

# Project 3 – Color Segmentation using GMM

Aaradhana Kannan Subramanian  
Meghana Madarkal  
Rohini Basu

## README:

To run the codes, download the folder submitted using [Google Drive](https://drive.google.com/drive/folders/1iCiO89OKSZWzho2BTC2S8Tp6rk9XjJCW?usp=sharing). It contains all the different codes mentioned below. The python files must be kept in the same folder as the folders containing the training images. Each file can then be run from the command line.

**Link:** <https://drive.google.com/drive/folders/1iCiO89OKSZWzho2BTC2S8Tp6rk9XjJCW?usp=sharing>

## Set of training images:

The folders “green\_train”, “yellow\_train”, “orange\_train” and frames used to create the training images are included in the [google drive](https://drive.google.com/drive/folders/1iCiO89OKSZWzho2BTC2S8Tp6rk9XjJCW?usp=sharing).

## Files included:

1. takeimage.py : Code to cut out buoys from frames to generate train set.
2. 1D\_gauss.py : Code to generate random gaussian data samples and find means and standard deviations using EM.
3. 1D\_gauss\_green.py: Code to detect green buoy using 1D gaussian on green channel.
4. 1D\_gauss\_yellow.py: Code to detect yellow buoy using 1D gaussian on green and red channels.
5. 1D\_gauss\_orange.py: Code to detect orange buoy using 1D gaussian on red channel.
6. 3D\_gauss\_green.py : Code to detect green buoy using 3D gaussian on all RGB channels
7. 3D\_gauss\_yellow.py : Code to detect yellow buoy using 3D gaussian on all RGB channels
8. 3D\_gauss\_orange.py : Code to detect orange buoy using 3D gaussian on all RGB channels
9. 3D\_gauss\_all.py : Code to detect all buoys using 3D gaussian on all RGB channels

## Videos included:

The [google drive](https://drive.google.com/drive/folders/1iCiO89OKSZWzho2BTC2S8Tp6rk9XjJCW?usp=sharing) also contains the following videos:

1. detectbuoy.avi :The given video
2. 1D\_gauss\_green.avi : 1D\_gauss\_green output
3. 1D\_gauss\_yellow.avi : 1D\_gauss\_yellow output
4. 1D\_gauss\_orange.avi : 1D\_gauss\_orange output
5. 3D\_gauss\_green.avi : 3D\_gauss\_green output
6. 3D\_gauss\_yellow.avi : 3D\_gauss\_yellow output
7. 3D\_gauss\_orange.avi : 3D\_gauss\_orange output
8. 3D\_gauss\_all.avi : 3D\_gauss\_all output

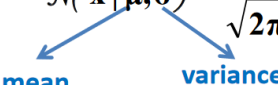
**Importing Modules:** The following modules were used.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
import os
from imutils import contours
```

### Introduction:

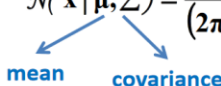
A dataset can have a distribution that is characterized by several clusters. Taking one mean (one gaussian) and estimating parameters is not the ideal way to model such a dataset. For example, if we take a dataset which has two significant means at value 218 and at value 250, taking the average might give us a value around 221. This is not the correct estimate. Using multiple gaussians with means at 218 and 250 will give a more accurate representation of the distribution of data in the dataset.

**The 1D gaussian formula is given by :**

$$\mathcal{N}(x | \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$


However, in some cases, when different sets of data, each with different numbers of clusters all describe one feature, it is more prudent to take a multivariate gaussian to model the data across the three sets. This way, we get a better understanding of how the clusters are spread out in the total dataset.

**The multivariate gaussian formula is given by :**

$$\mathcal{N}(x | \mu, \Sigma) = \frac{1}{(2\pi|\Sigma|)^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$$


This is applicable in our case where we look at an image with three color channels: Blue Green and Red. For example, if we look to detect the yellow buoy, we check for the number of clusters in each channel. Then, we use a three-dimensional gaussian to model the total image. In order to detect the probabilities of a pixel lying in a cluster and to estimate the means and co-variances, we use the Expectation Maximization algorithm.

The steps used to perform EM algorithm are detailed below:

1. Initialize the means and covariances with some random values. The covariance matrix must be of the shape (dim, dim) where dim is the number of dimensions of the gaussian. We store these values in a dictionary type data structure called 'parameters'.

2. **E Step:** The gaussians are combined as follows:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k)$$

Number of Gaussians
Mixing coefficient: weightage for each Gaussian dist.

We need to find the responsibilities. These are the corresponding probabilities for a given value  $\mathbf{x}$ . We can do this using Bayes rule as follows:

$$= \frac{\pi_k \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x} | \mu_j, \Sigma_j)} \quad \text{where, } \pi_k = \frac{N_k}{N}$$

These are stored in an (n\_feat, K) array called 'cluster\_prob'. Here, n\_feat refers to the number of rows in the dataset.

3. **M Step:** Now we need to calculate the new updated means and covariances. We can do that using the following formulae:

$$\mu_j = \frac{\sum_{n=1}^N \gamma_j(\mathbf{x}_n) \mathbf{x}_n}{\sum_{n=1}^N \gamma_j(\mathbf{x}_n)} \quad \Sigma_j = \frac{\sum_{n=1}^N \gamma_j(\mathbf{x}_n) (\mathbf{x}_n - \mu_j)(\mathbf{x}_n - \mu_j)^T}{\sum_{n=1}^N \gamma_j(\mathbf{x}_n)} \quad \pi_j = \frac{1}{N} \sum_{n=1}^N \gamma_j(\mathbf{x}_n)$$

4. Now we evaluate the log likelihood. Our aim is to increase the log likelihood function until the change in likelihood falls below a certain value. The log likelihood is calculated as follows:

$$\ln p(\mathbf{X} | \mu, \Sigma, \pi) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k) \right\}$$

The log likelihood is appended to an array called log\_likelihoods. The difference in log likelihood is found by subtracting the recent value from the last stored value.

This process is iteratively repeated until the difference in log likelihood falls below a certain value or the maximum iterations are reached. The final values returned are the estimated means and covariances of the given dataset.

## Generation of data and Evaluation of means and standard deviations using GMM and EM:

In this case, we use three 1D gaussians to generate three random datasets. The given ground truth values are as follows:

**Means:** 0, 3, 6

**Standard Deviations:** 2, 0.5, 3

We then concatenate these data values to arrange them in a column form. The EM algorithm is executed for K=3. The estimated means and variances are printed on the console. The obtained output is as follows:

```
Mean 1: 3.170345464207857
SD 1: [0.22631275]
Mean 2: 6.492257745502069
SD 2: [5.36180102]
Mean 3: -0.7088554514888479
SD 3: [2.49928468]
```

We can see that the generated means and standard deviations are close to the ground truth values. Thus, the ground truth parameters are approximately recovered.

## Creating Train Set

The train set images are cropped out for each buoy from multiple frames. The goal is to get the different shades of the buoy represented in the training set so that the model accommodates for these clusters of pixel values.

In the code to crop the images, we use a mouse click event to obtain pixel locations from the image and store it in a list. Then we crop out this area and store it as an image. We take different regions from the buoy to form the train set.



Fig. Cropping process



Fig. Train set examples

## Buoy detection

### Using 1D gaussian:

For each buoy, the detection pipeline is divided into training and testing. To train any given model, we need to first decide the kind of gaussian we need to use to model the data. In this case, we use a 1D gaussian and use the EM algorithm to find the points lying in the clusters.

We start by taking in images from the folder containing the training set images. These images are further cropped to get a close crop of the actual buoy itself. We then plot the histograms of each channel of this image set to get an idea of the number of clusters present. We can add up the number of clusters from each channel's histogram and decide the total number of clusters (K), for the buoy. This denotes the number of gaussians we need to use to detect this buoy in frame.

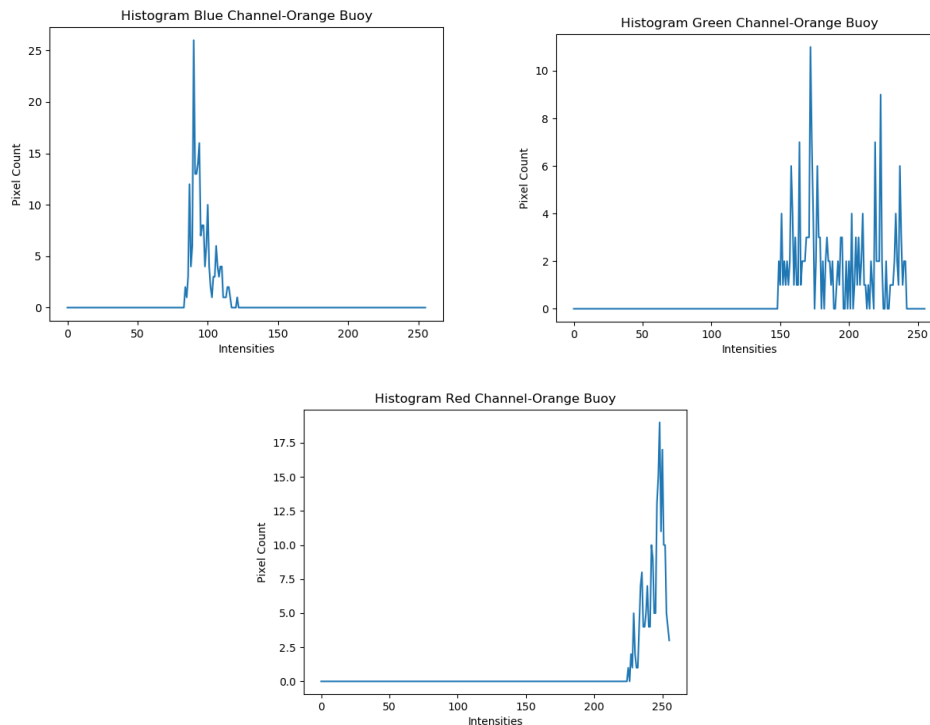


Fig: Histograms for each channel for the orange train set

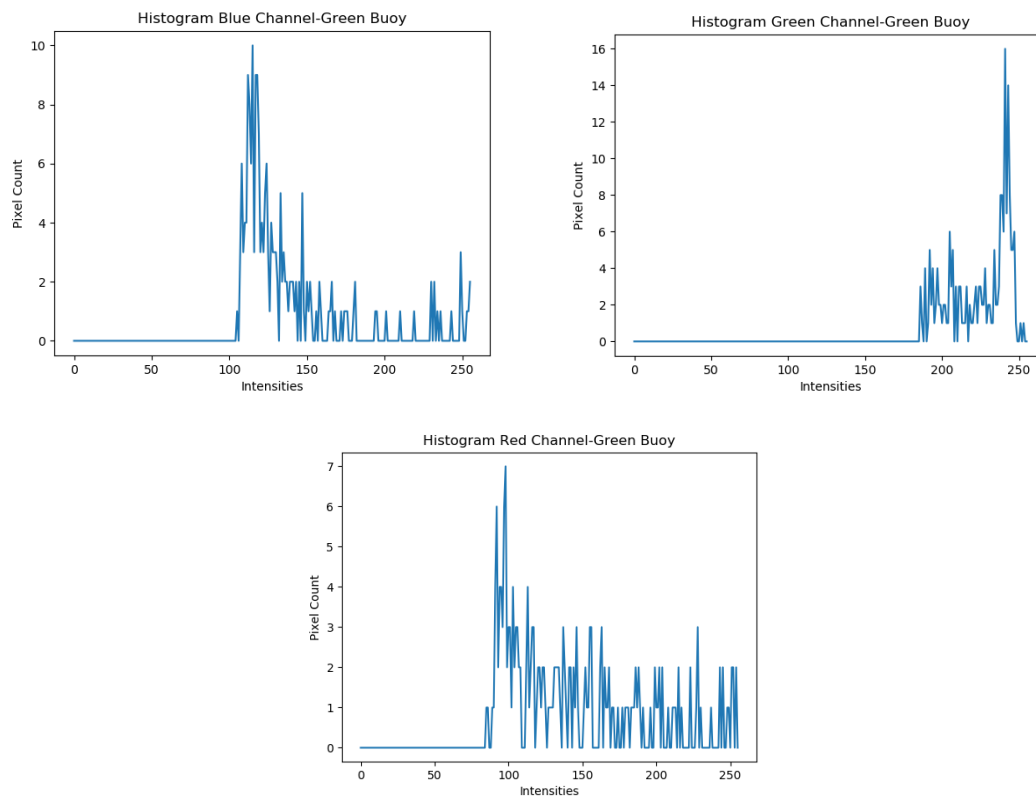


Fig: Histograms for each channel for the green train set

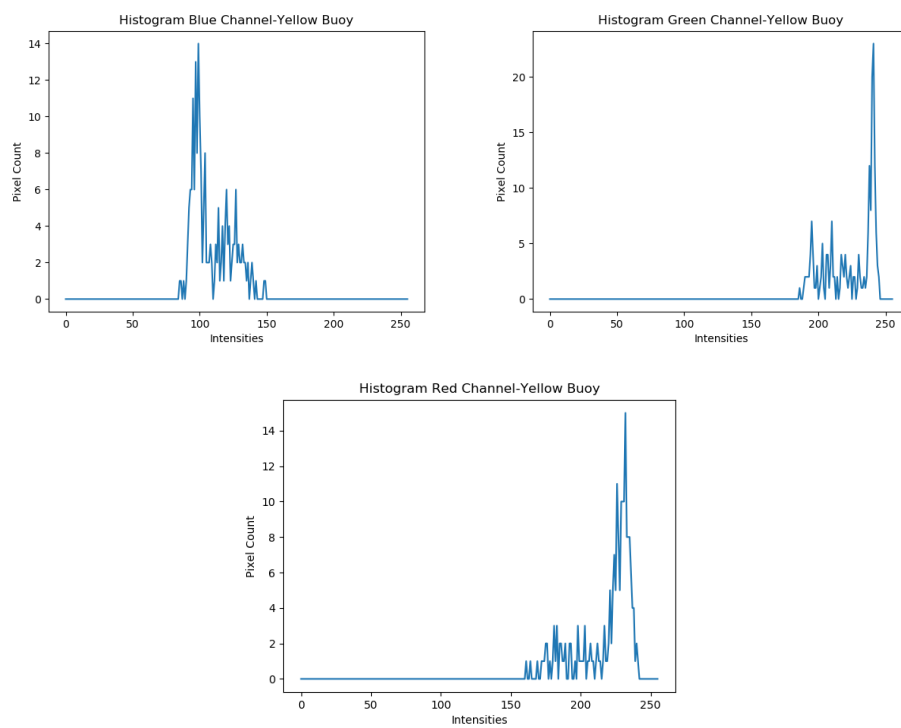


Fig: Histograms for each channel for the yellow train set

Then we extract the channel of the image we want to work with. Since this is the 1D case, we arrange the extracted channel(s) in a single column. For green buoy, we only use the green channel, for orange we use the red channel and for yellow, we stack the data in both red and green channels to form one column.

Once K value is decided, we initialize the K means and standard deviations (since it is a 1D case) using randomly generated numbers and start the EM algorithm. The algorithm runs until it converges to the given value (bound = 0.0001) or until it reaches the maximum number of iterations specified (1000). When it exits the loop, the last calculated mean and standard deviation values are stored in a '.npy' file.

To test the obtained values, we read the parameters from the stored files and re-evaluate the gaussian for every frame in the given video. We then calculate the log likelihood. Now we see what the probability is of a point lying in a cluster. We take the maximum value of calculated probabilities, scale it down by a heuristic value, and assign the 255 for every pixel that has a probability greater than this value. We fill these values into the corresponding channels of the blank output frame.

Now, we need to detect the contour of the obtained detected mask image. We blur the image and detect edges using Canny edge detection. Then, we use findContours() function to do this. Once the contours are detected, we sort them. Then, we pick the first contour from the obtained list of contours and try to evaluate the complete hull formed by these points using the convexHull() function. After that, we give these set of points as input to the minEnclosingCircle() function to get the approximate center and radius of a circle that encloses the hull. We then plot the circle around the detected buoy using cv2.Circle() function. The output is displayed frame by frame and then written to a video file as well.

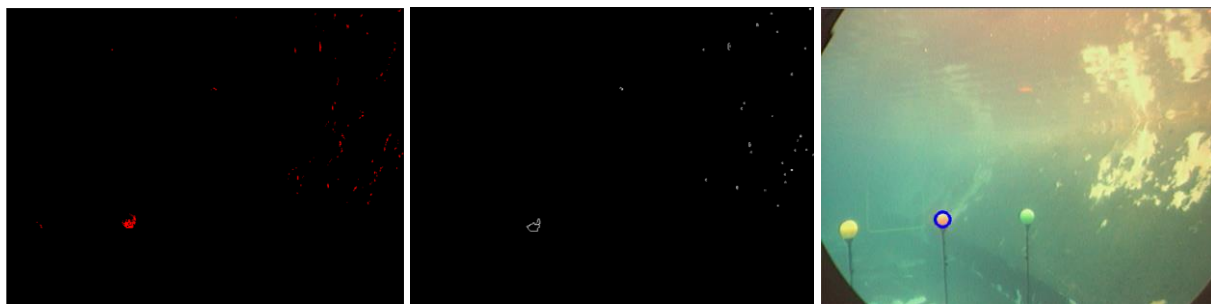


Fig : Detected orange buoy using 1D gaussian

### Using 3D Gaussian:

We use the same pipeline as the 1D case. The only changes occur in the way we send the input data. Instead of sending just one column of the image, we send all the three channels. Each channel is converted into a column of its constituents and appended to a stack. Finally, we have a 2D array with three columns (each corresponding to one channel) and rows equal to the total number of pixels in each channel.

The multivariate gaussian's initial parameters are then initialized using random values of the appropriate dimensions. The EM algorithm is then evaluated, and the rest of the steps are the same as 1D gaussian case.

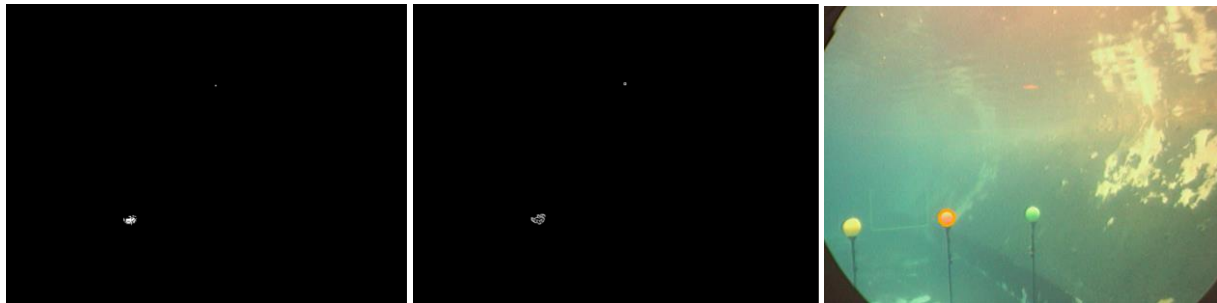


Fig: Detected orange buoy using 3D gaussian

### Outputs:

Shown below are the buoy detection outputs for different frames in the video for the 1D and 3D case.

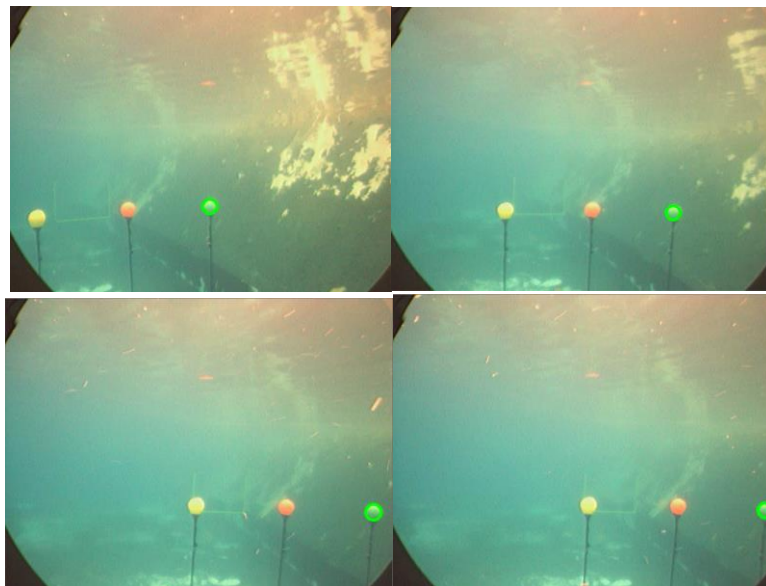


Fig: Detected green buoy using 3D gaussian



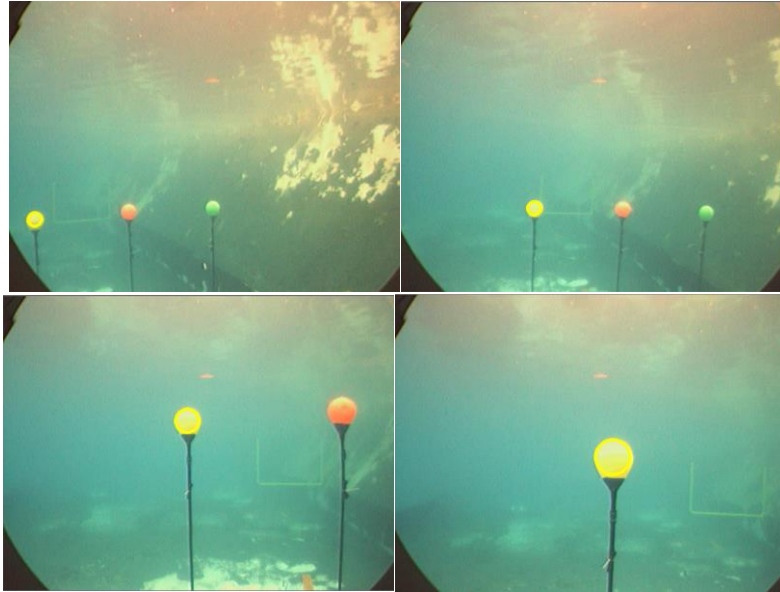


Fig: Detected yellow buoy using 3D gaussian

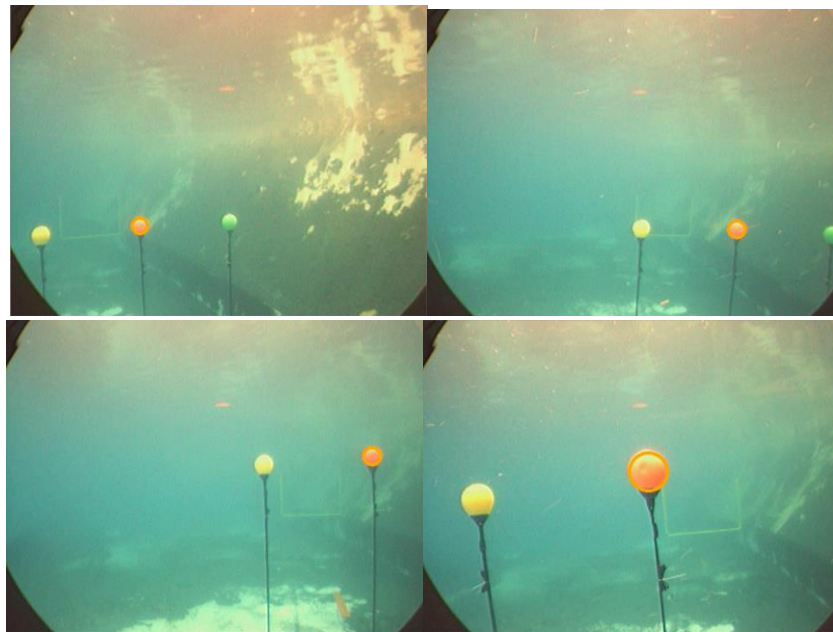


Fig: Detected Orange buoy using 3D gaussian

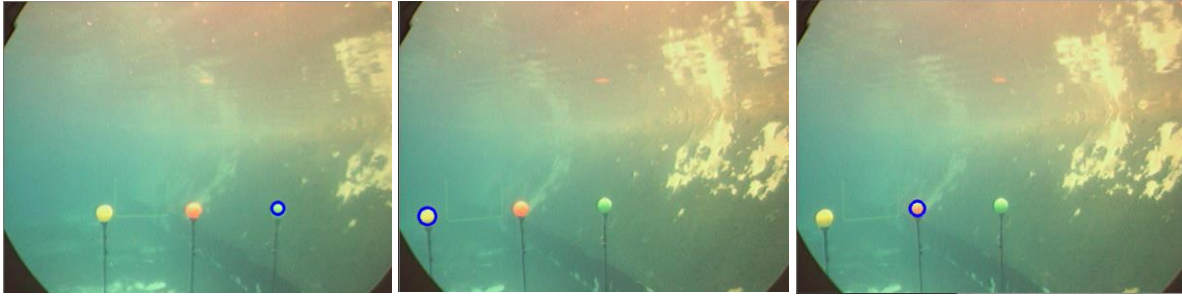


Fig: Detected buoys using 1D gaussian

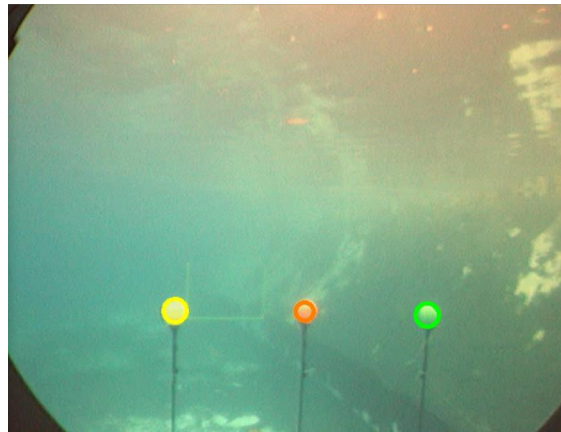


Fig: All Detected buoys using 3D gaussian

### **Inference:**

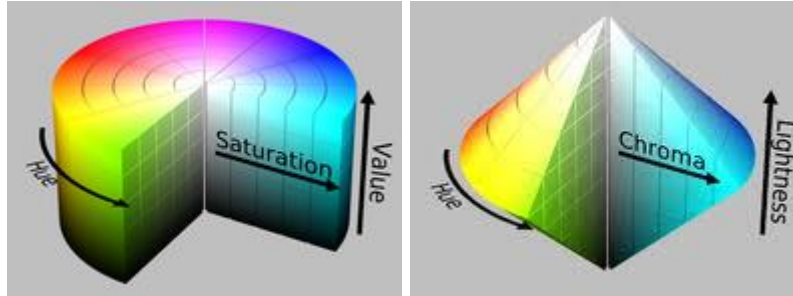
It can be noticed from the output videos that for the 1D case, the results for green buoy detection were suboptimal. The green color was predominant in the background as well as in the yellow areas of the image (including the yellow buoy). Thus, during contour detection, the code detects areas in the background that match similar criteria as the green buoy.

In the 3D case, since all the channels of the green buoy are trained together, for the number of clusters specified, it increases the likelihood of finding pixels that belong to the buoy alone. Since the buoy has color components that belong to channels other than the green channel, these components are specific to the buoy alone and aren't found in the background. Thus, buoy detection is more robust.

Similar results are observed for both the yellow and orange buoy detection as well.

### Working with other colorspace:

The color segmentation technique will yield better results in other color spaces. For example, converting to the HSV, HLS or YCrCb colorspace will give us more control over the colors we are attempting to segment.



For example, in the HSL colorspace, we have more control over the color we want to isolate. The hue range is specific for each color, the saturation component gives us the intensity and the lightness value can help with brighter and darker shades of the same color. Whereas, in the RGB space, we need all the three values to specify an exact shade.

### Challenges Faced:

- Preparation of dataset: A lot of issues were faced while reshaping the dataset to fit gaussians over. At first, we struggled with understanding how a multidimensional gaussian would accommodate a three-channel image. Later on, we understood that each channel needs to be represented by a column and wrote our code for the gaussian estimation accordingly.
- Adjusting parameters: Even after training the model with the appropriate number of clusters, the output was not always satisfactory. We fixed this by adjusting the dividing factor while assigning values to pixels until the output was satisfactory. We also tried various filters to denoise the image before edge detection to get a satisfactory contour detection.
- Writing to video: We had to try various codec schemes with different fps until the XVID scheme worked.

### References:

MVN: [https://en.wikipedia.org/wiki/Multivariate\\_normal\\_distribution](https://en.wikipedia.org/wiki/Multivariate_normal_distribution)

EM Algorithm: <https://www.ics.uci.edu/~smyth/courses/cs274/notes/EMnotes.pdf>

GMM concepts: <http://www.vlfeat.org/overview/gmm.html>

Python dictionary data structure: [https://www.tutorialspoint.com/python/python\\_dictionary.htm](https://www.tutorialspoint.com/python/python_dictionary.htm)

Color models: [https://scc.ustc.edu.cn/zlsc/tc4600/intel/2015.1.133/ipp/ipp\\_manual/GUID-AE698C04-81DB-402B-88E7-2BEED820D4DF.htm](https://scc.ustc.edu.cn/zlsc/tc4600/intel/2015.1.133/ipp/ipp_manual/GUID-AE698C04-81DB-402B-88E7-2BEED820D4DF.htm)