ENPM 673 Project 3

Rene Jacques Zachary Zimits

April 2019

1 Data Preparation

1.0.1 Buoy Cropping

The first step for this project was to crop each buoy out of the larger frame in order to create training and testing data sets. This was done using event handling in python. Each frame was plotted using matPlotLib. Extracting mouse click events made it possible to look for three separate clicks, extract the (x, y) coordinates at the click location, and then calculate the radius and center of a circle defined by those three sets of coordinate pairs. Using this method it is possible to define a bounding circle for each buoy in each frame and crop the image down to contain only the buoy on a black background. This decreases the amount of extraneous data that needs to be processed for each image, decreasing the likelihood of false positives.

1.0.2 Average Color Histograms

The next step was to compute the average color histogram for each color channel for each buoy separately. This mean extracting the red, blue, and green color channels from each cropped buoy image, averaging the value of each image for each color channel, and then plotting these average values as histograms. The resulting graphs show which ranges of each color channel are most likely to indicate that a pixel belongs to a specific buoy.

2 Gaussian Mixture Models and Maximum Likelihood

2.1 Expectation Maximization

2.1.1 Test Data Generation

To generate data samples for Gaussian Mixture Model (GMM) fitting using Expectation Maximization (EM) the np.random.normal function was used. Since the goal was 3 data sets we set the values of μs and σs to be 50,125,210 and 40,15,30 respectively. This returned a number of points randomly distributed according to the input values. Concatenating all of these values together creates a graph with multiple peaks, to which Gaussians can be fit by running the values through the EM algorithm.

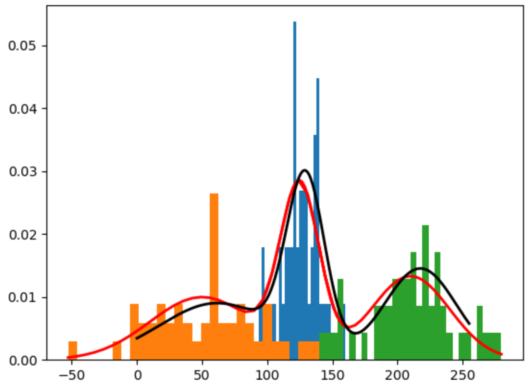

Figure 1: Probability Density Function

In the figure above the three different data sets can be seen in the histograms in three different colors. The red line represents the pdf using the given values and the red line represents the pdf with our calculated values of 61.56,129.44, 217.62 for mu and 44.29,16.65,27.52 for sigma

2.1.2 EM Implementation

The EM algorithm is implemented as follows. First the μ, σ, and mixture coefficient for each of the desired number of Gaussians to be found. Each of the μ values is initialized as the corresponding Gaussian number multiplied times 75, each of the σ values is initialized as the corresponding Gaussian number multiplied times 50, and each of the mixture coefficients is initialized as 0.5:

```
def init_params(num_gauss):
   mus = np.zeros(num_gauss)
   sigmas = np.zeros(num_gauss)
   mix = np.zeros(num_gauss)

   for i in range(num_gauss):
      mus[i] = (i+1)*75
      sigmas[i] = (i+1)*50
      mix[i] = 0.5

   return mus,sigmas,mix
```

These initialized parameters are then fed into the expectation function, which classifies the training data based on the probability that it should belong to a certain Gaussian:

```
def E(x,mu,sigma,mix):
   r = np.zeros([len(x),len(mu)])

   for i in range(len(x)):
      for k in range(len(mu)):
         den = 0
         for j in range(len(mu)):
            den += mix[j]*gauss(x[i],mu[j],sigma[j])
         r[i,k] = (mix[k]*gauss(x[i],mu[k],sigma[k]))/den

   return r
```

The outputs of the expectation function are then input into the maximization function, which updates the μ, σ, and mixture coefficient for each Gaussian.

```
def M(x,r,mu,sigma,mix):
   for k in range(len(r[1])):
      Nc = np.sum(r[:,k])
      mu[k] = mu_calc(x,r[:,k],Nc)
      sigma[k] = sigma_calc(x,mu[k],r[:,k],Nc)
      mix[k] = mix_calc(Nc,len(x))

   return mu,sigma,mix
```

Where Nc is the sum of all of the data within each separate Gaussian grouping. The new μ values are calculated using:

```
def mu_calc(x,r,Nc):
   return (1/Nc)*np.sum(r*x)
```

The new σ values are calcualted using:

```
def sigma_calc(x,mu,r,Nc):
   return np.sqrt((1/Nc)*np.sum(r*(x-mu)**2))
```

And the new mixture coefficients are simply calculated (where n is the number of data points) using:

```
def mix_calc(Nc,n):
   return Nc/n
```

All of these functions are brought together and looped through until the iteration limit has been reached, at which point the algorithm returns the calculated μs and s for each of the Gaussians:

```
def EM(x,num_iter,num_gauss):
   mu,sigma,mix = init_params(num_gauss)
   for i in range(num_iter):
      r = E(x,mu,sigma,mix)
      mu,sigma,mix = M(x,r,mu,sigma,mix)
   return mu,sigma,mix
```

2.1.3 GMM Fitting Using EM

This EM implementation can now be used to recover the model parameters used to generate the 1-D Gaussian data set described earlier. Graphing the recovered Gaussians against the data set and the Gaussians used to generate the data set shows that the recovered parameters are valid.

2.2 Learning Color Models

2.2.1 Number of Gaussians

2.2.2 Red Buoy

For the red buoy we chose to fit 2 gaussians for the red channel, 3 for blue, and 2 for green.

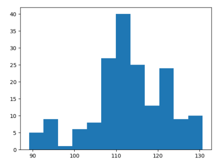

Figure 2: Blue Channel

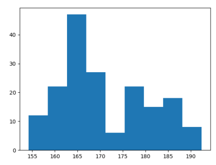

Figure 3: Green Channel

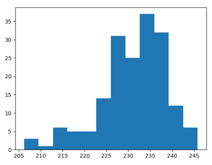

Figure 4: Red channel

2.2.3 Yellow Buoy

For the yellow buoy we chose to fit 2 gaussians for the red channel, 3 for blue, and 2 for green.

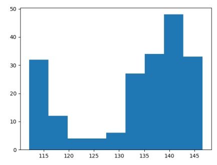

Figure 5: Blue channel

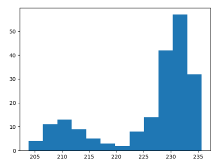

Figure 6: Green Channel

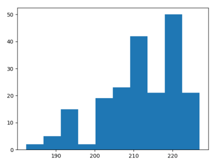

Figure 7: Red channel

2.2.4 Green Buoy

For the green buoy we chose to fit 2 gaussians for the red channel, 1 for blue, and 2 for green.

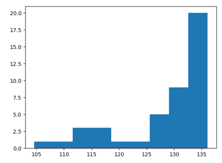
Figure 8: Blue channel

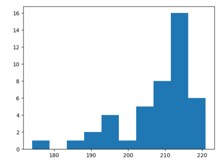
Figure 9: Green Channel

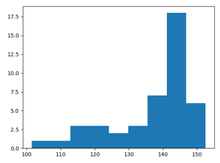
Figure 10: Red channel

2.2.5 Buoy Model Plots

2.2.6 Red Buoy

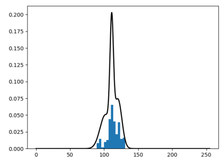
Figure 11: Blue Channel

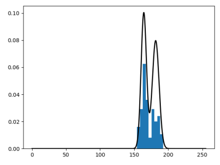
Figure 12: Green Channel

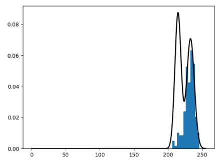
Figure 13: Red channel

2.2.7 Yellow Buoy

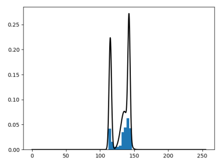
Figure 14: Blue channel

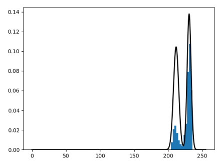
Figure 15: Green Channel

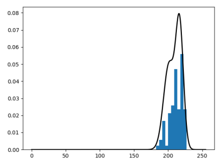
Figure 16: Red channel

2.2.8 Green Buoy

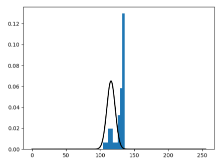
Figure 17: Blue channel

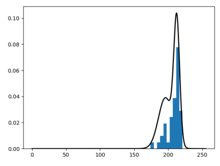
Figure 18: Green Channel

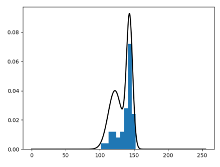
Figure 19: Red channel

3 Buoy Detection

3.1 Pipeline

The pipeline for buoy detection works as follows. For each frame of the video a probability graph image is generated that is the same size as the frame. This graph is used to store the probability that a pixel belongs in the buoy by setting its value to be its probability times 255, which means that brighter pixels have a higher probability of being buoys and darker pixels have a lower probability.

To increase accuracy every pixel is not checked individually. Instead the mean value of square groups of pixels is used to calculate that groups probability. So the frame is scanned by a window of size 10 pixels by 10 pixels and the probability for all of the pixels within that window is computed based on the mean value of the region.

For each window, the mean red, blue and green values are computed and stored separately. Then for each color Gaussian in the respective buoy model the probability that the region belongs to the buoy is computed for the red, green, and blue channels using the corresponding μs and σs. Depending which buoy is being tested, different weights can be applied to the different color channel's probabilities. This is so that for specific buoys more focus can be applied to a specific color channel, e.g. the red buoy and the red color channel. Each of the individual color channel probabilities are then added together so that only one probability remains for the group of pixels. This final probability is then multiplied with 255 to create a corresponding color value which is then applied to the pixel region in question. This is repeated for the entire frame, producing a probability graph represented by intensities of white for each buoy.

This probability graph is then processed to isolate the brightest pixels in the graph, which should be the most likely to be in the buoy. This is done by first initializing a new empty image that is the same size as the probability graph. This image is where the brightest pixels will be stored for future processing. The graph is then processed in a loop that stores all of the x and y positions where the pixel intensities are above a certain threshold. This threshold starts at 0% intensity and grows towards 100% intensity. So as the loop continues the amount of pixels stored will decrease. If the the number of pixels that are above the threshold reaches the desired number (e.g. 10 pixels left) then those pixel values are set to 255 in the new brightest pixels graph and then the loop will be broken out of.

Due to the colors in the original video, it is possible that some of the brightest pixels in the probability graph will be false positives. These false positives could be anywhere in the image, which will throw off any buoy location detection. To remedy this, cluster detection is performed on the image of the brightest pixels. This is a basic cluster detection since the number of data points that need to be processed is small. The cluster detection works by checking if the straight line distance between a bright point and every other white point in the image is less than a threshold value. If this is the case then the points are added to a temporary storage list. After a point is checked against every other point the length of the cluster list is checked with the length of the storage list. If the length of the storage list is greater than the length of the cluster list then the cluster list gets set as the storage list. This way only the largest cluster will be returned once every point has been checked against every other point. The centroid and radius of the cluster are then calculated in order to find the location and a value (the radius) that can be used to create a region of interest around the cluster. The region of interest within the original unprocessed image is then processed again using the methods described above with different thresholds in order to obtain more of the pixels that belong to the buoys and therefore obtain a tighter contour fit.

3.2 Bounding Contours

Once the region of interest in the image has been processed again the center and radius of a bounding circle are calculated using cv2.minEnclosingCircle. The circle defined by these parameters is then drawn on each buoy in the original image to visualize the buoy tracking.

3.3 Results

The buoy fitting works well for most of the input video, as seen in Figures 20 and 21. (Video clips of the buoy fitting can be seen by following this link: https://drive.google.com/open?id=1B74v-04E9FFo2aLzhXXFB1zVIYXmNT1X)

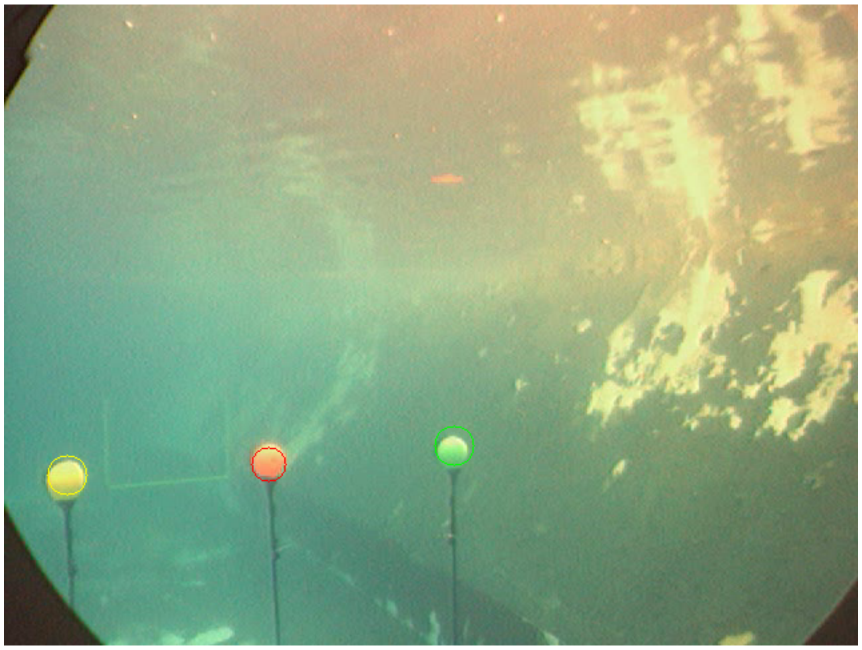

Figure 20: Contours Fit to Each Buoy

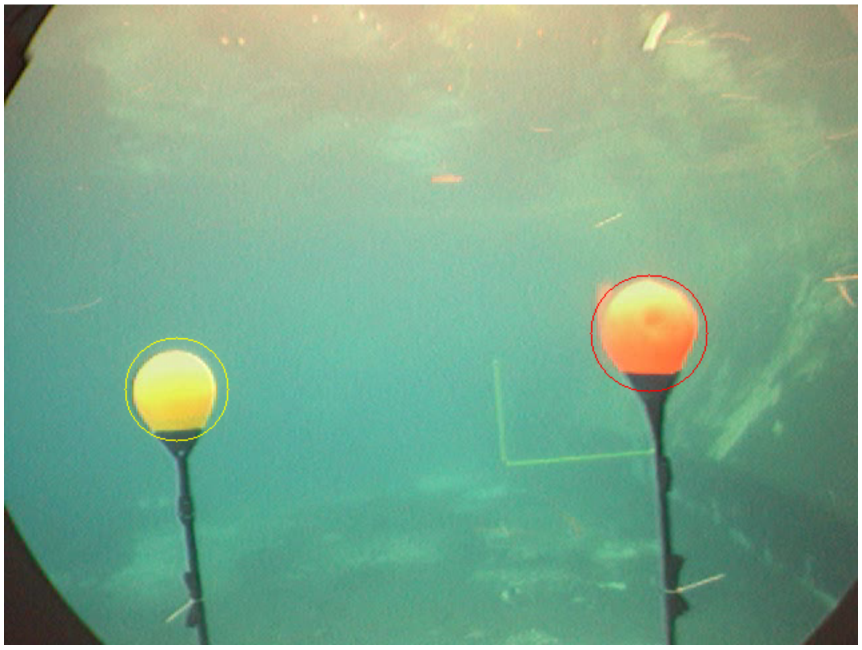

Figure 21: Yellow and Orange Contours Fit to Buoy

The enclosing circle does not always line up perfectly with the buoys. This is due to the shapes of the buoys not being perfect circles and is also thrown off by the lighting on the buoy in each frame. The technique used to locate the buoys can sometimes be confused by brighter false positive pixels in the image that can change the radius and positioning of the bounding circle as seen in Figure 22.

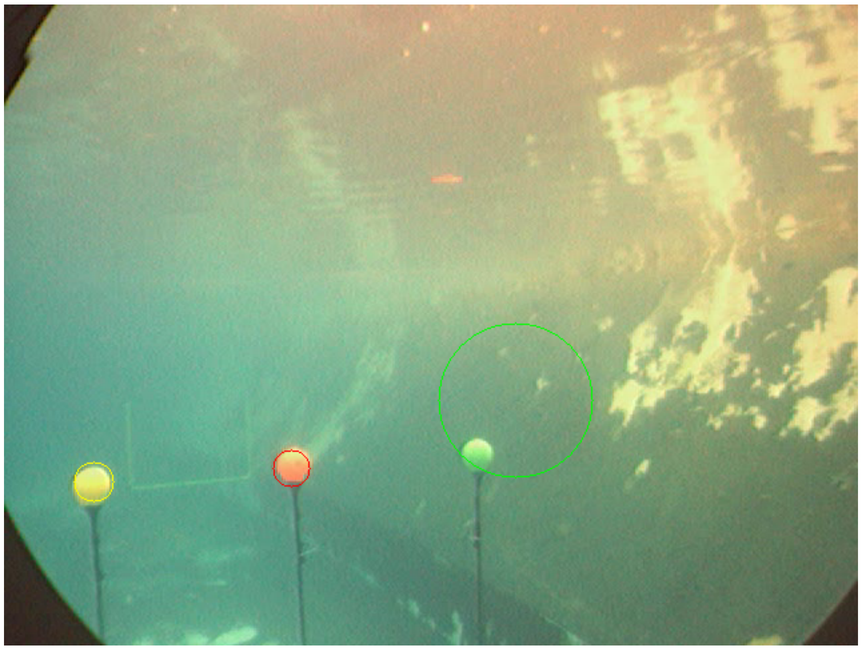

Figure 22: Green Contour Fit Using False Positive Green Buoy Points

It is also possible for these false positives to occur when the buoy being searched for is no longer in frame. For example, the green buoy leaves the video around a quarter of the way through and does not return. However several frames return false positives for the green buoy which results in a contour being fit to an area that does no contain any buoys. This is mainly due to the small training size available for the green buoy, as it has a lot less frames to be trained on then the other two buoys as well as the fact that the green buoy matches the background more than the other two buoys. Two examples of green buoy false positives can be seen in Figures 23 and 24.

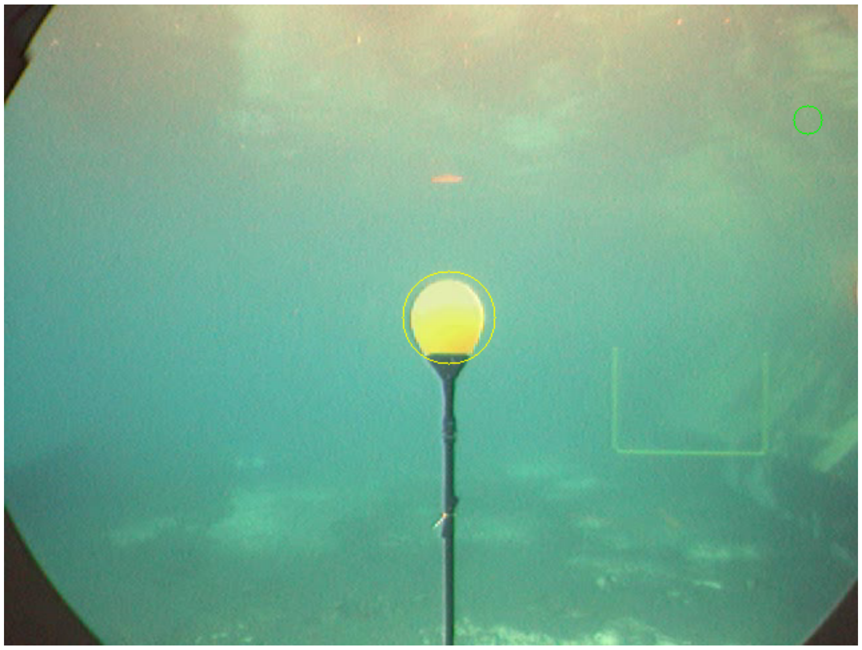

Figure 23: Yellow Contour Fit to Buoy with Green False Positive

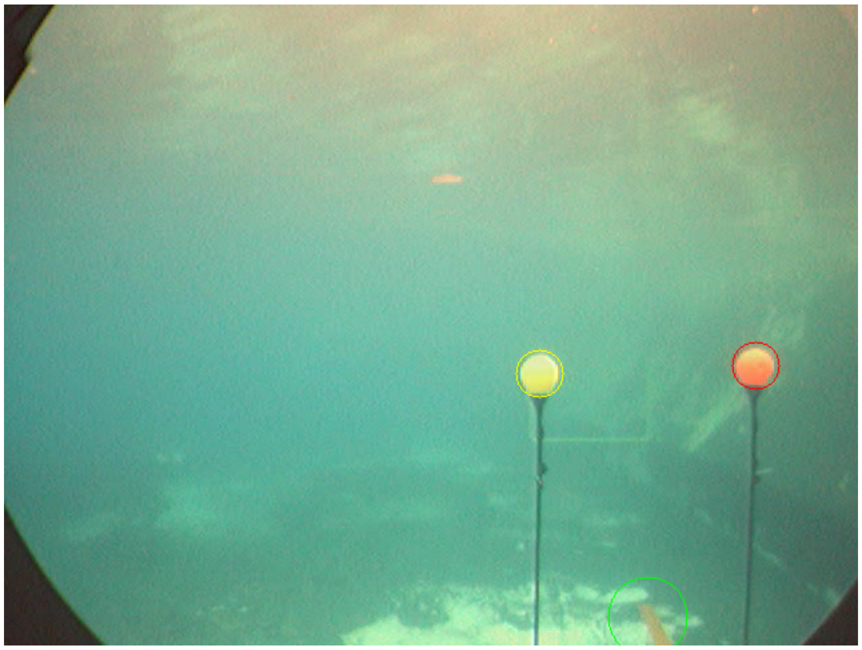

Figure 24: Yellow and Orange Contours Fit to Buoys with Green False Positive on Orange Debris

4 Further Analysis

While our project only analysed the RGB color space for these images future work could be done to develop other models based on different color spaces such as HSV or LAB. Colors react to different color spaces differently so while the blue background might read very highly in probability using an RGB model if one of more channels with switched or added from another color space a larger gradient might be detected.