Project #1

Sensing & Estimation in Robotics

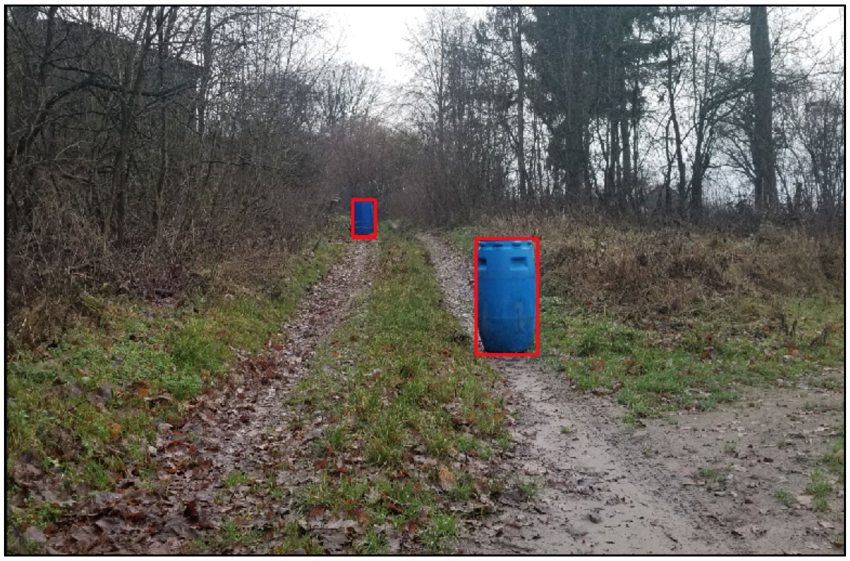
COLOR SEGMENTATION

Arshia Zafari
(A11167578)
ECE 276A
2/01/19

INTRODUCTION

The emergence of intelligent systems and modern computer vision technology has revolutionized the way machines interact with the world. Many of these systems utilize computer vision techniques to analyze visual data, draw meaning behind it, and make decisions, and much of what goes into this process involves detection and classification of certain objects within an image. In the context of this project, we will create a program to be able to detect a blue barrel within in a set of images, as in **Figure(1)**. This involves training a probabilistic model to analyze each image, segment the pixels according to color and create a mask according to the probability of the pixels belonging to the blue barrel or not. And with this mask, the barrel will hopefully be visible enough to be detected to create a box around it.

This report will dive into tackling barrel detection using a binary class, single Gaussian classifier in the HSV colorspace. It will also discuss the trials and attempts of various other methods, comparing the results using multiple color classes as well as through other color spaces.


Figure(1). Barrel detection

PROBLEM FORMULATION

The formulation of this classifier follows a Naive Bayes generative model given in **Eq.(1)**. Given each pixel in the image, assumed to be independent of one another, find the probability of the pixel belonging to a specific class, barrel or non barrel, and pick the class that maximizes that probability. These likelihoods are represented using a single standard Gaussian distribution **Eq.(2)**.

$$h(x) = argmax_y \, p(y,x) \quad \textbf{(1)}$$

$$p(y,x) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu)} \quad \textbf{(2)}$$

The parameters needed to train our model and compute these distributions include, for each class, the mean of the data **Eq.(3)** and the covariance of the data **Eq.(4)**. For the estimation strategy, the classifier will use a MAP estimate to maximize the likelihood of the parameters belonging to a given class, therefore we must also utilize the prior probabilities of those classes as our final parameter, **Eq.(5)**. The full list of parameters we need are listed below:

$$\mu_i = \frac{1}{N_i} \sum x_n \tag{3}$$

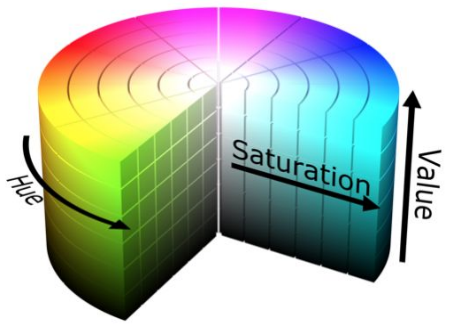
$$\Sigma_i = \frac{1}{N_i} \sum (x_n - \mu_i)(x_n - \mu_i)^T \tag{4}$$

$$\pi_i = \frac{N_i}{\sum N_i} \tag{5}$$

x_n = *individual pixel*
N_i = *total pixels for class i*
μ_i = *mean of class i*
Σ_i = *covariance of class i*

TECHNICAL APPROACH

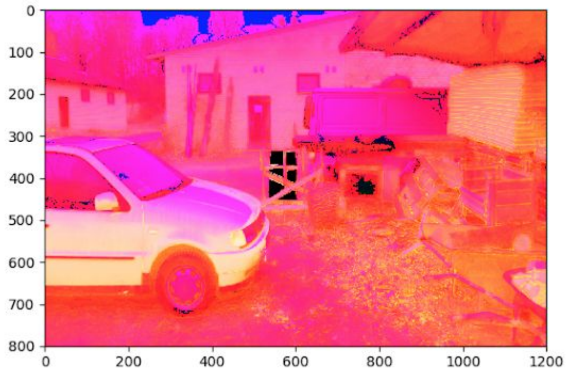
Color Segmentation:

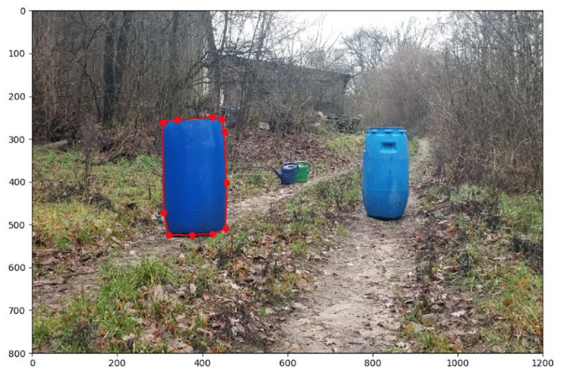


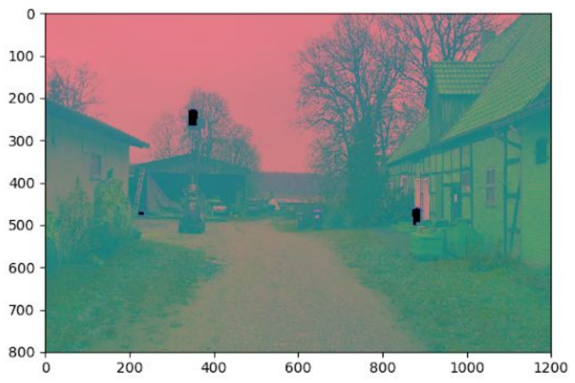
Figure(2). HSV colorspace

The colospace we will be implementing the segmentation in is HSV (Hue, Saturation, Value). The standard HSV colorspace has a cylindrical representation, shown in **Figure (2)**, where hue is the color expressed as an angle, saturation is the amount of grayness in the color, and value describes the brightness or intensity. Looking at images in the HSV space, the representation does a good job of picking up shading and blue hues in different lighting conditions. Although other color spaces like YUV would be a good option since it can strictly measure the blueness value of pixels, the value indices for HSV seemed most discriminant and visually the brightest so it was a proper choice for this application. Still, YUV was implemented as a comparison (see *Experimentation*).

To begin training we must take a subset of the training set of images as a validation set to test the results of training, the validation set here is the last six images, *41 - 46*. To gather the data needed for training the model, we use a script, *training_class.py*, which goes through each image of the training set (images *1 - 40*) and use "roipoly" to draw regions and collect the barrel pixels in the HSV colorspace, concatenate them and store them in a numpy array. In the script, the user is prompted to input whether multiple regions are needed to be drawn for images with multiple barrels. Once all barrel pixels are collected for a given image, all other pixels are assigned to the non-barrel class. The figure below show the training process in three images:







Figure(3). Creating the mask

It is important to be thorough when labeling image regions, as illustrated in the three images of **Figure(3)**; this includes a well defined region, not just a rectangle (top right) and to be meticulous where barrels have other objects blocking them (top left). We want to create masks around the objects to only get the blue barrel pixels, however a tiny bit of variance can still be beneficial, making the data more robust to noise. The labeling process is probably one of the most crucial parts to color segmentation because it determines much of how we wish to represent the pixels using the generative model.

After collecting the pixels, we end up with roughly 459,778 total barrel pixels and 37,940,220 total non-barrel pixels. Now just looking at this, the priors for these class distributions will be heavily biased toward the non-barrel class (0.11% compared to 98.9%), but let's move forward and see the effects. With the data collected into two large numpy arrays, we compute the mean and covariance across all the pixels for each channel, this is done in the script *Gaussian_param.py*. The final values for the parameters are shown in **Table(1)** and **Table(2)**:

Barrel	H	S	V
Mean	214.29094932	0.76332089	0.51774014
Cov	3.4283947e+02	2.5570089e-01	-4.0205122e-02
	2.5570089e-01	3.7853697e-02	1.7175753e-02
	-4.0205122e-02	1.7175753e-02	4.1811788e-02

Table(1). Parameters for barrel class

Non-barrel	H	S	V
Mean	84.74903621	0.26474935	0.43755096
Cov	7.2624369e+03	-5.2536650e+00	1.0120593e+00
	-5.2536650e+00	3.5106048e-02	-1.7558389e-02
	1.0120593e+00	-1.7558389e-02	5.83516089e-02

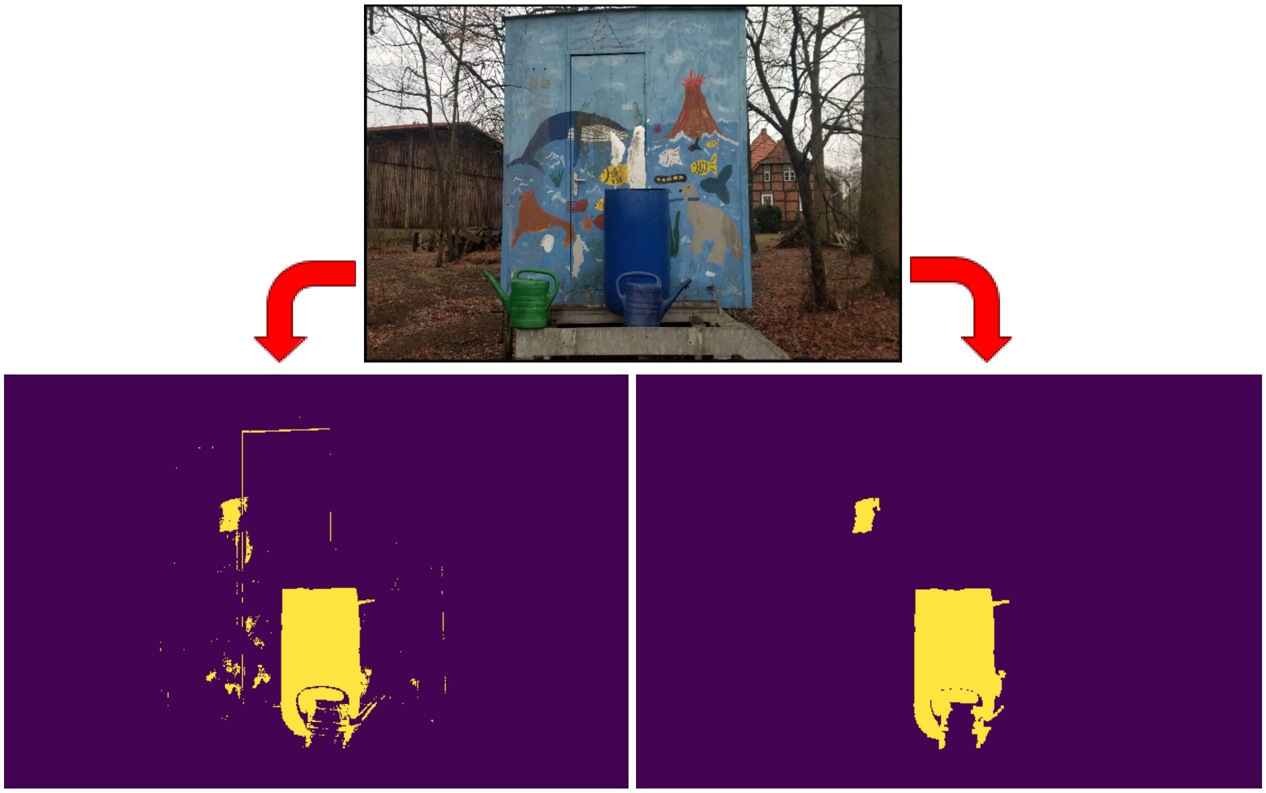
Table(2). Parameters for non-barrel class

Once the parameters have been computed, they are loaded into the main barrel detector script, *barrel_detector.py*. Two different likelihood functions are used, one for barrel and one for non-barrel, each defined by the computed means and covariances of the class distributions, and these models are the basis for how each pixel will be assigned.

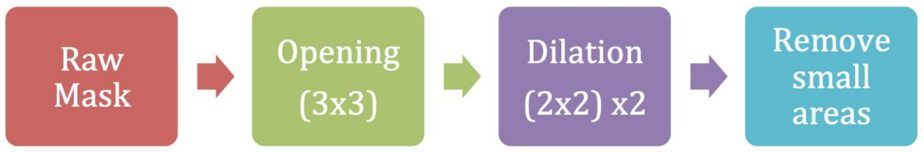
To create the actual segmentation mask, we loop through each pixel of the image and compute the likelihood using the MAP estimate that the specific pixel belongs to the class distribution of barrel or not barrel that we generated. If the likelihood of the pixel belonging to the distribution defined by the barrel is larger, then we assign a 1 to the mask corresponding to the pixel index of the image, otherwise the mask value is 0 (in uint8 [0 1] corresponds to [0 255]). The image is vectorized and values for the Gaussian model are precomputed to speed up performance time. This is all done in the function *segment_image*.

Post-Processing:

Once the segmentation process has created a mask, it is bound to have noise and misclassified pixels. Take one of the training images, image *28.png* for example, **Figure(4)** shows the mask for the image before and after post-processing. Of course being that it is an image that we trained on the mask does quite well, however there is still some pixels that were incorrectly identified as the blue barrel class. For this we must apply several post-processing techniques. After playing around with various OpenCV morphological transformations, the optimal set of post processing techniques follows the block diagram in **Figure(5)**.



Figure(4). Image *28.png* raw mask (left),
mask after post-processing (right)



Figure(5). Order of post-processing

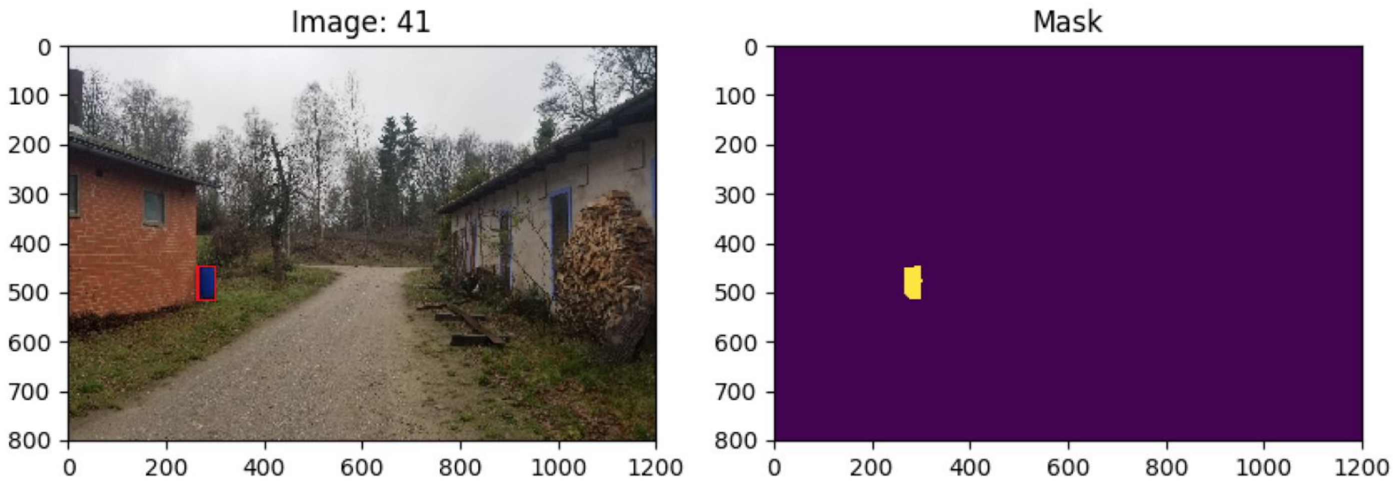
We first take the raw mask and apply "opening" which simply uses a combination of erosion and dilation to remove tiny pixels of noise, using a 3x3 kernel window. Next, we take another kernel of size 2x2 and apply two iterations of more dilation to help close and fill in missing pixels of the barrel where there should be, like cases where thin branches are in front of the barrel. Finally, we use *regionprops* to identify and remove any region of pixels that are smaller than an area of 1000, this helps eliminate any final "blue-like" regions that could be a false positive, like the watering cans.

Barrel Detection:

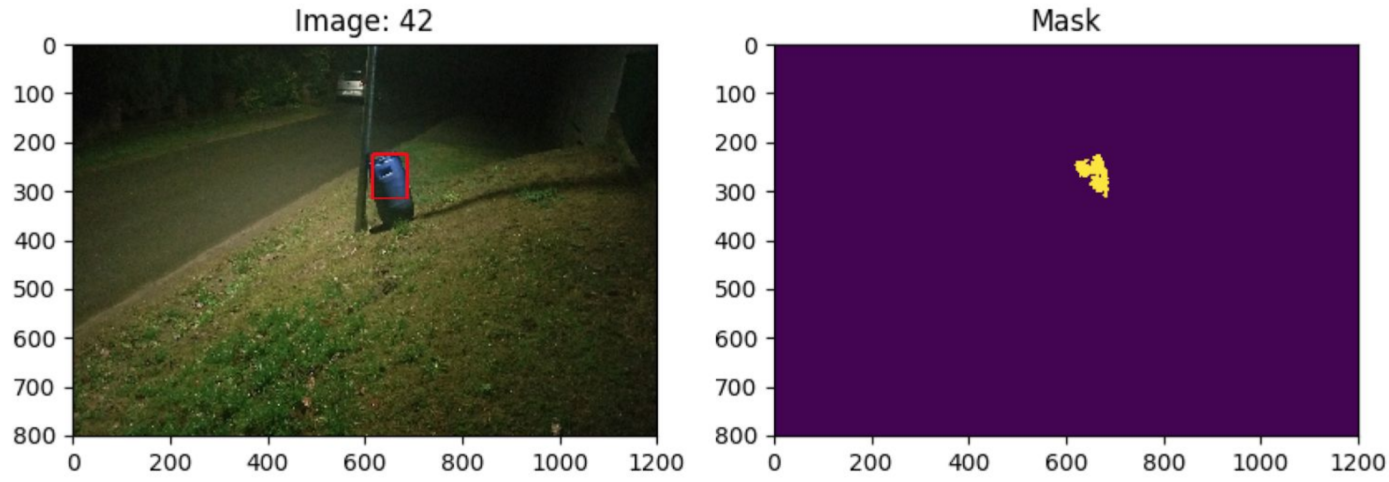
Once the image has been successfully processed through color segmentation, detecting the barrel is quite simple thanks to useful image analysis modules. In the *get_bounding_box* function, we call in an image, apply the segmentation process discussed previously and receive a mask. A successful mask will be able to show the contour of barrel clearly, so in order to actually detect the barrel we must box that region. Using the module *skimage* and its built in functions, *label* and *regionprops*, the functions label all connected regions in the mask and identifies the region proposal. Evaluating the area of the region proposal once more with a similar requirements as before, are of at least 1000 and no greater than 10000, guarantees the box be formed on only the barrel in case there still remains any false positives after segmentation, also in case *regionprops* identifies multiple regions as one large region. Finally with the proposed region, we can use *region.bbox* to identify the coordinates of a rectangle around the proposed region in the form, (x1, y1) = top left, (x2, y2) = bottom right, and append all bounding box coordinates from left to right in a list (the coordinates are presented as (y1, x1, y2, x2) for the autograder).

RESULTS

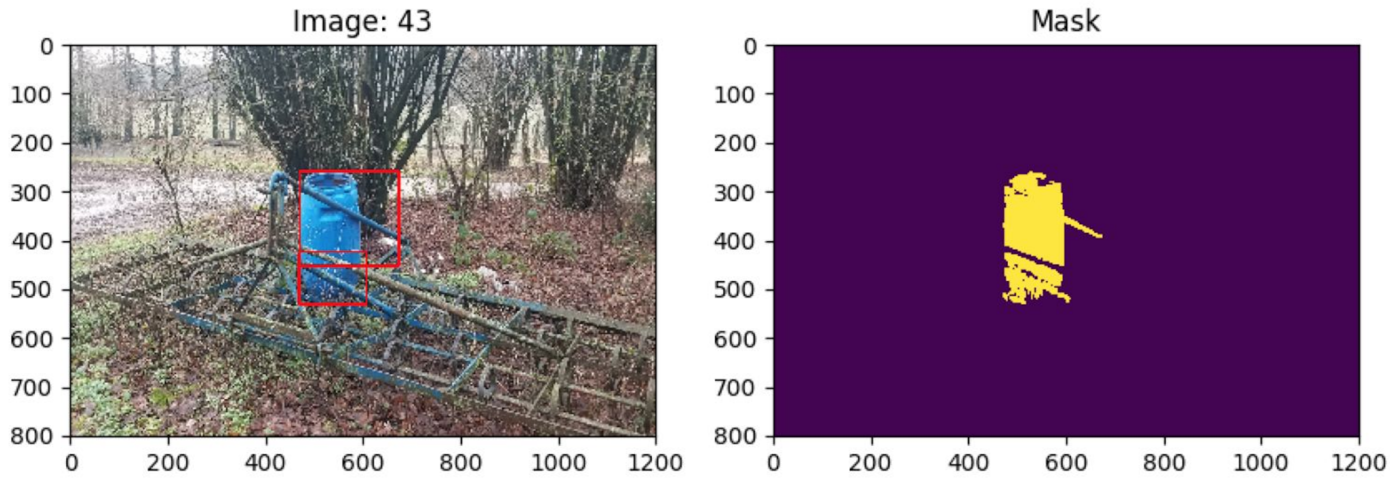
The following images in **Figure(6)** show the results of the trained classifier on the validation test set, images *41-46*. Keep in mind the classifier did not train on these images, so the data is completely new and as stated in the *Problem Formulation*, we are fitting these new pixels to a generative model that we created using the training set:



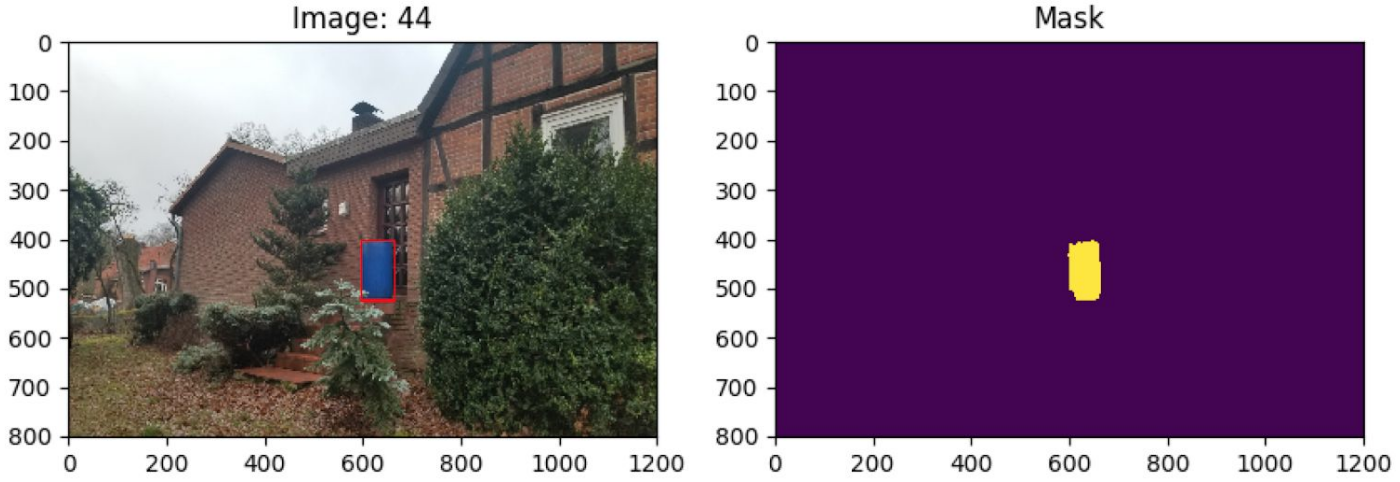
Bounding box coordinates: [[263, 447, 301, 515]]



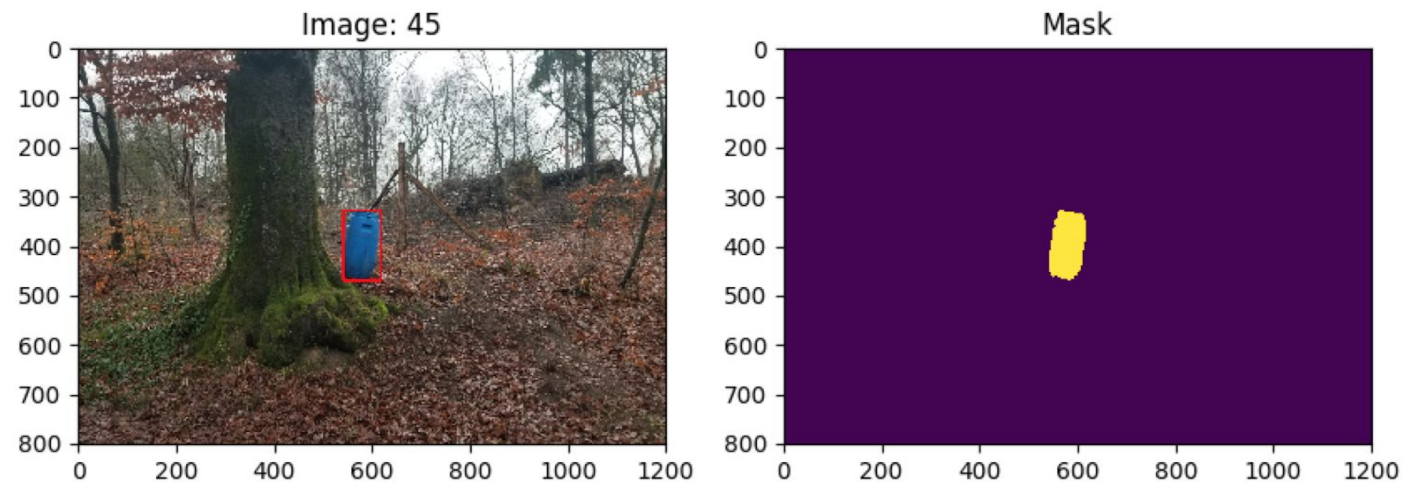
Bounding box coordinates: [[615, 224, 686, 314]]



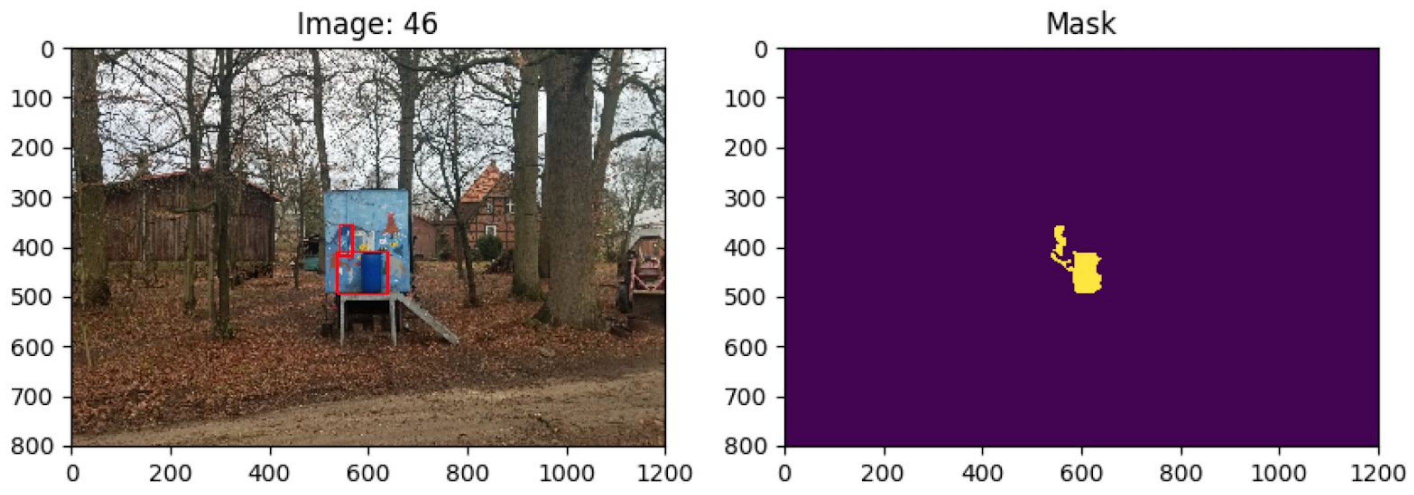
Bounding box coordinates: [[470, 258, 672, 452], [469, 424, 607, 531]]



Bounding box coordinates: [[596, 403, 664, 524]]



Bounding box coordinates: [[540, 328, 616, 469]]



Bounding box coordinates: [[542, 356, 568, 418], [537, 409, 639, 494]]

Figure(6). Classifier results on validation set

These final results show that the color segmentation and bounding box process was not perfect but in all, quite successful. For all six test images, the barrel was identified and boxed in the image.

Images *41*, *44*, and *45* shows the most success, with a mask and a bounding box visibly accurate to the barrel's location in the image. This is because of proper lighting and there not being any other barrel-blue pixels in the image.

Image *42* shows only part of the barrel identified and boxed. This is because of the shadowing on the barrel, ie. the value index of the pixels, was too dark to be identified as a barrel pixel, however the pixels that were more visible in the light were picked up nicely.

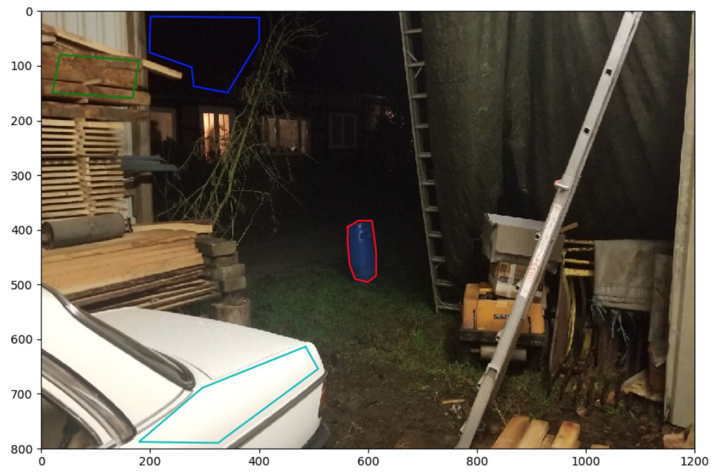
Image *43*, would be considered the least successful of the six. The mask picked up the blue from the beams of the scaffolding. This doesn't mean our classifier was poor, rather if we had created a barrel vs. non barrel blue class it might have distinguished between what is and what isn't the barrel pixels. Also if we had implemented more extensive post-processing techniques such as "closing," we could close the gap in between the two regions separated by the metal rod. This is why our bounding box function essentially detected two regions instead of one.

Image *46* does fairly well but isn't perfect either. It was able to mostly detect the barrel in contrast to the blue mural behind it, which is quite impressive, but the bounding box includes some pixels behind the barrel, making the box a lot larger than it should be. Our post processing techniques seem to work well but we could have made the bounding box function a little more tighter to fit the box along the rectangular part of the region.

EXPERIMENTATION

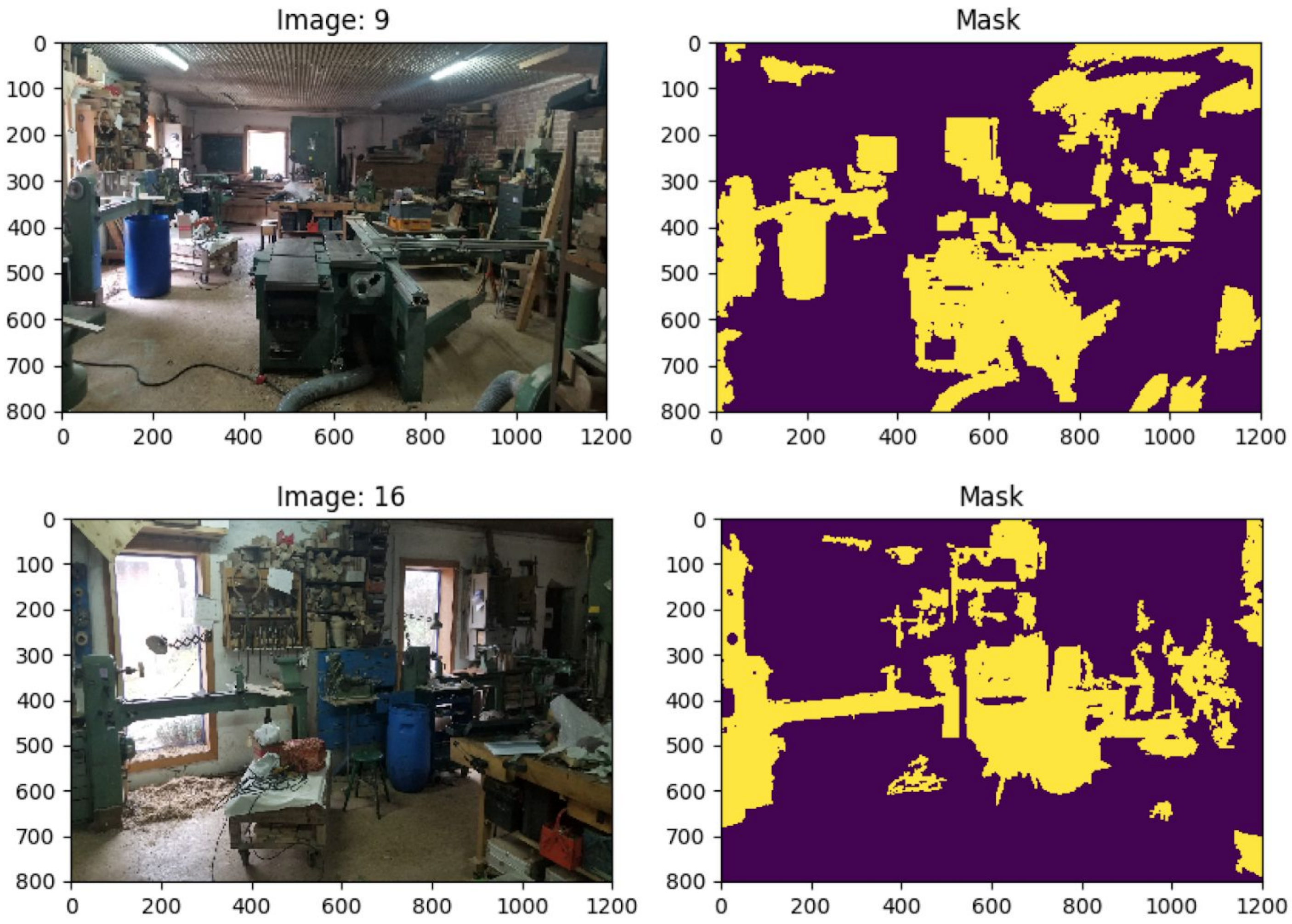
As decent as the results may show, it is important to note some of the trials and attempts of other methods and understand why they didnt work to further verify why our results are the way they are.

In practice, it is not ideal to segment an image with thousands of color values to only two color classes. It is too naive to generalize a large variety of data, such as non-barrel pixels, to one class. In theory, a multi-class classifier should perform better since we are generating more expressive models for our data. For our purposes, instead of two classes lets try implementing four color classes: blue pixels for the barrel, light/white pixels for the sky and other light, dark/black pixels for shadows, and brown/reddish/green for the ground and other brownish objects. For example, in **Figure (7)**, let's draw four regions and collect pixels in the same way as before:



Figure(7). Barrel pixels (red region), light pixels (cyan region), dark pixels (blue region), brown pixels (green region)

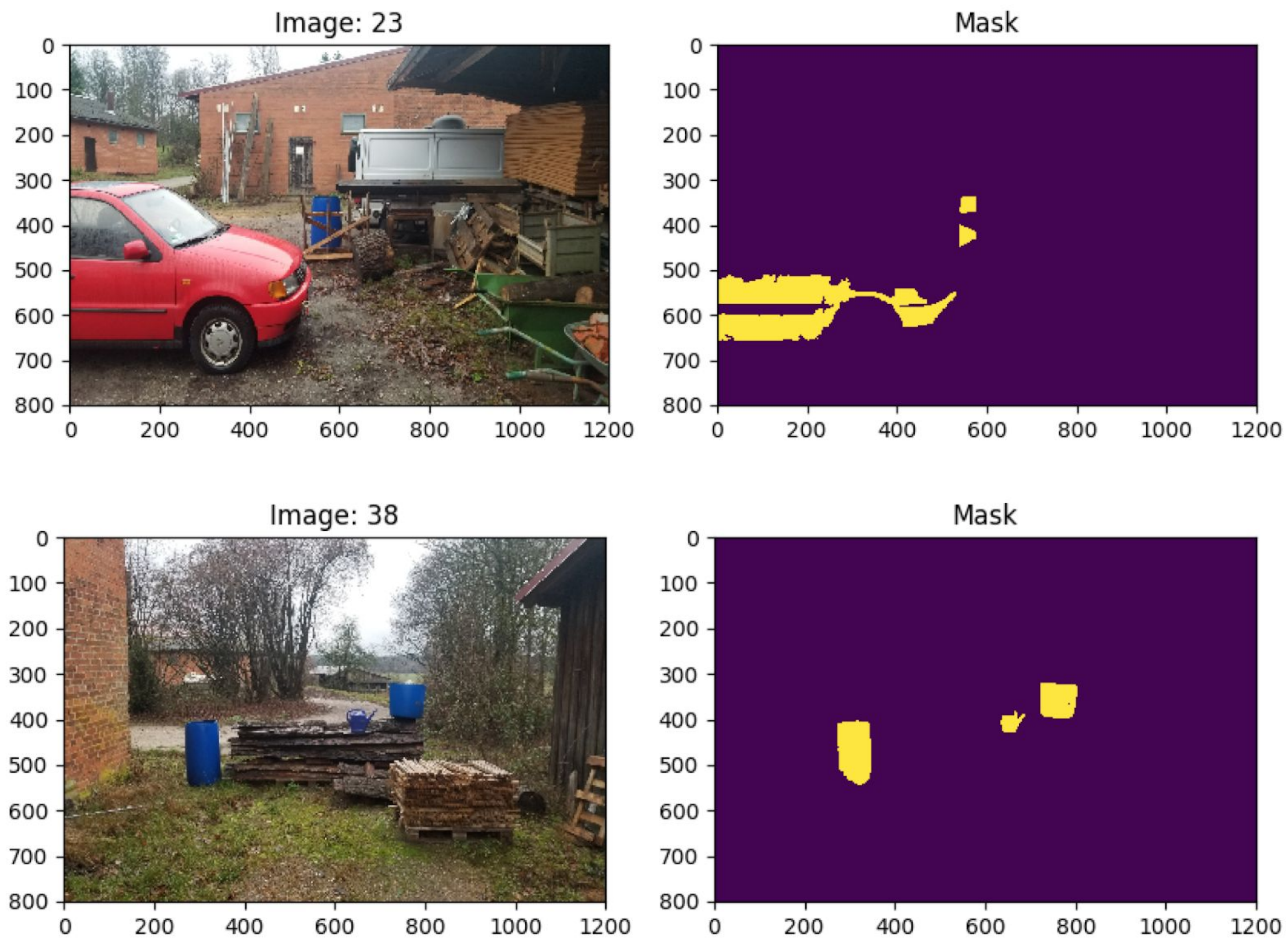
Labeling these pixels using the same training set, we end up getting around 300,000 barrel pixels, 900,000 light pixels, 700,000 dark pixels, and 800,000 brown pixels. These datasets are a lot less biased than the binary case where we had around 460,000 barrel pixels compared to 37million non-barrel pixels. We would think that a more evenly representative set of data would generate models that are more specific to that color. Lets see the results of the multiclass classifier on a few training images shown in **Figure (8)**:



Figure(8). Image *9.png* (top) and Image *16.png* (bottom) classified using four color classes

The results actually perform a lot worse than with two classes; a lot of dark pixels were classified as barrel pixels. Why is this the case? Although our models may be more specific, the more equal representation of the classes is not an accurate depiction of the image data and training data as a whole. Also choosing the right color classes is very important as there might be too many similarities across the HSV channels for blue and black, specifically the value channel. Maybe a more discriminative choice of classes would give us better results, which is why how we choose to label and initialize the data for our classifier is so critical.

Another attempt for better results can be analyzing the images through a different colorspace, lets try YUV. Lets label images in the exact same was as the HSV, now back to binary classes, except reading in images through a YUV colorspace. For a little background, YUV entails the luminance of the pixel (Y), and the chroma or color itself (U,V). The results of the same classifier using YUV trained parameters is shown in **Figure(9)**:



Figure(9). Image *23.png* (top) and Image *38.png* (bottom) classified using the YUV colorspace

The results seem to work pretty well for image *38* but for image *23* it seems to identify some red pixels on the car as the blue barrel. For that specific training image, the luminance of the red pixels might be too similar to that the blue pixel class, which leads to the misclassification. Although the chromas for red and blue are the most different, the variance across the luminance channel is not large enough to pick up the discrepancy, at least for that image. It shows that even on an image where the classifier is trained on, it can still misidentify data.

<u>CONCLUSION:</u>

This project has shown that using color segmentation, we can successfully train a classifier to identify an object like a blue barrel within a set of images that has not been seen before. We took a set of 40 images, classified the pixels, and built a generative model that represents the density of each class, barrel and non-barrel, as a single Gaussian distribution. Most of the technical approach to detecting the barrel went into the actual color segmentation because that is where the pixels are assigned a distribution. If we create a well defined mask from segmentation, detecting the object becomes quite easy with python modules that have been already created. The results show that using a binary-class, single Gaussian Naive Bayes classifier, we can successfully identify a barrel from a set of images. More accurate generative methods can be used, such as a Gaussian Mixture model, but for our purposes the single Gaussian did the job. What is amazing here is that this is just a glimpse of what current computer vision technology entails. Modern technology has expedited this process to be able to do color segmentation and object detection in real time, and identify a variety of objects in real time videos, so tens of images per second. Hopefully with more advancements in computer vision, we can apply these concepts to further the progression of machine learning to have artificial systems understand and see the world the way a human does.

***REMARKS:**
The results on autograder show a clear discrepancy with what has been presented, as there is failure across all bounding box tests. Up until just a day before this assignment was due, I had no idea what the issue was because clearly my functions were creating proper bounding boxes. After exploring in depth with a TA, the problem was with how the images were being read initially during the labeling process, using plt.imread rather than cv2.imread, so RGB rather than BGR. So there was no time to fix the issue, especially since we were given no information on how the autograder reads in images from the beginning. What is odd is that, per documentation straight from the OpenCV website below, HSV is calculated using min and max arguments across the RGB channels so it shouldn't matter if they are read in as RGB or BGR. However it still may be the case that the final HSV channel representation of a pixel is flipped between the two methods and a simple swap of the elements would not work because of the non-linearity of the conversion. In essence, reading the images using plt.imread gives better results than what is shown on Gradescope. The purpose of this assignment was to understand the application of data classification on an image to be able to segment it and extract meaningful information, and I believe that is exactly what this report shows.

$$V \leftarrow max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V - min(R,G,B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - min(R,G,B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - min(R,G,B)) & \text{if } V = B \end{cases}$$

https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor