ENPM673 Project 3

Justin Albrecht and Brian Bock

Spring 2020

Table of Contents

1 Introduction

This project uses a Gaussian Mixture Model to probabilistically learn and then detect colors. The sample video, `detectbouy.avi` (viewable in the media folder) shows the view from an underwater camera in a large pool with 3 bouys of different colors - yellow, orange, and green.

2 Data Preparation

2.1 Region of Interest

We manually selected the region of interest (ROI) for each bouy for each frame of the video (Figure 1).

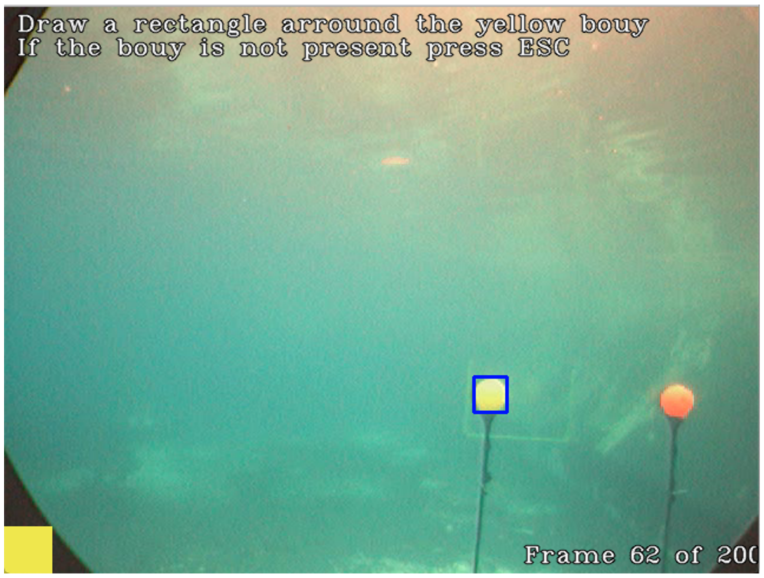
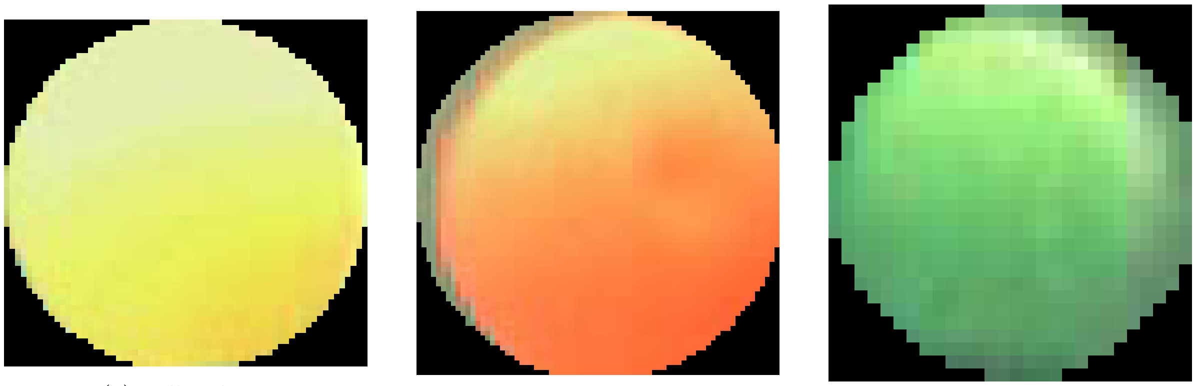

Draw a rectangle arround the yellow bouy
If the bouy is not present press ESC
Frame 62 of 200

Figure 1: Region of Interest Selection

You can view a portion of this process here: https://www.youtube.com/watch?v=gAHzZghxUaw
The frame is copied and cropped to the ROI, and then masked in an elliptical shape with major and minor axes equal to the width and height of the rectangular ROI. The elliptical masking eliminates the water captured in the rectangular ROI around the spherical buoy. This rounded image (Figure 2) is then saved into it's respective color folder. This process was repeated for each buoy in every frame. The yellow buoy is visible throughout the entire video, while the orange buoy is off-screen for a few frames, and the green buoy is only visible for the first 43 frames.



(a) Yellow buoy (b) Orange buoy (c) Green buoy

Figure 2: Sample result images from the ROI process

2.2 Division into Training and Testing Data

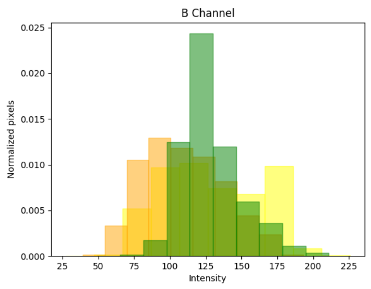
The rounded buoy images are then randomly divided into Training and Testing data folders such that the ratio of Training to Testing images (for each color) is approximately 70:30. The breakdowns for each color are listed below.

Color	# of Training Images	# of Testing Images
Yellow	140	60
Orange	124	54
Green	31	14

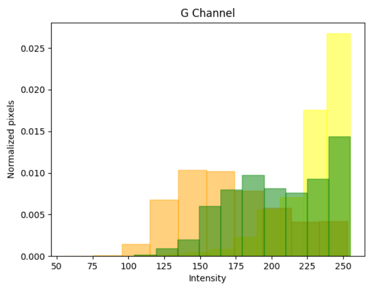
2.3 Reading in image data for GMM

Notes: Recognizing that our data might be more distinct in other color spaces, we built-in the option to work in either BGR or HSV color space, which can be changed via a simple toggle. For the sake of brevity, we'll be referring to image color channels as just BGR (regardless of the mode) so that we don't need to refer to both BGR and HSV. The steps are equally applicable for each color space.

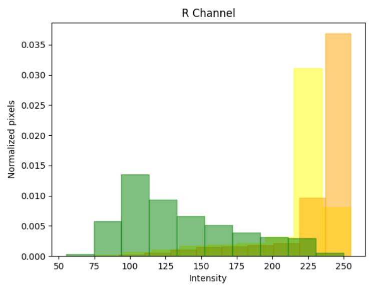
2.4 Image Histograms and Scatter Plots



(a) Blue Channel Histogram

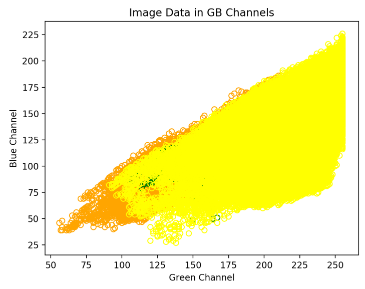


(b) Green Channel Histogram

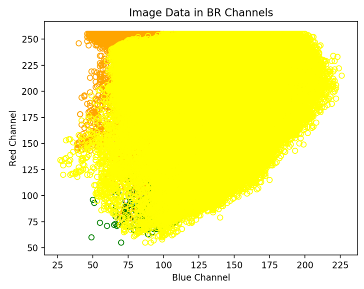


(c) Red Channel Histogram

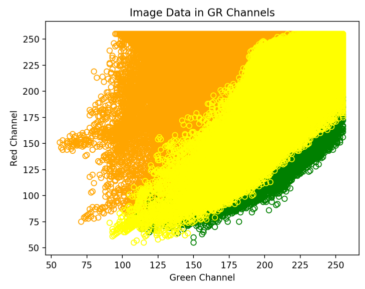
Figure 3: Pixel Intensity Histograms in each color channel for each color buoy



(a) Blue Green channel scatter plot



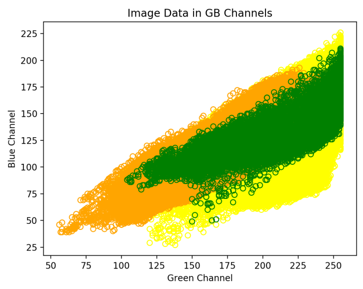
(b) Blue Red channel scatter plot



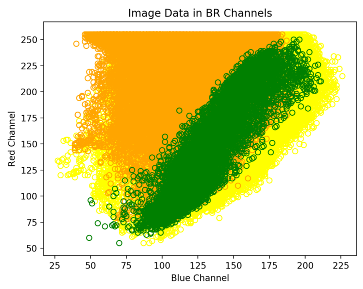
(c) Green Red channel scatter plot

Figure 4: Scatter Plots for each channel combination

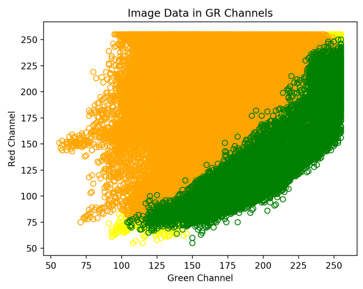
The data is dense, and the order we plot the colors affects the visibility of the data. We can switch the order and re-plot to get a better sense of the data.



(a) Blue Green channel scatter plot



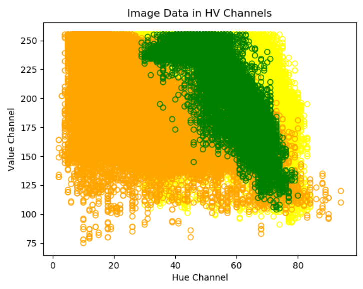
(b) Blue Red channel scatter plot



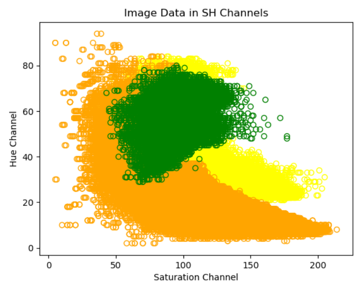
(c) Green Red channel scatter plot

Figure 5: Scatter Plots for each channel combination

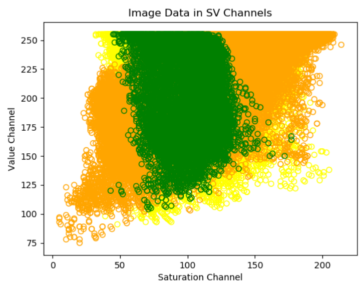
From Figures 4 and 5, it is clear that there is tremendous overlap in our data points in each channel combination. We can also look at our data in HSV color space.



(a) Hue Value channel scatter plot



(b) Saturation Hue channel scatter plot



(c) Saturation Value channel scatter plot

Figure 6: Scatter Plots for each channel combination

Again, the data is pretty muddled. Remember that Python can't see the colors we've plotted these scatter plots with (the buoy colors) - those are just for our own visualization.
We used channels (G,R) for identifying the orange and yellow buoys, and then (B,G) for the green buoy. We experimentally determined that this combination of channels produced the best results. Future iterations of this project might improve green's performance with a single channel (G) Gaussian, or with Guassians spanning all 3 color channels.

3 Learning color models

This work is based on the mathematics demonstrated in the following sources: [1–10]

3.1 Gaussian Mixture Modeling

The goal of this step is to generate a model that can be used to determine whether a pixel with given BGR values is likely to belong to the same color set as one of the buoys. We can accomplish this by using mixed Gaussian distributions. The mixture modeling allows us to generate K Gaussian distributions and combine them into a single density function. Each of the distributions is also weighted depending on it's importance in describing the overall data set.
For our data we decided to use two channels for each color group. So our model will be described as a combination of K multivariate distributions where each distribution will have the form:

$$\mathcal{N}(x_i|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)|\Sigma|}} \exp\Bigg(-\frac{1}{2}(x_i - \mu_k)^T \Sigma^{-1}(x_i - \mu_k)\Bigg) \tag{3.1}$$

Σ_k is the covariance matrix and μ_k is a vector for the mean in each channel.
The combined distribution is:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k) \tag{3.2}$$

where π_k is the corresponding mixture weight for that distribution.

4 Implementing Expectation Maximization Algorithm

In order to find values that explain our K number of 2-D Gaussians for each color we used the Expectation Maximization Algorithm. This algorithm employs an iterative approach to gradually increase how well the model fits the data.

4.1 Initializing starting values

The training data set for each color is comprised of a $2 \times n$ matrix, which represents the n pixels in all of the training images, and two of the color channels. The channels we use depends on the color we are training on. For example we use the G and R channels for orange but the G and B channels for green.
Before we can start the E and M steps we need to choose appropriate starting values.

- For the mean we divide the matrix into K sections, and then take the mean of each channel in the section. This results in K vectors

$$\mu_k = \begin{bmatrix} |x_k[channel1]| \\ \\ |x_k[channel2]| \end{bmatrix} \tag{4.1}$$

- For the covariance matrix we use a scaled identity matrix.

$$\Sigma_k = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \tag{4.2}$$

- For the mixture weights, we assume that each mixture contributes equally to each data point.

$$\pi_k = \frac{1}{K} \tag{4.3}$$

4.2 Expectation (E-Step)

For the E-step we calculate the responsibility matrix. This is a $n \times K$ matrix, where r_{ik} describes the amount that a pixel, x_i, belongs to the K^{th} cluster.

$$r_k(x_i) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{k=0}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)} \tag{4.4}$$

To simplify notation, we'll define N_k:

$$N_K = \sum_{i=0}^{n} r_{ik} \tag{4.5}$$

4.3 Maximization (M-Step)

For the M-step we compute new values based on the responsibilities matrix. The formulas for the new values are as follows:

$$\mu_k = \frac{\sum_{i=0}^{n} r_{ik} x_i}{N_k} \tag{4.6}$$

$$\Sigma_k = \frac{\sum_{i=0}^{n} r_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{N_k} \tag{4.7}$$

$$\pi_k = \frac{N_k}{n} \tag{4.8}$$

To test the fit of the model we can calculate the log-likelihood. This number will increase as the model better fits the data.

$$\text{log likelihood} = \sum_{i=0}^{n} \ln\Big(\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)\Big) \tag{4.9}$$

As the program iterates, the change in log likelihood converges towards zero. The program stops when the difference in log likelihood between iterations has dropped below 5.

5 Buoy Detection

5.1 Determine Probability Thresholds

We use our pre-trained data and run our Test images through it. Our `determineThresholds` function runs our Test images through our Trained Gaussians, and produces the probability that each pixel belongs to those Gaussians. The probabilities are summed over each of our K Gaussians. By sorting the list of these probabilities we can determine a minimum threshold that will describe a pre-determined percentage of the test data. By increasing the percentage from the test data we can improve our recognition in the buoy but we will also have move false positives. We found that using a threshold that allowed 60% of the training data to be classified was a good balance for false positives.

5.2 Color Segmentation

We return to looking at the entire original video, one frame at a time. We convert the frame into a flat 2 channel array, to make it easier to work with. We then create a new black image with the same dimensions as the original frame for each color, which will be the canvas on which we draw new pixel. If any pixel in the original frame meets our buoy color thresh criteria, that pixel coordinate in the black image is colored to match the buoy's color (Figure 7). The first few frames of the video are very noisy with the pool sidewall closely matching several buoy colors.

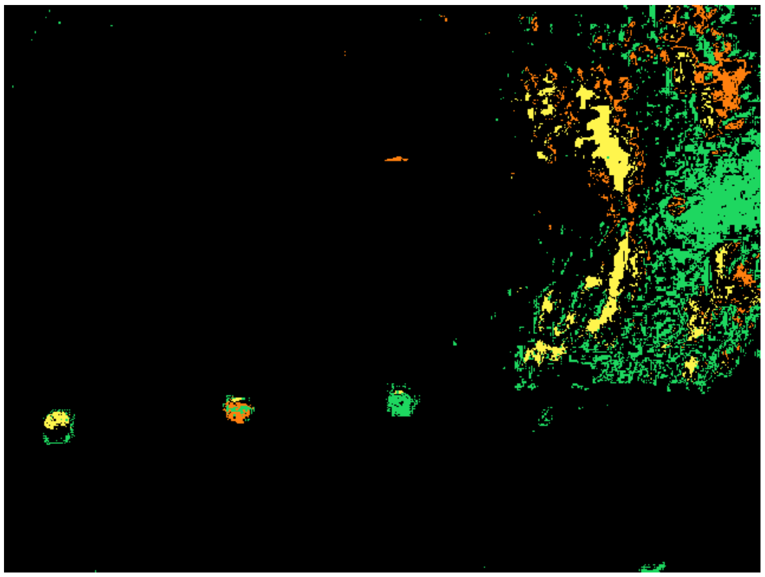


Figure 7: New frame with just Gaussian determined pixels colored

5.3 Contour Detection

Take our new image (Figure 7) and create a binary image where any previously colored (non black) pixel becomes white (Figure 8).

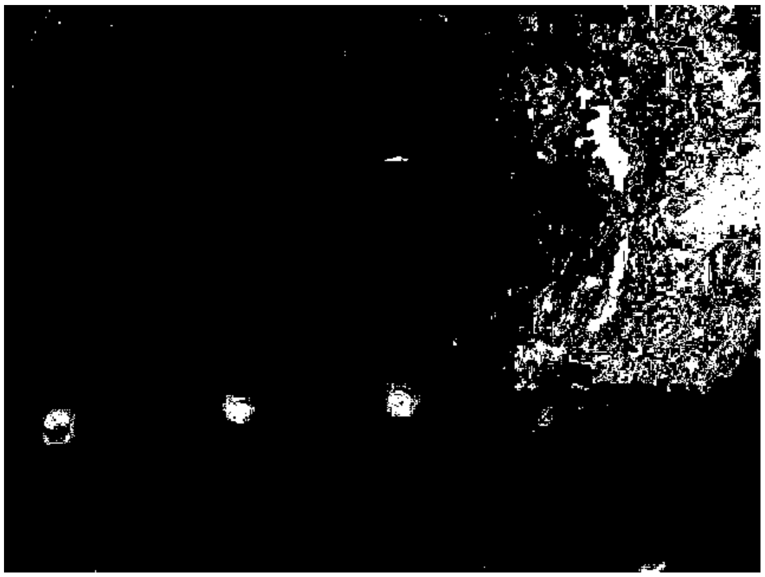


Figure 8: Binary Frame

To reduce noise, we apply a small Gaussian blur on this image and then run `cv2.findContours`, which returns us all of the contours in the image (Figure 9).

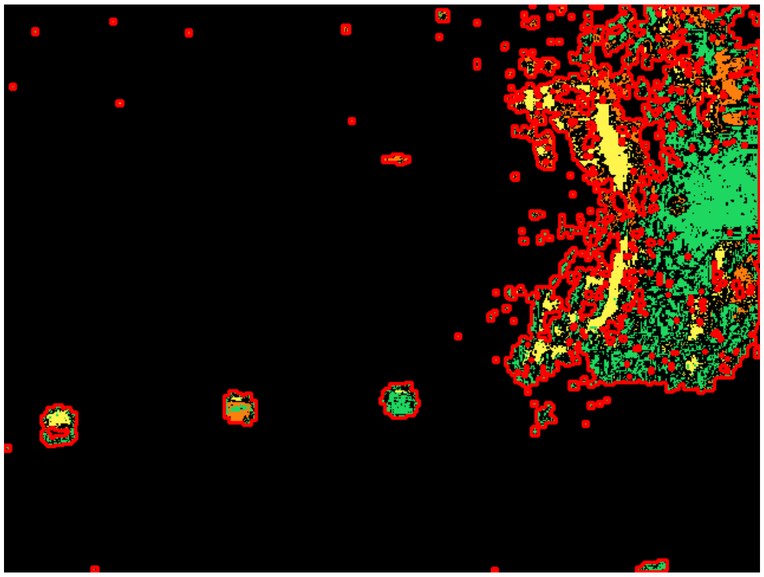


Figure 9: Initial Contours

Many of these contours are tiny and erroneous, so we sort our contours and only keep the largest 8 (Figure 10)

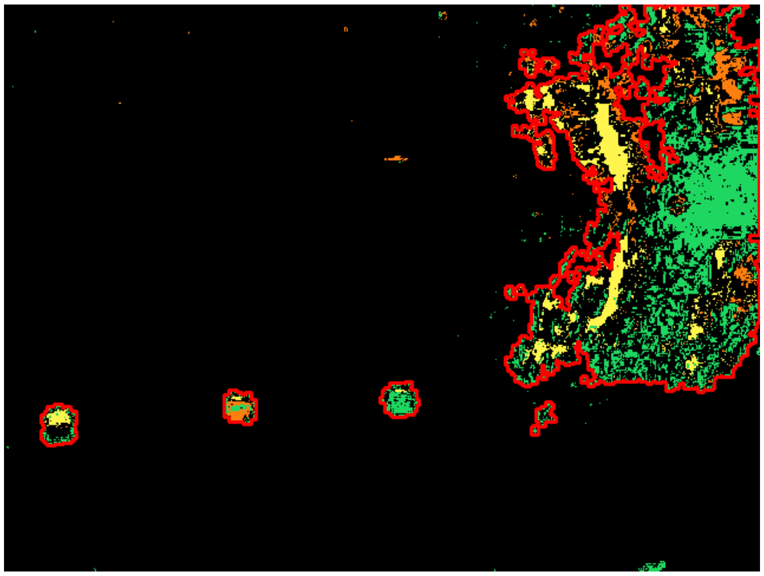


Figure 10: 8 Largest Contours

We use `cv2.minEnclosingCircle` to find the center and radius of the minimum enclosing circle for each contour. We know that our buoys are mostly circular, and so the area enclosed by the contour around a buoy should be comparable to the area defined by that minimum enclosing circle. Any contour whose area differs from it's min enclosing circle area by less than an experimentally defined threshold is likely a buoy. We then identify the color contained within that circle, and use it to define the color of the circular contour we draw (Figure 11).

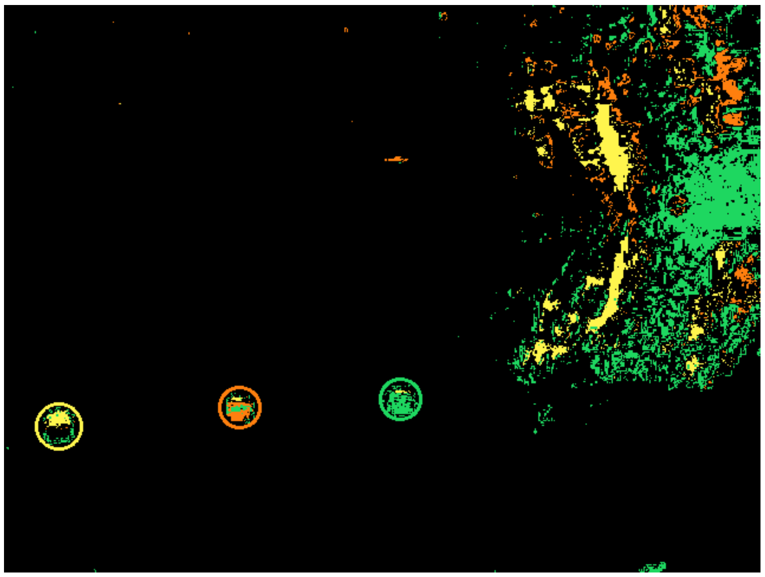


Figure 11: Buoys Traced with Colored Circles

You have the option to output the data with the rough drawn Gaussian determined pixel colors. For cleaner output, you can toggle `solid=True` in the top of `buoy_detection.py`, which will render solid circles on a black canvas (Figure 12).

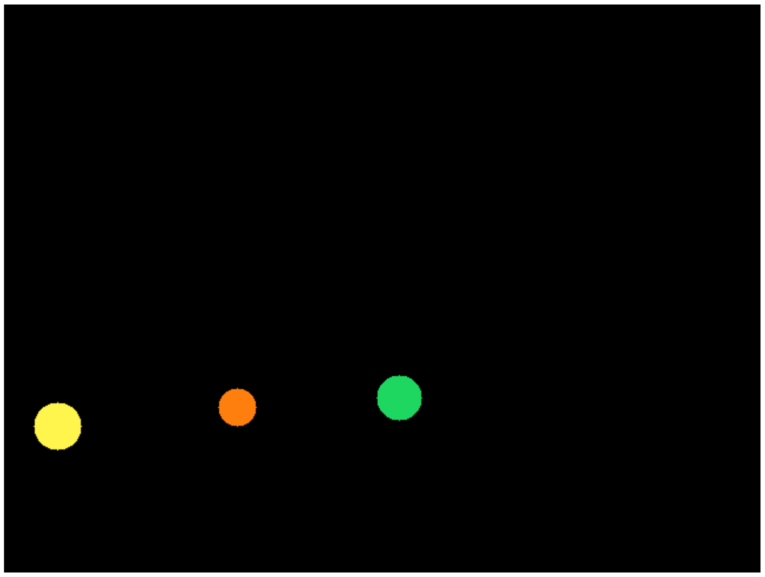


Figure 12: Buoys Replaced with Solid Circles

You can view both of these output versions in the Videos section. The new frame is exported side by side with the original video for side by side viewing (Figure 13).

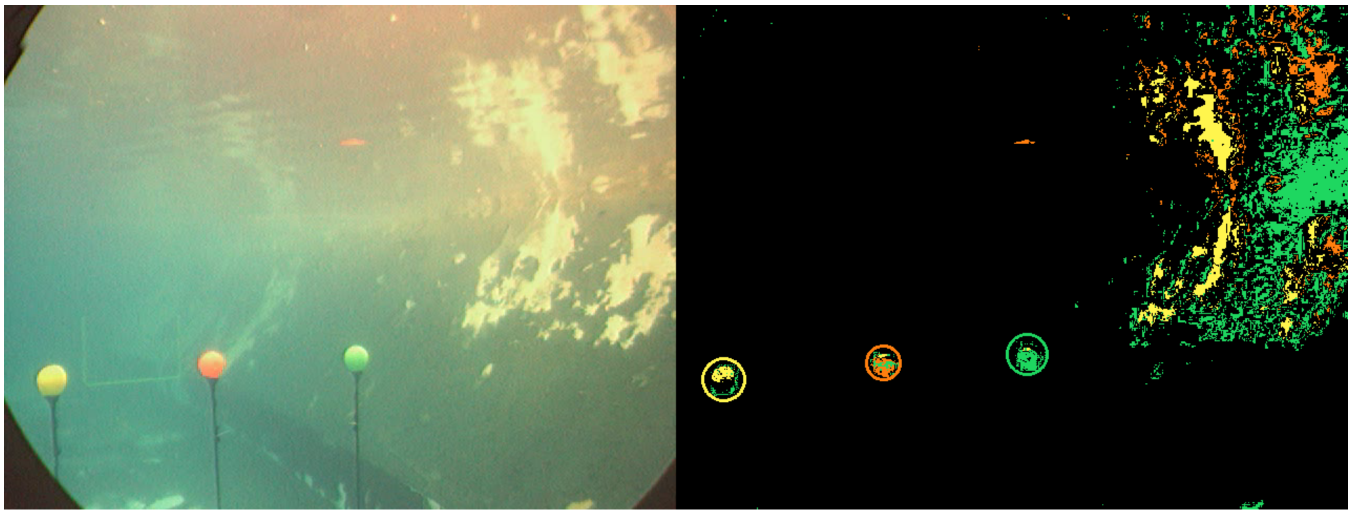


Figure 13: Final Frame

6 Videos

Original Video: https://youtu.be/JEC2yy3dHiw
Buoy Region of Interest Selection: https://youtu.be/gAHzZghxUaw
Buoys with Colored Rings: https://youtu.be/UGUHVrVdI2U
Bupys with Colored Circles: https://youtu.be/aP2YAfDPRnQ

7 Code

You can view the files related to this project at: https://github.com/BrianBock/ENPM673_Project3

8 Additional Reading

These are some additional resources which were helpful in the comprehension of this material.
https://www.youtube.com/watch?v=kkAirywakmk
https://www.youtube.com/watch?v=SKNUd6YUOpY
https://www.youtube.com/watch?v=qMTuMa86NzU
https://www.youtube.com/watch?v=JNlEIEwe-Cg
https://www.youtube.com/watch?v=EWd1xRkyEog

9 References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer Science+Business Media, LLC, 2006. [Online]. Available: http://www.rmki.kfki.hu/~banmi/elte/bishop_em.pdf

[2] F. C. Chang, "Fitting a Mixture Model Using the Expectation-Maximization Algorithm in R." [Online]. Available: https://tinyheero.github.io/2016/01/03/gmm-em.html#fitting-a-gmm-using-expectation-maximization

[3] M. S. Charifa, "Week 6: Clustering K-Means, EM Algorithm and its Applications," March 2020.

[4] P. Smyth, "The EM Algorithm for Gaussian Mixtures." [Online]. Available: https://www.ics.uci.edu/~smyth/courses/cs274/notes/EMnotes.pdf

[5] E. Fetaya, J. Lucas, and E. Andrews, "CSC 411 Lectures 15-16: Gaussian mixture model & EM." [Online]. Available: https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec15_16_handout.pdf

[6] Visualization and P. Lab, "Gaussian Mixture Model (GMM) using Expectation Maximization (EM) Technique." [Online]. Available: http://www.cse.iitm.ac.in/~vplab/courses/DVP/PDF/gmm.pdf

[7] N. J. Sanket, "Color Classification." [Online]. Available: https://cmsc426.github.io/colorseg/#colorclassification

[8] R. Sridharan, "Gaussian mixture models and the EM algorithm." [Online]. Available: https://people.csail.mit.edu/rameshvs/content/gmm-em.pdf

[9] H. Permuter, "Lecture 2: GMM and EM." [Online]. Available: http://www.ee.bgu.ac.il/~haimp/ml/lectures/lec2/lec2.pdf

[10] R. Zemel, R. Urtasun, and S. Fidler, "CSC 411: Lecture 13: Mixtures of Gaussians and EM." [Online]. Available: https://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/13_mog.pdf

Appendices

A Enlarged Plots

The 3x1 plots earlier in this report may be difficult to read in detail, so they are reproduced full size here:

A.1 Histograms

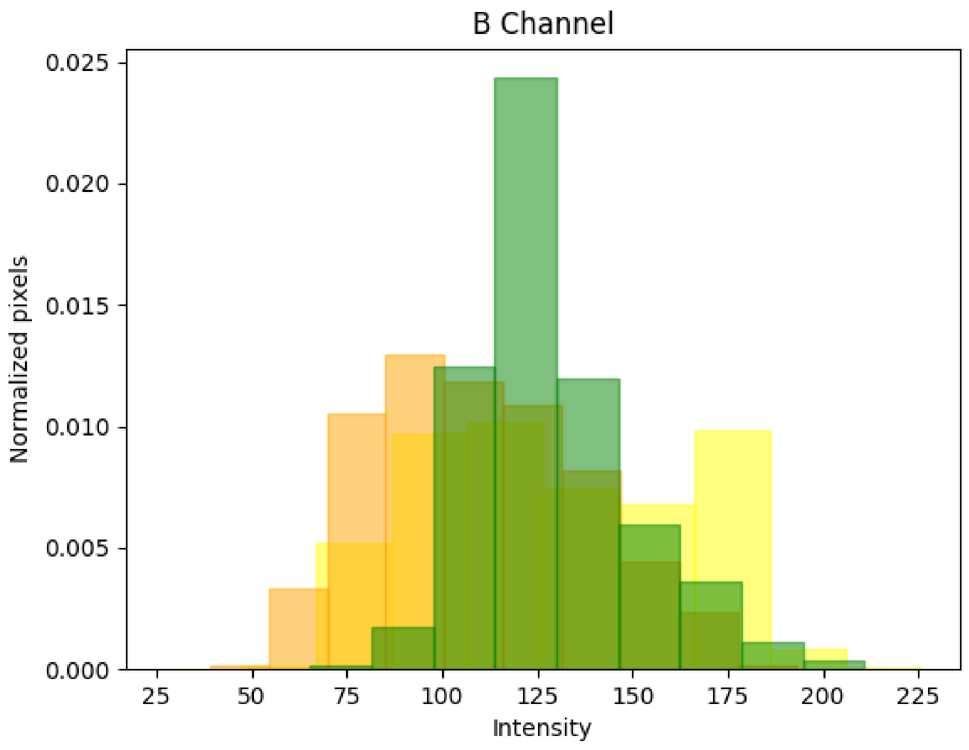


Figure 14: Blue Channel Histogram

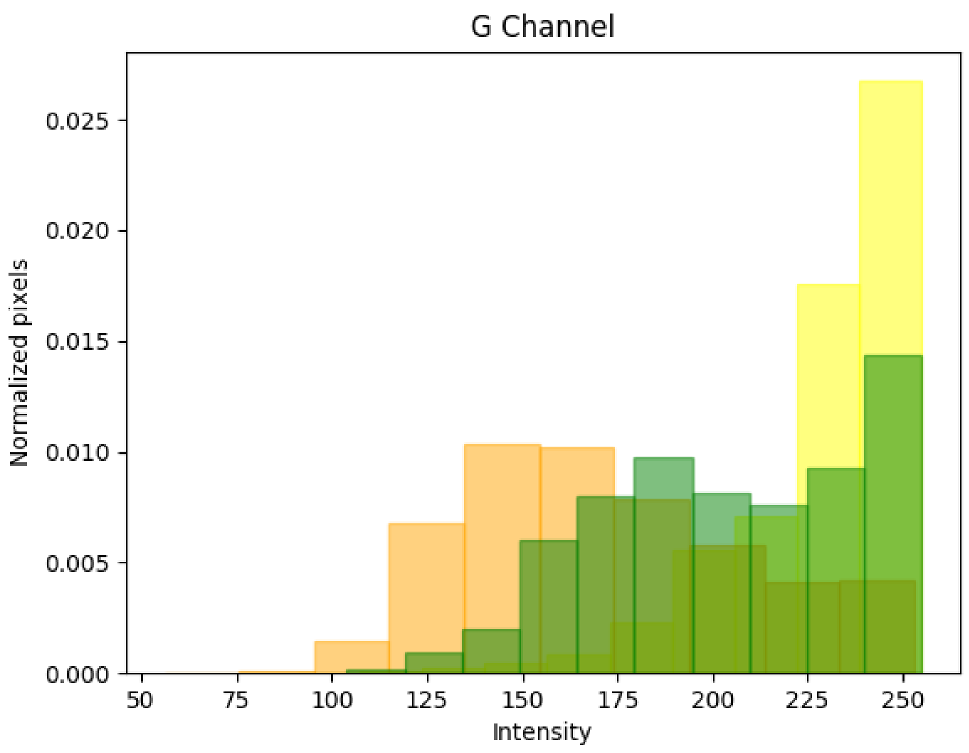


Figure 15: Green Channel Histogram

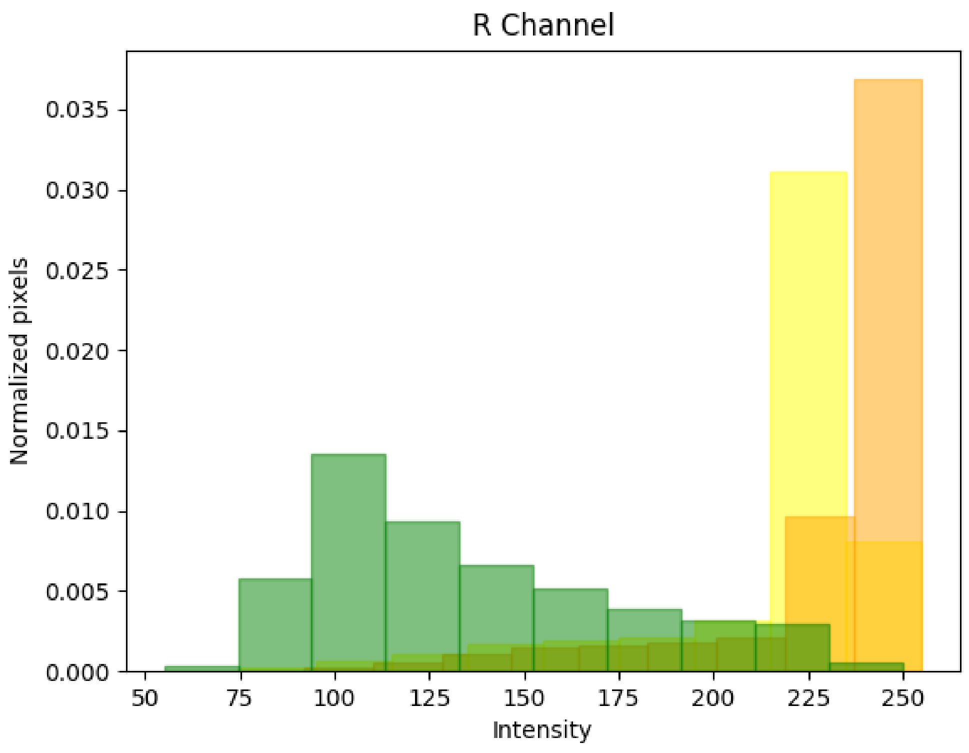


Figure 16: Red Channel Histogram

A.2 Scatter Plots

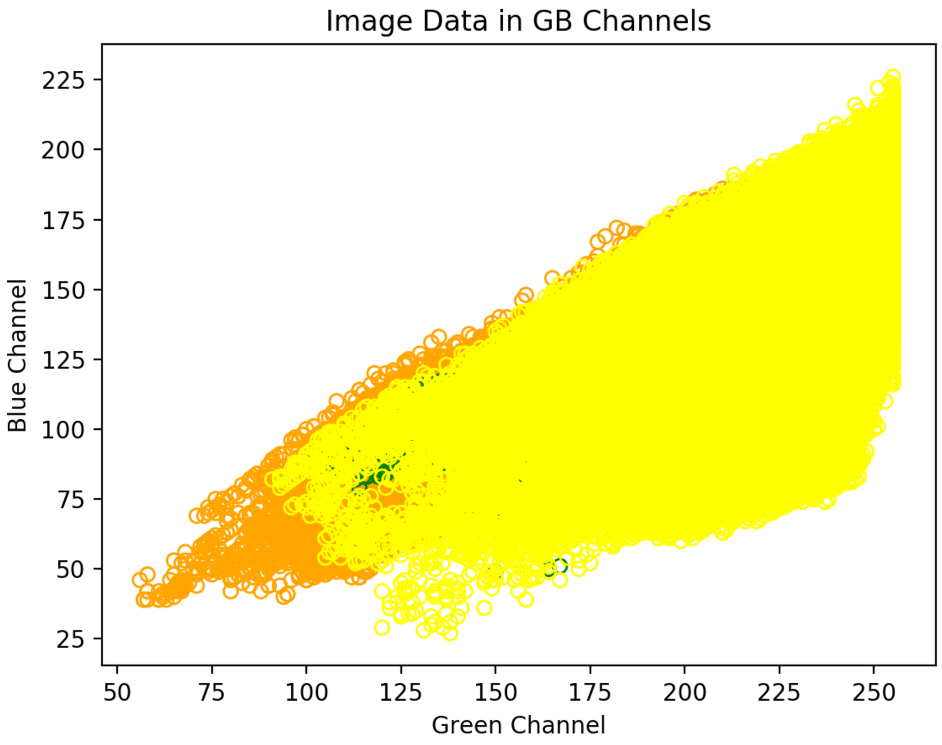


Figure 17: Blue Green channel scatter plot

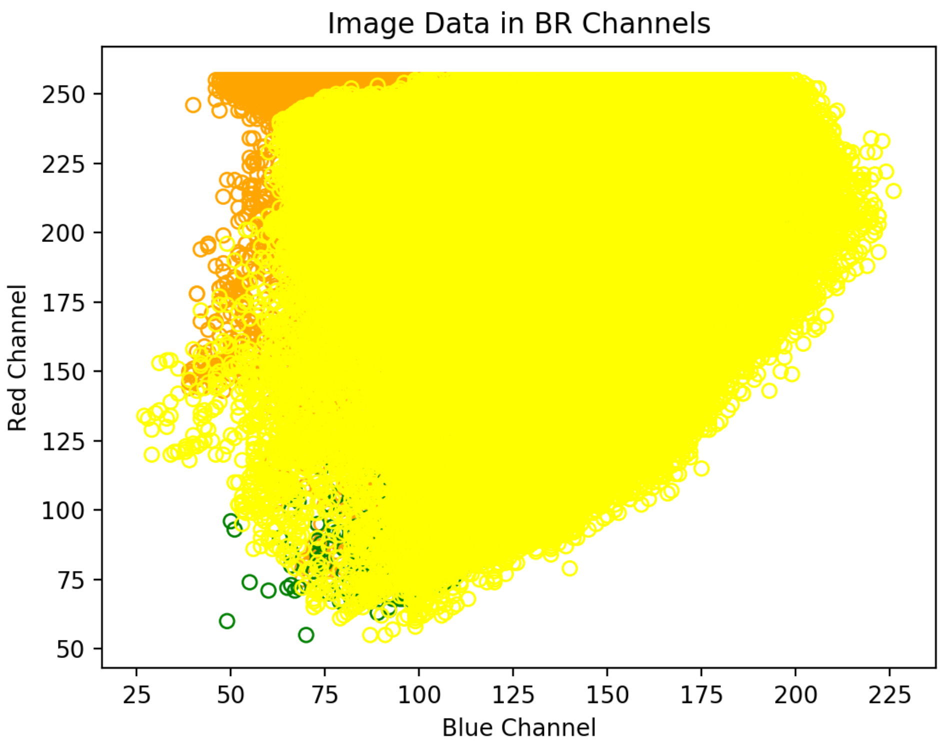


Figure 18: Blue Red channel scatter plot

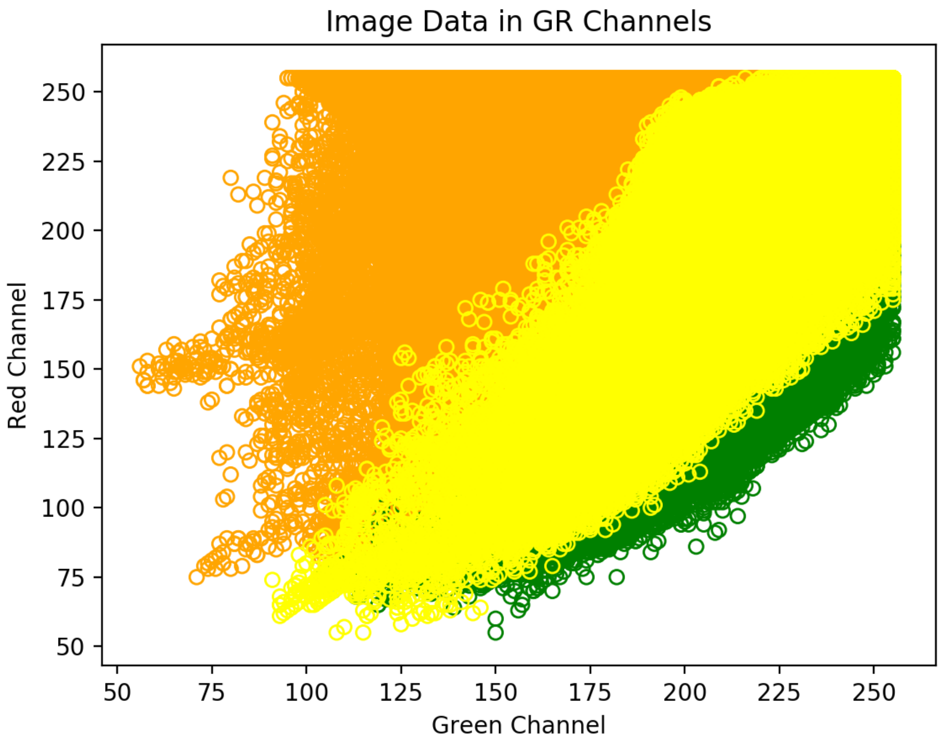


Figure 19: Green Red channel scatter plot

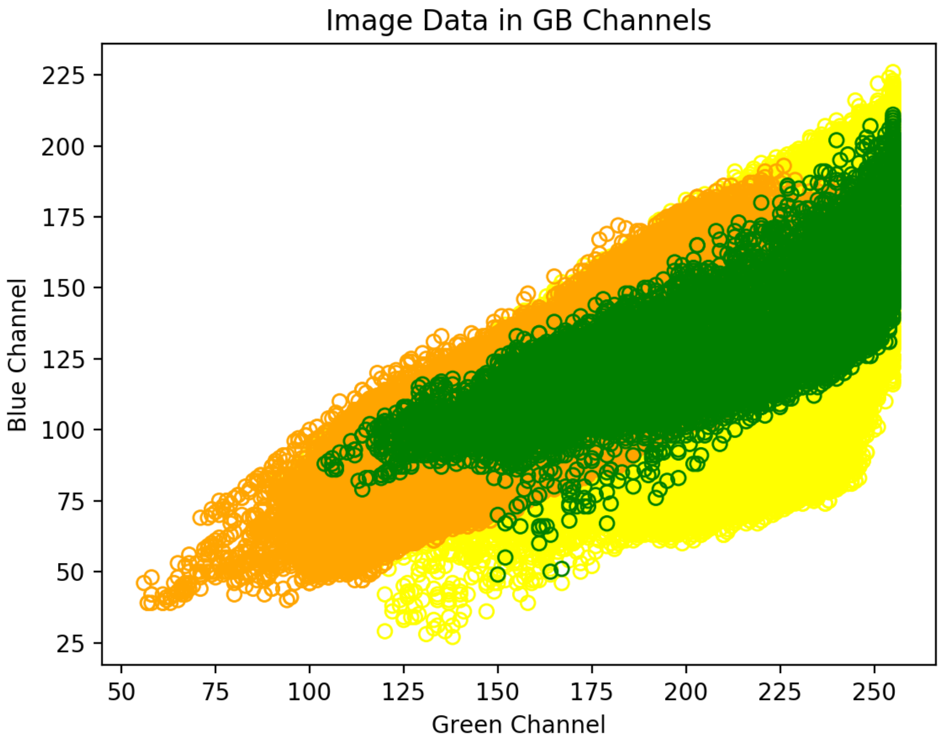


Figure 20: Blue Green channel scatter plot

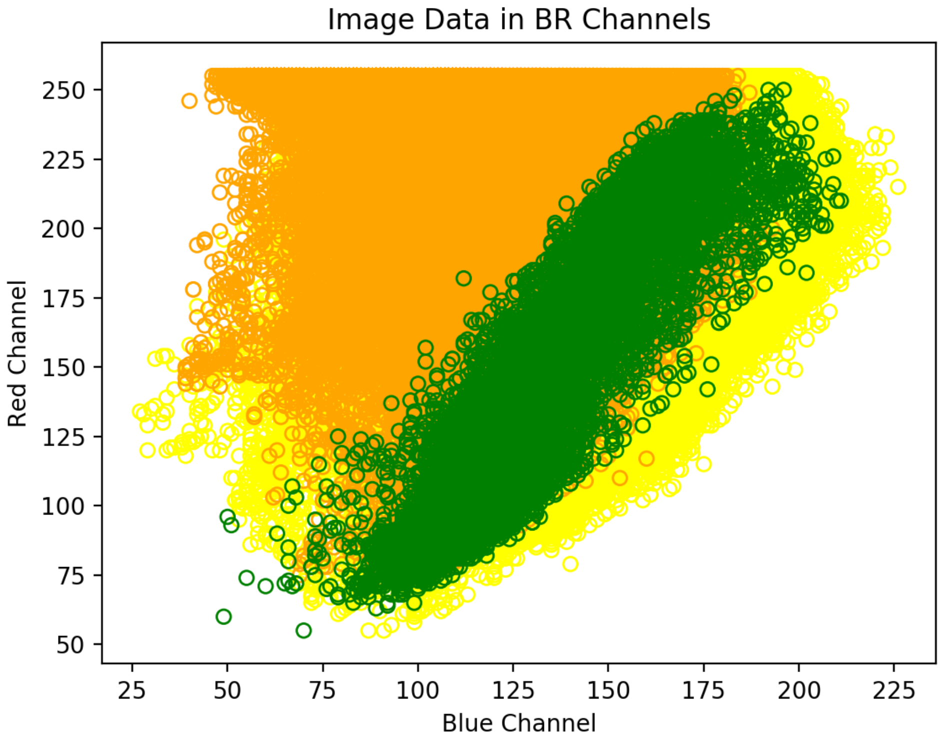


Figure 21: Blue Red channel scatter plot

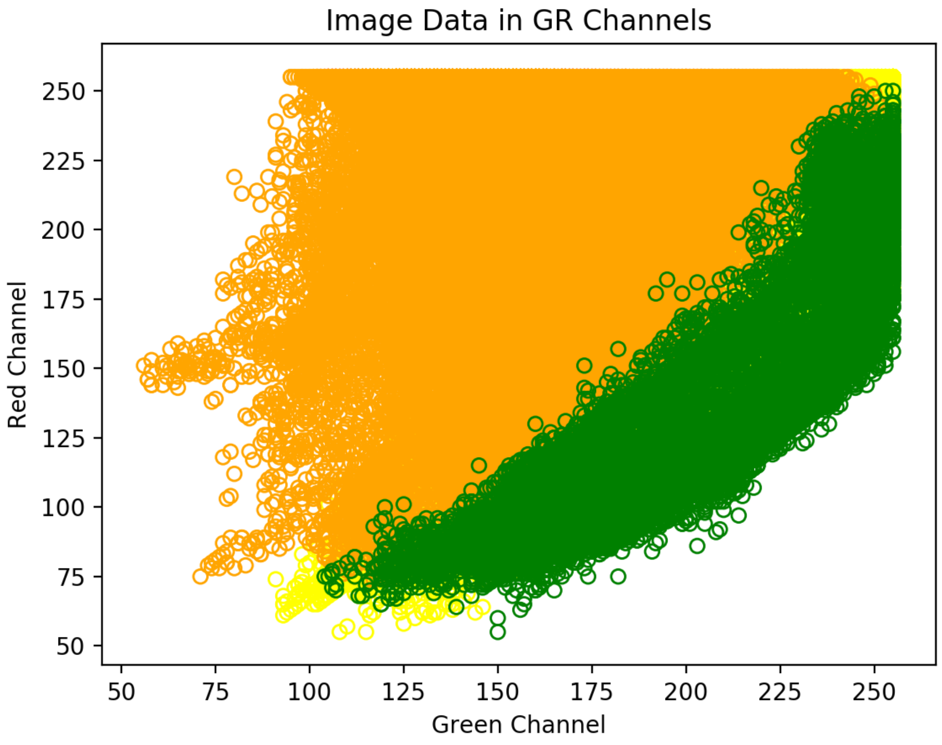


Figure 22: Green Red channel scatter plot

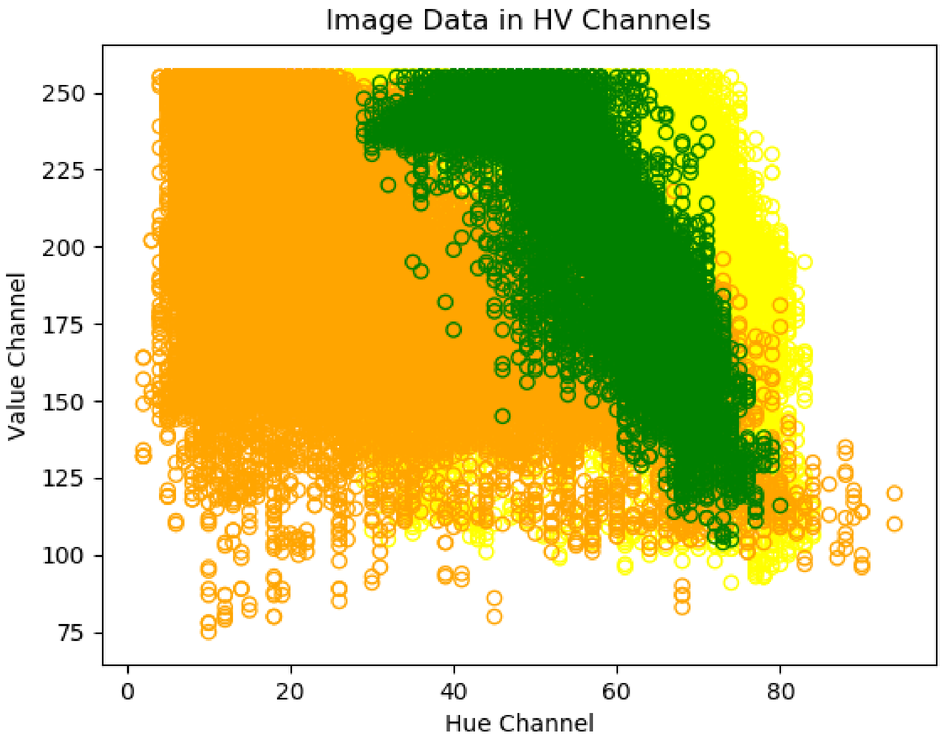


Figure 23: Hue Value channel scatter plot

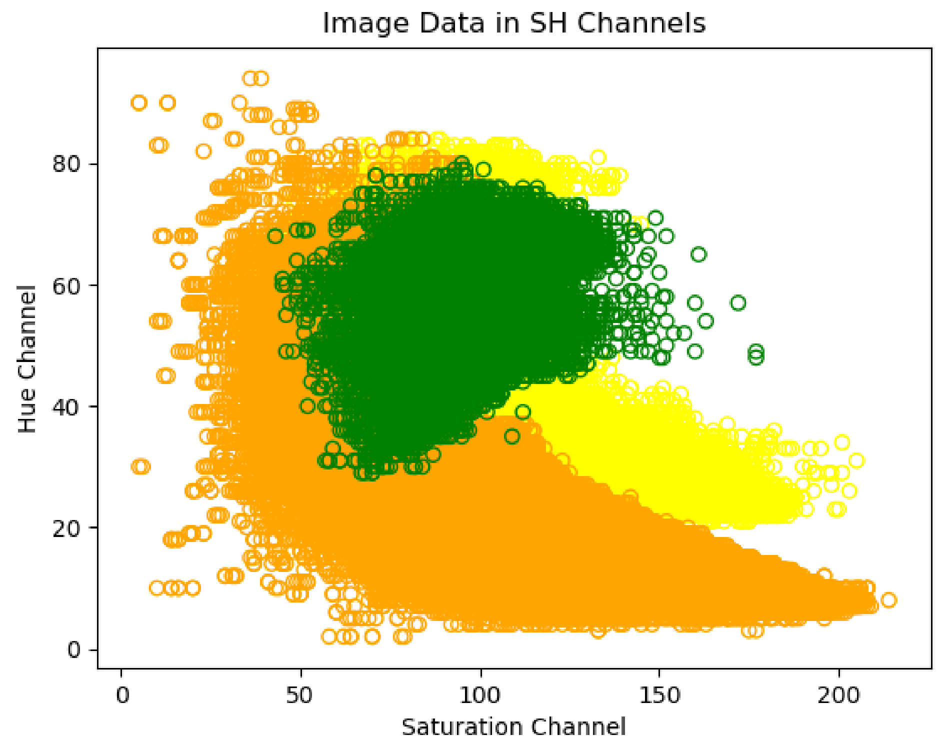


Figure 24: Saturation Hue channel scatter plot

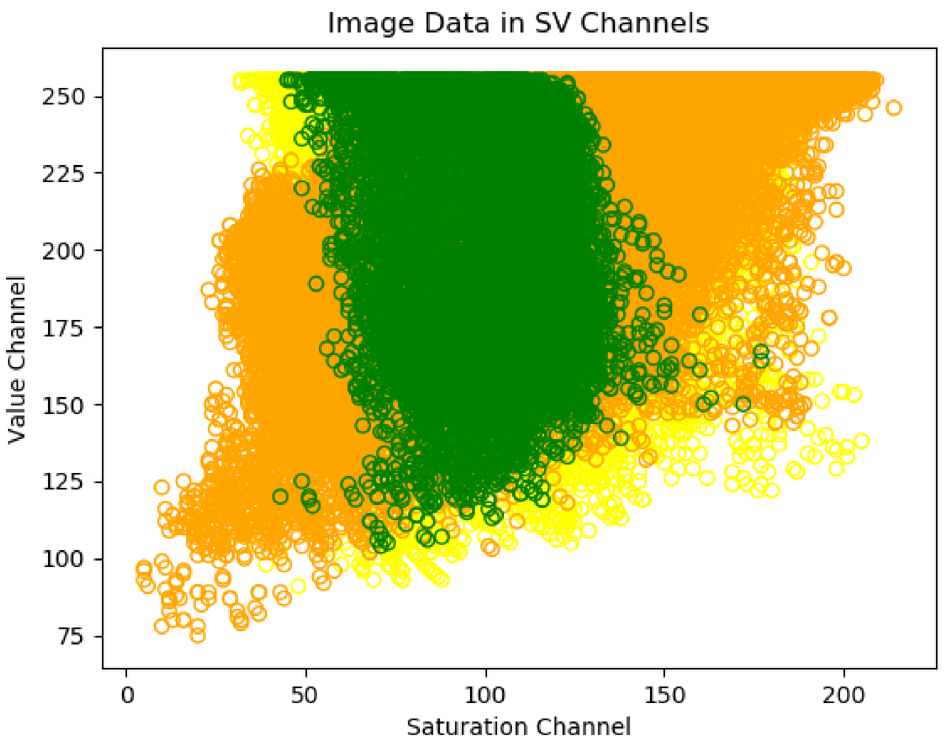


Figure 25: Saturation Value channel scatter plot