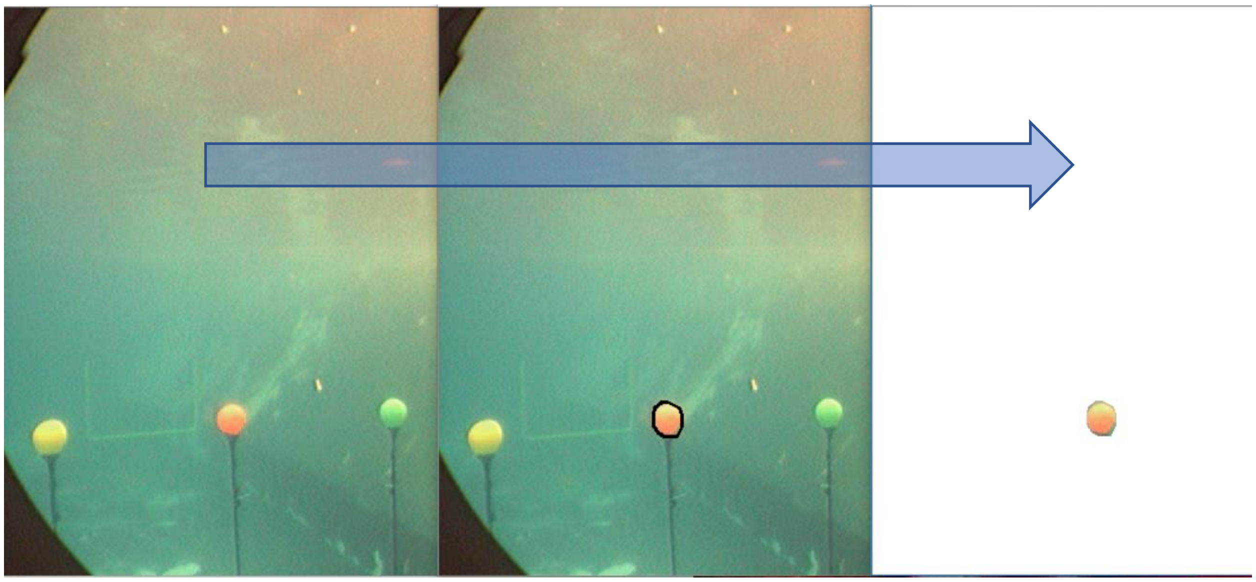
Akash Atharv, Rishabh Choudhary, Sanchit Gupta

Project-3 Report

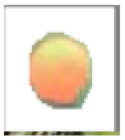
Color Segmentation using Gaussian Mixture Models and Expectation Manimization

Aim:
The aim of this project was to perform color based segmentation using Gaussian Mixture Model and Expectation Maximzation techniques. In this technique we are first learning the color distribution in each of the buoys and then test the trained model on the test data set and then on the original frame.

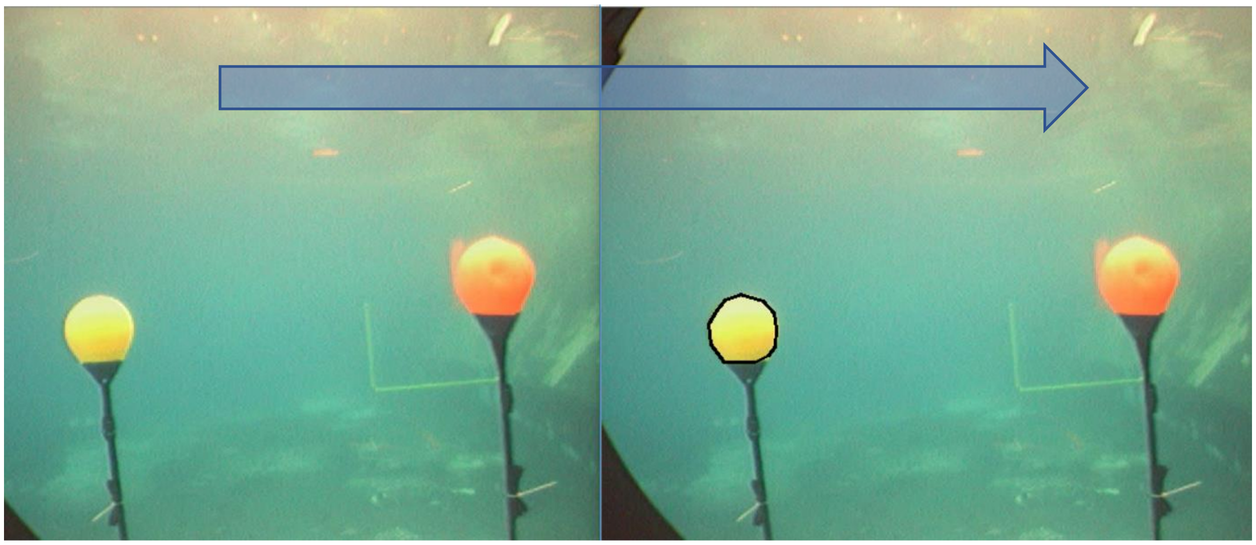
Part1 - Preparing Data:
For preparaing the data, we used event handeling in Python. Using the left click of the mouse, we go around the boundary of the buoy we want the data for and make a bounding shape on top of it and also save the coordinates of this bounding shape. Once we have the coordinate points we draw a contour using these points and then using bit wise operations we crop out the buoy on a white background. The same operations are shown below going from left to right.

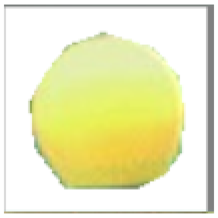


We then find the min value of the pixel where there is a non-zero value and the max value of the pixel location where there is a non-zero value and then using these values crop out a square smaller image of just the buoy, which is shown below:



Some more examples of the cropping data are shown below captured at different times in the video to give a rich data set.





Different examples of data set collected:

The code for the same is saved in the file: “Crop_out_Buoys.py”, this file needs an image of the frame to run.

Part2 – Training the Dataset for different Buoys

Python modules needed to run this code:

- *Matplotlib (plt) for histogram plotting*
- *Os for file manipulation*
- *Imutils (contours) for convex hull plotting*

After loading the images into the program, we reshaped/cropped the image so that the background region also cropped and present in the images can be removed, without removing this part the background will also be trained with the buoys and give bad results in the final output.

The cropped buoys trained images are then reshaped to be in the shape (nx*ny, 3) from (nx,ny,3), where the “nx” is the number of pixels in the x axis and “ny” is the number of pixels in the y axis. This is done to get the images in 2D matrix.

Deciding on the number of clusters

To decide the number of clusters for the Gaussian, we used histogram to get a general intuition and then played around with the values until we got a good result. The color profile histogram of all different buoys with three channels is plotted separately. This histogram is then used to discern initial distinguishing features between buoys. Separate histograms generated for each buoy is depicted below:

Thus, we use this information to further divide the data into different clusters for application of further operations such as Expectation Maximization

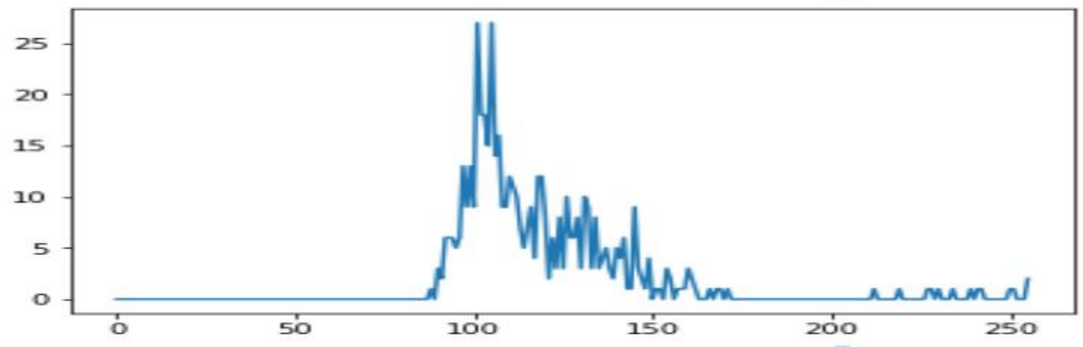


Fig: Histogram for Yellow buoy for blue channel

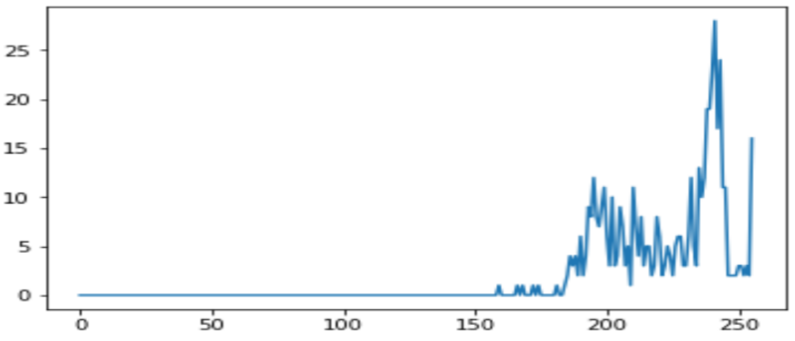


Fig: Histogram for Yellow buoy for Green channel

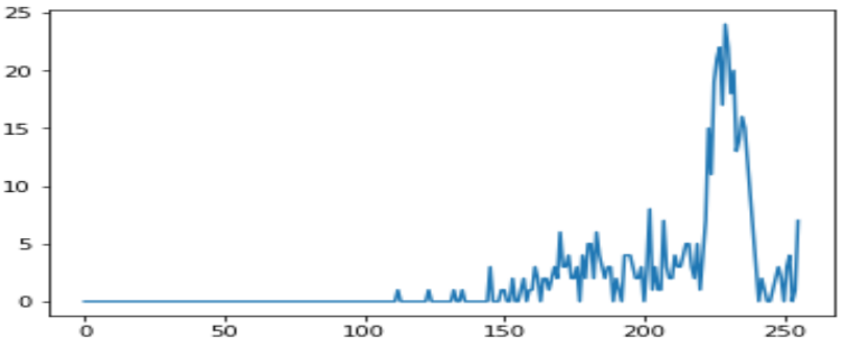


Fig: Histogram for Yellow buoy for red channel

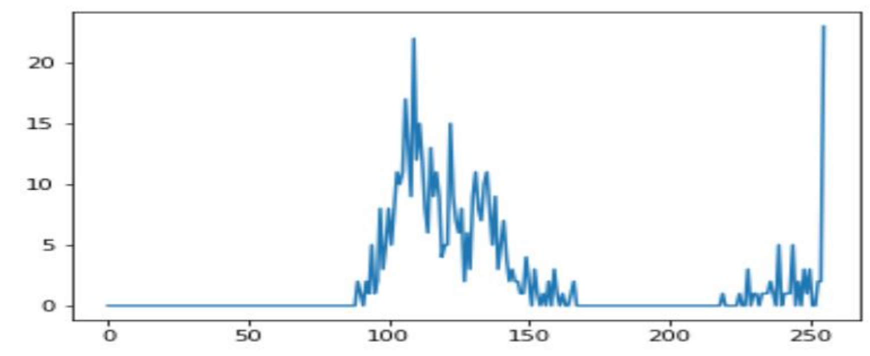


Fig: Histogram for Orange buoy for blue channel

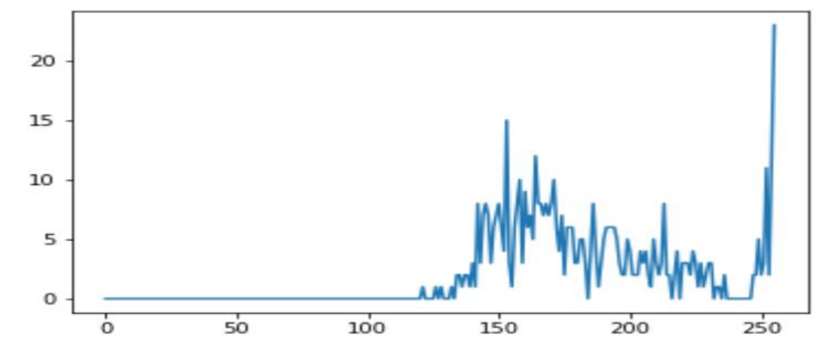


Fig: Histogram for Orange buoy for Green channel

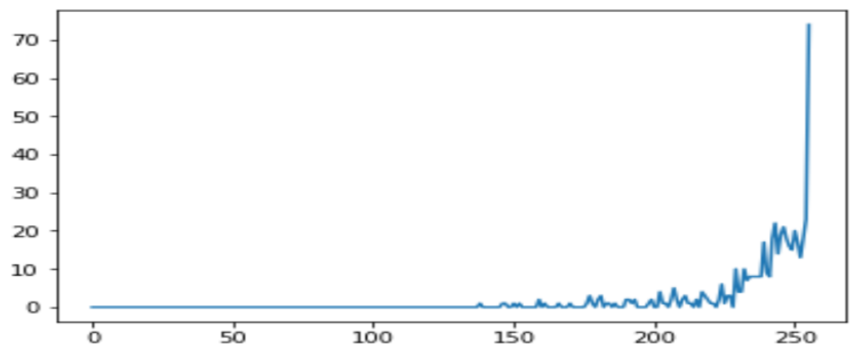


Fig: Histogram for Orange buoy for red channel

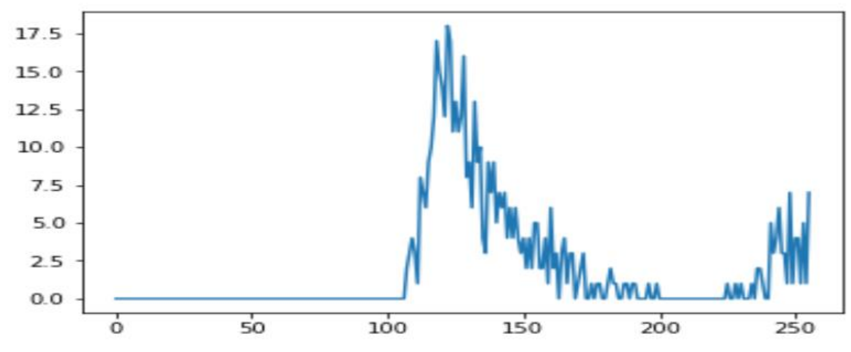


Fig: Histogram for Green buoy for blue channel

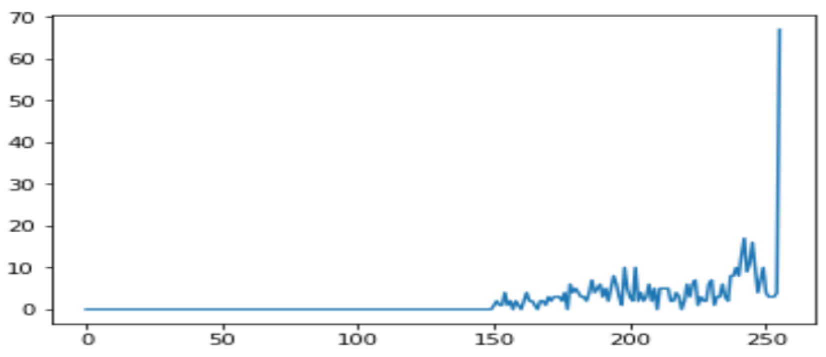


Fig: Histogram for Green buoy for green channel

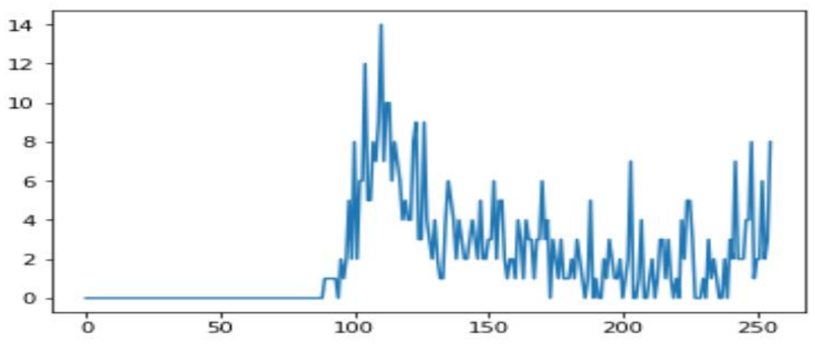


Fig: Histogram for Green buoy for red channel

Univariate Gaussian:

Once we have an idea about the number of clusters we will be training for each colored buoy, we start with the training of the data set. To perform this, we first try to fit a univariate gaussian to the data set.

The univariate data set contains only a single column vector of data points.

The equation for the univariate gaussian takes mean and variance as an input and gives the Pdf (Probability density function) for each data entry as an output. The equation can be seen below:

$$\mathcal{N}(\mathbf{x} \,|\, \underset{\text{mean}}{\boldsymbol{\mu}}, \underset{\text{variance}}{\boldsymbol{\sigma}}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

To run the univariate gaussian in the expectation maximization, we first have to initialize the value of the mean and the variance. The variance is initialized by a random value between 1-255 read as a float value and the mean is initialized with any random value from the column of the initial image that we have.

After the initialization, we calculate the pdf using these initial values and then update the value of variance and mean with these values. This whole first step can be broken down into 2 smaller steps of the EM algorithm.

Multi-variate Gaussian:

For the multivariate gaussian we read the initial image and form the image in the shape of (nx*ny, 3), instead of the variance, the multivariate gaussian requires a co-variance matrix as an input. To initialize this co-variance matrix, we take a random value from 1-255 and then multiply it with the identity matrix. Rest everything is similar with the univariate gaussian.

The whole EM algorithm is broken down into 2 steps:

The E Step:

The E-step or the estimation step takes in the value of the variance and the mean and outputs the pdf, which is different for different clusters, the same is combined using:

$$\mathbf{p}(\mathbf{x}) = \underbrace{\sum_{k=1}^{K}}_{\text{Number of Gaussians}} \underbrace{\pi_k}_{\text{Mixing coefficient: weightage for each Gaussian dist.}} \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)$$

Now we can evaluate the posterior probabilities called responsibilities using Bayes Rule:

$$\mathbf{p}(\mathbf{k} \mid \mathbf{x}) = \frac{\mathbf{p}(\mathbf{k})\mathbf{p}(\mathbf{x} \mid \mathbf{k})}{\mathbf{p}(\mathbf{x})}$$

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

These values are calculated and then returned for each iteration to the M step.

The M step:

In the M step, the input values which are the pdf returned from the E step are taken and new values of the co-variance/ variance and mean are calculated. The formulas for the same are given below:

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

With these parameters log likelihood is calculated and stored in a list as a column vector and is maintained and updated in each iteration.

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

The values of the log likelihood when differs from the previous iteration by less than 0.001, then we say that the convergence is reached and do not run more iterations and if not, then the calculated new mean and co-variance/variance data is given to the E step to again calculate the new pdf.

When Convergence Reached:
When the convergence is reached while training the dataset. We break out of the training loop and save the values of co-variance/variance and the mean in an "npy file". This completes the training set with the number of clusters, dimension of the gaussian, mean and the value of co-variance fixed for a certain color of buoy.

Testing Data set:
Now for testing the trained model on the data set we select an image from the test data set, this for the initial part should be the one of the cropped buoys and then when you get acceptable results, must be a frame from the original image.

Implementing the trained algorithm on the data set, detection buoy:

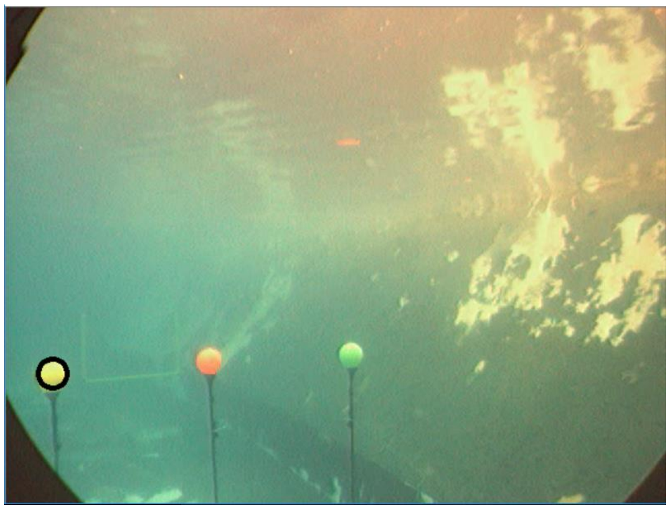
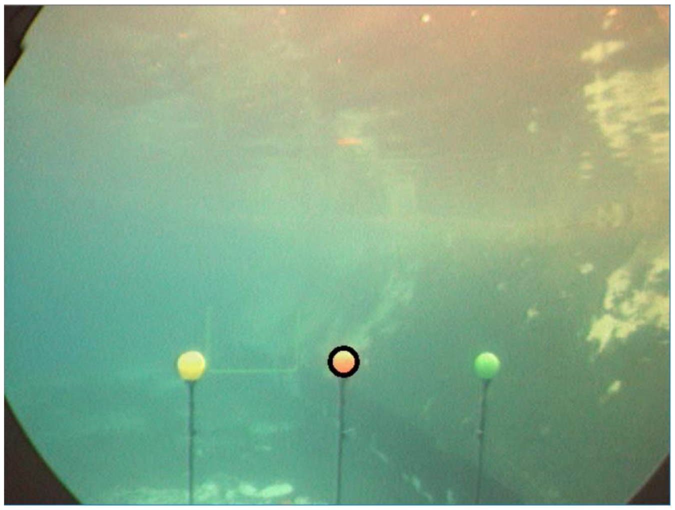
For implementing on the original frame of the video, we read the frame and then convert it into the following shape- (nx*ny, 3) where nx is the number of pixels in the x direction and ny is the number of pixels in the y direction. Then those columns vectors are selected which are trained for that particular buoy. The value of the co-variance and the mean are read from the files saved in the training set and used to calculate the probability of each pixel of the frame to tell whether it is present in any of the trained cluster or not. All these probabilities are stored in separate columns and are summed to find the probability if a pixel is present in any of the cluster or not. If the probability of a particular pixel is more than a certain scaled value, then that particular pixel is written 255 on a black image which has the same shape as out input frame.

Now all the places where the buoy is detected is marked in white on a black background. This image is then blurred to get a more uniform result using a Median Blur and then canny edge detection is done to get a finer edge of the detection. Then on this image find_contour function is used, and the detected contours are then sorted from left to right as they appear in the image frame.

Once, the left most contour is detected a convex hull is used on it, we then get the radius of these convex hulls from the function. The first radius which is more than a certain fixed value is used to draw a circle on the detected convex hull using OpenCV and the radius found from the convex hull. When no convex hull is detected with radius more than the pre-selected value.

The same is repeated for all the frames in a video, each of these frames with a printed circle on the detection is then written in a video (.avi) file using openCV's Video writer function. This video is stored in the same folder as the python program.

Output from the 1D

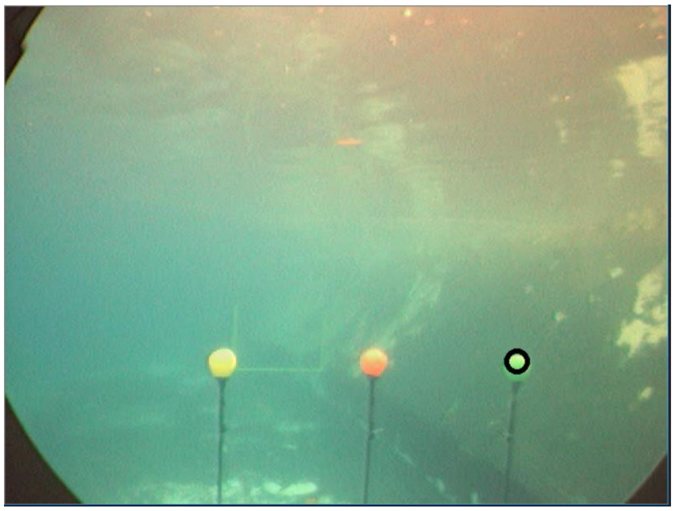


Fig: Output of different buoy detected using a 1D gaussian

Output from multivariate Gaussian:

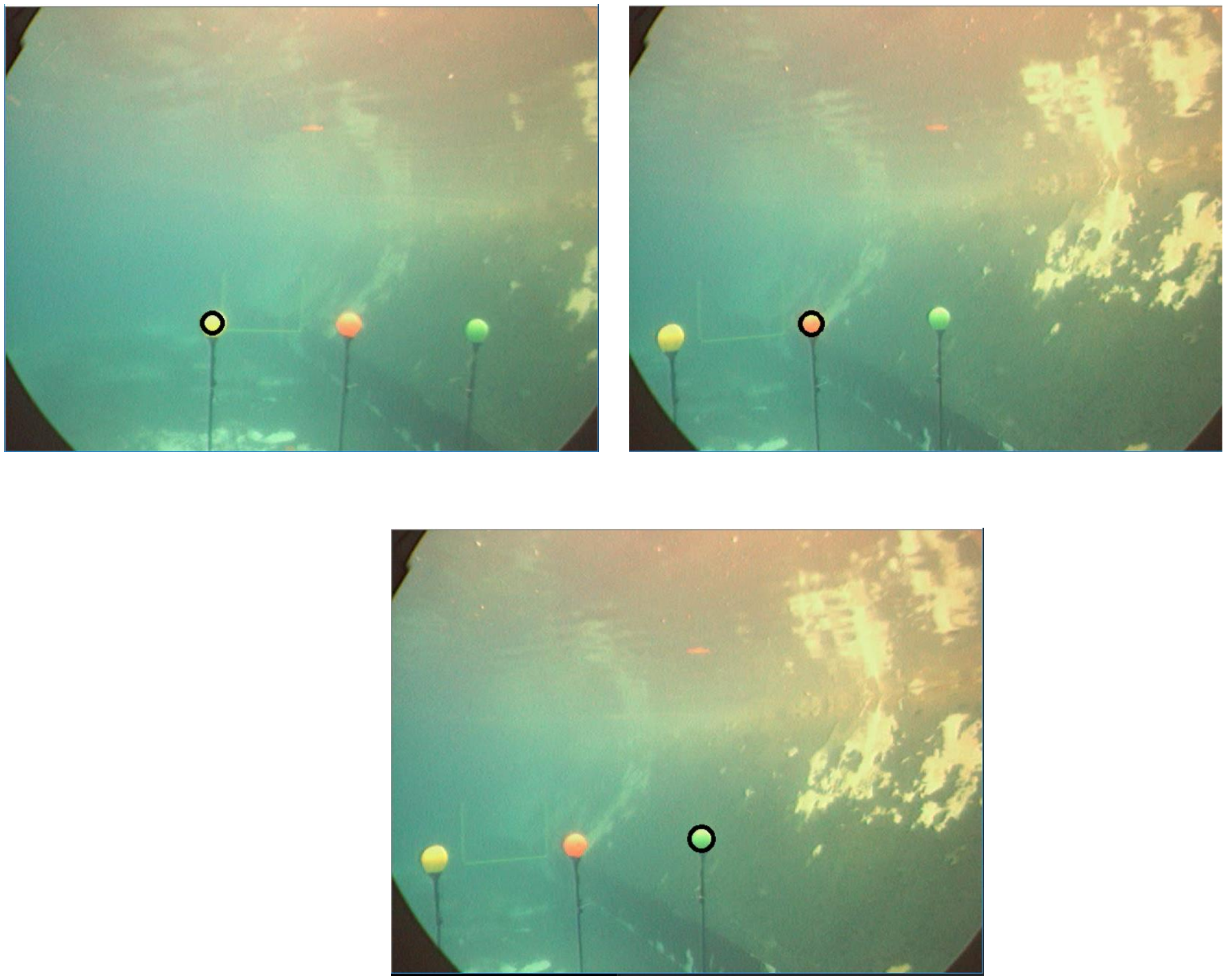


Fig: outputs from the multivariate Gaussian for different buoys

Observations from the results:

In the 1D gaussian, the trained models for the different colored buoys are not detected very well and the result often merges with the background or is not detected. The green buoy results are not at all optimal in the 1D gaussian training.

The results obtained after training a multivariate model are much better and robust, they fit the data set better and do not get lost in the background noise.

Final Result:
All the detections were merged into a single frame video, the output for the same is shown below in specific colors:

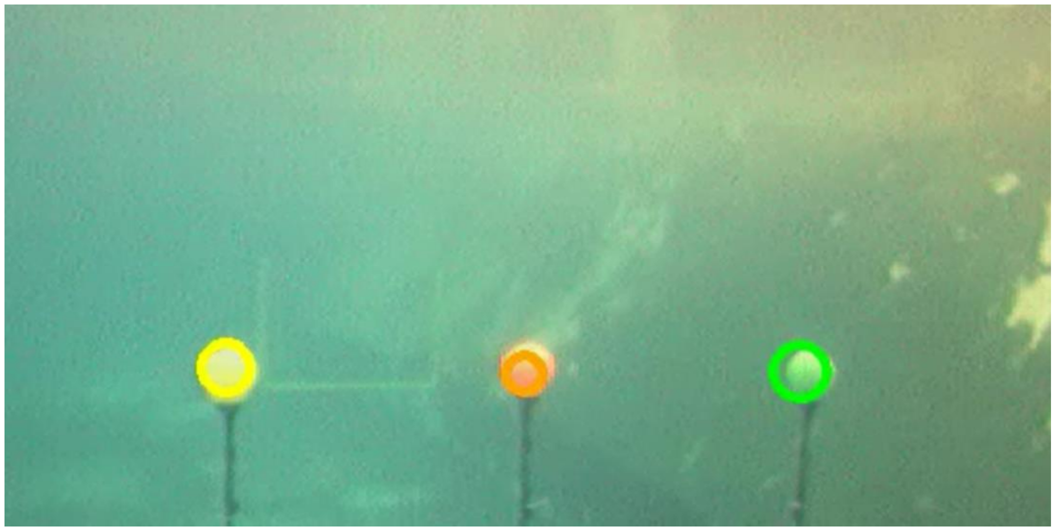


Fig: Final Output

Challenges Faced:

- **Deciding on the number of clusters:**
  To decide the number of clusters in each channel we had to run the training data a lot of times, sometimes the convergence wasn't even reached and sometimes a singular value of the determinant was reached of the co-variance matrix. In the end we did get some idea using the histogram of different channel for different buoys and selecting the number of clusters using them.

- **Co-variance Matrix dimensions:**
  We were confused about the concept and the dimensions of the co-variance matrix, we initially thought about making the co-variance matrix for each pixel, but that code often gave us error and wasn't able to converge.

- **Drawing the circle around the detection:**
  To draw the circle after detecting the pixel location, we had to try several different method before we could zero on the find contour and then sorting them from left to right, we further put another check on top of it for the radius because there were times when the certain buoy wasn't in the frame and it was detecting something from the background. This fine tuning took us a lot of time.

- **Deciding the dividing factor for detection:**
  The final deciding factor for deciding if a pixel probability was above or below the certain value was very vague

and we had to guess it by printing the probability values on the console. The largest value in the diagonal of the console was selected and used to divide in the end.

Further Analysis - Use of different color space:

For color segmentation operations we generally do not prefer to work in RGB mode because similar looking color have completely different characteristics based on the value between 0-255. A different color space like HSL has the value of the intensity separated from the color and hence in the histogram equalization only intensity is present not color. This helps to club the similar looking colors together which helps in detection and segmentation. Similar to HSL, YrCbCr is also a popular choice which is used for segmentation.

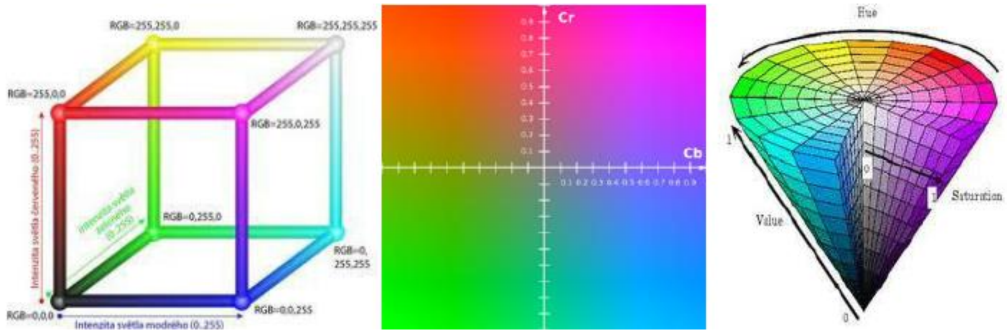


Fig: a) RGB b) YrCbCr c) HSL

The best way still for color segmentation will be to create our own color space and then try to train the pictures according to that.

References:

- Event Handling, mouse click for data preparing:
  https://www.pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/
  https://stackoverflow.com/questions/28327020/opencv-detect-mouse-position-clicking-over-a-picture
- To understand the GMM and the EM concept and get the formulas for the same:
  http://www.cse.iitm.ac.in/~vplab/courses/DVP/PDF/gmm.pdf
  https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html
  https://www.youtube.com/watch?v=REypj2sy_5U&t=35s
- Convex hull calculation:
  https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/shapedescriptors/hull/hull.html
- Find contours understanding of the Tree hierarchy:
  https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html