# Segmentation and Bounding Box Approximation of Blue Barrels Using a Single Gaussian Approach

Jared Leitner ( jjleitne@eng.ucsd.edu)

ECE276A: Sensing & Estimation in Robotics

## I. Introduction

Sight is the primary sense humans use to navigate around their environment and carry out tasks in an effective way. In order for robots to accomplish meaningful tasks in the real world, they must be able to make sense of their surroundings. Computer vision is an area of study concerned with developing methods that allow computers to extract high level information from photographs, in a similar way to humans.

Humans are excellent at recognizing a large number of varying objects and localizing the positions of these objects in their line of sight. Providing this ability to computers will enhance their capacity to understand their environment, and ultimately perform useful tasks.

This project focuses on a small problem of object recognition, in which the objective is to develop a program that can detect and localize blue barrels within an image. There are two steps in this process: 1.) Classify each pixel in the image as belonging to either the "barrel" or "non-barrel" class. 2.) Given this segmented image, determine which regions are indeed barrels and exclude other objects of similar color. This is an interesting application of computer vision, and the proposed method can be generalized to detect other objects of interest.

For this project, the training images were provided by the course instructor, Nikolay Atanasov. These images consist of single or multiple blue barrels set up in different environments with varying backgrounds. The images were hand labeled in Matlab by identifying the locations of the barrels and distinguishing between "barrel" and "non-barrel" pixels. A single Gaussian probabilistic model is trained for each of these classes using Maximum Likelihood Estimation (MLE). At test time, Bayes rule is implemented using each of these trained distributions to classify unseen pixels. Bounding boxes are obtained by computing the "barrel-ness" of each detected region. This procedure is explained in detail in a later section.

The rest of the report is structured as follows: Section II formulates the problem at hand in a mathematical sense. Section III describes the approach to solving this problem. Section IV discusses the results of the model and potential further work.

## II. Problem Formulation

This section provides a mathematical formulation of the problem at hand. The goal is to learn the parameters of a Gaussian distribution for each class of pixels, which are "barrel" and "non-barrel". Each pixel is represented as a 3-dimensional vector, corresponding to RGB values of that pixel. Therefore, the Gaussian model for each class will be a 3-dimensional distribution over the RGB values.

The parameters of each Gaussian are determined using MLE. These parameters include the mean $\mu_i$, covariance $\Sigma_{\boldsymbol{i}}$, and prior probability of each class $\pi_i$. The following equations describe how these parameters are found for each class. Each $x_j^{(i)}$ is a 3-dimensional vector corresponding to the *j*-th pixel in class *i*.

$$\boldsymbol{\mu_i = \frac{1}{n}\sum_j x_j^{(i)}}$$

$$\Sigma_{\boldsymbol{i}} = \boldsymbol{\frac{1}{n}\sum_j \left(x_j^{(i)} - \mu_i\right)\left(x_j^{(i)} - \mu_i\right)^T}$$

$$\boldsymbol{\pi_i = \frac{c_i}{n}}$$

$$\boldsymbol{i \in \{barrel, \quad non - barrel\}}$$

Put into words, the mean and covariance for each distribution are equal to the mean and covariance of the training pixels for the corresponding class. The prior probability for each class is equal to the number of training pixels for that class ($c_i$) divided by the total number of training examples between both classes ($n$).

During test time, Bayes rule is implemented as follows to determine the class of each pixel.

$$\boldsymbol{i^* = argmax_{\ i}\ P_{X|Y}(x \mid i) * \pi_i, \ \ i \in \{barrel, \quad non - barrel\}}$$

$$\boldsymbol{P_{X|Y}(x \mid i) = N(x;\ \mu_i, \Sigma_i\ )}$$

## III. Technical Approach

This section describes how the previous formulation was put into practice using the provided training data. The training process was carried out in three steps: 1.) Labeling the image pixels using Matlab's *roipoly* function. 2.) Separating the pixels into different arrays according to their class label. 3.) Computing the MLE estimates for each class.

The *roipoly* function in Matlab takes in a picture as input and provides a GUI for hand selecting a certain polygonal area of interest. The function outputs an array with dimensions equal to that of the input image. This array contains 1’s within the selected polygonal region and 0’s elsewhere. This array is saved as an image and used to separate the training data in the next step. This procedure was carried out in the command line of Matlab for each training image.

Separating the pixels according to their labels was carried out in Python, and this process can be seen in the Jupyter Notebook called *blue_barrel_parameter_estimation.ipynb*. This resulted in two NumPy arrays each of size 3 by the number of training pixels for that class. I separated the data into a training and validation set where the validation set comprised of the first 5 photos and the training included the remaining 41. This split resulted in 347,713 examples of “blue barrel” pixels and 37,914,963 examples of “not blue barrel” pixels.

Once the data was properly separated, the parameters for each of the single Gaussians were calculated using the formulas presented in the previous section. This process only takes a few seconds, as NumPy’s broadcasting abilities allow calculations carried out on arrays to occur quickly. These steps are shown in the above mentioned Jupyter Notebook.

A multivariate Gaussian function was implemented in order to calculate the probabilities of different pixels given the estimated parameters. This function is called *multi_gauss*, and can be viewed in the same notebook. The function takes a single pixel as input along with the parameters of an individual Gaussian model and returns the probability of that pixel for that distribution.

When it comes time to localize blue barrels within an image, the first step is to segment that image by assigning each pixel into one of the classes. This is carried out using Bayes rule and the previously mentioned Gaussian function along with the prior probabilities. This results in a binary mask of the image, and examples of this are shown in the following section. The function that carries this out is called *segment_image* and can be found in the file *barrel_detector.py*. The second step is to determine which areas of the mask are indeed barrels and obtain the bounding box coordinates for each barrel. The Python module *cv2* provides functions for finding the contours of objects within an image and provides information about these objects such as the area, center, width and height. This information is utilized in order to determine the “barrel-ness” of each of the segmented objects. The function *get_bounding_box* is used to carry out this process. The first step is to remove any noise from the segmentation and smooth out the larger segmented regions. This is carried out with functions called *dilate* and *erode* from the *cv2* module. Then, any objects with a very small area are removed because they will not be barrels. Finally, an aspect ration check is carried out to remove any objects that are excessively tall and thin or short and wide. Bounding boxes are then determined for the remaining objects.

Examples of both the segmentation and bounding box process are shown in the following section.

## IV. Results and Discussion

As previously stated, the data was separated into a training and validation set, where the validation set comprised of the first 5 images. Results of the segmentation and bounding box for 3 of the 5 validation images are shown below.

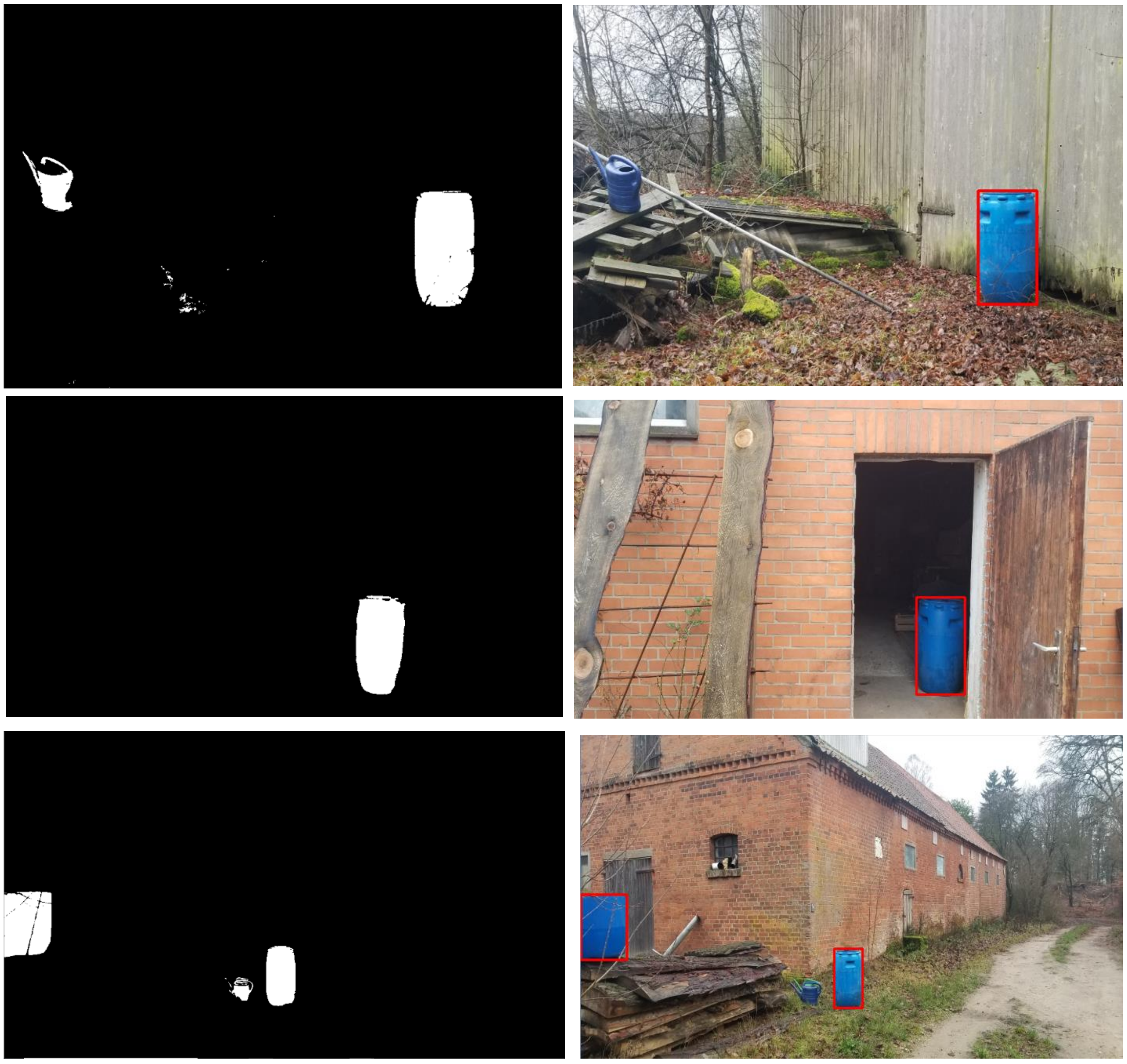

Figure 1: Accurate validation results

Evidently, the blue barrel detector was able to accurately segement the validation images and determine the bounding boxes by eliminating non-barrel objects.

The following are examples from the training data where the barrel detector did not perform as well.

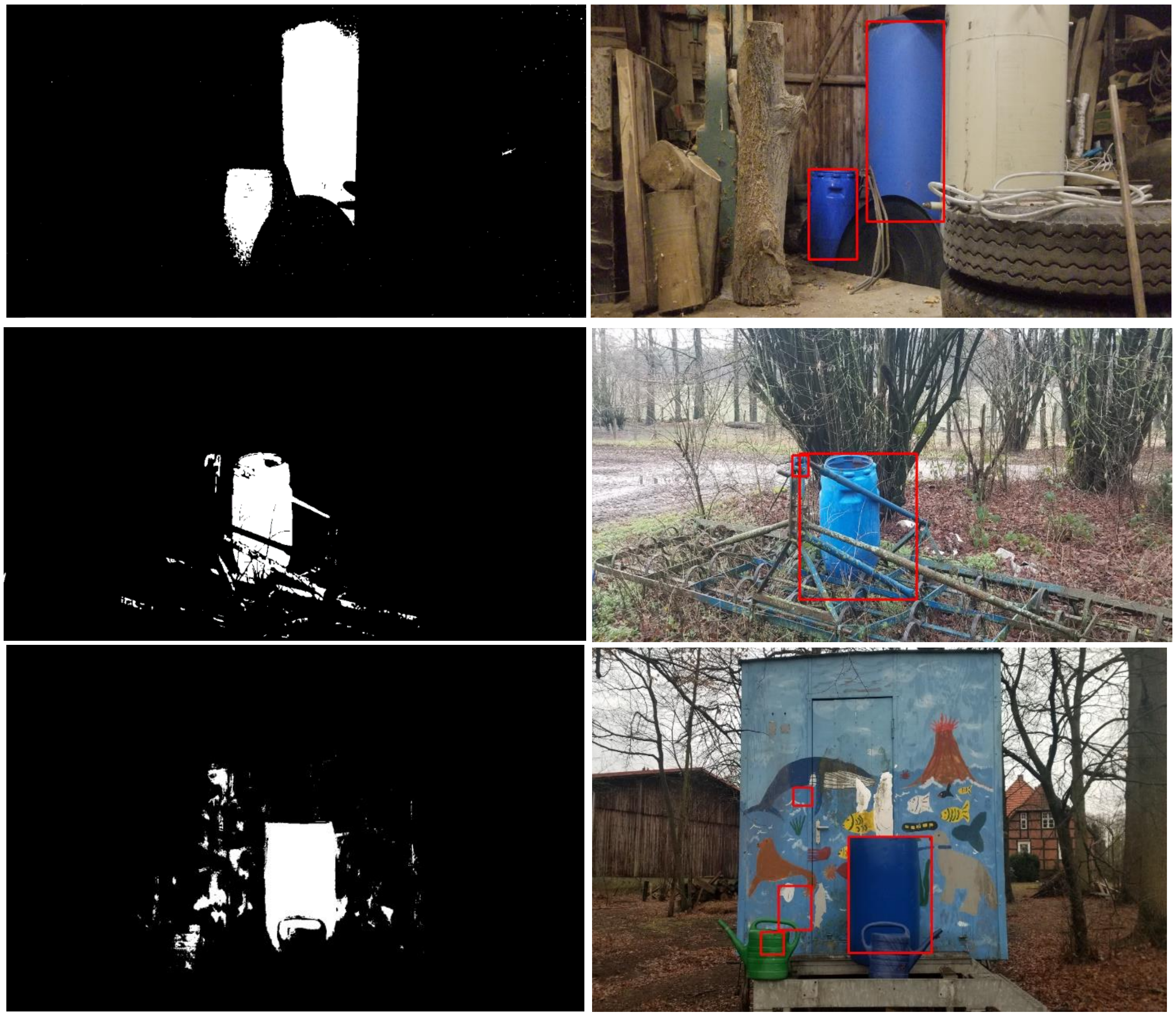

Figure 2: Inaccurate training results

The classifier fails for examples when the blue barrel is positioned in front of or behind another blue object. It will also detect objects that are not barrels but have a similar shape to a barrel, as seen in the first example. Overall, the segmentation performs well for binary classification. The overall accuracy can be improved by introducing more classes and implemented more a more involved post-segmentation processing step.