Project 1 Report: Color Segmentation

ESE 650 Learning in Robotics, 2018 Spring

Dian Chen 1/25/2018

1. Introduction

Gaussian Mixture model is a classic parametric model for classification. With multiple Gaussian components, it has the ability to capture more sophisticated pattern in data than a naïve model with only a single Gaussian. In this project, I used a GM model to predict whether a pixel has high enough probability from a distribution of a red barrel, and after that we examine the whole picture to detect barrels. I trained a GM model with 3 equally weighted Gaussian components, using EM algorithm to iteratively estimate the parameters. Additionally, a simple linear regression model is trained on the given barrel distances and calculated barrel areas from upstream detection. Finally, given a new unseen image, the system can detect red barrels in it with estimation of their distances to the camera.

In section 2 all the subsections are ordered in consistence with the whole pipeline, and all detailed description of algorithms are given in these following sections.

2. Method Description

2.1. HSV Color Space

For color segmentation purpose, it is usually better to manipulate pixel values in spaces other than RGB space since lighting condition is a big variance in photos of real scenes. HSV (Hue, Saturation, Value) space is a good choice since it separates color information from intensity or lighting, thus maintaining some invariant information of a particular color under different lighting conditions.

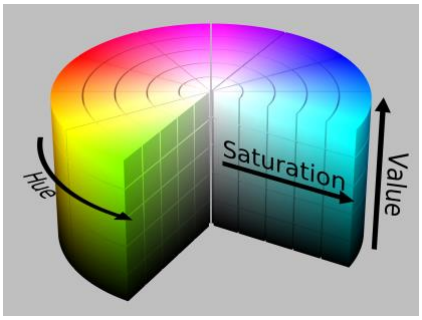


Fig. 1. HSV representation. (From Wikipedia)

In both training and testing, loaded images are first converted to HSV space from

RGB space using a simple call to matplotlib.colors.rgb_2_hsv.

2.2. Gaussian Mixture Models

In this project, I used one GM model to classify pixels as "red" or "non-red", with red pixels labeled from barrels in training images. Instead of using a fixed set of covariance matrices Σ's and only estimate a set of μ's, I chose to estimate both Σ's and μ's. I decided to use 3 equally weighted Gaussian components as the hyper-parameters, which yielded pretty good results on both training and test set.

$$K = 3, D = 3$$

The shape of training examples are selected as

$$X \sim (3, N), \qquad N = 915,263$$

2.2.1. Training

The training of my GM model consists of 5 steps, the first 4 being the standard steps: 1) Initialization; 2) E-Step, estimate the current distribution to construct the lower bound; 3) M-Step, update parameters to maximize log likelihood; and 4) Check convergence. I added a fifth step, which is run several rounds of training with different initialization, and pick the most frequent convergence as the final optimal parameters. In this way it's more likely that I could avoid being trapped in a local optima.

1) Initialization

   Proper initialization certainly helps convergence. To initialize μ's, I first compute a range of all positive pixels (red) in 3 channels, multiplied by 3 random numbers between 0 and 1, and finally add to the mean of all positive pixels. This guarantees that initial μ's have reasonable range

$$\mu = \begin{bmatrix} \max(H) - \min(H) \\ \max(S) - \min(S) \\ \max(V) - \min(V) \end{bmatrix} .* \begin{bmatrix} random_1 \\ random_2 \\ random_3 \end{bmatrix} + \begin{bmatrix} \text{mean}(H) \\ \text{mean}(S) \\ \text{mean}(V) \end{bmatrix}$$

   To initialize Σ's, I took advantage of the fact that Σ can be decomposed in the form of USU^T, in which U is a set of orthogonal basis and S is a diagonal matrix. These two matrices can be generated randomly and multiplied together.

2) E-Step

In this step, we should first compute the Gaussian Mixture probabilities of each data point under each Gaussian components:

$$g_k^i = \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}} \exp\{-\frac{1}{2}(x_i - \mu_k)^T \Sigma^{-1}(x_i - \mu_k)\}$$

With the help of numpy arrays, we can compute N probabilities of one Gaussian component in a vectorized way:

```
# compute powers of exponentials in fully vectorized form
X_minus_mu = X - mu
X_Cov = -0.5 * np.dot(X_minus_mu.T, A)

powers = np.sum(np.multiply(X_Cov.T, X_minus_mu), axis=0)
```

After that, we compute:

$$z_k^i = \frac{g_k^i}{\sum_{k=1}^K g_k^i}, \qquad z_k = \sum_{i=1}^N z_k^i$$

Again, this can be vectorized as follow (g and z are of shape (N, K)):

```
# sum up probabilities from K mixture components
g_sum = np.sum(g, axis=1, keepdims=True)

# compute reletive weights z
z = g / g_sum

zk = np.sum(z, axis=0, keepdims=True)
z = z / zk
```

Now we've completed the "estimation" in one iteration.

3) M-Step

In this step, we maximize the log likelihood by updating parameters as follow:

$$\mu_k = \frac{1}{z_k} \sum_{i=1}^N z_k^i \, x_i$$

$$\Sigma_k = \frac{1}{z_k} \sum_{i=1}^N z_k^i \, (x_i - \mu_k)(x_i - \mu_k)^T$$

Again, by clever manipulation of linear algebra and numpy arrays, we can vecotrize the computation as follow:

```
# M Step: update mu and sigma in fully vectorized form
mu = np.dot(X, z)

for k in range(num_Gaussian):
    X_minus_mu = X - mu[:, k, np.newaxis]
    X_minus_mu_weighted = X_minus_mu * (z[:, k, np.newaxis].T)

    sigma[:, :, k] = np.dot(X_minus_mu_weighted, X_minus_mu.T)
```

Now we've completed the maximization part of the iteration

4) Check convergence

   A reasonably small change of the parameters can be viewed as a signal of convergence. To measure the change, I composed the following delta value:

$$\text{delta}_k = \|\Delta\mu_k\|_{l2} + \|\Delta\Sigma_k\|_F$$

   Once all K delta's are below a predetermined threshold, the loop terminates.

5) Multiple rounds

   Theoretically the EM algorithm cannot avoid local optima. To prevent acquiring such parameters, I set up multiple EM rounds and wrote a script to manually inspect the occurrence of parameters. The average of most frequent set of parameters is chosen as the final estimation result. Statistically this cannot prevent the result from being local optima either even if it occurred more times, but in practice this set of parameters does yield good predictions so we can say we've indeed found a global optimal point.

As an implementation-wise detail, I chose to use normalized pixel values between 0 and 1 instead of 0 and 255. In this way I could get reasonable Gaussian probabilities, otherwise the probabilities are too easy to approach exponentially to zero and floating point error would occur.

2.2.2. Initial Prediction

Having found the parameters of our GM model, we can simply predict a pixel by thresholding its Gaussian Mixture probability:

$$c_i = \begin{cases} red, & if \sum_{i=1}^{N} g_k^i > thresh \\ nonred, & if\ otherwise \end{cases}$$

For an $H \times W$ image that usually has millions of pixels, it is again necessary to

vectorize the computation:

```
for k in range(K):
    # calculate in advance inverse of sigma
    A[:, :, k] = np.linalg.inv(sigma[:, :, k])
    # calculate probabilities in fully vectorized form
    g[:, k] = gauss_prob(mu[:, k, np.newaxis], A[:, :, k], X)
```

where X is a reshaped (3, $H \times W$) array of pixels.

2.3. Post Processing

2.3.1. Morphological Operations

After the initial prediction, we can obtain a binary mask of shape (H, W) indicating the red and non-red pixels in an image. However, this raw mask is inevitably noisy and disconnected. Background color that resembles that of our barrels may be included, while barrels occluded by handrails may be dissected into two patches. To overcome such evil facts, I used morphological dilation and erosion to post-process the raw binary mask.

For example, in training image "3.11.png", we can get these following post-processing results:

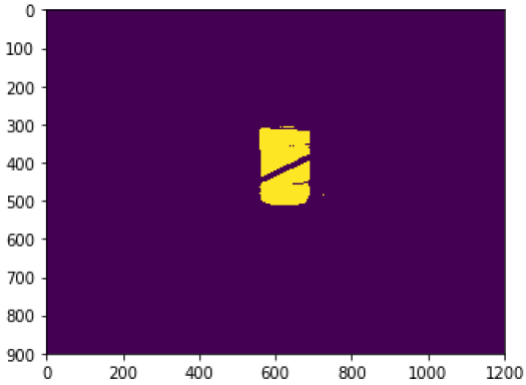


Fig. 2. Before

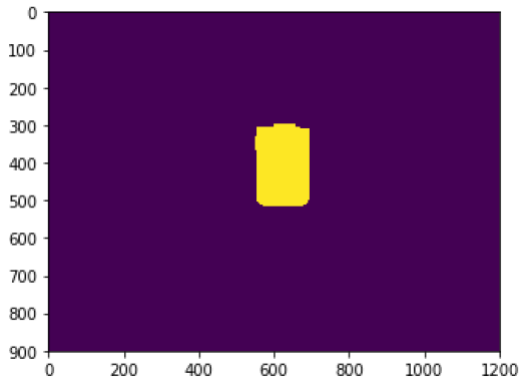


Fig. 3. After

Apparently this helped the detection of a barrel partially occluded by a handrail:

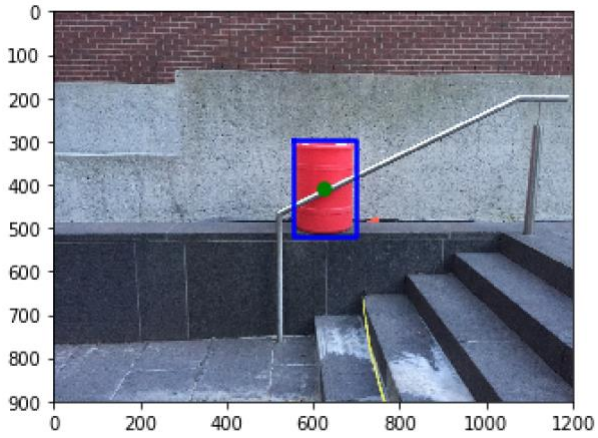


Fig. 4. Detection of "3.11.png"

As for suppressing noises, I also managed to get good effects:

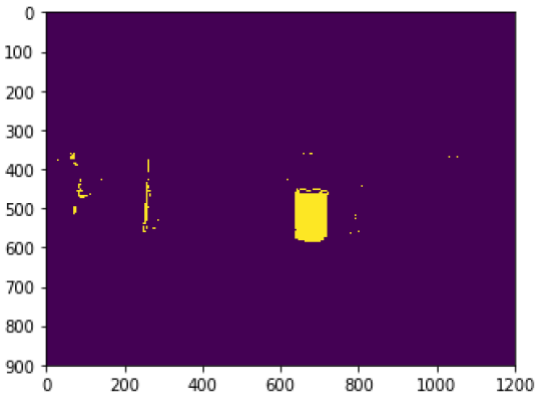


Fig. 5. Before

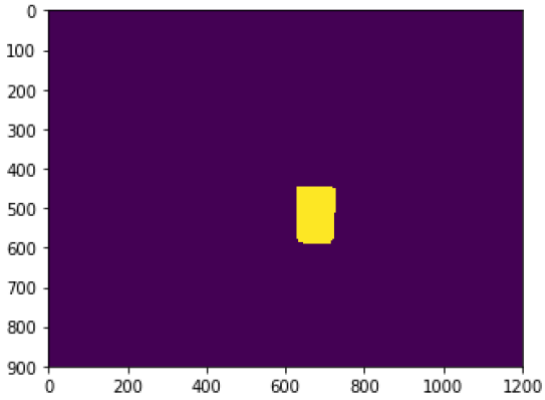


Fig. 6. After

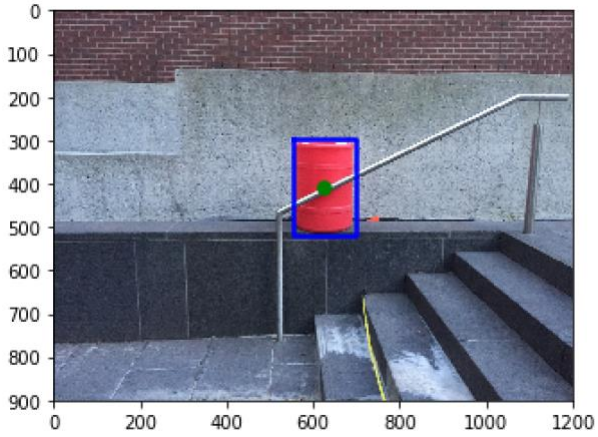


Fig. 7. Detection of "5.4.png"

2.3.2. Regionprops and Filtering

This scipy function helps to determine patches and their geometric properties in a binary mask. Inevitably, predicted masks still contain small patches of noise even after erosion and dilation and over erosion or dilation is certainly not the solution. A clever way is to use the "area" property of regions derived from regionprop, and filter out those that have too small areas to be convincing barrels but more likely noise patches. One could play with the threshold value.

```
# filter regions by areas
for one_prop in props:
    if one_prop.area > 2200:
        props_selected.append(one_prop)
```

2.4. Final Barrel Detection

After filtering by area, the remaining regions are determined to be the detected barrels! The position of the centroids and areas can be obtained conveniently from the region's properties, and bounding boxes can be drawn using four extreme coordinates derived also from the properties.

2.5. Linear Regression for Distance Estimation

Using pinhole camera model, the distance of an object to the camera is proportional to the inverse of square root of its area:

$$d \sim \frac{1}{\sqrt{A}}$$

With distances given in training images and corresponding areas derived from barrel detection, we can fit a linear model to this set of data. A constant intercept is also included to account for all the effects that make the physics in real world not perfectly homogeneous.

$$\hat{d} = k\frac{1}{\sqrt{A}} + b$$

After fitting the linear model, we can predict the distance of any barrel detected in a new unseen image.

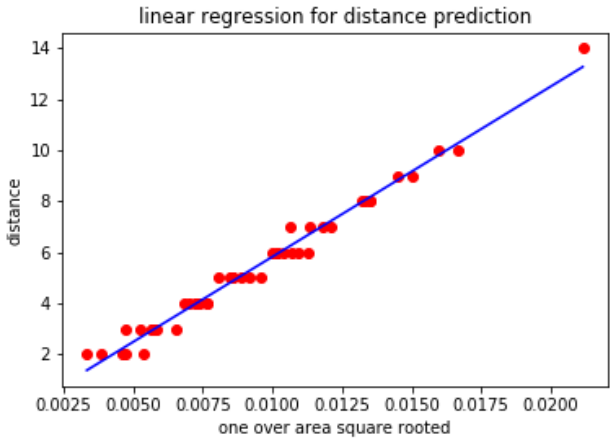
3. Results

3.1. Training Set

In all 50 training images, the trained model achieved pretty good results. Only 3 of them have picked up other objects as barrels in addition to the correctly detected barrel. The other 47 images are successfully detected, with bounding box nearly perfectly placed around the barrels. Interestingly, the "EXIT" signs and red cones in GRASP Lab are correctly omitted by my detector.

The resulted pictures for the training set are omitted from this report. Please feel free to refer to the "Training_Set_detection" folder for all the detection results.

As for distance prediction:

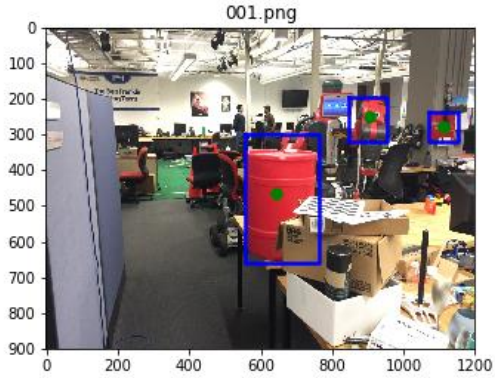


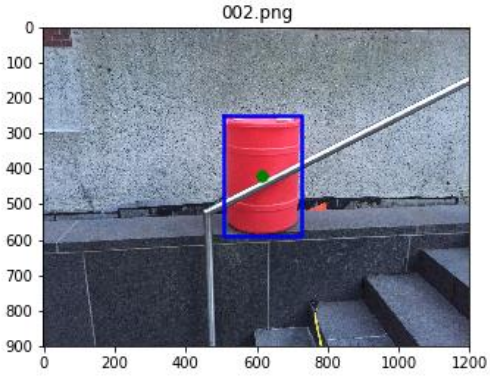
slope = 668.4214, intercept = -0.8564

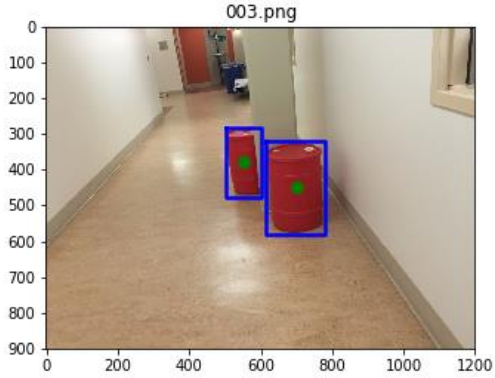
correlation coefficient = 0.9911, standard error of the estimated gradient = 13.215

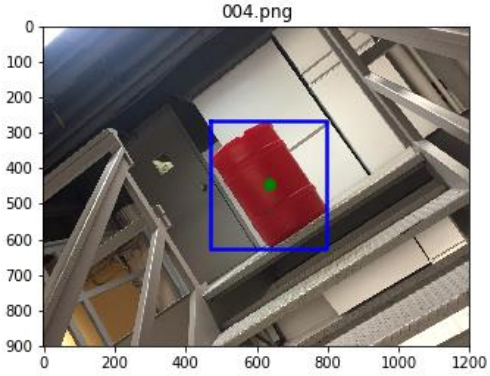
3.2. Test Set

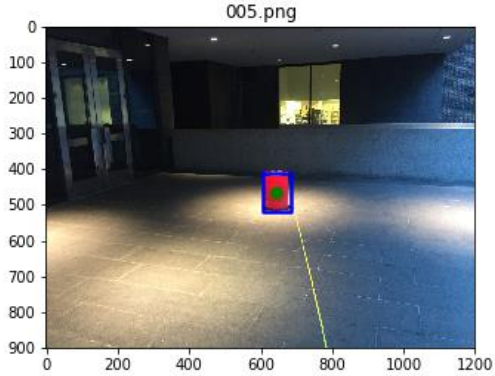
The barrel detection and distance prediction of test set are as follow. Only "001.png" has incorrect background objects being picked up in addition to the correct barrel. Barrels in the other 9 images are satisfactorily detected. Interestingly, the "EXIT" signs are correctly omitted by my detector.

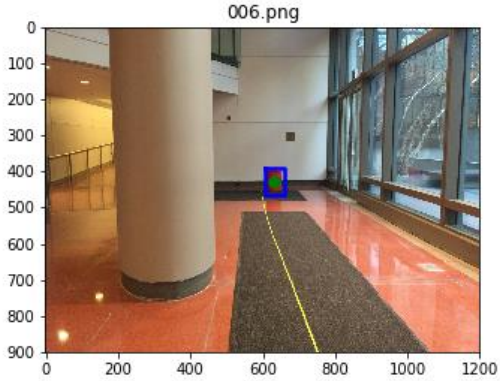

001.png
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

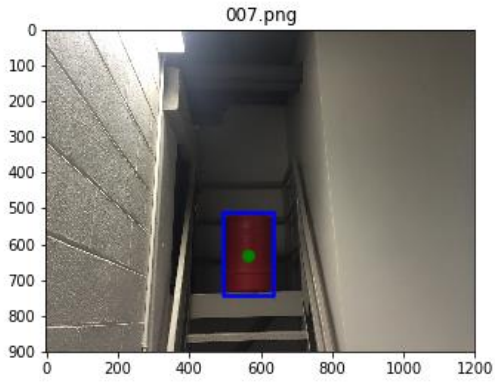

002.png
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

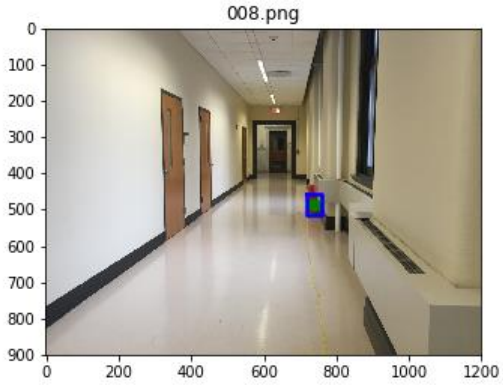

003.png
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

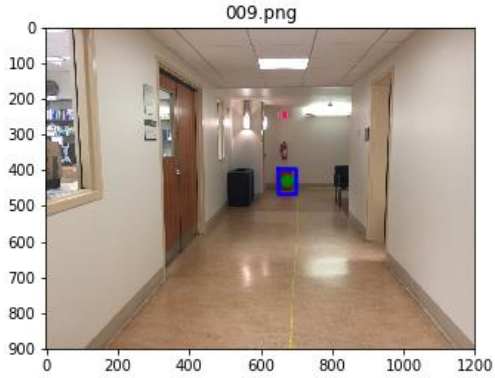

004.png
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

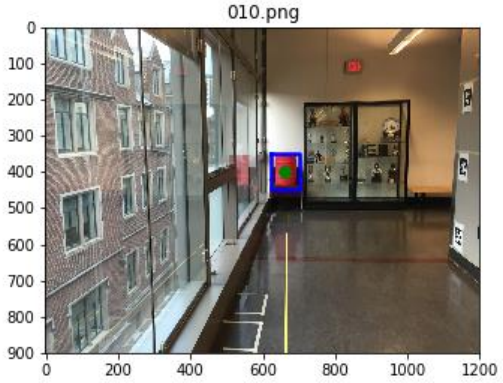

005.png
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200


006.png
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200









For Image 001

Barrel 1 (centroidX, centroidY) = (905.7213, 251.5462) distance = 6.5843

Barrel 2 (centroidX, centroidY) = (1109.1091, 276.6338) distance = 7.3794

Barrel 3 (centroidX, centroidY) = (644.9406, 463.1594) distance = 2.0368

For Image 002

Barrel 1 (centroidX, centroidY) = (616.3688, 417.7105) distance = 1.6933

For Image 003

Barrel 1 (centroidX, centroidY) = (553.0666, 376.6941) distance = 4.3895

Barrel 2 (centroidX, centroidY) = (700.2888, 446.6227) distance = 2.5735

For Image 004

Barrel 1 (centroidX, centroidY) = (633.5496, 444.2222) distance = 1.6801

For Image 005

Barrel 1 (centroidX, centroidY) = (645.1287, 464.2735) distance = 6.5103

For Image 006

Barrel 1 (centroidX, centroidY) = (631.4632, 426.8507) distance = 10.0183

For Image 007

Barrel 1 (centroidX, centroidY) = (564.8232, 628.5718) distance = 3.0082

For Image 008

Barrel 1 (centroidX, centroidY) = (738.7484, 486.8709) distance = 13.0751

For Image 009

Barrel 1 (centroidX, centroidY) = (672.8920, 427.4724) distance = 10.6086

For Image 010

Barrel 1 (centroidX, centroidY) = (661.6096, 398.5158) distance = 6.7699

4. Discussion

4.1. General Comments

The trained model achieved quite good results on both training set and test set, with only 3 glitches in all 50 training examples and 1 glitch in 10 test images. Apart from the glitches, the barrels that should be detected are still successfully detected.

In my opinion, using only pixel values as features is a relative simple way of composing data points. A triplet of pixel values alone doesn't justify a pixel's belonging to a barrel. If something having exactly the same color as the barrel in training images, it will definitely be pick up by our detector, since the object and barrels are just color patches to our detector. If we really want to distinguish barrels from other objects with similar colors, we should include other features that make barrels stand out for being a "barrel". I think this is a rare situation where incorrect results largely come from too simple features instead of insufficient training data or improperly designed models.

4.2. Noise Pixels

As discussed in 2.3.1, noise pixel can be effectively suppressed using morphological operations. Playing around with the threshold or prediction helps a little, but erosion and dilation play a major role for noise suppression.

4.3. Occlusion

As discussed in 2.3.1, occlusion by thin objects like handrails can be fixed by proper erosion and dilation. However, if a major part of the barrel is occluded then

morphological operations certainly won't help anymore. In this case the centroids derived from regionprop will be off the real centroids since those centroids are computed using effective pixels. So, in this case a bounding box would be more appropriate for computing the centroids, but to determine if major occlusion happens is another difficult task that pixel values won't be informative enough for modeling.

4.4. Distant Barrels

In my training and test set, distant barrels are successfully detected. However, I'm pretty sure that if the barrels go more far away from the camera they will be eventually missed by my detector. Again, if the color patches reduce to small enough our naïve detector won't be able to tell if those are real barrels or noise patches. This can be solved only if we have more sophisticated data features as discussed in 4.1.

4.5. Distance Prediction

The distance predictions are coarse, the main reason being that the ground truth distances are relatively coarse and the area calculation itself is also imprecise and noisy. In addition, some ground truth distances are measure by tape measure lying on the ground while the picture is taken from tilted ways which may correspond to incorrect distance. Nevertheless, the coarse linear relation is still capture by my data points.