ENPM673

Project – 3: Underwater Buoy Detection.

Submitted by –

| Chayan Kumar Patodi |
I Prasanna Marudhu Balasubramanian |
| Saket Seshadri Gudimetla |

Introduction :

The idea of the project was to develop a pipeline for underwater buoy detection. We were provided with a video sequence , that has underwater footage , which we had to process and work on with certain methods and approaches.
Since it was mentioned that color segmentation techniques wouldn't work in such an environment , we were suggested to work with 1-D Gaussian Models and Gaussian Mixture Models.

Data Preparation :

As suggested , our very first step was to break down the whole video into frames. Since the fps of the video was 5 , we got 200 frames from the video.
To achieve this , we wrote a code , which would create a Folder , titled "Frames" and the broken down 200 frames , would be stored in the Original Frames folder inside this folder.

Our next task was to separate the frames in training and test sets. We were asked to create different folders for different colors of the buoys. So we created three folder titles 'Green','Red','Yellow' , inside the Frames folder . Inside those folders we created the training and test folders for splitting the frames.

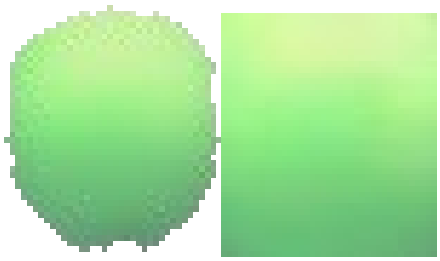
The catch here was that , we can't split the images in a similar fashion for each color , since the green buoy was only present for the initial 42 frames in the video , whereas red and yellow buoys are present throughout the video. So , to create a dataset for green buoy we have to consider only the initial 42 frames , whereas for red and yellow buoys , we'll be using all the 200 frames.

So for the green dataset , we considered every i+2nd frame , and we took 40 frames in consideration , so to we had to split 20 images in a 70-30% ratio as suggested ,and we got 14 Training Images and 6 Test Images.
Similarly , for red and yellow buoy , we considered every $i+10^{th}$ frames , and we took 200 frames in consideration , so we again had to split 20 images in a 70-30% ratio , which gave us 14 training images and 6 test images.

We stored all those images in the folder's created from the code itself. The real task for data preparation began once we had the dataset. Now we had to crop the buoys , so as we can use them to train our model in the future steps.We will be cropping 7 images out of the 14 training images.

We tried multiple methods to crop the buoys from training images. We created a separate code , which can be found under the name "Event_Handling.py".We created a few functions , which when called , will apply filters and morphological operations on the frames and then will show us the image , and the user will be asked to mark points on the image , once the user selected n points (we selected 10-15 points for better bounding polygon) , the points would get stored in a numpy array , as suggested in the project description. From there we would use that numpy array to get the ROI. The problem here , was that we were getting a tight polygon , but it was on a black background , which altered our histogram values . So we created another function here , to make the transparent background , we worked with the alpha channel and we were able to successfully eliminate the background from the image. Even though the background was eliminated , we couldn't use those

images for training the data . The pixel values of the transparent background was [0,0,0] , which was the issue with the 1st attempt. We tried eliminating the all rows with [0 0 0] , but even with that , our histogram was not giving us the desired output. So after applying all the approaches to use the tighter polygon , we gave up and we created a function to crop a rectangle from the buoys and we tried to get the best cropped image we could and we worked on the same.



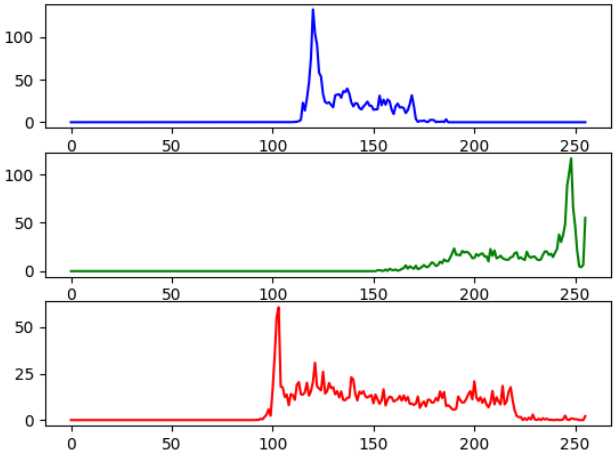
The above mentioned codes and folders could be find attached with this report and are as follows :
1. **Frames.py** : generates the folder and saves the images and also calls in the functions from another file and asks the users to crop the buoys (green , red and yellow respectively).
2.**Event_Handling.py** : this contains all the function we tried to use and we used to crop the images.
3. **Frames** Folder : Has all the folders inside , with original frames and the training , test and cropped set.
4. **OGAttempt** : This folder contains our initial dataset , with tighter polygons in the cropped images folder.

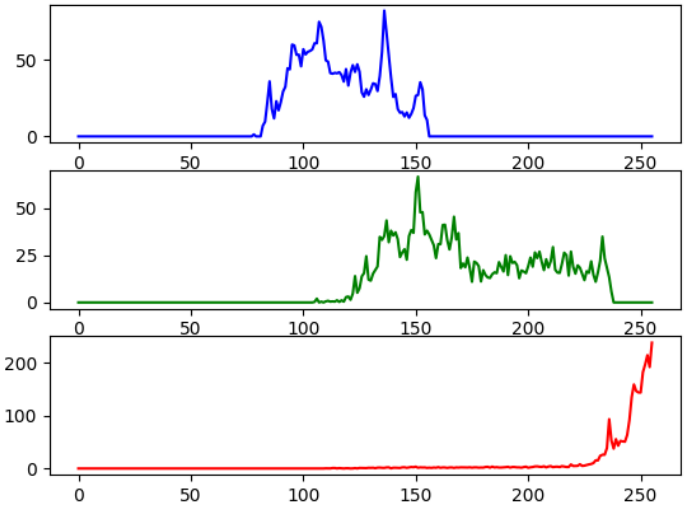
Once we were done with the data , we went on to the next step.Our next step was to generate an average histogram for each channel of the samples images.We created a function called hist , which would read the image and return the histogram values. We can't plot this histogram yet , since we want a average histogram in our case. We get all the histogram values for all the 7 cropped images , we store it and we add it together. Once we have summed the histogram values for all the images , we divide it by n , where n is the number of cropped images or the number of images we are taking the histogram values. We then plot that average histogram , using subplots , to make the picture more clear for analysis.

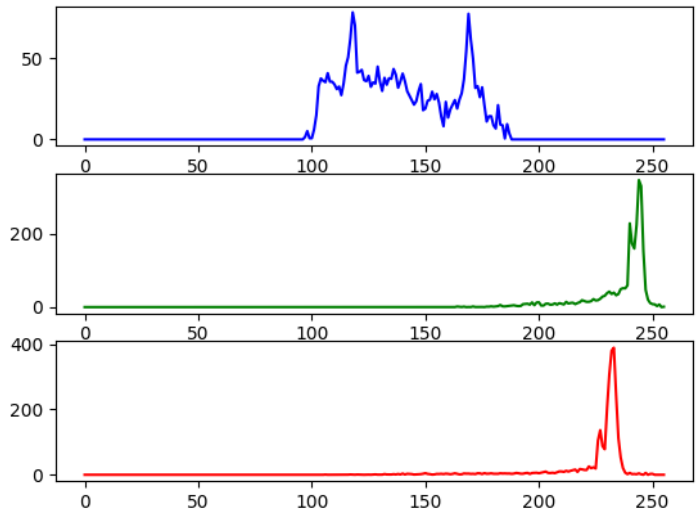
Once we run our code , we got three figures . Figure 1 gives you the average histogram for green buoy , Figure 2 gives you the average histogram for red buoy and the 3rd figure gives you the histogram for yellow buoy.

If you look at the 1st figure , you can see that the green intensities are dominant. Green has the peak at the higher intensities on the histogram. This shows us that the green color is dominant in the green buoy. We will be using this inference for detecting the buoys in the future steps.
Similarly , if you look at the figure 2 , you'll find the red color to be dominant. The red color has peak values after 240 , which forms a peak and thus will be helpful to detect the red buoy in the video.
Similarly , if you look at the 3rd figure , you'll notice that both red and green channels have peaks in the histogram after 200 and are dominant. This proves the fact that , we have to use both red and green channels for yellow buoy detection in the video , as also suggested by the project description.

Histogram gave us the idea , which channel intensities were dominant in which buoy and how the intensities varied across different channels for different colors.







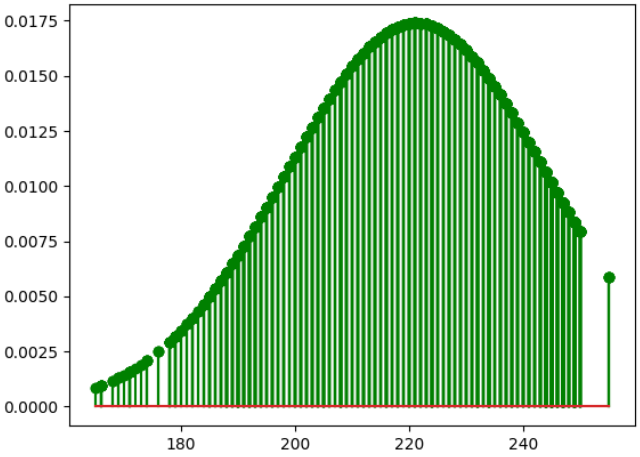
The above code can be found in a file named "**Histogram.py**" , attached with this report.

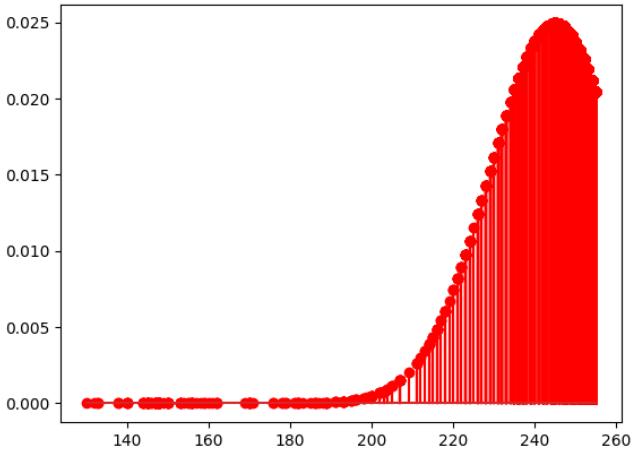
Our next step , was to create a model and segment the buoys using that model. Considering a 1-D Gaussian was used to model the color distribution of the buoys , we created a code to do the same.

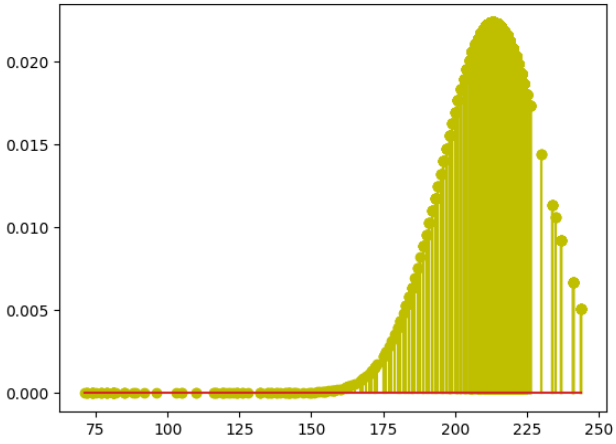
We created 2 functions , initially , which would take in the samples as input and give us the model parameters , which will be given as input to our 2nd function , I.e Gaussian1-D , which generates a Gaussian distribution for the samples.

Here we are calculating the mean and standard deviation of the samples and also their gaussian distribution. We will use this mean and standard deviation in the detection pipeline. For now , we are plotting the Gaussian distribution in the same color as that of the buoy.
Once you run the code , you'll get three figures again. Each figures represent a gaussian distribution for the buoys and can be compared to the histogram as well. You'll be able to see that the gaussian forms over the dominant parts of the color-space.

Figure 1 gives the gaussian distribution for green buoy's green channel , figure 2 gives you the gaussian distribution for red buoy's red channel , whereas figure 3 gives us the gaussian distribution for
yellow buoy's , green and red channel combined , since we know that the yellow buoy is formed using green and red color's . If we compare our gaussian distributions with histogram , we can clearly see how the gaussian is uniformly distributed over the dominated channel.



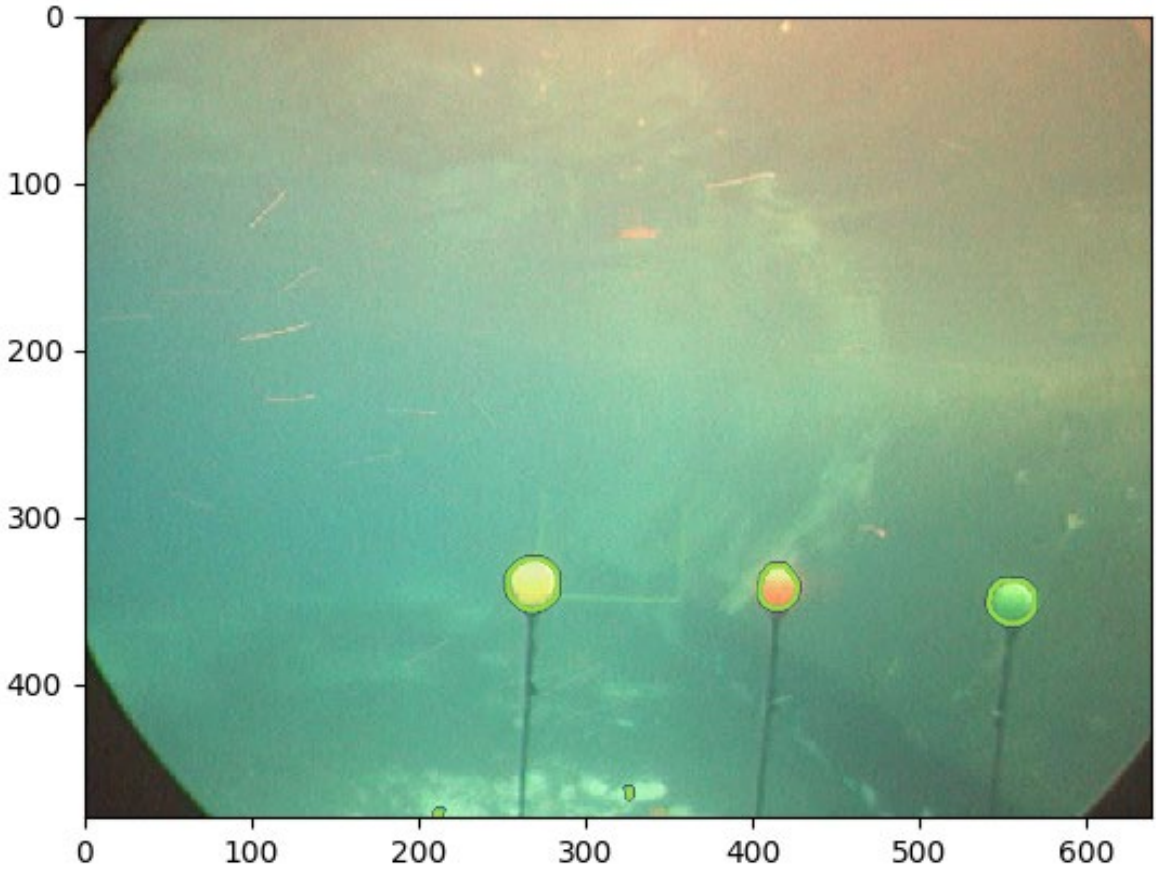


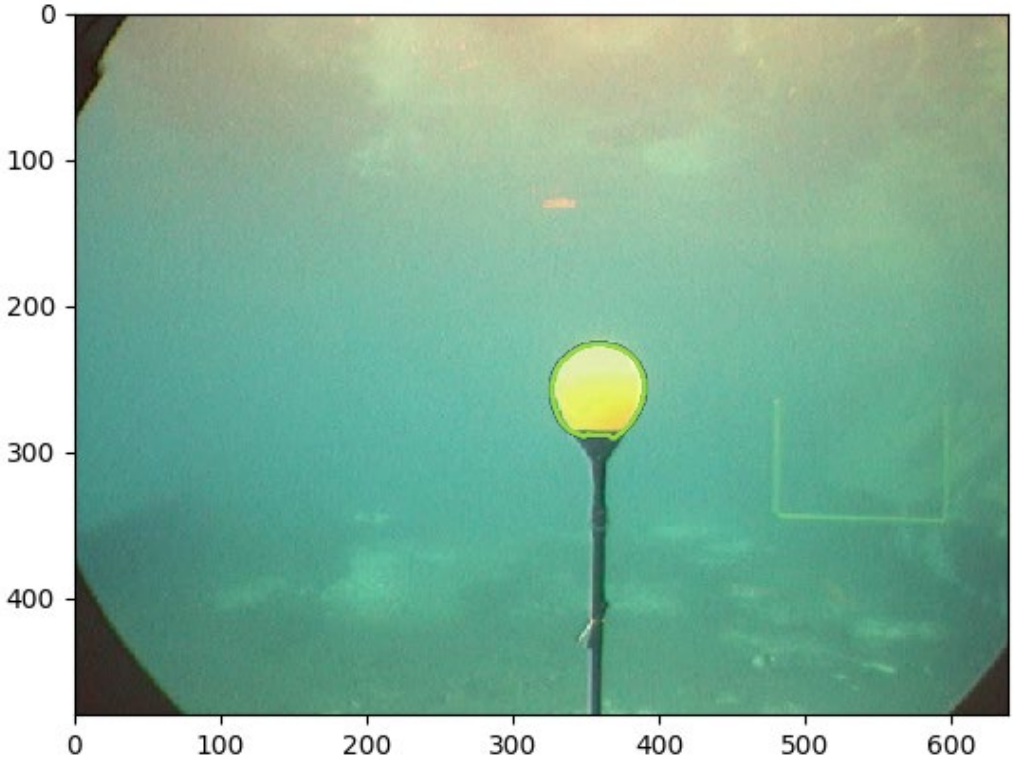


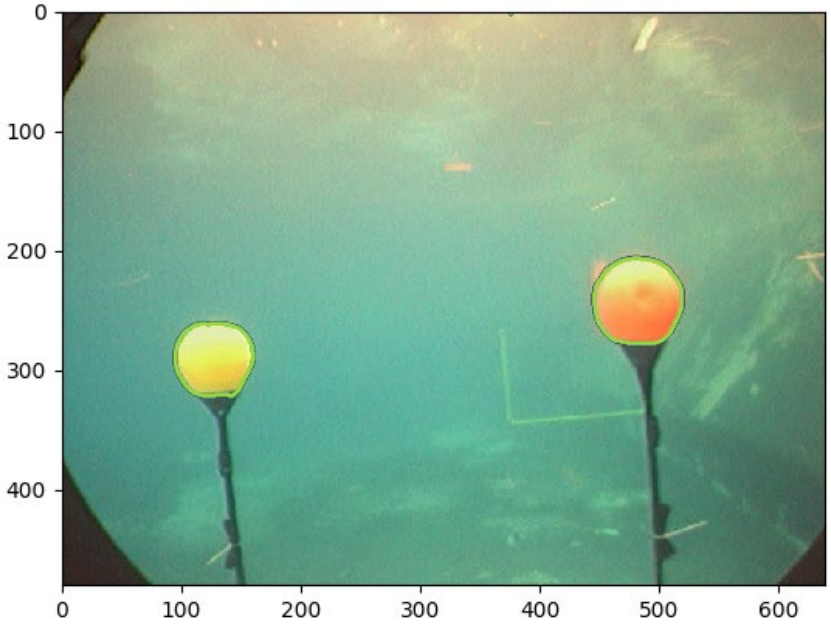
The above code can be found in the file titled '**1-D_Gaussian.py**'
Since , we have created a model , we have trained it , now it was the time to test it . So we wrote a code and we used the mean and standard deviation from previous file .

Our basic idea here , was to read the test frame , apply filters and morphological operation on the frame. Extract the channels and compute the Probability map using the 1-D gaussian function and mean and standard deviation generated from before. We then used the hit and trial method , to achieve a threshold for detection and then we did the OR operation b/w red and green probability true false values and got a final image , which detected all the three buoys. We compare the probability of the color being present in the buoy and save the true false value against that and plot those values.We use function like plt.contour to draw the contours on the image

We couldn't find a way to use the cv2 function for draw contour , since our final image is nothing but a bunch of 1's and 0's , whereas , the cv2 function requires input in the form of (m*n*p) and thus we had to use the plt.contour function for the same.
We were able to detect the contours properly for most of the test images and very less noise was seen in the output.




0
100
200
300
400
0
100
200
300
400
500
600


0
100
200
300
400
0
100
200
300
400
500
600

The code for the above said things could be found in the file titled '**Model_Test.py**'.This file returns all the 18 test images , with detected buoys.

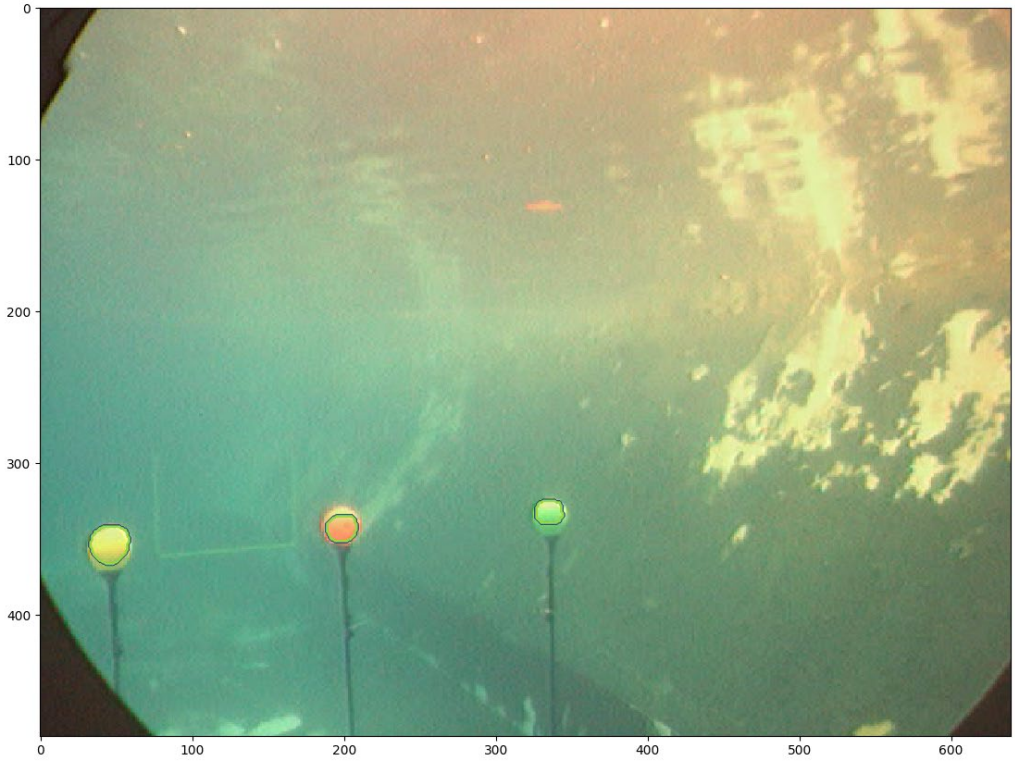
Once we were able to detect the buoys in the test images , we used the very same method to create the detection pipeline. Instead of passing the test images , the passed all the frames from the video. We plotted the contours using plt.contour in this case as well and we were able to detect the buoys using this pipeline which is generated as an output of this code file. We have used plt.animation to show the output.

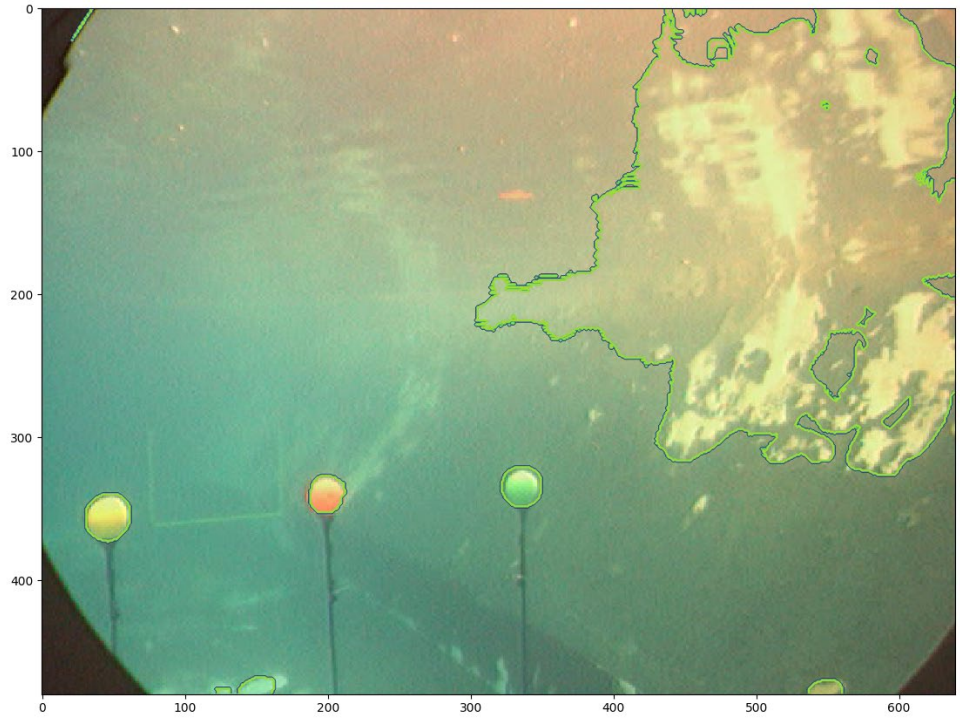
We tried one more method apart from the probability map. We used the mean and standard deviation of the Gaussian and tried thresholding the values of the 1-D Gaussian for detection as suggested by the project description.We got some outputs , but the output had a lot of noise and thus we tried the probability map strategy.

While working on the detection pipeline , initially the dataset we used , gave us some values of mean and standard deviation. When we used that model parameter for the detection , we were able to form the contour properly , i.e tighter contour on the buoys , but we also had noise in the video.

To eliminate the noise we changed the dataset again and we calculated the mean and standard deviation again. Using the new parameters we were able to elimiate the noise from the video , but we also compromised on the bounding of the countours on the buoys. With this new dataset and model parameter , we got a noise free output , it's just that the detection was not properly bounded.

The codes for the above could be find under the file name titles '**Detection.py**' , which gives us the noise free output , with buoys detected , whereas the file titled '**Detection_Noise.py**' , which gives us the proper bounded buoys with noise in the video.


0
100
200
300
400
0
100
200
300
400
500
600


0
100
200
300
400
0
100
200
300
400
500
600

To see the output videos , you just have to run the above files and the **animation** would start.

The problems we faced till this part of the project were:
1 Cropping the Dataset (Tried multiple methods as explained before.)
2. cv2 takes in the image as BGR , whereas matplotlib reads the image as RGB , so everytime we had to use any of these functions , we had to convert the colorspace accordingly.

Gaussian Mixture Models and Maximum Likelihood Algorithms:

3.1 Expectation Maximization:

This part of the project demands us to write a program to implement the EM algorithm for a given number of clusters or gaussians to fit the data-set.
A GMM or a gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of finite number of Gaussian distributions with unknown parameters. EM is a well defined algorithm designed to get around the problem of GMM(formed by unlabeled data) that the GMM does not know which points came from where. The overview of EM algorithm is somewhat like this -
The first step is to assume random components and for each point a probability of that point belonging to a particular gaussian or cluster. Then the parameters for a particular cluster get updated based on this soft probability value and are then used to compute the new gaussian. The process is iterated till a best fit is found that can either be found using the observation for the given set of points a best fit gaussian encompasses the entire set of points or a log-likelihood function.
The formulas and the steps for the above explained algo are as follows -

- Initialise the means, co variances and sizes of clusters
- Analyse the E-step and compute the following -

$$\gamma(z_{nk}) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_j)}$$

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

Where D is the dimension of the gaussians

- In the las step or the mstep the new means, covariances are updated for each cluster and a log likelihood function is checked for convergence. The formulas for te m-step to compute

the new or updated means, variances and fractions are as follows -

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k^{new})^T (x_n - \mu_k^{new})$$

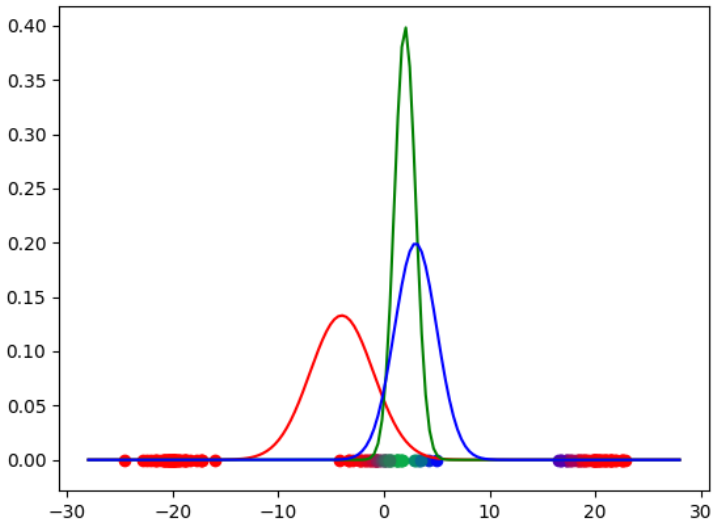
$$\pi_k^{new} = \frac{N_k}{N}$$

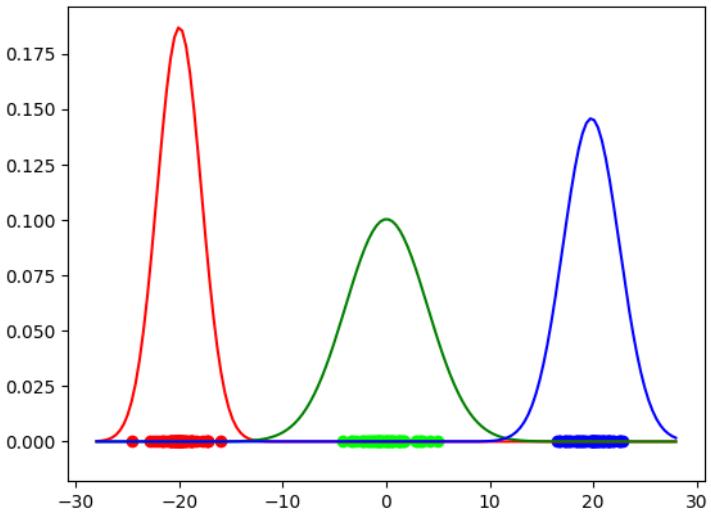
$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

And the log-likelihood function is given by,

$$ln(p(X|\mu, \Sigma, \pi)) = \sum_{n=1}^{N} ln\{\sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k)\}$$

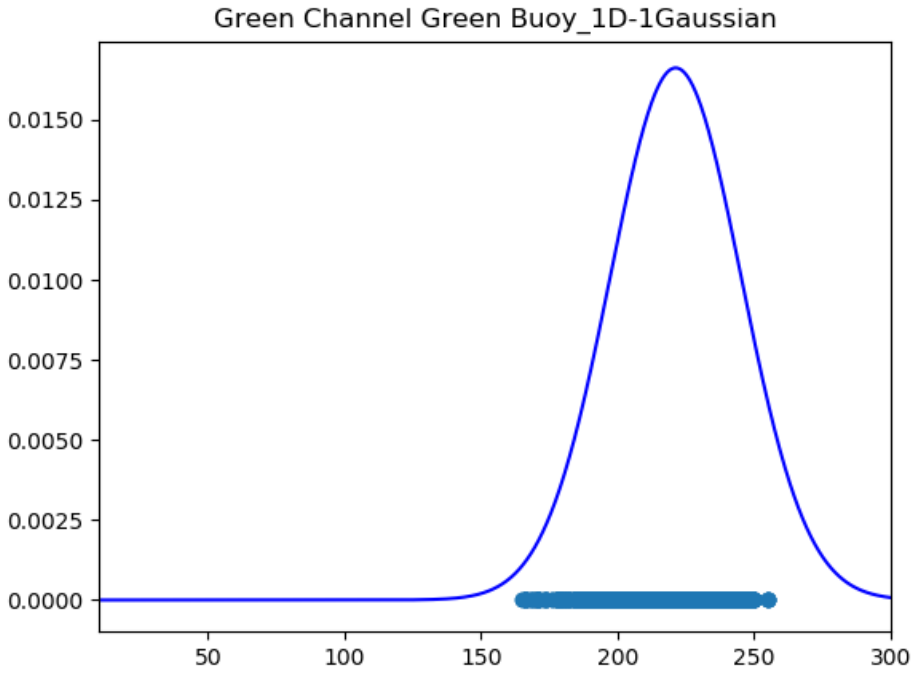
Using the steps shown above we have fitted 1, 3 and 4 1D gaussians to our data set with varying levels of accuracy observed in the end, out which 3 1D gaussians gave us the best fit.

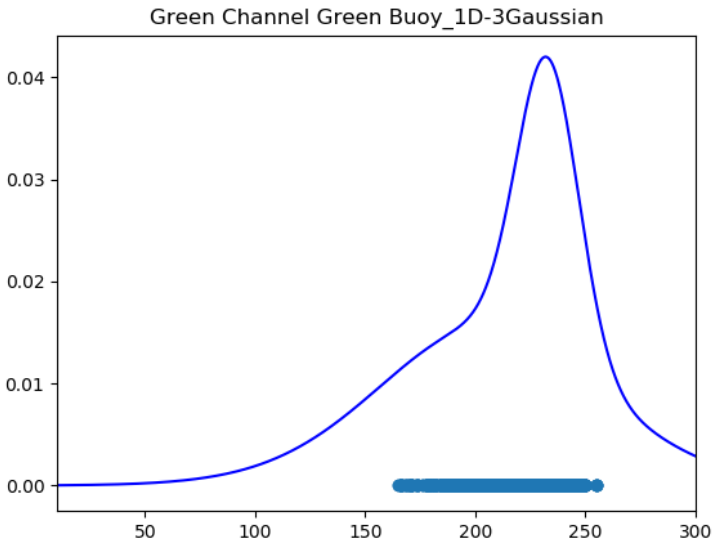


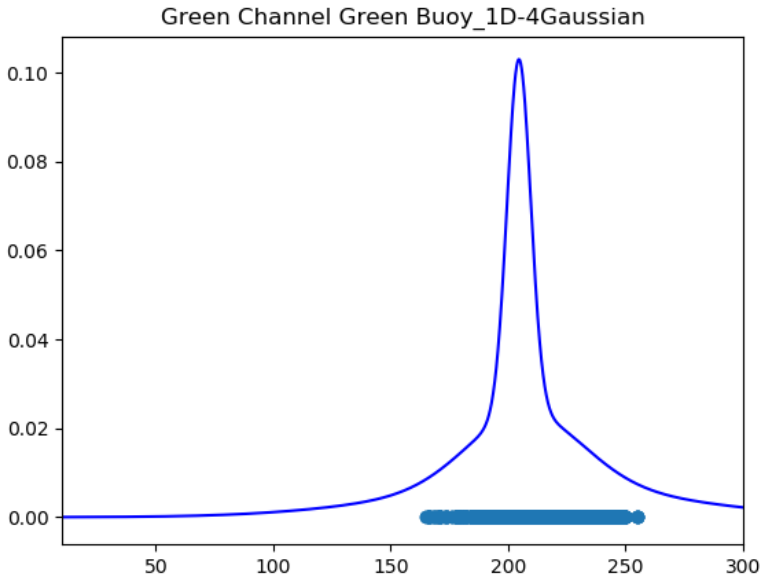


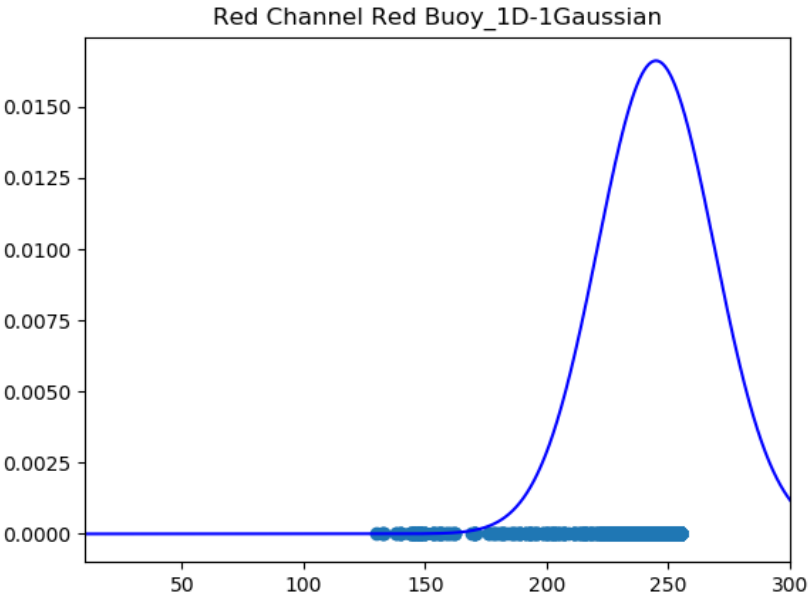
The code for the above discussed topic , can be found under the file name titled '**EM.py**'. It gives us 2 outputs.The 1st figure is gaussian for some random data , whereas the 2nd figure is the figure after gaussian fitting the data points and after the final iteration.

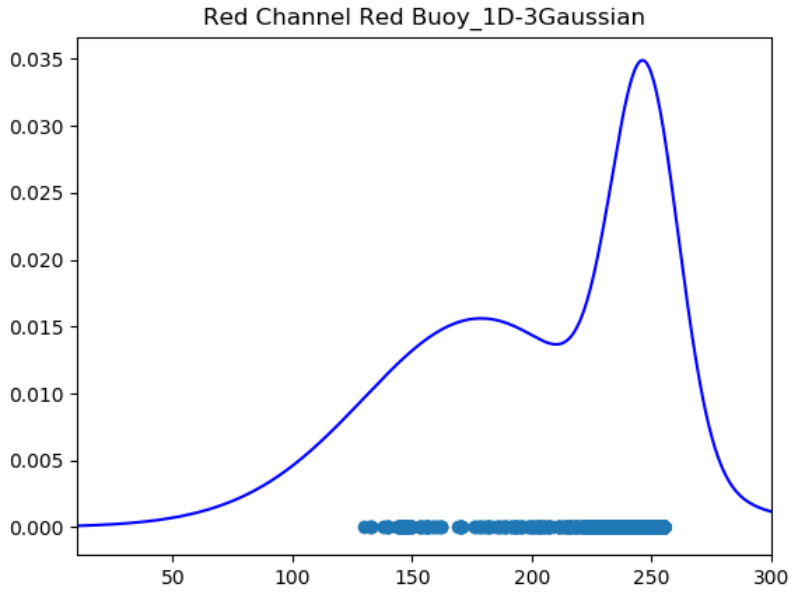
The mean and variance also has changed drastically. Our ground truth values for sigma were [3,1,2] whereas from this gaussian fitting the sigma we get is [2.13,3.9726,2.734] , whereas the mean was –[4,2,3] and the new mean was [-20.013,0.0244,19.819].

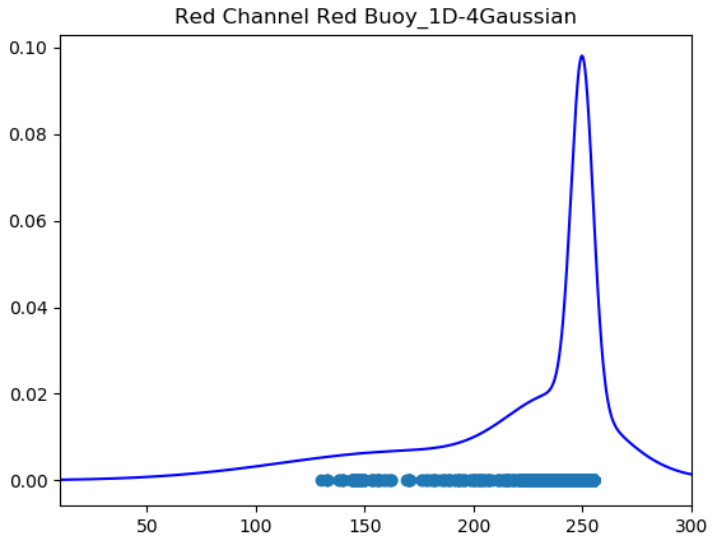


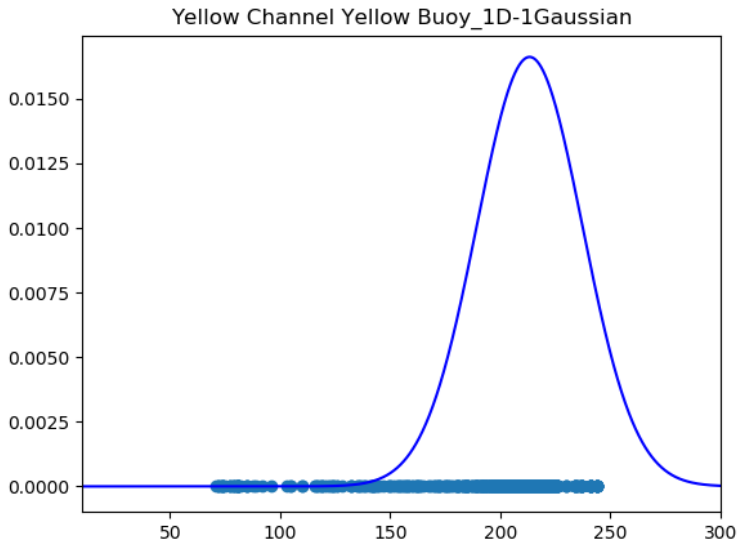

Green Channel Green Buoy_1D-3Gaussian
0.04
0.03
0.02
0.01
0.00
50
100
150
200
250
300

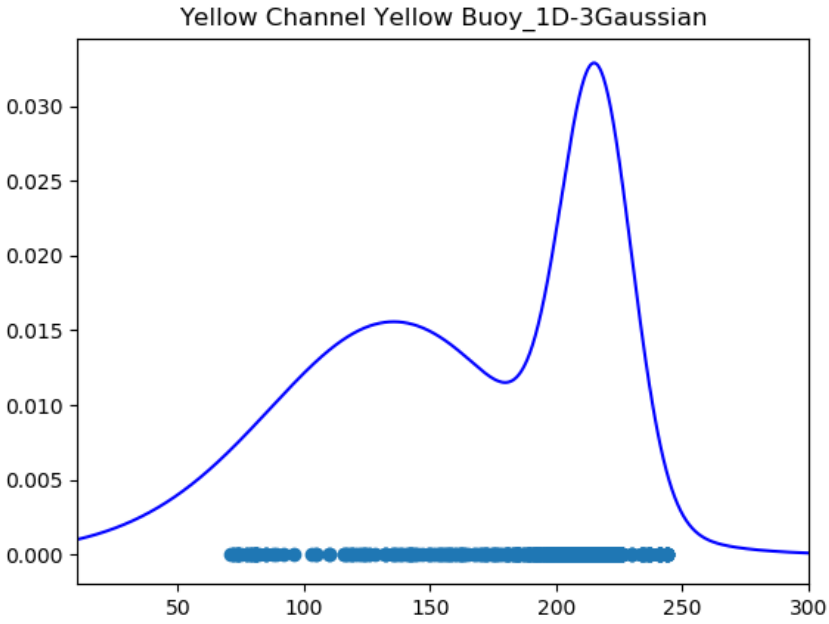

Green Channel Green Buoy_1D-4Gaussian
0.10
0.08
0.06
0.04
0.02
0.00
50
100
150
200
250
300

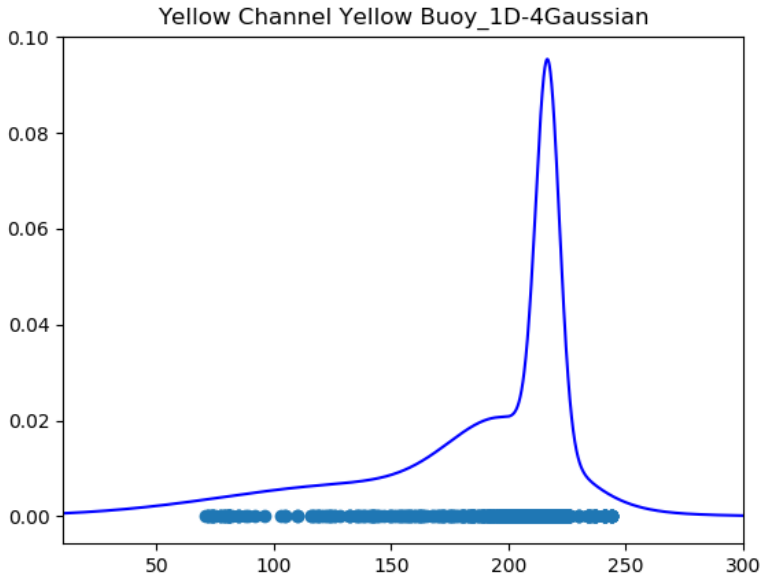

Red Channel Red Buoy_1D-1Gaussian
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
50
100
150
200
250
300


Red Channel Red Buoy_1D-3Gaussian
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
50
100
150
200
250
300


Red Channel Red Buoy_1D-4Gaussian
0.10
0.08
0.06
0.04
0.02
0.00
50
100
150
200
250
300


Yellow Channel Yellow Buoy_1D-1Gaussian
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
50
100
150
200
250
300





3.2 Learning color models

This part of the problem requires the use of clustering via GMM . Based on the color histograms generated in the previous sections the optimum number of gaussians and its dimensions must be decided.

The multivariate gaussian distribution is a generalization of the 1D gaussian distribution in higher dimensions. The multivariate normal distribution is often used to describe, at least approximately, any set of (possibly) correlated real-valued random variables each of which clusters around a mean value. It is given by the following equation,

$$N = \frac{1}{(2\pi^{n/2})\Sigma_c} * exp(-0.5(x_i - \mu_c)^T(\Sigma_c)^{-1}(x_i - \mu_c))$$

The multivariate gaussian can also be used to fit 1D gaussians on our dataset. However, based on the observation it was hard to decide the dimension of the gaussian to be used, let alone the number of gaussians. Thus, we decided to experiment with multiple 1D gaussians and observe which one fits the dataset accurately.
We tried fitting 1, 3 and 4 gaussians respectively, the outputs of which can be seen as follows -

Based on the shapes and how well the data points are enclosed by the combined pdf, the optimum combination for our dataset is 3 1-D gaussians.
We tried fitting 3D gaussians by changing the dat-set to a set of [R G B] intensities, but we were not able to incorporate it.

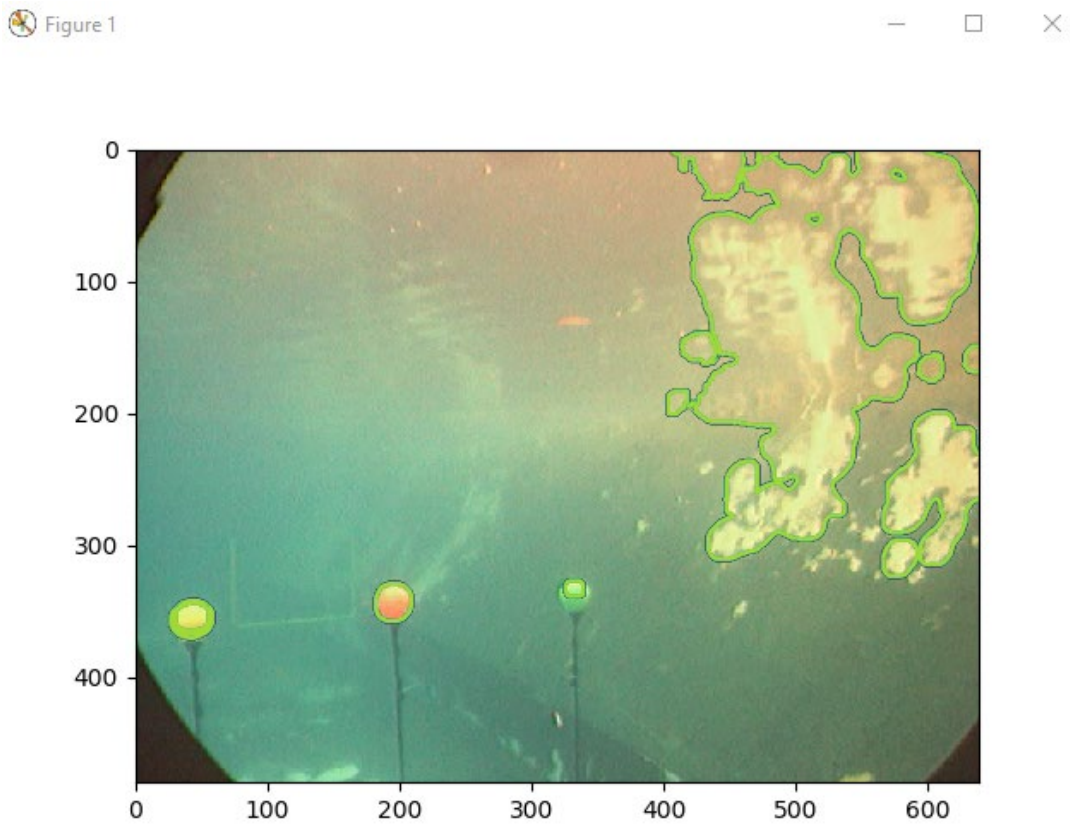


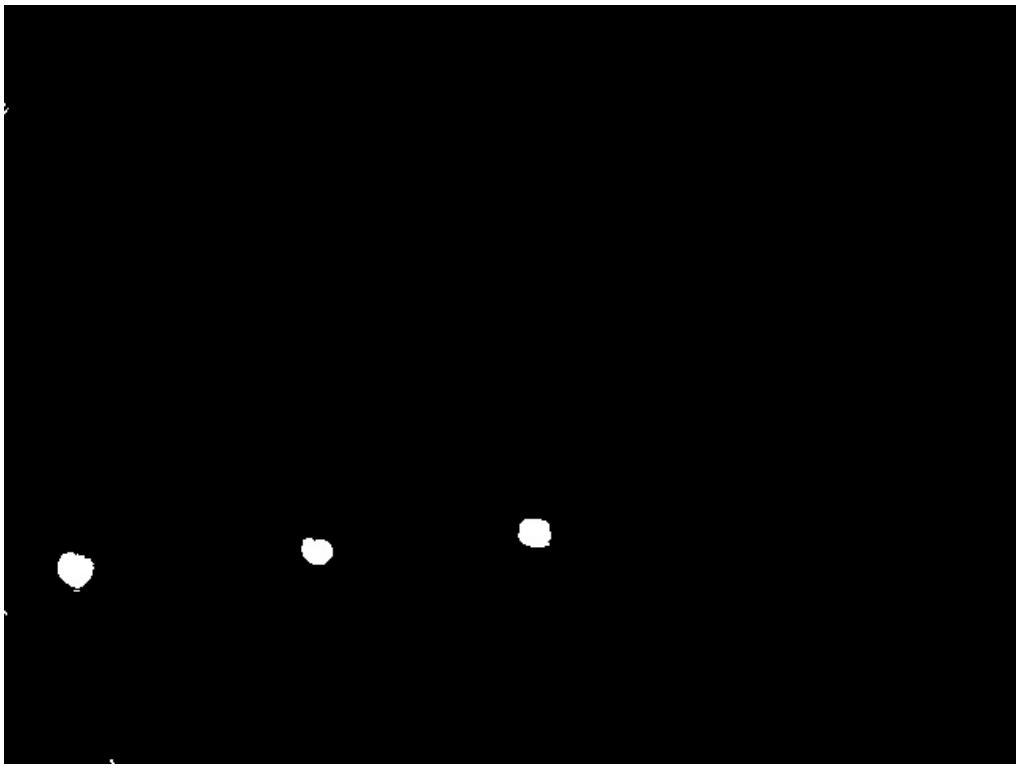
Figure 1 GMM 3 Gaussian Contour Fitting.

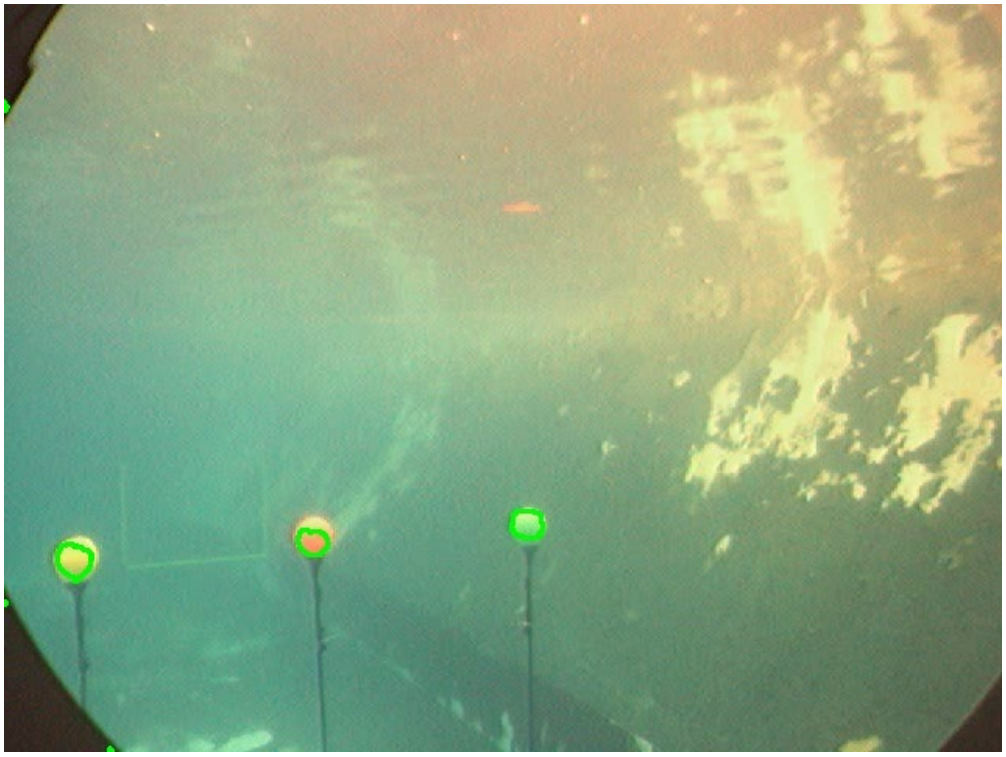
Further Analysis:

Yes , our current pipeline considers only RGB images , and yes , it may become inaccurate at some extents . The sole reason for that is the effects of lighting.The buoys have different pixel values , depending on how the light from the surface hits them and how the light varies.
A dark day , might generate more dark colors , whereas a sunny day , might give you a better toned down bright color , for the very same buoy. This results in the problem when you are trying to threshold the values for detection.

The other color-space representations , are more feasible in such scenarios.
For example , if we consider HSV space , where H stands for Hue , S stands for saturation and V stands for value(brightness). The use of HSV would be more adaptive to the changes in the images such as lighting , shadows and similar noises. The hue and saturation remains stable , despite the changes in lighting , which only affects the V part of this colorspace. Using HSV color space , would definitely give us better results.

We tried to implement this , so we wrote a code for it . We wrote the code , for the very 1st frame ,since it had the maximum lighting and thus noise. The same approach can be applied to the whole video as well. This was our very first attempt , with little bit of tuning the color range. If we would have worked on this instead of the RGB color space , then we could have got more better results in shorter span of time.





The code for this could be found in the file name titled '**hsv.py**'.

Similarly , we could have worked on the L*a*b colorspace , which takes in Lightness in consideration , as well as the a & b , represents the green-red and blue-yellow color components respectively.
There are other color spaces such as YIQ (Luminance-Hue-Saturation) and RGBY (this could have been better in our case since RGB and Yellow , would have helped us in the Yellow Buoy detection.)

Apart from RGB color space , we can work on various other colorspace , as discussed above and would have got a better results.