ENPM 673
Perception for Autonomous Robots

Project 3 Report

Color Segmentation using Gaussian Mixture Models

Akshay Bapat (UID: 116215336)
Patan Sanaulla Khan (UID: 116950985)
Kulbir Singh Ahluwalia (UID: 116836050)

4 April 2020

Dataset generation

Trimming the frames

In order to generate the dataset, we used the library "roipoly" in python.

A folder of all the frames of the video was made which had 200 frames as the video had a frame rate of 5 frames per second and had a duration of 40 seconds. Frames for the training data were chosen manually. The first 42 frames have all three buoys; hence, we included every third frame in the training set. After frame 42, we include every tenth frame in the training set.

We used roipoly to manually trim out the yellow, orange and green buoys respectively. After that, we got images of size 480 by 640 pixels which only had the chosen area of interest while all other pixels were black.[2]

From the previous step we have the images stored as per the ROI in each frame of the video. Now from the ROI we need to read the intensity of each frame and store the intensity of each pixel in a .CSV file. We can get the intensity information from the .read() method and store it in the .CSV file. Now given a pixel point we are expected to generate a probability of the intensity such that it belongs to a particular class.

In order to generate the dataset, we took only the non-black pixel values from every image and appended them to a csv file. Thus, three csv files named "*yellow_buoy_dataset*", "*orange_buoy_dataset*" and "*green_buoy_dataset*" were generated with only the (R,G,B) values of the pixels corresponding to yellow, orange and green respectively.[1]

The histograms we generate from the training data are shown in images 2, 1 and 3.

Gaussian Mixture Models

Each pixel in an image has a value denoted by x = [R G B]. We have three colour classes yellow, orange and green. The goal is to calculate a probability that a given pixel belongs to a colour class "C_l" denoted by P(C_l|x).
Using Bayes rule we have,

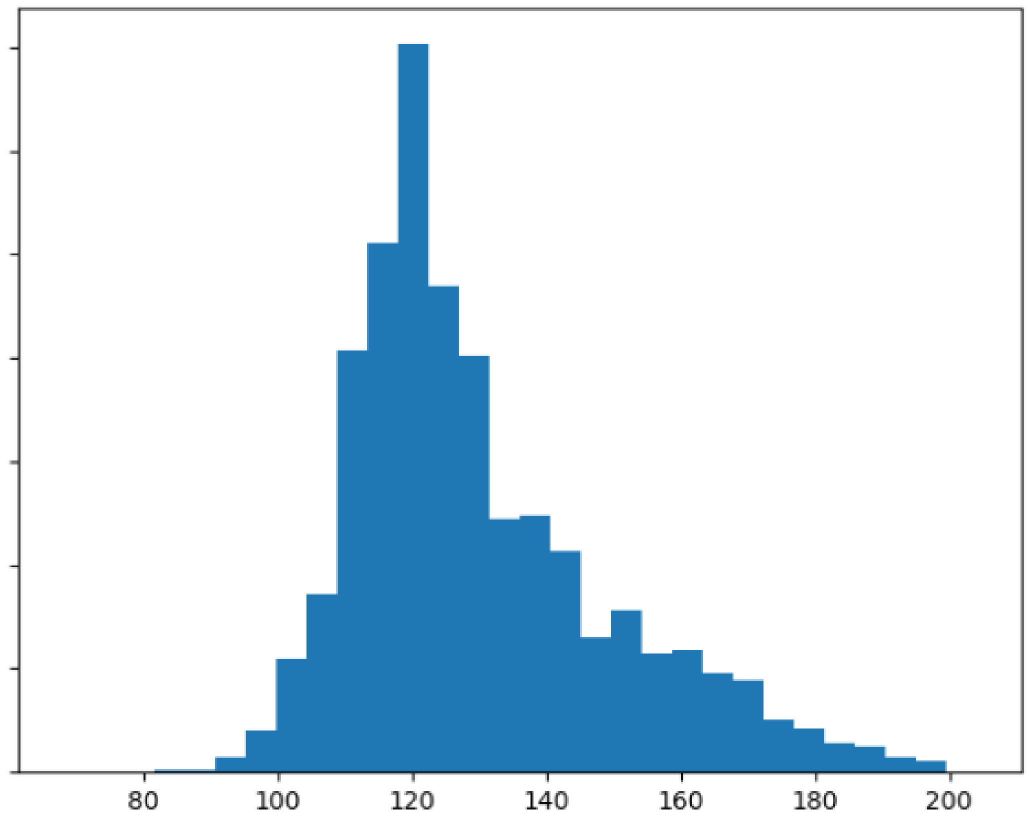

Figure 1: Histogram - Blue Channel

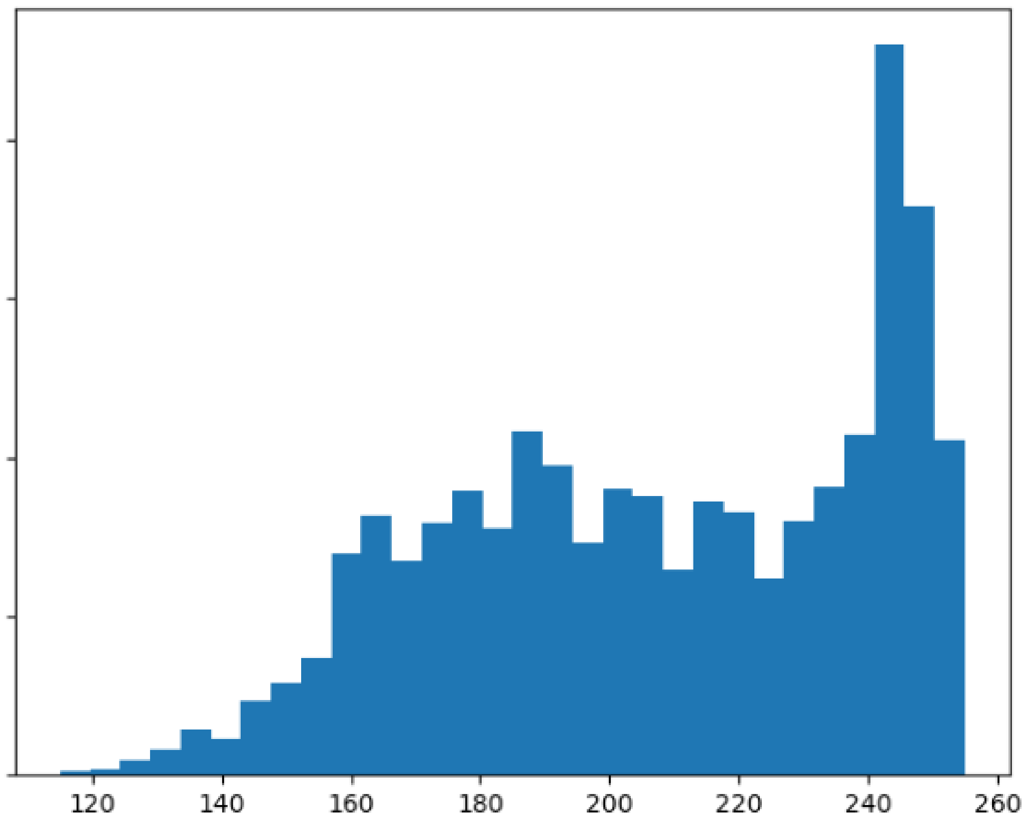

Figure 2: Histogram - Green Channel

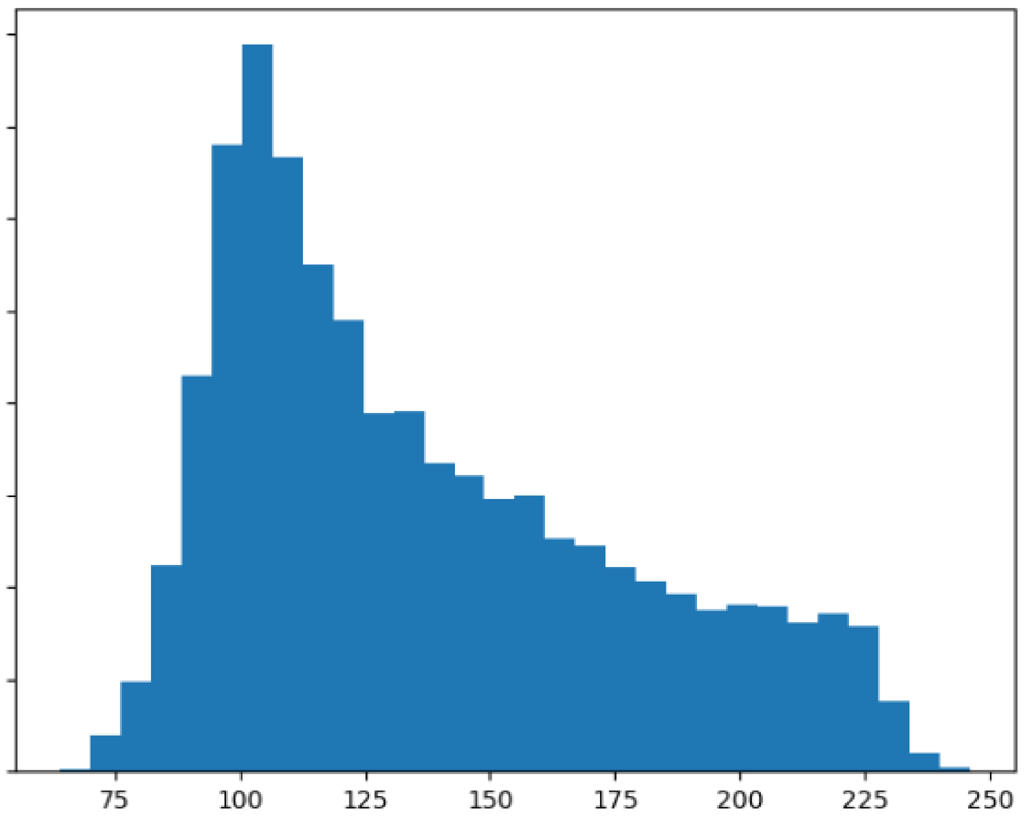

Figure 3: Histogram - Red Channel

Hence from the Bayes rule we have,

$$P(C_l|x) = \frac{P(x|C_l)P(C_l)}{\sum_{i=1}^{3} P(x|C_i)P(C_i)} \tag{1}$$

where C_i is, the probability of a class C_i occurring and the probability that a C_i occurs can be given as 1/(no. of classes), which in our case is 1/3. Since the denominator of $P(C_l|x)$ is the same for all three classes, we ignore the denominator. The resulting probability will simply be scaled by a common factor for all three color classes.

The probability $P(x|C_i)$ is modelled as a Gaussian distribution. This method might not work well under varying lighting conditions, because a single Gaussian does not properly model all color classes. Hence, we use a weighted sum of multiple Gaussian models [3]. In our project, we assume a mixture of three Gaussian models. Where, the new model is given as:

$$P(C_l|x) = \sum_{i=1}^{n} \Pi_i N(x, \mu, \Sigma_i) \tag{2}$$

where,

$$N(\pi_i, \mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^3|\Sigma|}} exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right] \tag{3}$$

In this, Π_i is a weighting coefficient and $N(x, \mu, \Sigma_i)$ is the i^{th} Gaussian Model. This expression represents a GMM of k different Gaussians.

And Σ_i is the co-variance matrix which is generated based on the values of the R, G, B saved in the training data and μ is the mean matrix, which has the mean values generated for each R, G, B in the same order of a pixel intensity matrix (x).

Σ_i, or co-variance matrix can be easily calculated by using the following formula.

$$\Sigma_i = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^T \tag{4}$$

where, N is the number of training pixels.

Expectation Maximization

We find these Π_is and the K Gaussian Model parameters using Expectation maximization, which is similar to K-means algorithm to cluster the given samples. To perform the EM we follow the steps described below.[3]

1. Randomly Choose a set of (Π_i, μ_i, Σ_i)
2. Calculate $\alpha_{i,j}$, where j is the data point (or the pixel), and i is referred to the cluster or the class.

$$\alpha_{i,j} = \frac{\Pi_i P(x_j|C_i)}{\sum_{i=1}^{K} \Pi_i P(x_j|C_i)} \tag{5}$$

3. Calculate new cluster parameters:

$$\mu_{i,j} = \frac{\sum_{j=1}^{N} \alpha_{i,j} x_j}{\sum_{j=1}^{N} \alpha_{i,j}} \tag{6}$$

$$\Pi_i = \frac{1}{N} \sum_{j=1}^{N} \alpha_{i,j} \tag{7}$$

$$\Sigma_i = \frac{\sum_{j=1}^{N} \alpha_{i,j}(x_j - \mu_i)(x_j - \mu_i)^T}{\sum_{i=1}^{N} \alpha_{i,j}} \tag{8}$$

4. Repeat steps 2 and 3 until convergence. Where convergence is defined as the case when

$$\Sigma_i ||\mu_i^{t+1} - \mu_i^t|| \leq \tau$$

Here t is the iteration number and τ is some user-defined threshold.

Project Code

In the expectation maximization code (EM.py), we assume three models and start with randomly generated (using module *random*) sets of model parameters, (μ_i, Σ_i). We start with same values of weights for each model. That is, we set all π_i as $\frac{1}{3}$. Then, we perform the expectation maximization steps as discussed in the previous subsection in a loop. We terminate the loop when the difference between the norm of difference between mean vectors (μ) of consecutive iterations drops below a threshold.

This algorithm has to be run on the training data set and takes a considerable amount of time to converge and generate the model. Hence, this algorithm is saved as a separate module in the code base. We generate the model and save its parameters in three yaml files, one corresponding to each color model. The yaml files contain a dictionary containing information about the weights, means and covariance matrices of the model.

For 1-D Gaussian, instead of the RGB pixel vectors as used in 3-D Gaussian, we only consider the G value for the green buoy, and the (R+G)/2 value for the yellow and orange buoys. These values are then used to calculate the probability. The output of 1-D Gaussian is the same as 3-D Gaussian, except that it does not perform as well and many frames do not yield an output in which we detect the buoy.

In the main module of the project (generate_colorClass_video.py), we parse the yaml files and generate mean vectors, covariance matrices and weights for that model. We use these parameters to find the probability that each pixel in the test video belongs to a color class. We generate a new binary image where pixels that have probability greater than a set threshold are white and rest are all black. We filter this binary image using a median filter of size 5 to get rid of noise in the image (which is predominantly salt-and-pepper noise). Then, we perform Canny edge detection and generate contours around the edges. Finally, we draw circles around these contour centers and generate the output video.

Images 4 and 5 represent the same frame of the input data. Image 4 is the binary image obtained after performing probability thresholding, median filtering and Gaussian blurring. This image is further processed by perform-

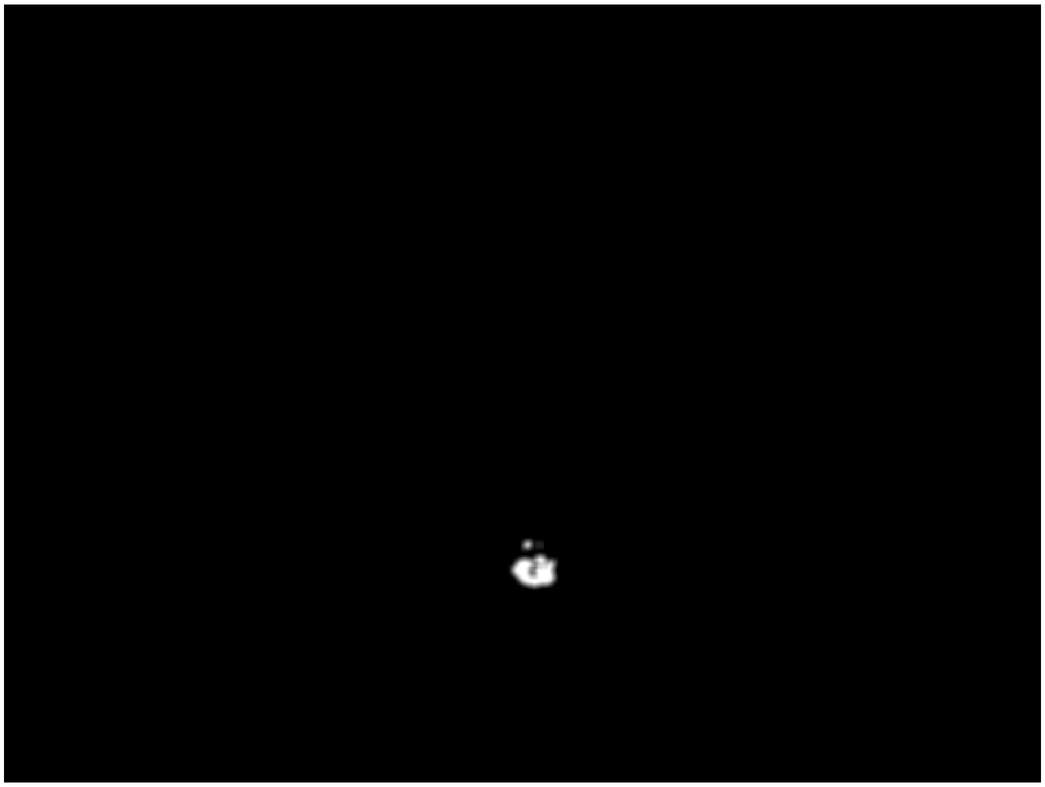

Figure 4: Binary intermediate image

ing edge and contour detection. The center of the contour is obtained and a circle is drawn on the original frame, which is seen in image 5.

Output Videos

The output videos can be found at **Google Drive**.

Interesting Problem

The binary image we get from simply applying the threshold has a lot of noise in the background. Hence, we cannot perform edge detection on this

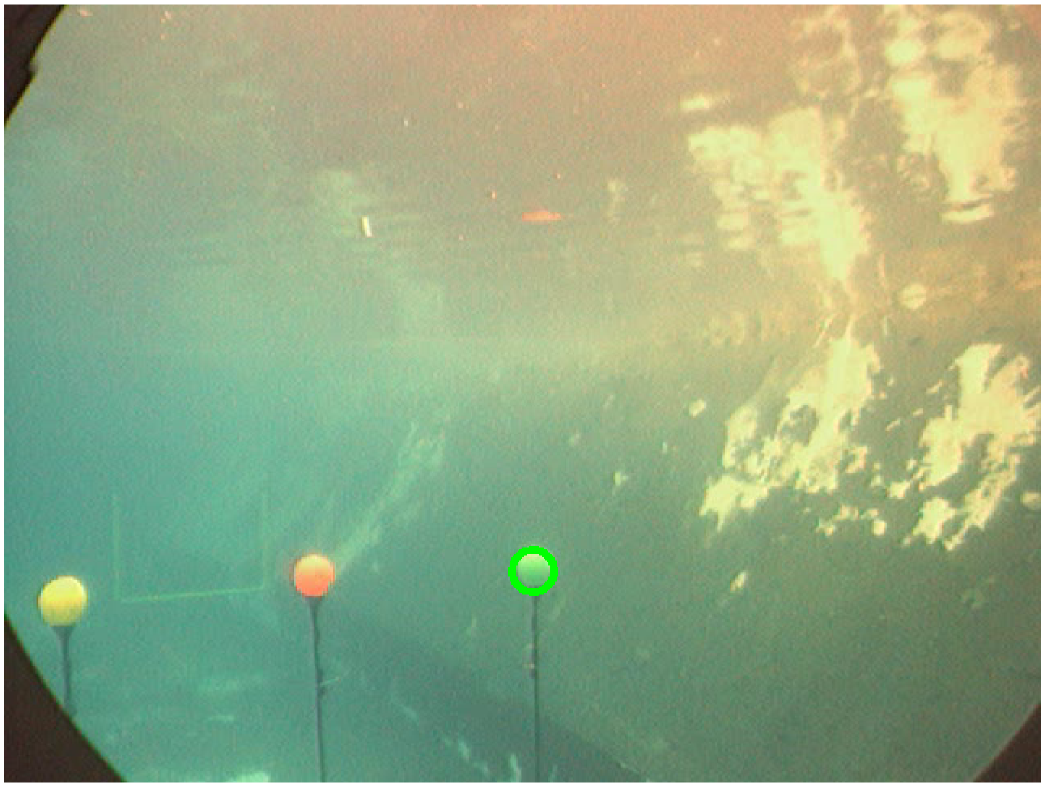

Figure 5: Output image

image directly, which is why we perform median filtering followed by Gaussian blurring.

The green buoy has a color that is similar to the water in the background, which causes a lot of background pixels to have a probability that is close to the pixels of the green buoy. This results in false positives and generates a lot of noise in the binary image generated for the green buoy. We solved this problem by increasing the threshold for the expectation maximization convergence as well as for the binary image generation.

Bibliography

[1] CSV Files with Python - Read, Write & Append
`https://www.youtube.com/watch?v=0Vl0iwkXrQ8`

[2] Selecting a ROI from a given frame
`https://github.com/JCardenasRdz/roipoly.py/blob/master/roipoly.py`

[3] Color Classification using GMM
`https://cmsc426.github.io/colorseg/colorclassification`