]
Left Bottom position is:  [188L, 608L]
Right Top position is:  [112L, 655L]
Right Bottome position is:  [188L, 655L]

The barrel is 7.998750 meters away from the camera.
```

Test 8(succeed in detection)



```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [222L, 528L]
Left Bottom position is:  [499L, 528L]
Right Top position is:  [222L, 705L]
Right Bottome position is:  [499L, 705L]

The barrel is 2.194603 meters away from the camera.
```

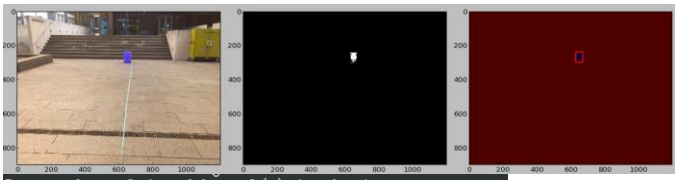
Test 9 (detection missing)



The missing is caused by my red barrel testing algorithm, I simply check the size ratio and drop some contours with strange size. However, I thought my DFS algorithm can combine two contours together but the combined one is also filtered out by my test algorithm. After improving my testing part, the performance becomes better.

```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [569L, 629L]
Left Bottom position is:  [718L, 629L]
Right Top position is:  [569L, 691L]
Right Bottome position is:  [718L, 691L]

The barrel is 4.079899 meters away from the camera.
```

Test 10(succeed in detection)



```
Detected total 1 red barrel(s) in the image.

Left Top position is:  [239L, 628L]
Left Bottom position is:  [298L, 628L]
Right Top position is:  [239L, 669L]
Right Bottome position is:  [298L, 669L]

The barrel is 10.303475 meters away from the camera.
```

VI. Brief Summary

The color segmentation looks like pretty straightforward, the main and clear pipeline is labeling red barrel pixels in the training set first, then construct probabilistic model via certain group of data. Applying the constructed model for testing image, each pixel will gave its label based on the probability value in different classes. When obtaining a binary image shows up the possible red barrel region, we implement image processing to filter out regions that are not red barrel and return true contours. Finally, combine the position information and the fundamental knowledge in Computer Vision and perception, we predict the possible physical distance of each detected red barrel.

However, words are always easier than behaviors. I truly met multiple problems during the process, such as I have no idea how to use Python to deal with image, and bad implementation to construct GM. When comes to the image processing, various cases of red barrel truly make me crazy.

Finally, after multiple times of testing and with the help of cross validation set, I figure out that the abundant and various training data, correct training algorithm and generated model, suitable threshold and weights of different GMMs, as well as good image processing strategy will bring high accuracy of red barrel detection.