IMAGE PROCESSING FOR WHITE LINE DETECTION IN A GRASSY FIELD

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Computer Science

By

Rodi Murad

In Partial Fulfillment of the Requirements
for the degree of
MASTER OF SCIENCE

Major Department:
Computer Science

July 2019

Fargo, North Dakota

North Dakota State University
Graduate School

Title

MAGE PROCESSING FOR WHITE LINE DETECTION IN GRASSY FIELD

By

Rodi Murad

The supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Jeremy Straub
Chair

Dr. Pratap Kotala

Dr. Majura Selekwa

Approved:

July 1, 2019
Date

Dr. Kendall Nygard
Department Chair

ABSTRACT

Image processing for object detection and decision making based on the observed remains a challenge today. As participants in the Intelligent Ground Vehicle Competition in 2018 and 2019, we were tasked to build an autonomous vehicle capable of maneuvering through a grassy course delineated with white lines and laid out with obstacles. To remain within the white line boundaries, we used OpenCV and Python for image processing and white line detection. To accomplish the task, we used image processing techniques such as image filtering, blurring, and edge detection. Although this algorithm was produced for the specific application of detection edges of white lines and obstacles found throughout the course, it can be used for the detection of edges of various objects for shape comparison for identification, as well as quality control of produced goods.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Objects in space reflect light in various wavelengths outside and within the visible spectrum of the humans' visual system [9]. The reflected wavelengths within the visible spectrum allows the human eye to visualize [9] and the brain to interpret the surrounding space, make decisions, and perform actions based on what is seen. The development of computer vision came from interest in developing computational methods to replicate the human visual system to build autonomous systems that can perform the tasks that are currently done by humans [4].

This directly relates to our project which involves building an autonomous vehicle that is capable of maneuvering around a course using computer vision for white line detection and edge detection of obstacles. In addition to computer vision, we also utilized other sensory devices such as LIDAR and a global positioning system for obstacle detection/avoidance and navigation through a non-delineated section of the course.

In computer graphics, the red, green, and blue (RGB) color model is most widely used due to its simplicity [5]. However, in RGB the color and intensity are not separate which introduced a challenge when attempting to measure the difference between color and greyscale. Due to the challenges of using RGB color model in computer vision, there are other color spaces that have been developed to counter such issues such as HSV, HSL, and others. HSV and HSL are most widely used.

For the purpose of this project, my main task was to use computer vision for the detection of the white lines that delineate the course path as well as the detection of edges of the obstacles for maneuvering within those boundaries white avoiding the obstacles. In order to achieve this, we used the OpenCV, Numpy, and matplotlib libraries along with Python programming language, and implemented various techniques that were available for image processing. The methods used included the conversion of the image from an RGB color model to HSV, HSL, and grey color spaces, followed by a combination of image smoothing, masking, and morphological transformation for the detection of the edges. The edges detected include edges of the white line as well as obstacles that were scattered along the course path for testing vehicle's obstacle avoidance ability.

Given the wide range of applications for which computer vision is being used, the importance of this project goes beyond the development of an autonomous vehicle that is capable of traversing a

course without crossing the white lines and avoiding obstacles. This project is as much about computer vision as it is about understanding the environment in which it is being used, understanding the method better and for which environment some of the methods tend to be more effective than others, as well as establish an understanding of the use of their combination to achieve the desired results. Gaining the capability of de-noising images may provide an improved edge detection as well as feature extraction.

Furthermore, it would potentially allow the detection and comparison of objects based on shapes [21], contour matching [22], and other parameters of interest. For example, in manufacturing, the use of computer vision and edge detection for product quality control is used to ensure that produced items fall within an acceptable defect margin based on predefined shapes. This is then compared to the produced object's shape and its detected contours. Another example usage is shape-based object detection in x-ray scanners for airport security or in healthcare.

CHAPTER 2. BACKGROUND AND RELATED WORK

2.1. Visible Spectrum

The electromagnetic (EM) spectrum contains a range of types of electromagnetic radiation. The range of radiation includes radio, microwave, infrared, visible light, ultraviolet, X-ray, and gamma-ray [1]. The difference in the electromagnetic radiations is the energy (wavelength) at which they travel, where long wavelength is attributed to low energy, while short wavelength is high energy. Although not all are familiar with the various types of EM radiation, at some point in life most have been encountered most of them [1], [9].

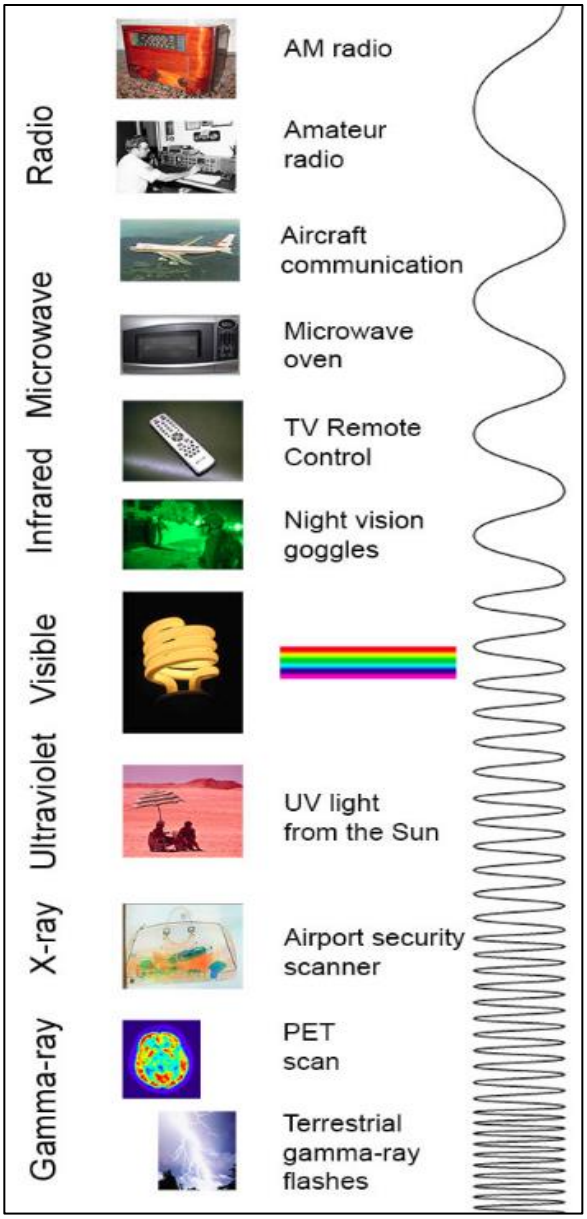

Figure 1. Electromagnetic Spectrum [1]

From Figure 1 we see that the visible spectrum can be found between the ultraviolet and infrared EM radiation types, where the ultraviolet attributes to the radiation emitted by the sun, while the infrared radiation is emitted by objects that emit heat that is much lower in energy than the sun, such as the human skin [1], [6].

For the purpose of this project, we are only interested in the EM radiation that falls within the visible spectrum which is the portion that is visible to the human eye [9]. Within this spectrum, the primary colors are red, blue, and green, while all other colors that we can see are the result of various combinations of wavelengths from the primary three colors [6].

2.2. Color Model and Spaces

RGB is a three-dimensional cube where the colors are described in Cartesian coordinates. It is an additive color model [7] developed by computer graphics researchers to more closely align to the human perception of color based on the human's ability to process three independent channels for colored information conveyance [2], [6].

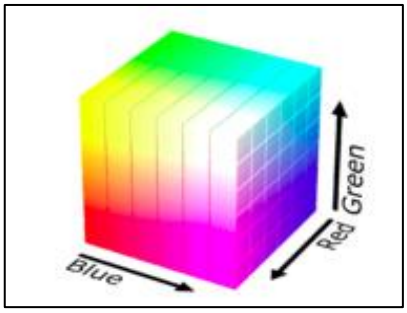

Figure 2. RGB Color Cube [2]

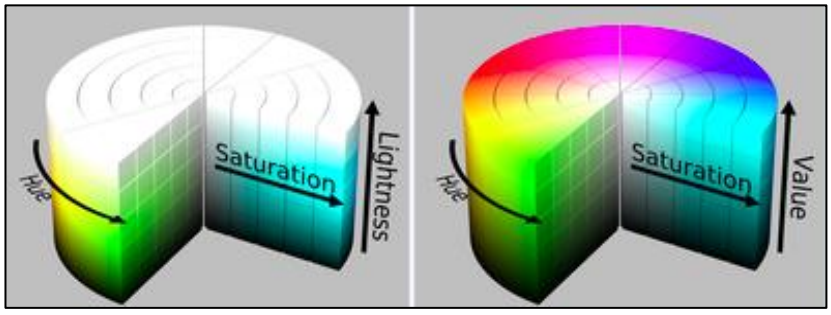

Figure 3. HSL (left) and HSV (right) [3]

HSV and HSL are an alternative representation of the RGB color model and are thought to more closely resemble the human perception of colors [3]. The hue of each color is placed in a cylindrical model with an angular range of 0 - 360 degrees [7] revolving around the central axis of neutral colors which represent black at the bottom, white at the top, and various shades of grey in between. While the saturation represents the intensity of the color, which increases moving away from the central axis. In HSV the value represents the mixture of the various colors, while in HSL the colors are more aligned with the Natural Color System and are placed at a half value of lightness, while 0 and 1 represent black and white respectively. HSV resembles paint color mixing, while HSL attempts to resemble the natural color system [3].

Lastly, greyscale is a neutral color space where pixel values lie between 0 and 255 [8], which are black and white respectively, while any values in between represent various intensities/shades of grey. Therefore, computation performed on grey color space are applied in two dimensions and makes the process of image processing less intensive [8].

2.3. Image Processing

Image processing is the application of various methods for noise reduction given that the retrieved images and video frames from digital cameras tend to have varying levels of noise. The noise can be attributed to conditions such as lighting, sensors of a digital camera, rapid movement of the camera, etc. [10], [11], [13]. Therefore, in order for us to obtain the desired information from the retrieved images, we must apply various processing techniques to remove any unwanted information using the OpenCV library to reduce/eliminate any unwanted information.

Prior to applying any image processing techniques/methods, a depiction of the image is shown in Figure 5 for the purpose of understanding where the noise is found in the image. In the original image we only see various shades of green grass, while in the binary form as seen in Figure 5 we can clearly visualize the noise that must be removed in the grassy area between the white lines prior to us being able to use the retrieved image for white line detection, obstacle detection and path mapping.

Figure 4. RGB

Figure 5. Binary (RGB)

2.4. Related Work

Since the development of computer vision, due to the interest in intelligent machines replicating human tasks [4] by interpreting what is being visualized in digital images, numerous research and various techniques have been developed to tackle the problems that arose throughout, and research continues to take place in the subject [12], [13], [14], [15].

For autonomous vehicles the main challenge is to ensure that the algorithm detects the white line markings [11] that delineate a path, ensure that the vehicle remains within the boundaries as well as perform object identification for obstacle avoidance, road sign interpretation, pedestrian identification, etc. Given that our project mainly focuses and white line/edge detection, we have looked at work done by others for white line detection and edge detection. This research includes methods such as a double circular operator [13], multilevel morphological fuzzy edge detection [12], and improved sobel edge detection [14].

2.4.1. The Double Circular Operator

The double circular operator focuses on the local luminance distribution around a white line, where the algorithm works with an assumed width of the white line with varying orientation. The camera is calibrated to where the focus is set to infinity and shutter speed appropriate to distinguish luminosity of the white line. The procedure steps start with converting the image to bird's eye view, after which smoothing of the image is implemented then the application of the double circular operator [13].

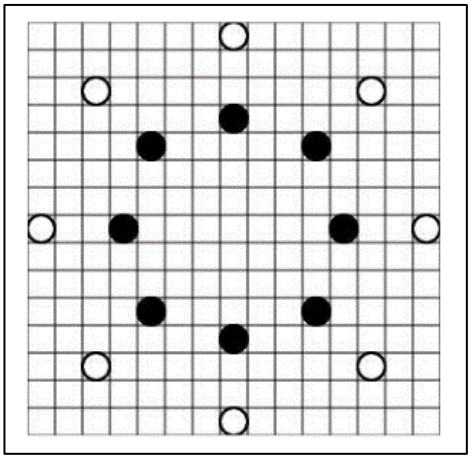

Figure 6. Double Circular Operator [13]

The double circular operator consists of a 15x15 pixel size to extract the white line from the image. As shown in Figure 6, the double circular method consists of white outer and black inner circles that are distributed radially in the pixel matrix. In the double operator method, d represents the diameter of the inner circle and d_o represents the outer circle, w represents the pixel width of the to be detected white line, and e is the marginal width at the edge. The luminosity of the inner circle is constant, while the luminosity of the outer circle is overlapping the white and are representative of the light and dark peaks. In order to detect the white line, the pixel is examined to match the center of the inner circle of the

operator and the distribution of luminosity of both circles, if the returned value are within the threshold, then it becomes a candidate point, while if the value returned is outside the threshold, then it is rejected and the operator moves to the next pixel for evaluation [13].

2.4.2 The Multilevel Morphological Fuzzy Edge Detection Method

The multilevel morphological fuzzy edge detection (MMFED) method uses several processes for edge detection. The MMFED algorithm enhances the image with the implementation of a transformation function based on threshold values. There are two threshold values based on the morphological function where one value used for the object in an image and another for the background. For image enhancement, a two-level edge detection process will be used. Using the grey image, the edge detection will be performed initially to define the structure of properties, while the second edge detection will use the morphological operator to detect a more accurately the fine edges. This ultimately results in a higher accuracy for extracting fine edges [12].

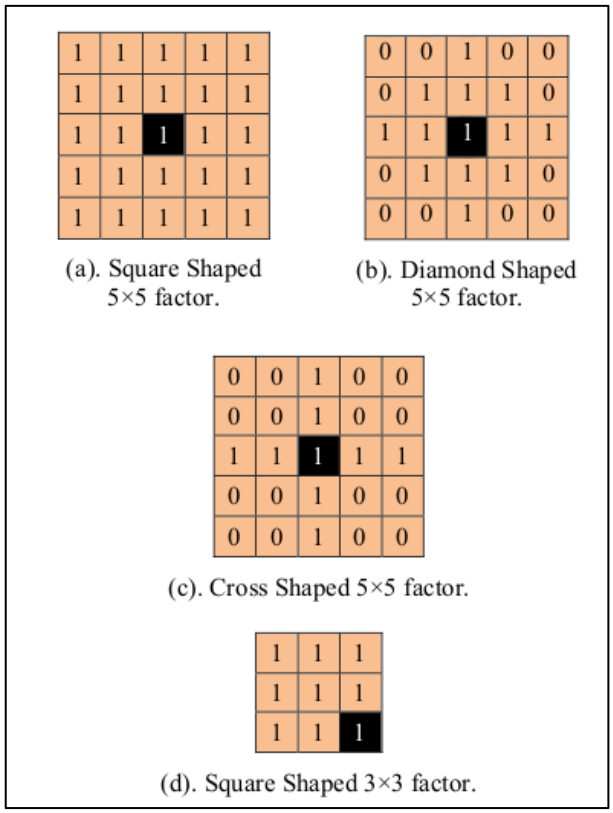

Figure 7. Structuring Factor

The MMFED method steps include a structuring factor, basic morphological operations, followed by the first and second edge detection processes. The structuring factor plays a vital role in the

implementation of the various morphological operations which include the dilation, erosion, opening, closing, etc. The structuring factor is a two-dimensional matrix of zeros, ones, or both with various shapes (square, diamond, cross), which are then used for image smoothing effects. The size of the structuring factor as shown can vary and is not limited to the 3x3 or 5x5 matrix, where the increase in its size directly affects the number of pixels that is affected by it [12].

The morphological operations include (dilation, erosion, opening, and closing), where dilation and erosion have opposite effects of increasing and decreasing the white region of the foreground respectively, while the opening and closing are another name for erosion followed by dilation and dilation followed by erosion respectively, and all four are ultimately helpful in removing noise [18]. Once the structuring factor has been chosen and the morphological methods (dilation and opening) have been applied to the image, then the first level edge detection method follows.

As mentioned previously two levels of edge detection are applied, where the first edge detection is applied to the blurry images to identify constant edges which produces thick and double edges and followed by the second edge detection method (improved edge detection algorithm). In the second level edge detection method, the morphological thinning operators (erosion and closing) are used, followed by the edge detection, which is used to find the fine edges that are isolated, and their removal is followed [12].

2.4.3. Improved Sobel Edge Detection

Sobel is one of several methods used for edge detection using the gradient method. The gradient is considered as an abrupt change in intensity [14], [16] of pixel values given that edges have higher intensity value than neighboring pixels. In the gradient method minimum and maximum values are searched for, where it uses a derivative approximation value for the determination of the edge [14], [16], which is where the gradient is maximum. For the determination if the gradient found is an edge or not, a threshold value is set as a boundary parameter. If the derivative approximation value exceeds the value of the set threshold, then the pixel and its location is recorded as an edge.

The Sobel method performs a 2-d spatial gradient measurement using a pair of horizontal and vertical gradient matrices with a 3 x 3 dimensions [14], [16]. Advantages of the Sobel operator are that, it has a smoothing effect and enhancement of the edges. The smoothing effect for random noise occurs

when found in an image using the average factor, while the enhancement (thickness and brightness) of the edge occurs due to its differential of two rows or columns [14], [16].

Based on prior work on Sobel, the improved Sobel image edge detection algorithm (ISIEDA) is implemented using image smoothing algorithms (opening and closing methods of the morphological transformation), followed by using an extended Sobel operator for edge detection, binary Otsu (thresholding method), and image optimization using the image fusion method [14].

Morphological transformation opening and closing methods use the erosion followed by dilation and dilation followed by erosion methods respectively. Erosion removes pixels that do not equate to 1 when overlaid by a specific kernel size, while the dilation increases the surrounding pixels when all pixels under the kernel equate to 1 [14], [17].

The extended Sobel operator added 6 directions to the traditional Sobel (0 and 90 degrees), where the added directions are 45, 135, 225, 270, and 315 degrees [14].

Otsu binary thresholding method is applied based on an image histogram in greyscale, where the selection is divided into two parts to obtain the maximum from the intraclass variance.

And finally apply the image fusion method, which involves the combination of three methods that are the Improved Sobel operator, Canny operator, and LoG (Laplacian of Gaussian) operator. Upon applying the three edge operators, the mean of the three methods is taken and superimposed on the final image [14].

CHAPTER 3. PROBLEM AND METHODS

3.1. Problem – White Line Detection on a Grassy Field

The main challenge is to detect white lines that delineate the course, avoid obstacles, and traverse a non-delineated section of the course using GPS. The retrieved video feed contains noise that may prohibit the clear identification of the white line using color alone; therefore, image processing is necessary to remove noise from the image while maintaining the visibility of the white lines as well as the edges of the obstacles.

Given that the course is a grassy area, as more vehicles travel through the course, such an action causes localized compaction of the grassy area. Combining such effects with lighting conditions (clouds, sun, etc.), various shades of grass, as well as the camera's sensors stabilization capability, results in much noise retrieved in the video feed. The noise found in each frame introduces a challenge; removing it while maintaining the contrast of the white lines and the ability of their detection throughout the traversed course.

In order to accomplish the task of white line/edge detection, a series of image processing methods used for noise removal. The methods utilized include a change of color space, thresholding, smoothing, morphological transformation, and edge detection. Therefore, to reach desired results given the potential for variability of weather and lighting condition as well as various shades of ground surface to which our autonomous vehicle will be exposed to, a trial and error method is necessary for obtaining the best results. The methods mentioned are introduced as well as the results of their implementation followed by the decision of choosing the best combination of the methods for our purpose.

For the implementation of the mentioned methods, we utilized the following libraries OpenCV, NumPy, and matplotlib developed for Python programming language. The image processing libraries' versions used are OpenCV 3.4, NumPy 1.15, matplotlib 2.2.3, and Python 2.7.

3.2. Systems – OpenCV, NumPy, matplotlib, Python

OpenCV is an open source computer vision library which began development in 1999 at Intel. In 2005 OpenCV was implemented in the Stanley autonomous vehicle built by the Stanford University racing team in cooperation with the Volkswagen Electronics Research Laboratory that won the DARPA

challenge. The OpenCV library is available for multiple platforms (Windows, OS X, Linux, etc.) and supports many programming languages and algorithms for computer vision and machine learning [23].

NumPy is a scientific computing package with Python, which contains N dimensional array object, function broadcasting (describes NumPy treatment of array during operations permitting scalar value combination with arrays), tools for integrating C/C++ and useful linear algebra, Fourier transformation, and random number capabilities. Some functions (e.g. addition of multiple images) can be done in NumPy and/or OpenCV, however, OpenCV provides better results since the operation in OpenCV is saturated, while in NumPy it is a modulo operation. In a saturated operation, minimum and a maximum are defined, where if the result of the addition or multiplication is greater than maximum then the value is set to the maximum, while in modulo operations the difference is taken without consideration to maximum/minimum value, where ultimately the saturated operation results in better results for addition/multiplication operations. However, NumPy allows us to generate various size arrays efficiently with minimal code requirement [24].

Maplotlib is a Python 2-D plotting library that provides functions for creating histograms, charts, plots, etc., and it is a numerical mathematics extension to NumPy. For the purpose of our application, we mainly use it for the purpose of plotting histograms since we are only concerned with white line detection in this particular project [25].

Python is an interpreted, general-purpose high-level programming language that emphasizes code readability and simplicity. Using the OpenCV-Python is the Python API for OpenCV which combines the best qualities of OpenCV C++ API and the ease of programming in Python as well as it allows the use of NumPy, as previously mentioned is an optimized library for numerical operations [26].

CHAPTER 4. EXPERIMENT

4.1. Color Spaces

For the purpose of our application, we only look into the conversion from RGB to HSV HSL, and grey color model/spaces since they are easier to work with than the RGB color model, while the grayscale allows for easier computation given that need not worry about the three-color channels. Due to the simplicity of the RGB color model, it is widely used in various electronic devices such as cameras and televisions, therefore, all images retrieved from our camera will be in RGB. Once the image frames are retrieved from the camera, we apply the color space conversion to the image for the purpose of applying different image processing methods for the detection of the white lines. As shown in Figures 8 – 11, depicting the four-color spaces and their visual difference.

Figure 8. RGB

Figure 9. HSV

Figure 10. HSL

Figure 11. Grey

Rather than just rely on the visual images, we also plot the images' histogram to visualize pixel value concentrations throughout the image (prior to applying edge detection) and compare the results between color spaces. This can show us which method performs better in normalizing pixel values throughout the image, where for grey the pixel values range from 0 to 255 depicting shades of grey, while

in HSV and HSL depicts the combination of colors within the same range. As seen in Figures 12-15, we show the concentration of pixel values in the image where this will serve as the baseline that will be used to compare the methods we apply to the color spaces for noise reduction.

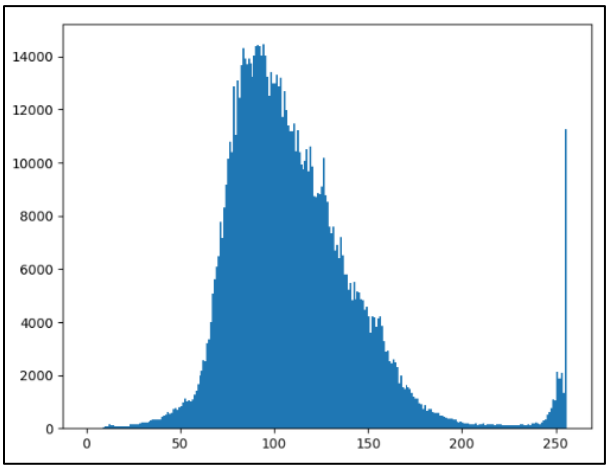

Figure 12. Histogram (RGB)

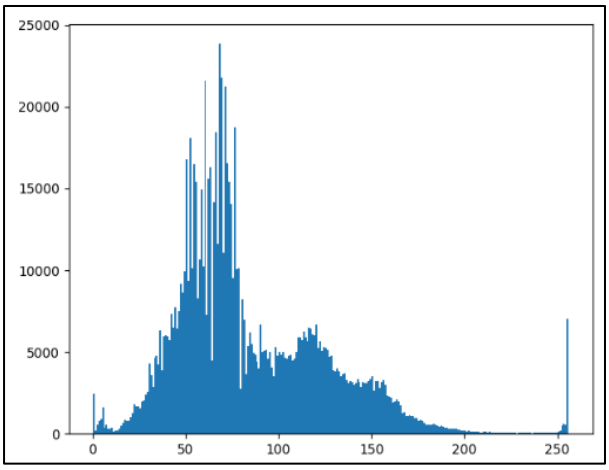

Figure 13. Histogram (HSV)

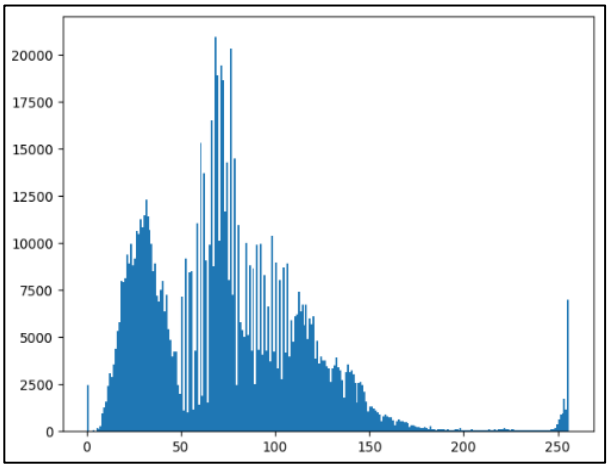

Figure 14. Histogram (HSL)

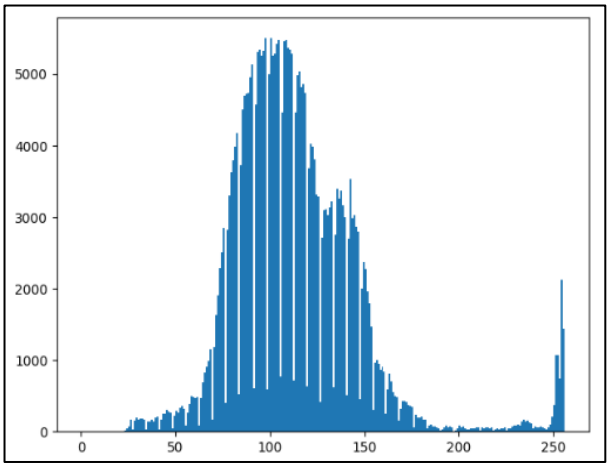

Figure 15. Histogram (grey)

From the histogram plots we can see that the distribution of pixel values varies depending on the color space used, which is expected since as seen in Figures 8 – 11 the retrieved colors in each color space differ greatly. The RGB in Figure 8 shows colors as we perceive them, HSV in Figure 9 depicts the image in a magenta color, HSL in Figure 10 depicts the colors in green, and the grey in Figure 11 depicts the image in various shades of grey.

4.2. Image Processing for Noise Reduction

Now that we have converted our image to the three different color spaces (HSV, HSL, and grey) from the original RGB color model, we will initially apply the Canny edge detection method and compare the results between the three-color spaces. This will allow us to gauge which color space will require the least amount of image processing for edge detection. However, as we can see in Figures 16-18 that the results are nearly identical with the exception of the grey where the difference can be found in the circled area. Given that the difference can only be found in the circled area, we will begin to apply various image processing methods available to all color spaces until we find which of the color spaces responds best to the image processing methods applied. Once we find the best response, we will abandon the other color spaces.

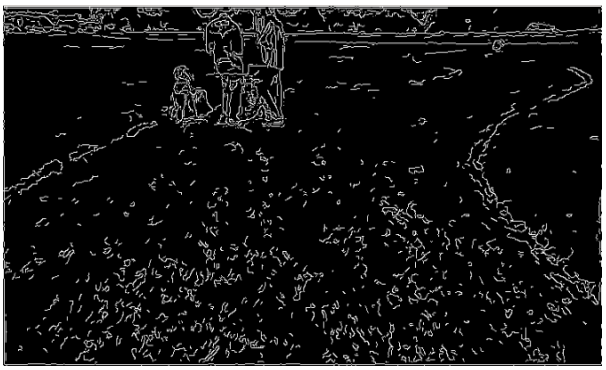

Figure 16. Canny Edge (HSV)

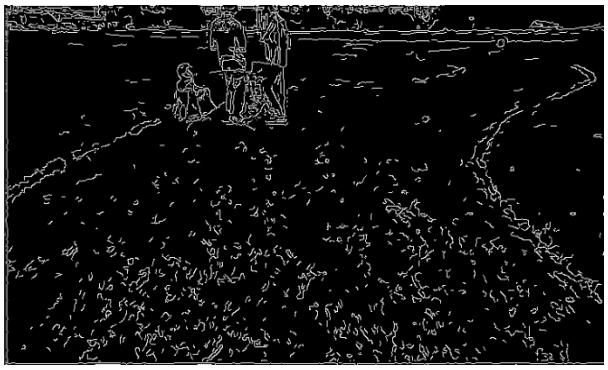

Figure 17. Canny Edge (HSL)

Figure 18. Canny Edge (grey)

From the results obtained using the Canny edge detection, we see that a great deal of noise was found in the grassy area between the white lines, therefore, we will next apply multiple methods of image filtering/smoothing to remove such noise.

Image smoothing is the application of pixel altering methods used to blur and filter images using mathematical functions that ultimately replace singular pixels that are detected as edges which are called

noise. Generally, this is done through the computation of same value that replaces pixel values under an area of the size of a square matrix used for the filter application (e.g. using the 2-D convolution method, take the sum of 25 pixels, compute their average, and replace the central pixel with the average value given we use a 5x5 averaging kernel size). For image smoothing, there are five methods (2-D convolution, image blurring, gaussian blurring, median blurring, and bilateral filtering). Each of the mentioned methods differ in how their averaging of pixel values are computed. Therefore, we shall discuss each in more detail and potentially choose the one with the best results for our application, however it most likely will require a combination of multiple methods to arrive at the best results.

4.2.1. 2-D Convolution Method

Using the 2-D convolution method we can filter the image for noise and edges. In OpenCV the 2-D convolution is applied using “cv.filter2D (src, -1, kernel)”, where “src” is the source image, -1 is the anchor position (position of the pixel of interest), and the kernel that will be used. It works by using a kernel (e.g. Figure 15) of a specified size, which in our case is a 5x5 matrix of ones and is convolved with our image from left to right, and top to bottom [23]. The kernel used will be applied on top of the pixel of interest, after which the pixel values surrounding the pixel of interest that fall under the matrix area are multiplied by the chosen kernel and each value is by the square of the kernel size then the sum of the resultant matrix is computed followed by the computation of the average value [23].

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 19. 2-D Convolution Kernel (5x5) [23]

Once the average value is computed, the value of the pixel of interest is replaced with the average value. Now we apply the 2-D convolution filter to the three-color spaces to compare the results to see if we can decide on which color space we will continue on using for the rest of our experiment. From Figures 20 – 22 we can see that a great deal of noise has been removed, while some noise remained. We varied our kernel size using a 3x3, 5x5, and a 7x7 matrices to see which will provide best results.

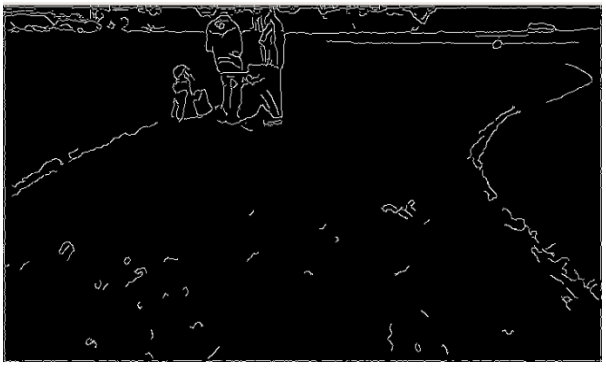

Figure 20. 2-D Convolution (HSV)

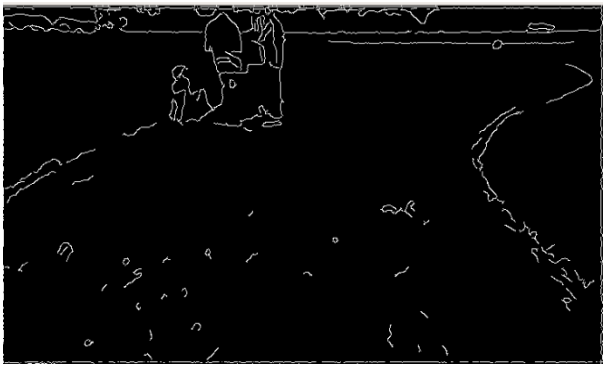

Figure 21. 2-D Convolution (HSL)

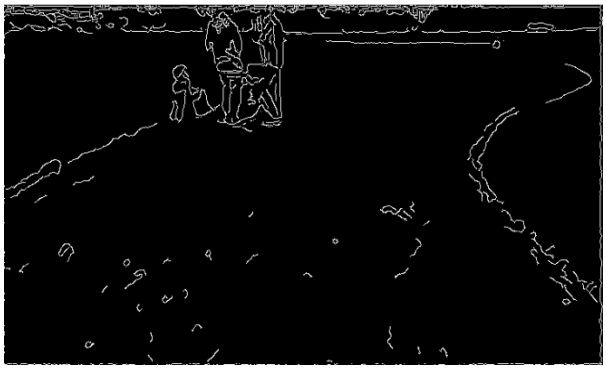

Figure 22. 2-D Convolution (grey)

Using the 3x3 matrix quite a bit of noise in our image, we then applied a 7x7 matrix, which resulted in a great loss of the white line, subsequently we applied a 5x5 matrix, which provided the best results by removing much of the noise, while keeping most of the white line detected. We then visually compared between the difference of noise reduction in three color spaces (HSV, HSL, and Grey), however, the difference was not great, therefore, we plotted the histogram of our images for each color space to see how were the pixel values altered by our method to compare to the baseline histogram plots as well as to compare to the other methods that we will apply in a later section.

From the histogram plots we can see the distribution of pixel values, where the Grey in Figure 25 has a more uniform distribution of pixel values between 60 and 125, while the hsv in Figure 23 has a high concentration of pixels between 50 and 75, and HSL in Figure 24 has high concentration between 25 and 50 and 65 and 75 where we can see that both lack uniformity, which can permit the assumption that they will be more difficult to work with.

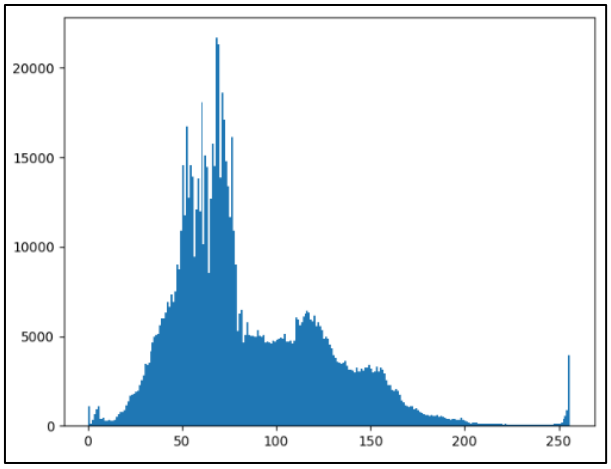

Figure 23. Conv-Hist (HSV)

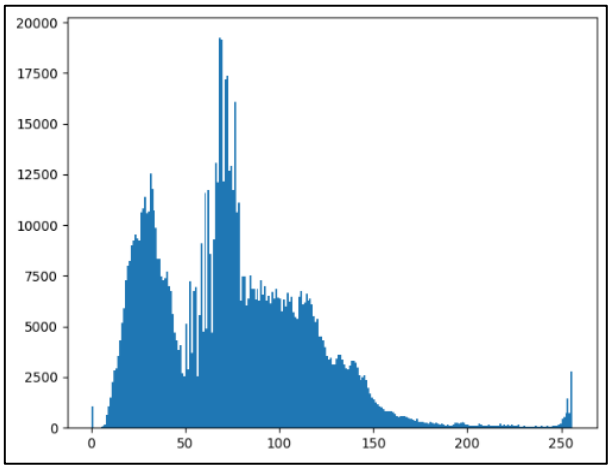

Figure 24. Conv-Hist (HSL)

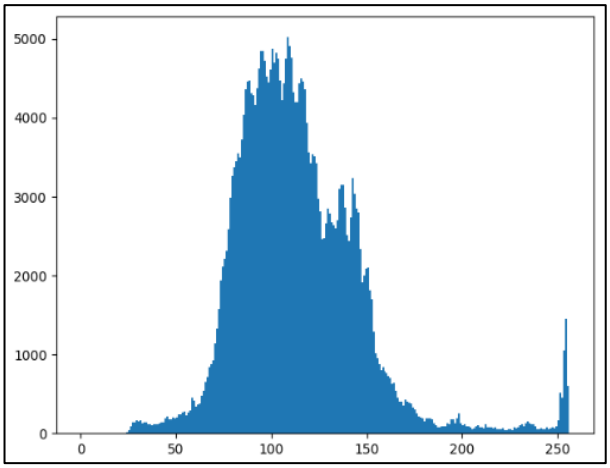

Figure 25. Conv-Hist (grey)

Comparing these histogram plots to the baseline plot above, we can see that a tremendous change is visible from the range (x-axis) as well as the number of pixels (y-axis) found within that range. As we progress through the experiment, we will compare each method to see which are more effective not only visually, but through the comparison of values obtained from the histogram plots as well.

4.2.2. Averaging Method

Here we use a method called averaging to continue our noise reduction process. The OpenCV averaging method is applied using “cv.blur(src, (x, y))”, where “src” is the source image and “(x, y)” denotes the kernel size used. It works similar to the convolution method where the kernel chosen is placed on top of the pixel of interest followed by multiplying the kernel with the pixels that fall under its area, followed by computing the average that replaces the central pixel with the computed pixels’ average

[23]. Ultimately, the difference in computing the average between the two methods lies in the fact that 2D convolution is a linear filter, while the averaging is a box filter.

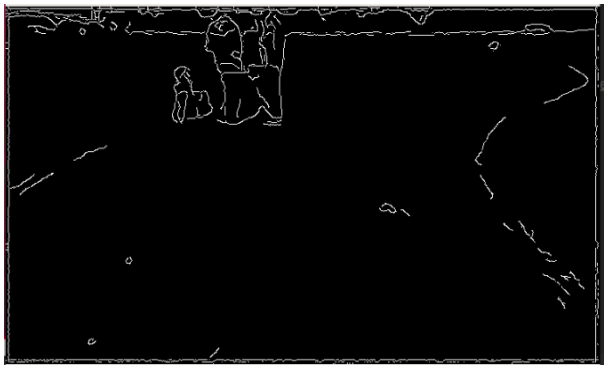

Figure 26. Averaging (HSV)

Figure 27. Averaging (HSL)

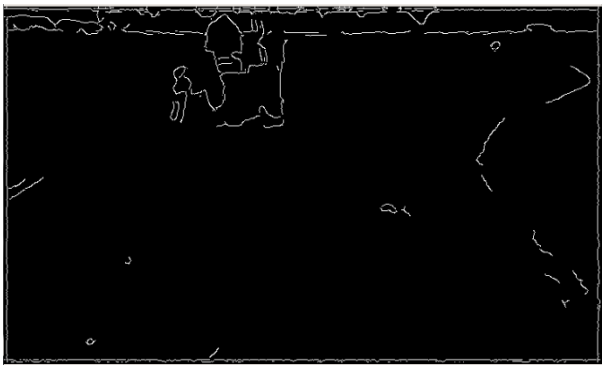

Figure 28. Averaging (grey)

Given that, we applied the method to see if it performs better than the 2-D convolution method used previously. From Figures 26 – 28 we can see that the averaging method reduced a great deal of noise from our image, however, we also lost much of our white line, which defeats the purpose for our experiment. We can also see from comparing the results that edge detection with the HSV color space retained more of the white line, while the noise of the grassy area between the white lines was reduced equally. However, given that the difference is not great, therefore it does not justify abandoning the other color spaces just yet.

We varied the size of the kernel again when using this method to find that a 3x3 kernel size retains much of the noise, while a 7x7 matrix results in a loss of the white lines, which lead us to use a 5x5 matrix, where in comparison it produces the best results. Therefore, we will look no further in terms of the kernel size for this specific method.

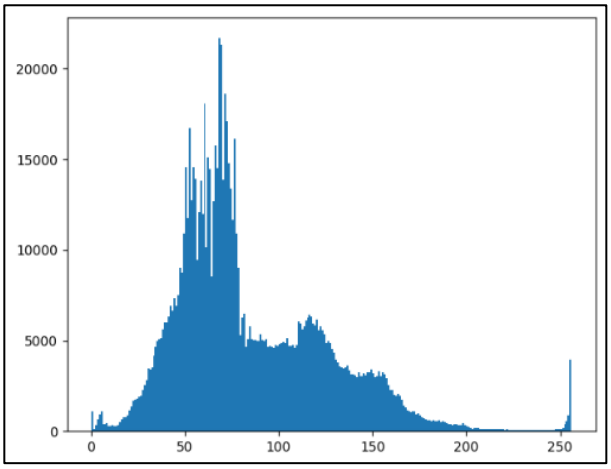

Figure 29. Avg-Hist (HSV)

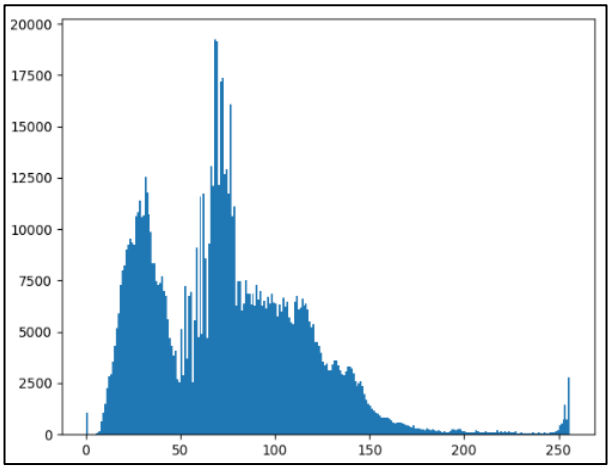

Figure 30. Avg-Hist (HSL)

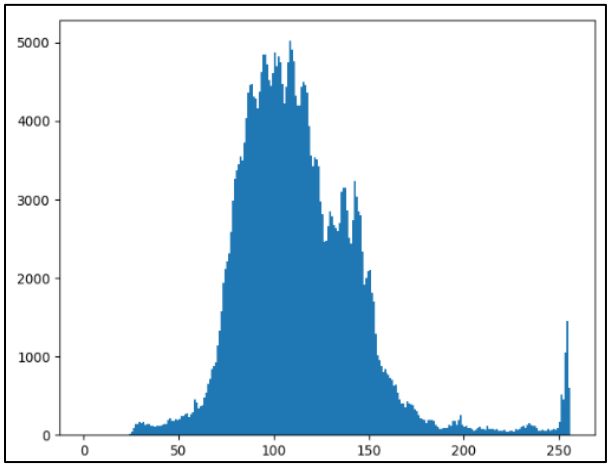

Figure 31. Avg-Hist (grey)

We also plot the histograms shown in Figures 29 – 31 for each color space after we have applied our smoothing method to compare the difference between the baseline and results after the application of the averaging method. A difference is noticeable while comparing to the baseline Figures, however not much difference was found between the 2-D convolution and the averaging methods.

4.2.3. Gaussian Blur Method

We move on to the next image smoothing method which is called Gaussian blur. The Gaussian blur method is a highly effective for removing gaussian noise from images. Gaussian noise is sensor noise that is attributed to poor illumination, high temperature, and transmission noise which is (electronic circuit noise). It works by using a spatial filter that takes the neighboring pixels and finds a weighted average that replaces the existing pixel values which ultimately reduces the high frequencies in fine-scaled images [23]. In this method we specify the size of our kernel that will be used, as well as the

standard deviation in x and y. We varied the kernel size and standard deviation for best results and reached the conclusions that the best suited kernel size is an 11x11 matrix, which was different from all the previous methods we used so far, while the standard deviation did not result in any noticeable difference so the values remained at 0.

Figure 32. Gaussian (HSV)

Figure 33. Gaussian (HSL)

Figure 34. Gaussian (grey)

Using the Gaussian method in all three-color spaces produced similar results as depicted in Figures 32 - 34, while for noise reduction and white line edge detection more was lost from the white line in the HSL and Grey, and the same was true for the previously used methods. Therefore, we might be inclined to use the HSV color space for the rest of our experiment, however, there are more methods that we will be using for image filtering before we can conclude which method will suit our application most. As indicated above that the best results were achieved using an 11x11 kernel size, while a 3x3 - 7x7 produced too much noise and a 13x13 resulted in a more than desired loss of the white line.

From the histogram plots in Figures 35 - 37, much different results obtained comparing to the previous methods. We can see that HSV, HSL, and grey peaks and gaps of pixel values found within the range 0 - 255 have been reduced which indicates that the altered pixel values became a part of other

pixel values within that range. Such results indicate that noise removal is highly effective while maintaining the white line, unlike the averaging method, where it was effective to remove noise, however, it resulted in higher than desirable loss of the white line.

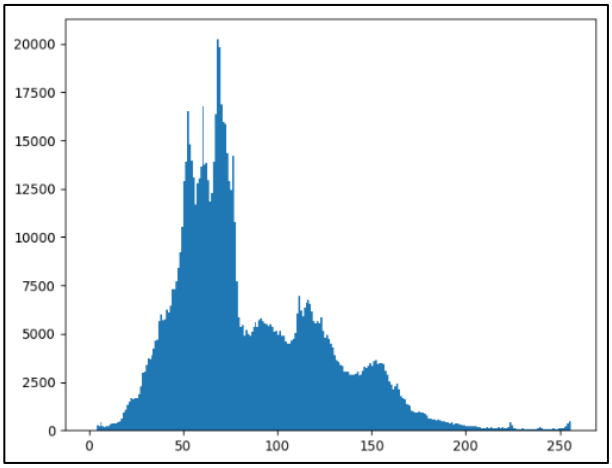

Figure 35. Gauss-Hist (HSV)

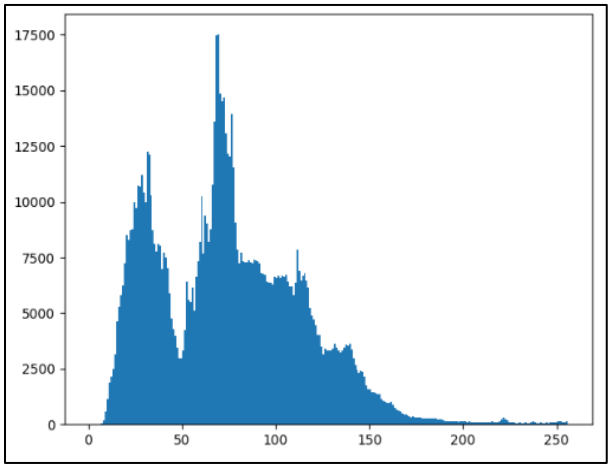

Figure 36. Gauss-Hist (HSL)

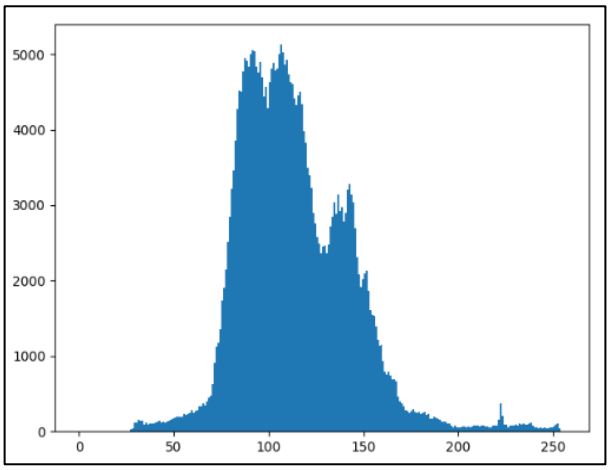

Figure 37. Gauss-Hist (grey)

4.2.4. Median Blur Method

Following the Gaussian blur method, we moved on to the median blur method, which is highly effective for salt and pepper noise removal. Unlike the previous methods used where an average was computed that replaced the central pixel, the median blur works by replacing the central pixel with a median value instead of an average value, while the rest of the method is the same [23]. In OpenCV, the median blur uses “cv.median(src, kernel)”, where “src” is the source image, while the kernel is the kernel size chosen for this method.

Once the kernel size is chosen, then a matrix multiplication of the two matrices is performed, followed by computing the median value and replace the value of the pixel of interest. The results from this method were favorable as depicted in the Figures 38 – 40. For this method we performed the same steps of varying the kernel size to arrive at the best results using an 11x11 matrix. However, for the median blur method, we should note that the method is slower than the methods used previously.

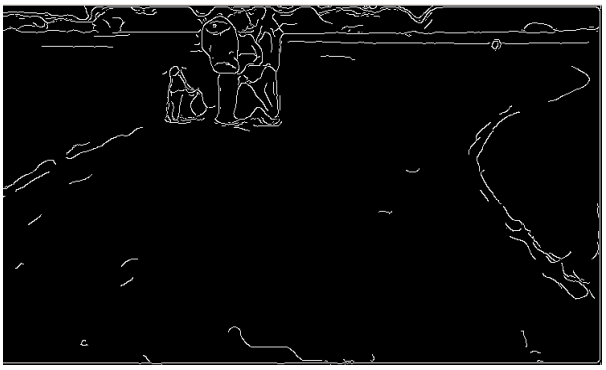

Figure 38. Median (HSV)

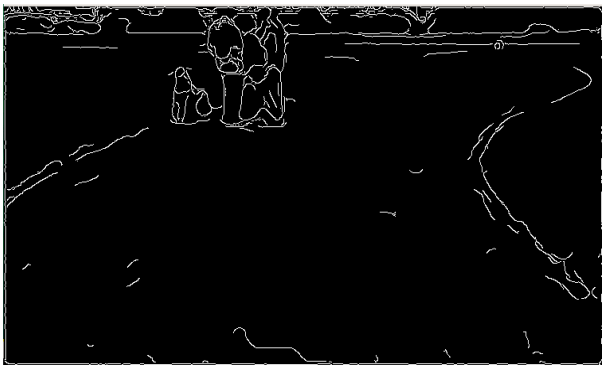

Figure 39. Median (HSL)

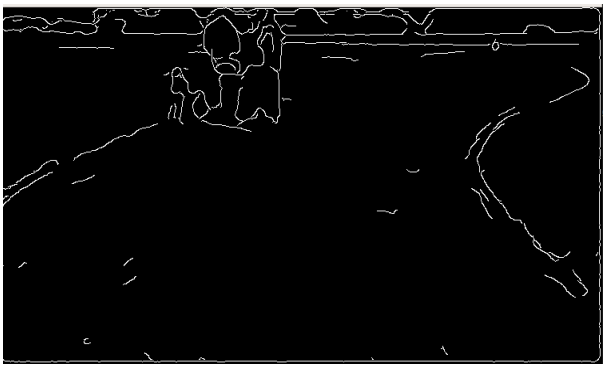

Figure 40. Median (grey)

Continuing with our comparison of the methods by plotting the histogram of the images for all color spaces, Figures 41 – 43 show the results obtained from the median method. We can see that the results are not as favorable as previously obtained from the Gaussian method - where peaks were reduced, and pixel values converged.

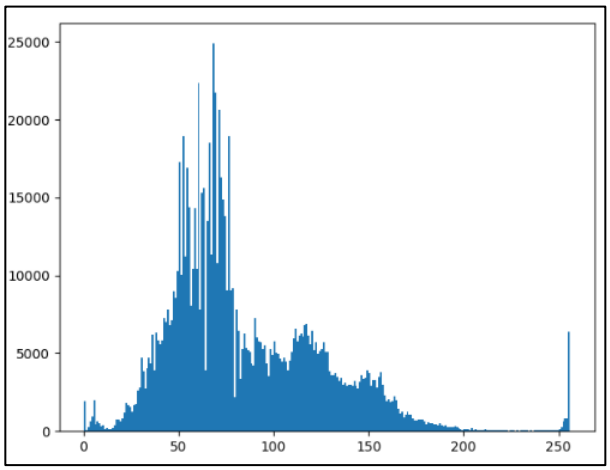

Figure 41. Median-Hist (HSV)

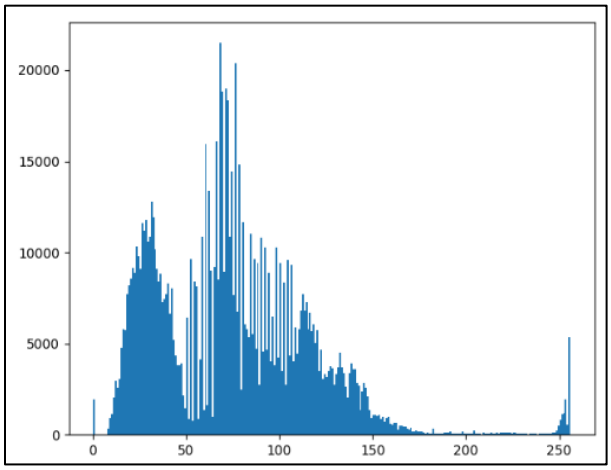

Figure 42. Median-Hist (HSL)

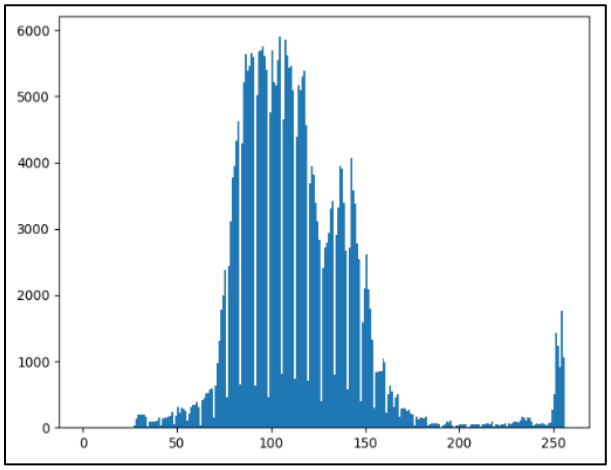

Figure 43. Median-Hist (grey)

4.2.5 Bilateral Filtering Method

Now we arrive at the final image smoothing method called bilateral filtering. The bilateral filtering is considered effective for removing noise while keeping edges intact. However, it is known to be slower than the other methods introduced above. It works by using the Gaussian method used above with an added functionality that considers only blurring neighboring pixels that have similar intensity to the central pixel, and therefore, the edge is preserved given that edge pixels have a large intensity variation [23]. In OpenCV is uses “cv.bilateralfilter(src, d, sigmaColor, sigmaSpace)”, where d is the diameter of the pixel neighborhood used for filtering, “sigmaColor” which is dependent on the value large or small will result in a larger or smaller area of semi-equal colors, and sigma space which also is dependent on the value large or small will influence the pixel value if their values are close enough. Therefore, when we set the value of “sigmaColor” to large and “sigmaSpace” to small we lose the edge of the white line as well as the

majority of the noise, while setting “sigmaColor” to small and “sigmaSpace” to large the obtained results are noisier. So, we set the values of “sigmaColor” and “sigmaSpace” to equal each other to balance the outcome as seen in Figures 44 – 46.

Figure 44. Bilateral (HSV)

Figure 45. Bilateral (HSL)

Figure 46. Bilateral (grey)

The results of the bilateral filter produced favorable results. As we can see from the three-color spaces depicted that the HSV color space tends to retain most of the white line when comparing to the returned HSL and grey color spaces after applying the bilateral filtering method. However, given that morphological transformation works with binary images, we will not settle for a specific color space just yet and will continue to work with all three-color spaces to see if there is much of a difference between converting the original image to grey or if there is added value in converting to the other color spaces then converting to grey.

Prior to moving on to morphological transformation methods in OpenCV, we will plot the histograms for each of the color spaces as we have been doing for comparison of the methods. As we can see from Figures 47 – 49 the histogram plots for the bilateral filtering method, the performance of the method provides better results in comparison to the median blur method, however, comparing to the

previously used methods, it seems that it does not provide as favorable results as the Gaussian blur does where the Gaussian reduce the singular peaks and has better convergence in pixel values.

4.3. Morphological Transformation Methods

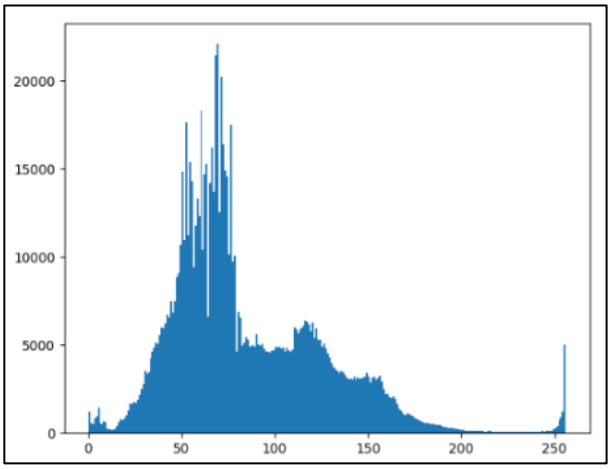

Figure 47. Bilateral-Hist (HSV)

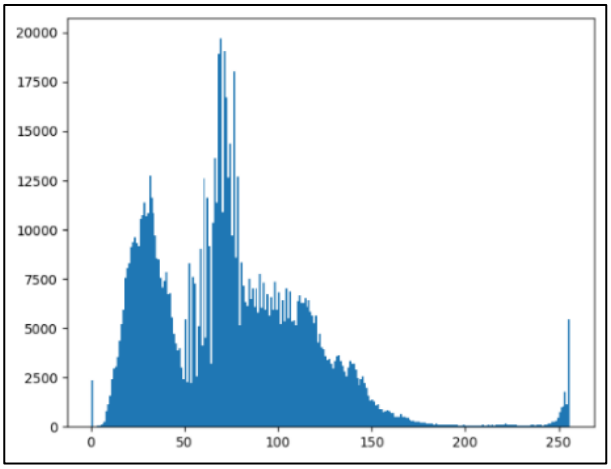

Figure 48. Bilateral-Hist (HSL)

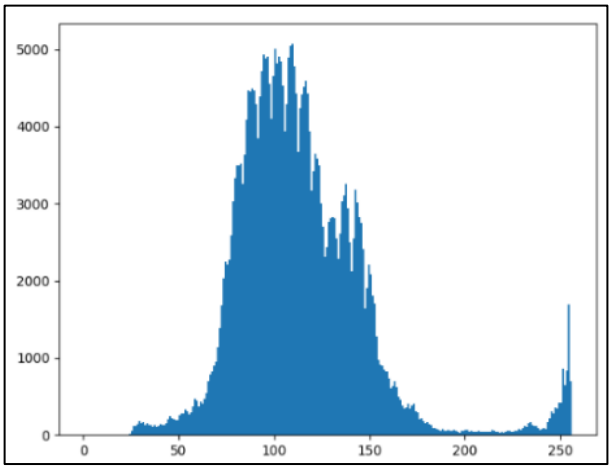

Figure 49. Bilateral-Hist (grey)

Morphological transformation algorithms are used to process geometrical structures based on the shape of the image. Previously it required the source image to be converted to binary prior to applying the method and if the image is not converted, it converted it automatically prior to the application of the method, however, that no longer is the case [17]. Therefore, we will use all color spaces when applying these methods. Depending on the method used, it works by removing pixels under the kernel that do not equate to 1 or adding pixels to increase the boundary of objects when pixels under the kernel equate to 1. The morphological transformation methods that we will use in this project includes erosion, dilation, opening, closing, and gradient.

4.3.1. Erosion Method

Erosion is the first morphological transformation method we will try. This method works by eroding/removing object regions' foreground boundaries, where for HSV and HSL color spaces it removes pixel values for each channel then merges them, while for the Grey color space it only has a single channel to work with. It works similar to the 2-D convolution method described above, where the kernel passes through the pixels in the image, however, rather than computing the average and replacing the pixel of interest value with the average, it searches for pixel value of 1, which means that for all pixels under the kernel size if all pixel values equate to 1, then the object remains, otherwise it is eroded [23].

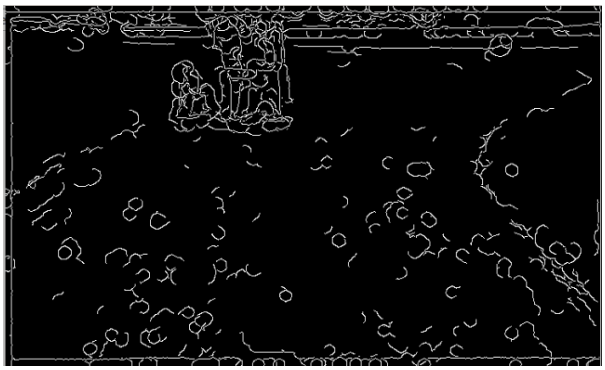

Figure 50. Erosion (HSV)

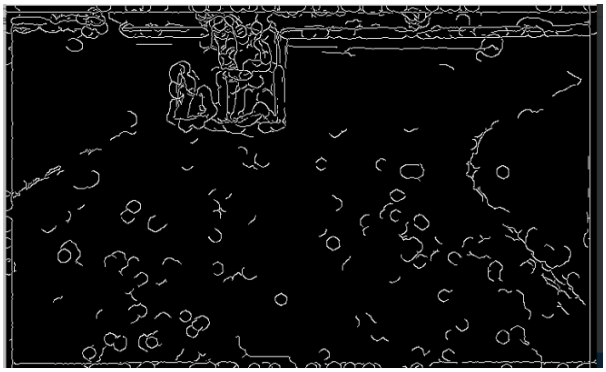

Figure 51. Erosion (HSL)

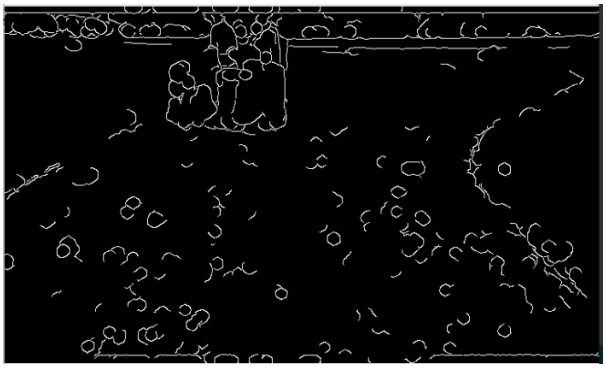

Figure 52. Erosion (grey)

From the obtained results of the application of the Erosion algorithm to the 3 color-spaces, we can see that the results are found similar, where noise in the images remained almost identical.

Now, for comparison of the methods used, as we have done previously, we plotted the image histograms for the three-color spaces. From Figures 53 – 55 we can see that our images still have a great deal of high peaks for singular pixel values, and it is not as effective as image smoothing methods, however, in comparison to our baseline in Figures 15 – 17, we can see that pixel values have converged and the range of high peaks have reduced which is indicative of some convergence/merging of color pixel values. This method, just as the previously introduced methods is not to be used as a standalone and solely relied upon to provide satisfactory results, however, as stated previously regarding other methods, the combination of this method along with others can provide better and possibly satisfactory results.

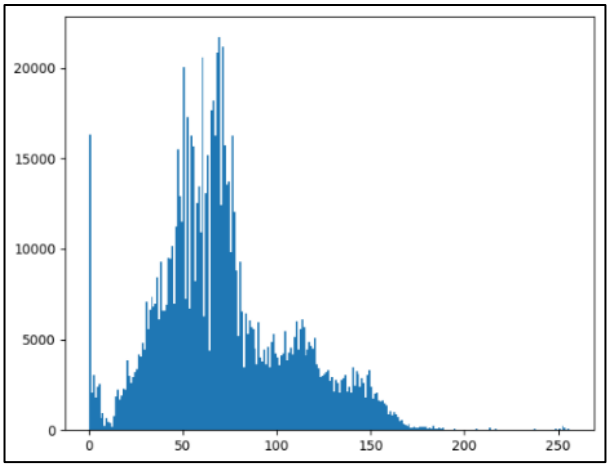

Figure 53. Erosion-Hist (HSV)

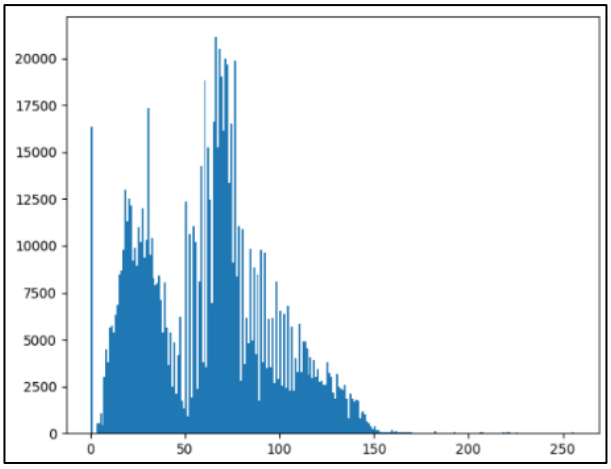

Figure 54. Erosion-Hist (HSL)

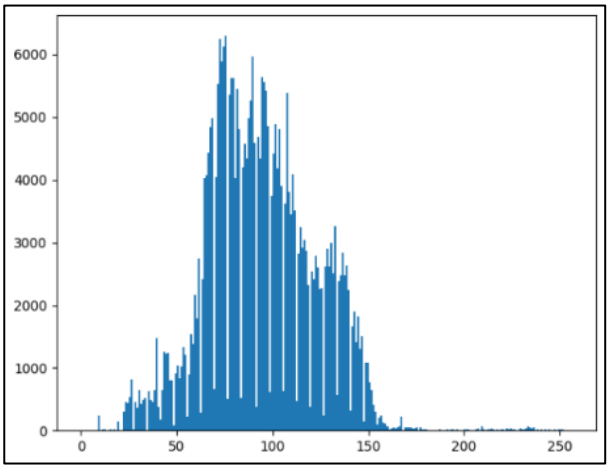

Figure 55. Erosion-Hist (grey)

4.3.2. Dilation Method

The dilation method performs the opposite of what was described for the erosion method. Rather than removing pixels in the foreground, the dilate searches for a pixel value of 1 under the kernel. If a pixel values of 1 are found, then the region increases under the kernel, otherwise it is ignored [23]. Given that our purpose to apply various methods to remove noise and detect white lines, the dilation should not be used as a standalone method since it will not help reduce noise in an image. However, for the purpose of understanding the outcome of the method we will apply dilation separately. Although we expect the noise level to increase rather than reduce as what occurred using the erosion.

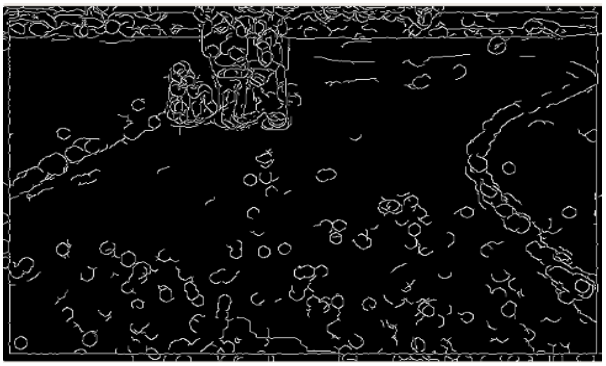

Figure 56. Dilate (HSV)

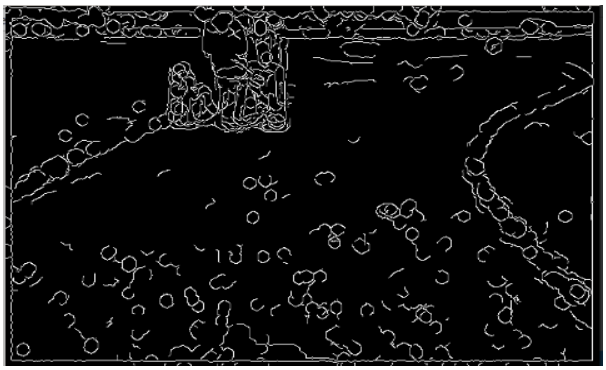

Figure 57. Dilate (HSL)

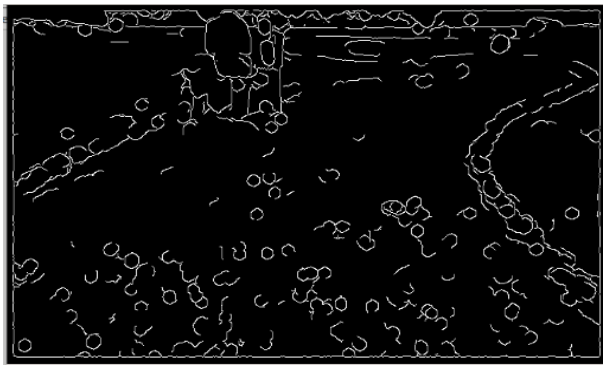

Figure 58. Dilate (grey)

As shown in Figures 56-58, using the Dilate method increases the Erosion, however, the white line has two edges instead of one.

From the image histogram plots for the dilate method we can similarly see an increase in pixel value peaks in comparison to the erosion method as seen in Figures 59 – 61. We can also see a tremendous increase in the pixel values ranging between 250 - 255 which accounts for the pixel values of white. This confirms the double edge for white lines in Figures 56 – 58 and can prove beneficial when

combined with other methods for Its detection (e.g. upon noise removal and retaining white line, result in increase of white line area).

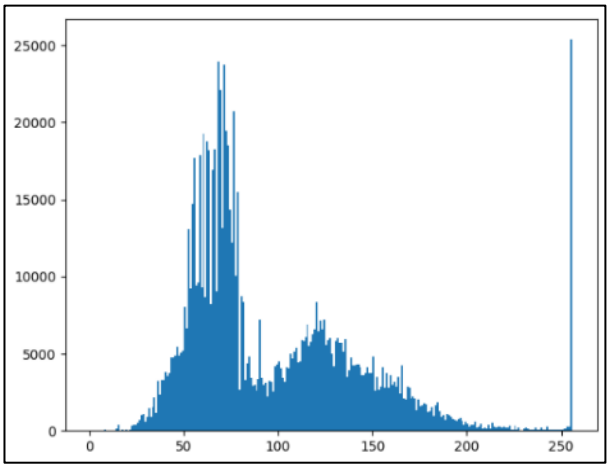

Figure 59. Dilate-Hist (HSV)

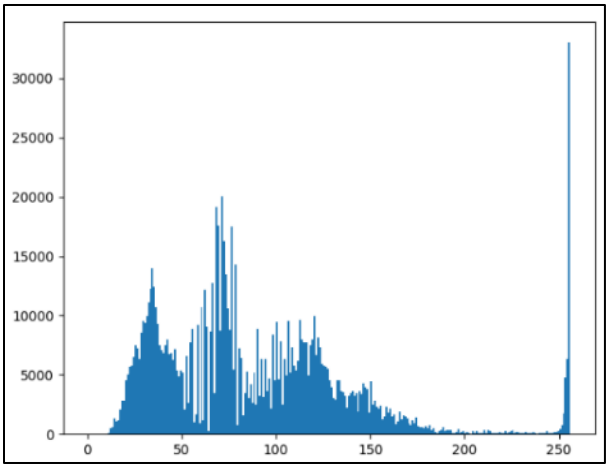

Figure 60. Dilate-Hist (HSL)

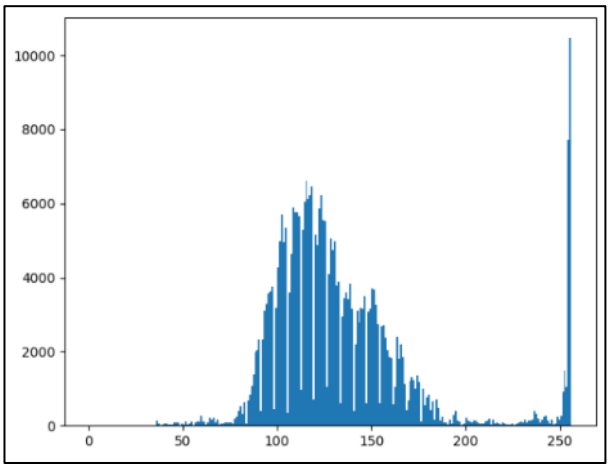

Figure 61. Dilate-Hist (grey)

4.3.3. Opening Method

The opening method is a method that performs erosion followed by dilation, where pixels' value of 1 are searched for within the image, if the value is found then pixels are eroded, upon completion, dilation is performed where it increases the area of the neighboring pixels if the value 1 is found [23]. This method is used to noise removal and increase the area surrounding the pixel of interest (e.g. white line).

We attempted the opening method where the results can be seen in Figures 62 – 64, which returned a higher level of noise, however reduced form circles as in erosion and dilation to specks, therefore, the gain we receive form implementing this method is the reduction of a two-step process of eroding the image first and followed by dilating it in a single step.

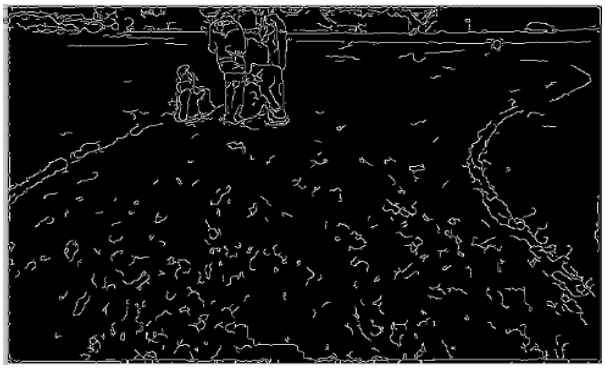

Figure 62. Opening (HSV)

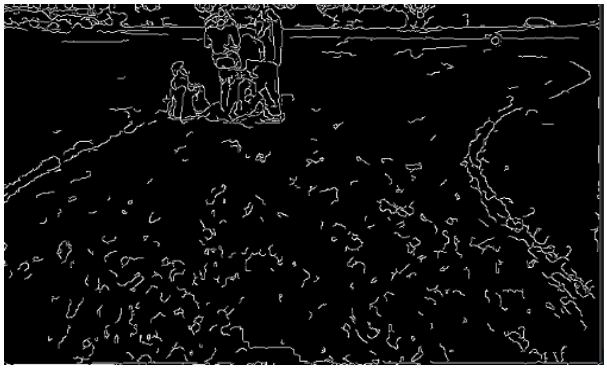

Figure 63. Opening (HSL)

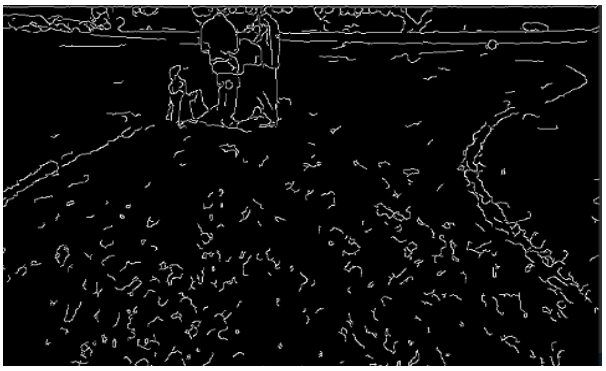

Figure 64. Opening (grey)

For comparison between morphological transformation methods, we can see from the histogram plots for all three-color spaces we can see an increase in peak value for singular pixels as shown in Figures 65 – 67. However, unlike the dilate method alone, given that this method applied erosion first then dilates the image, we do not see the same increase effect of the white line pixels therefore, this method may not be useful for us as the dilate method was.

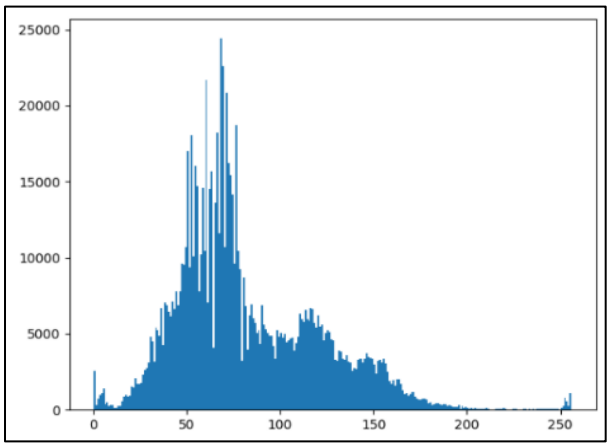

Figure 65. Opening-Hist (HSV)

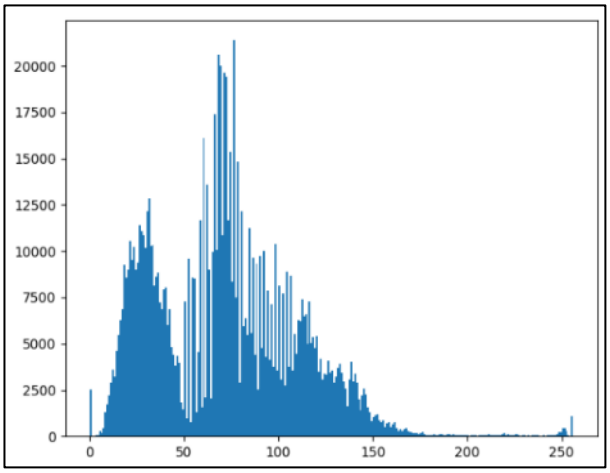

Figure 66. Opening-Hist (HSL)

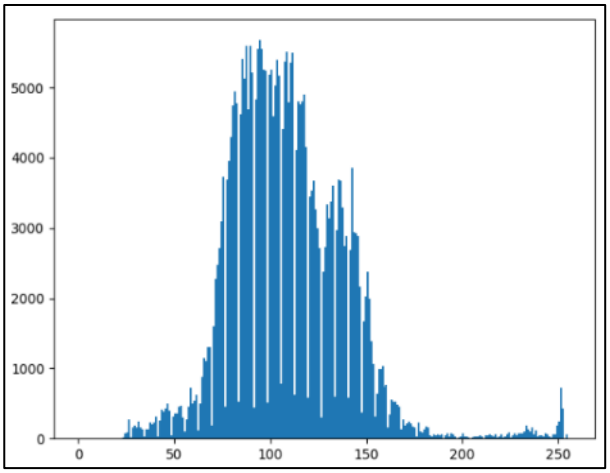

Figure 67. Opening-Hist (grey)

4.3.4. Closing Method

The closing method is the reverse operation of the opening method, where the operation of dilating the image occurs first, followed by eroding it [23]. Figures 68 – 70 show an increase in detected noise as edges, which was the case for the opening method as well, however, looking more closely at both outcomes from Figure 68 – 70 and 62 – 64, the closing method results in an increased area of the white line in comparison to the opening method. Also given the increase in noise, this method cannot be applied as a standalone for white line detection, however, it can be applied after other image processing methods are implemented. Such implementation of this method may produce desired results for our project.

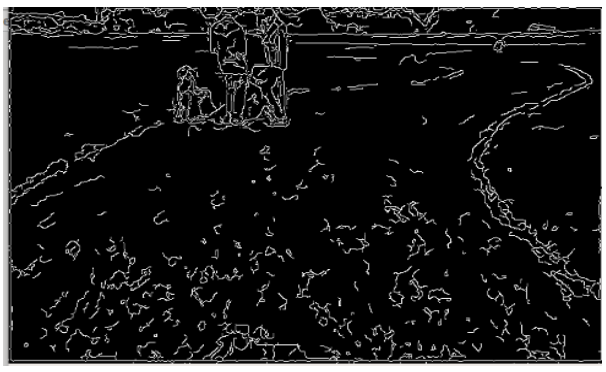

Figure 68. Closing (HSV)

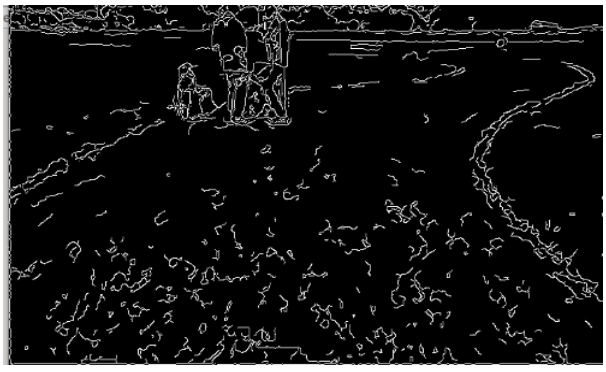

Figure 69. Closing (HSL)

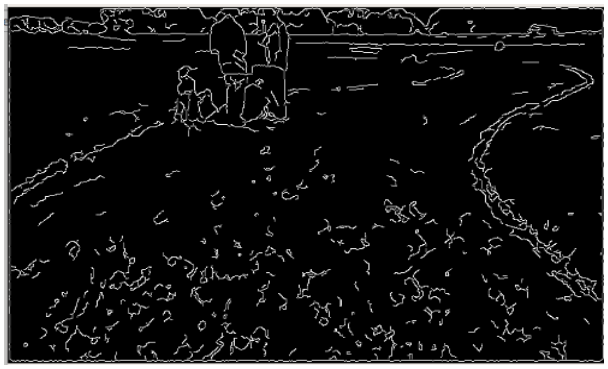

Figure 70. Closing (grey)

To better visualize what occurred with our image as seen in the applied canny edge detection, we plot the image histograms.

From the image histograms plotted for comparison as seen in Figures 71 – 73. The results of the histogram plots are similar to the histogram plots obtained from the opening method, however, looking more closely at the pixel values between the range 250 – 255, we can see an increase in pixel values peak for the white color, such a characteristic seem to potentially provide a desired result for us.

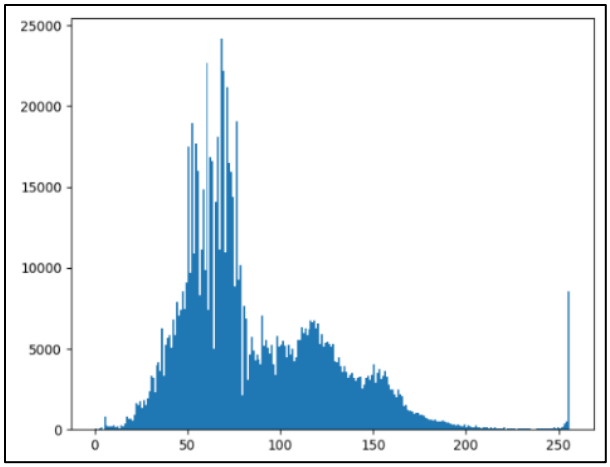

Figure 71. Closing-Hist (HSV)

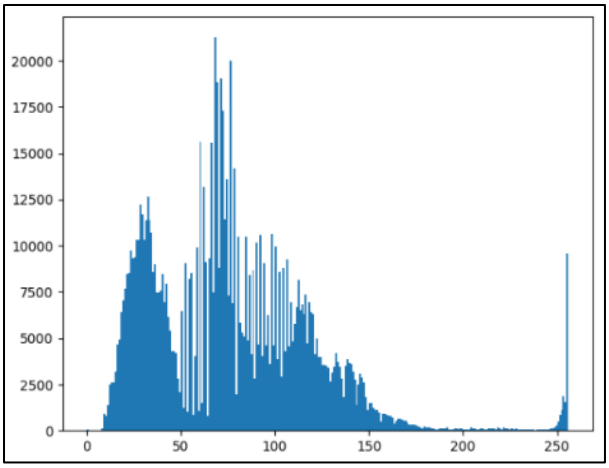

Figure 72. Closing-Hist (HSL)

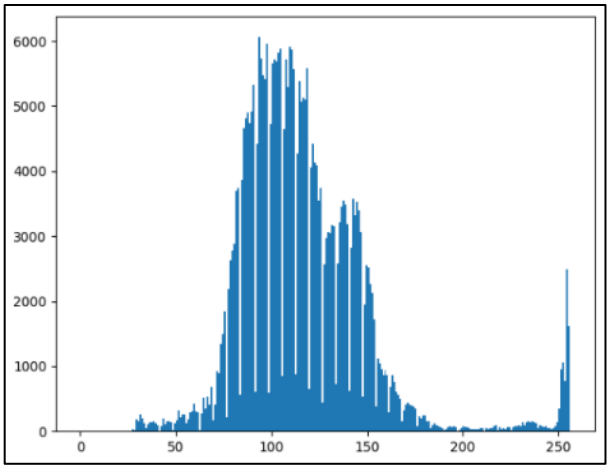

Figure 73. Closing-Hist (grey)

4.3.5. Gradient Method

The gradient method is the last method in the morphological transformation methods. It works by returning the difference between erosion and dilation methods. This difference returned is the result of eroding the central pixels while keeping the edges [23]. From this description we can say that using this method will result in an increase in contrast of the neighboring pixels. Therefore, given any noise remaining in the image, will result in the erosion of the central pixels, while giving the neighboring pixels the eroded values. This means that if we take the pixels under the kernel with value of 1, all the neighboring pixels will become 1, while the pixels under the kernel will become 0. We will attempt this method and compare results to conclude if it is of any value for our project.

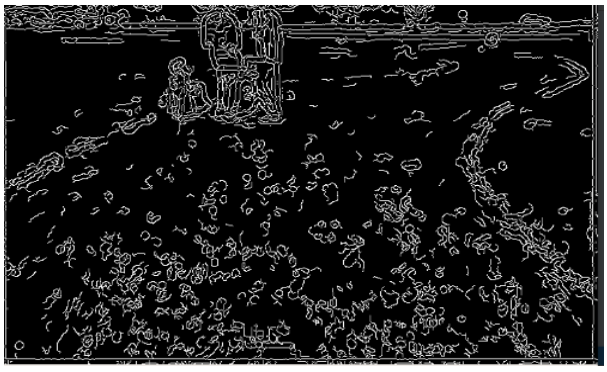

Figure 74. Gradient (HSV)

Figure 75. Gradient (HSL)

Figure 76. Gradient (grey)

Figures 74 – 76 show the results obtained using the gradient method. From the results obtained, noise reduction did not prove to be better than the opening method, however, once succeeding in removing the noise, this method can be used to white line detection for which it is often used and could prove beneficial for our purpose as well. Therefore, we will examine the image histogram plots for the Gradient method to view how the pixel values are altered using it.

From the image histograms for the gradient method as shown in Figure 77 – 79 depict the shift of the pixel values from a range of 2 – 255 to a range of 0 – 50 in all three-color spaces. Such an outcome is attributed to taking the difference between the erosion and dilation where the majority of pixel values took a value of the computed difference.

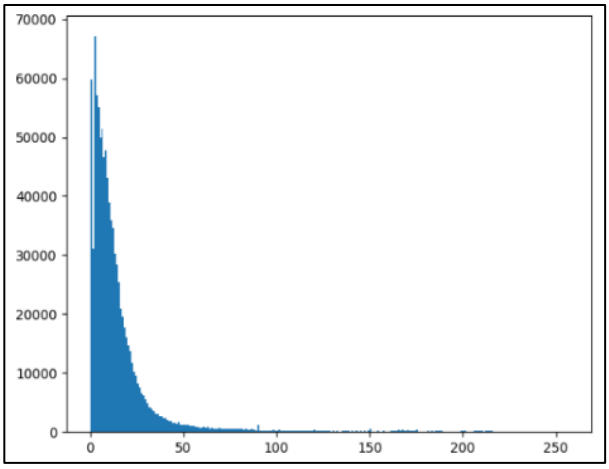

Figure 77. Gradient-Hist (HSV)

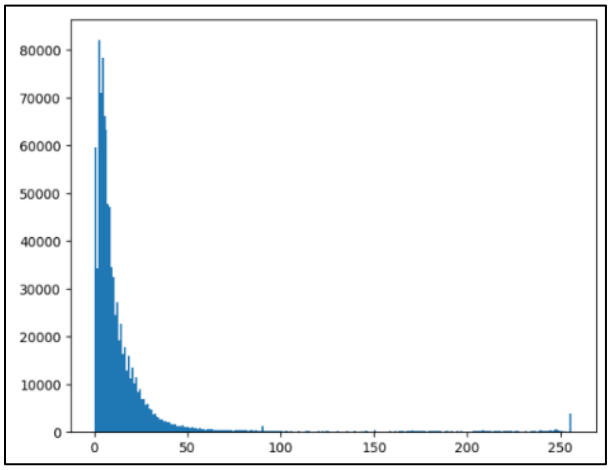

Figure 78. Gradient-Hist (HSL)

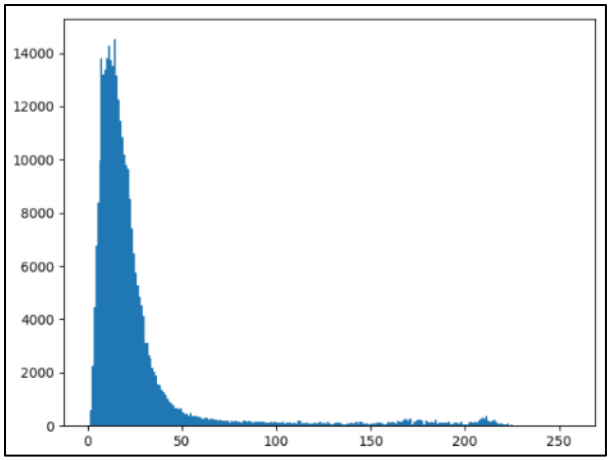

Figure 79. Gradient-Hist (grey)

Given such results, we have also plotted the images prior to applying the edge detection to view the results of the application of the gradient method. The results of the method can be viewed in Figure 80 – 82. From the images we can see that most of the colored pixels have been subtracted with mainly the edges remaining. Although much noise remained in the images as seen in Figures 74 – 76. However, the edges of the white line were preserved and can prove to be beneficial for our application.

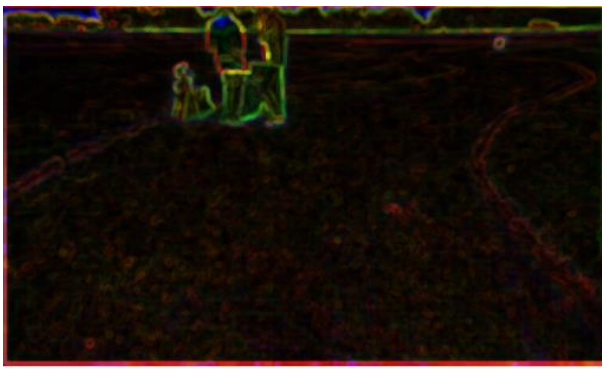

Figure 80. Gradient (HSV)

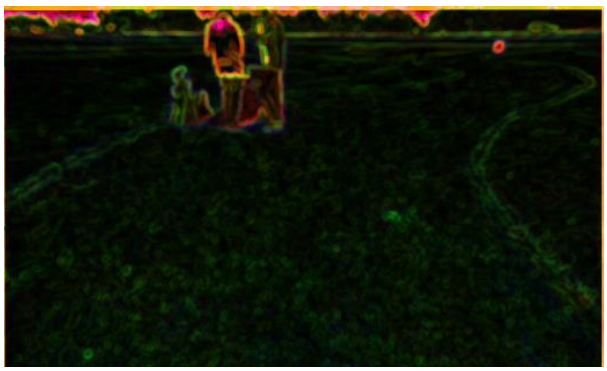

Figure 81. Gradient (HSL)

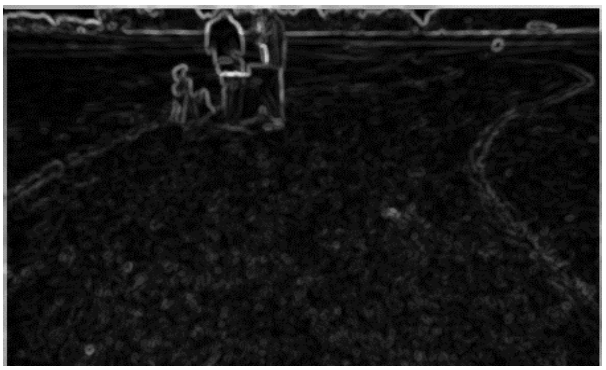

Figure 82. Gradient (grey)

Again, similarly to the dilate or erode methods, this method should not be applied as a standalone since it will not produce a noise free image but may potentially result in favorable results when combined with other methods.

4.4. Image Smoothing and Morphological Transformation Comparison

Now that we have applied all the image processing methods that we had in mind for which pixel value alterations occur and obtained numerical values from the histogram plots representing pixel values, we recorded the approximate values in a table for comparison between the baseline and all the image processing methods used. The table includes the methods used with the three-color spaces of interest. Each of the sections represents one color space and includes peaks of singular pixels' color value, the range for which peaks occur, and peaks for the white pixels at range 250 – 255.

Table 1. Histogram Plots' Peak Value Comparison for Image Smoothing Methods

	HSV	**Peak Range**	**White Peak**	**HSL**	**Peak Range**	**White Peak**	**Grey**	**Peak Range**	**White Peak**
Baseline	15000 – 24000	50 – 75	5400	12500 – 22000	60 – 70	7500	5000	75 - 150	2100
Convolution	15000 – 22500	50 – 75	4000	12500 – 19000	65 - 75	2500	4100-5000	85 – 125	1500
Averaging	15000 – 22500	50 - 75	4000	12500 – 19000	65 – 75	2500	3500 – 5000	75 – 125	1500
Gaussian	15000 – 20000	50 – 75	100	12500-17500	65 – 75	0	converged	75 – 125	50
Median	15000-25000	50 – 80	5000	15000 – 22500	65 – 75	5000	5000 – 5800	85 – 125	1900
Bilateral	15000 – 22500	50 – 75	5000	12500 – 20000	65 – 75	5000	4000 – 5000	75 – 125	1900

From the table we can see that a reduction in peak value from the baseline was the highest using the Gaussian blur method where the peak value for the number of pixels found at the color value was reduced by approximately 4000 for the HSV color space, 4500 for the HSL color space, and nearly converged all pixel values in Grey color space creating a gapless bell curve plot.

From the baseline histogram plots we could also visually see that pixel values did not distribute uniformly throughout the entire range of color values between 0 – 255 (e.g. 5000 pixels have a range value of 100). However, when we applied the image processing methods, we noticed that discrete color values tend to exhibit reduction by merging with neighboring color values. This behavior is expected,

however, not all methods created equal where outcomes differed. Of the five methods, median provides the worst results, while Gaussian the best, while the rest of the methods remain in between, however that is not to say that such results will be true for any conditions for which these methods are applied. From table 1 that can be confirmed about the median method where it has the highest peak for all color spaces used in this project.

Table 2. Histogram Plots' Peak Value Comparison for Image Morphological Transformation Methods

	HSV	**Peak Range**	**White Peak**	**HSL**	**Peak Range**	**White Peak**	**Grey**	**Peak Range**	**White Peak**
Baseline	15000 – 24000	50 – 75	5400	12500 – 22000	60 – 70	7500	5000	75 - 150	2100
Erosion	15000 – 22500	0, 50 – 75	0	12500 – 22500	0, 40, 50 – 75	0	4000 – 5000	75 – 125	0
Dilation	15000 – 25000	50 – 80, 255	25000	15000 – 20000	65 – 75, 255	25000	5000 – 6000	100 – 135	10000
Opening	15000 – 23000	50 – 80	1000	12500 – 20000	65 – 75	1250	5000 – 5500	85 – 125	900
Closing	15000 – 23000	50 – 75	8000	12500 – 22000	65 – 75	10000	5500 – 6000	85 – 125	2500
Gradient	70000	0 – 25	0	80000	0 – 25	5000	14000	15 – 25	0

For the morphological transformation methods shown in Table 2, although each method performs a different function and, therefore, they cannot be compared directly; however, we found value from comparing methods' pixel alteration capability from our images. For instance, the baseline row shows white pixel peak values are at 5400, 7500, and 2100 for HSV, HSL, and grey respectively, however, using the Dilation method, we can increase the white values to 25000, 25000, and 10000 for the three-color spaces. In addition, looking at the closing method, which as previously stated performs the dilation followed by erosion results in a peak white pixel values of 8000, 10000, and 2500 for the three-color spaces, and these results are post application of the erosion method. Such results show that these methods are also valuable for noise reduction as well as pixels' value alteration as we have seen the reduction in the singular peak color pixel values found in histogram plots.

Also, looking the opening method, we can see that it reduces the number of white lines tremendously, which is to be expected given that it applies the erosion method then followed by dilation. However, this allows us to conclude that applying the dilation method followed by erosion or the closing

method will result in higher contrast of the white line and reduces the potential of its loss (using HSV and grey), therefore, for our application it seems to have a positive outcome.

4.5. Masking Method

Color masking is the simplest method used for colored object tracking. It works by extracting the colored object from the rest of mage using threshold for a range of values of the color of interest. This is best used in HSV and HSL color spaces rather than the use of the BGR color model, since in BGR any color is constructed from a combination of the three primary colors which makes the extraction of a specific color more difficult [23]. Therefore, we attempted this method in two mentioned color spaces and compared the results between them to decide which of the color spaces is more suited for out application.

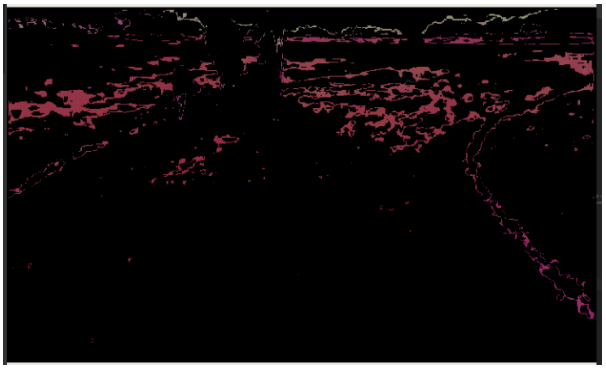

Figure 83. Mask (HSV)

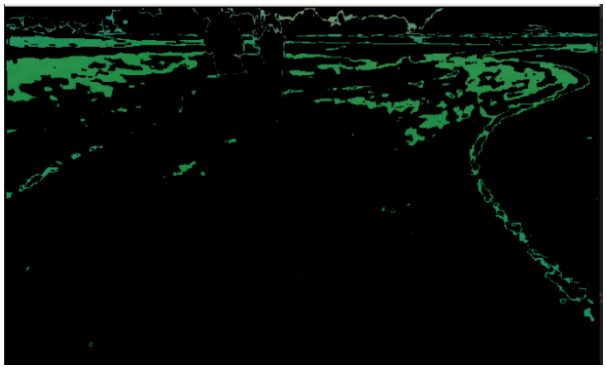

Figure 84. Mask (HSL)

As can be seen in Figures 83 and 84, the majority of the red and green colored areas in the HSV and HSL images respectively have been extracted, some regions remained, possibly with some noise reduction the results may prove beneficial to our project.

4.6. Edge Detection

There are multiple edge detection methods that we will use in our project to determine the best suited method. The methods include image gradient and the Canny edge detection methods where the image gradient includes Sobel, and Laplacian derivatives. Until now we have seen that the majority of images embedded in the report used some edge detection method, however, the method used was not yet introduced. Prior to introducing the method used previously, we will implement and discuss the other methods, compare their results, and conclude which methods suits our application most.

4.6.1 Sobel Edge Detection

The Sobel edge detection method is used to extract abrupt changes in the intensity of color within the image. It is a discrete differentiation operator that is applied at each pixel of the image, approximates its derivative, and separates the horizontal alignment by convolving with the image and returns edges at the point where the gradient is maximum [23]. We discussed more on this topic in the related works section for more detail, however, we will simply recall that it uses the horizontal and vertical matrices to perform the edge detection operation.

Figure 85. Sobel (HSV)

Figure 86. Sobel (HSL)

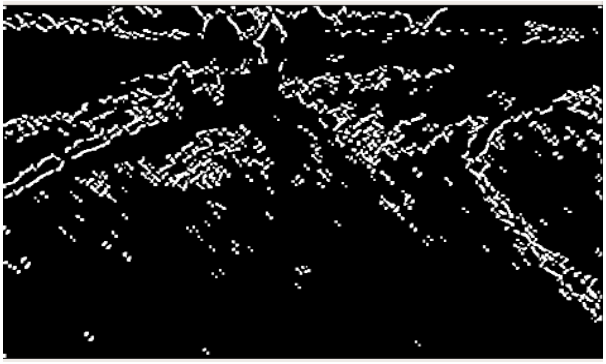

Figure 87. Sobel (grey)

From the results of this method shown in Figure 85 – 87, we can see that the Sobel method for edge detection does not provide suitable results without further image processing, however, we can see that after converting the image from HSV, HSL, and grey then using the threshold method to convert the image to binary, the best result obtained was using the grey color space.

4.6.2. Laplacian Edge Detection

Similar to the Sobel operator, the Laplacian operator is a derivative operator used for edge detection in an image. However, the difference is that the Laplacian operator takes the second derivative

of each derivative found using the Sobel derivatives. Also, unlike the Sobel operator which uses two kernels shown in Figures 88 and 89, where it uses one for horizontal edge detection the other for vertical edge detection, the Laplacian only uses one kernel that accounts for all orientations including diagonal edges as shown in Figure 90 [23].

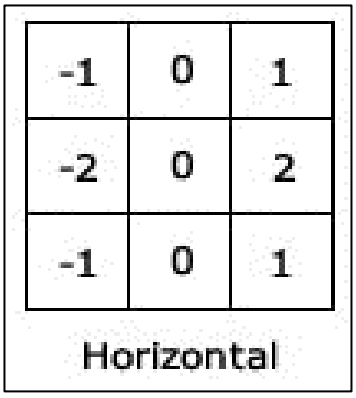

Figure 88. Sobel (H) [16]

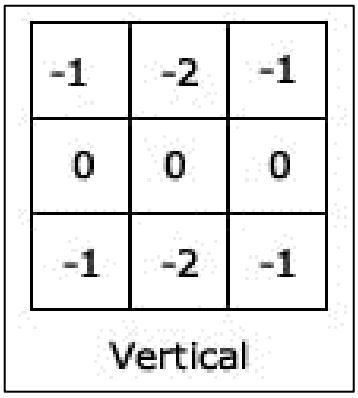

Figure 89. Sobel (V) [16]

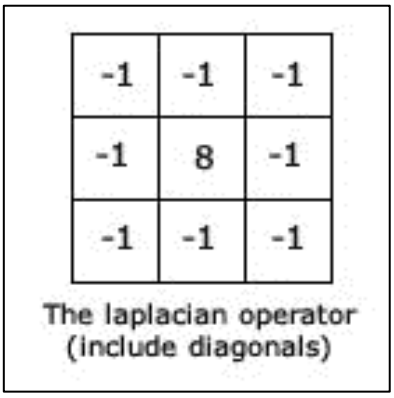

Figure 90. Laplacian [20]

Now that we established that the Laplacian's method uses a single kernel, we will apply the method to our color spaces and view the results of the applied method as shown in Figures 91 – 93.

Figure 91. Laplacian (HSV)

Figure 92. Laplacian (HSL)

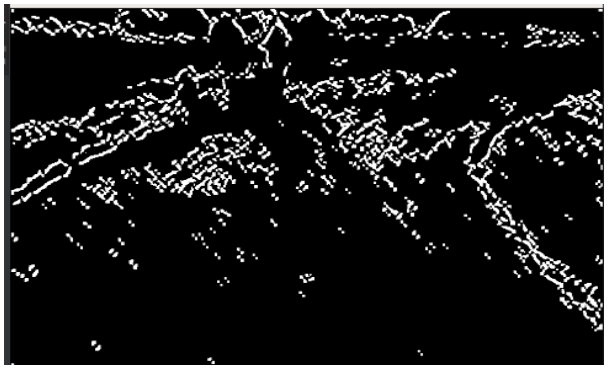

Figure 93. Laplacian (grey)

From the obtained results, other than the results obtained in Figure 93, the results are unfavorable, and more image processing will need to be applied prior to using the Laplacian, since much noise. However, prior to excluding any of the edge detection methods discussed so far, we will combine image smoothing and filtering, morphological transformation, and thresholding followed by the application of edge detection for the purpose of arriving at the best results.

4.6.3. Canny Edge Detection

From previous sections we have applied the canny edge detection method for the purpose of showing the noise found in an image upon the application of some image filtering techniques. Now we will discuss the Canny edge detection method used in more detail. This method is widely used for edge detection and works as a multi-stage algorithm, where initially an image smoothing technique is used and followed by finding the gradient of the image. In order to find the gradient, the Sobel kernel (Figure 88 and 89) in the horizontal and vertical orientation are applied to obtain the first derivatives. From the computed derivatives, the gradient is searched for in every pixel and every pixel's orientation.

Once the magnitude and direction are found, a full scan of the image is performed to remove any unwanted pixels where the local maximum is checked for. The local maximum is found by inspecting if the pixel value, its neighboring pixels, and determining if the pixel value forms a local maximum. Following the search for the gradient, a Hysteresis thresholding is applied, where the pixel of interest and pixels connected to it are checked if they pass through the maximum value, if yes, then an edge is detected, however, if the value fall within the maximum and minimum without intersecting the maximum value, then the value of the pixel is determined to be a non-edge and is discarded [23]. Since we have previously used the Canny method to show how the image processing methods worked by removing noise, we think that we have demonstrated enough results from the use of this method, and will not illustrate results by this method till a later section where we discuss a combination of the methods for image processing.

4.7. Combining All Methods

In previous sections, we have presented the results obtained from the application of each method individually. Now we will combine the various methods introduced until we arrive at the most suitable results for our application. This is done through a process of trial and evaluation, since the conditions

upon which we apply our methods continuously change and a single method will not produce the desired outcome.

Given the mixed results we obtained by applying the three-colors paces, and no single-color space produced the best results, we will continue to apply the image processing methods on all color spaces until we reach a different conclusion. Therefore, we applied the 2-D convolution and averaging methods as show in Figure 94 – 96.

Figure 94. Conv-Avg (HSV)

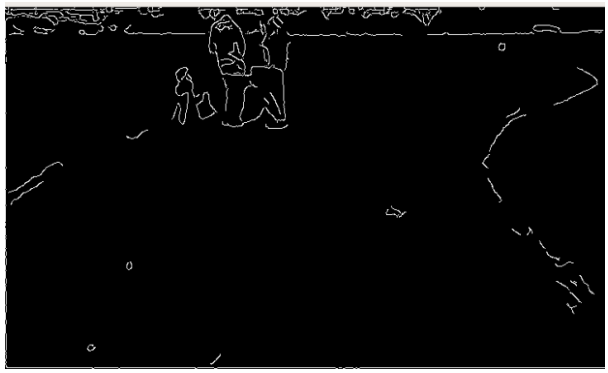

Figure 95. Conv-Avg (HSL)

Figure 96. Conv-Avg (grey)

From the results obtained and shown in the Figure 94 – 96 and comparing them with the single application of the convolution method or the averaging method alone from sections 4.2.1 and 4.2.2 it is evident that the use of both methods resulted in greater removal of noise, while maintaining most of the white line, however in the HSV color space more of the white line remains.

Next we combine the convolution method along with the Gaussian blur to determine if this combination produces better results. Using the Gaussian method, unlike previously we achieved good results using a kernel size of 11x11, however, given that the convolution method was applied prior to Gaussian, that kernel size is no longer adequate since much of the white line is lost. Therefore, we

reduced the size of the kernel to a 7 x 7 to maintain the detection of the white line while removing the majority of the noise from our image.

Figure 97. Conv-Gauss (HSV)

Figure 98. Conv-Gauss (HSL)

Figure 99. Conv-Gauss (grey)

As shown in Figure 97 – 99, we can see that interchanging the averaging method with the Gaussian did not produce much different results. therefore, next we will interchange the Gaussian method with the Median blurring method to see if the outcome is any different.

Figure 100. Conv-Median (HSV)

Figure 101. Conv-Median (HSL)

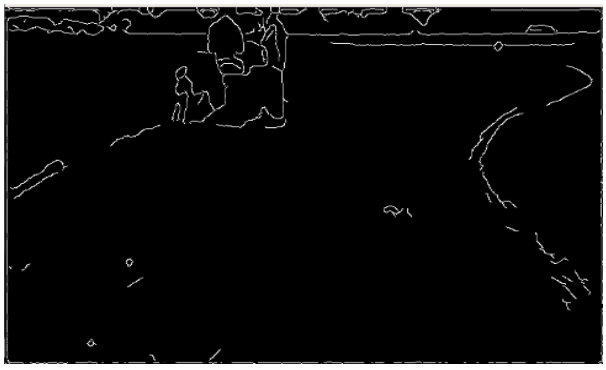

Figure 102. Conv-Median (grey)

Comparing the results in Figure 100 - 102 to the convolution-averaging and convolution-Gaussian methods' results, we can see that the outcome is less favorable using the convolution- Median method due to the added noise but resulted in a better white line retention. Also, we ran the video feed using our algorithm, it is evident that it is processing intensive; therefore, the playback of our processed image seems to have slowed down tremendously, which is also not favorable given that our autonomous vehicle must travel at a velocity between 1 and 5 miles per hour. We thought about reducing the kernel size at which the median blur processes the image, however, this introduces more noise to the image, therefore, it is not suitable either.

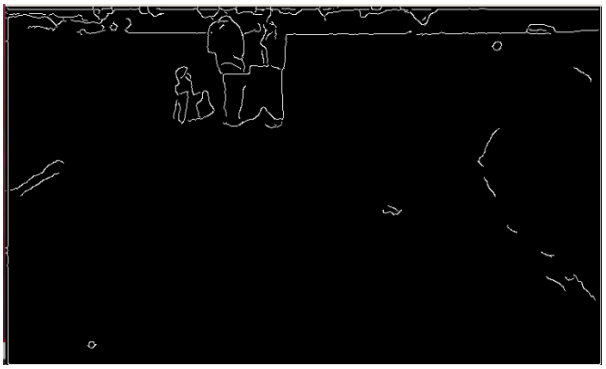

Figure 103. Conv-Bilateral (HSV)

Figure 104. Conv-Bilateral (HSL)

Figure 105. Conv-Bilateral (grey)

Next we will apply the combination of the convolution method and the bilateral blurring. As with the median blur method being an intensive image processing method, it is known that the bilateral filter also is a processing intensive method, however prior to discarding, we will make an attempt and compare our results.

Figures 103 – 105 illustrate the results obtained from the application of the convolution-bilateral filter method. We can see that the convolution-bilateral method results in greater erosion of white line, which is not a desirable outcome and therefore this combination will be discarded.

In the same fashion, we interchanged all the filtering and blurring methods (e.g. averaging and Gaussian blur, median blur and Gaussian, median blur and bilateral filter, etc.), followed by increase in color saturation, color masking, threshold, morphology transformation (gradient), canny edge detection and finally followed by dilation to increase the volume of the detected edges until we arrived at the final result that gave us the most satisfactory results for the edge detection using computer vision.

Figure 106. HSV

Figure 107. HSL

Figure 108. Grey

After many iterations and use of various methods, we finally arrived at a method that gave us the most satisfactory results. The methods we used precisely followed the following steps. We changed the default (RGB) color space to HSV, HSL, and grey color spaces, as depicted in Figures 106 - 108.

Following the conversion to the three-color spaces, next we applied the 2-D Convolution method to remove some of the noise that can be found in the image as we have seen previously when we applied this method followed by the edge detection to view the results. As indicated previously, the 2-D convolution method will convolve the image from left to right and top to bottom to replace the pixel of interest (central pixel) with a computed average of the pixels that fall within the kernel of a size used, which was a 3x3 matrix in our case. The results of the 2-D convolution method applied to the three-color spaces is shown in Figure 109 – 111.

Figure 109. Convolution (HSV)

Figure 110. Convolution (HSL)

Figure 111. Convolution (grey)

Comparing the images in Figure 106 – 108 with the results of the 2-D convolution cannot be done visually, however, as we have seen previously when we applied the Canny edge detection, much of the noise was reduced, where we will be able to see once we complete the application of the image smoothing.

Upon the application of 2-D Convolution method, we applied the Gaussian blur method to remove more noise from the image. The kernel used in the Gaussian method was of a size 11x11 where the Gaussian function is applied by multiplying each pixel value with values form the two-dimensional kernel that were computed using the Gaussian function which represents a normal distribution (bell curve).

Figure 112. Gaussian (HSV)

Figure 113. Gaussian (HSL)

Figure 114. Gaussian (grey)

The results of the Gaussian blur method application can be seen in Figure 111 – 113, where some of the sharpness of the image begin to disappear when compared to the image from the convolution method as well as the original images after the change to the three-color spaces.

We then followed by using the last method of image smoothing, which is the median blur. For this method we used a kernel size of 3, as we recall this method rather than replacing the central pixel with an average value, it replaces it with a median value of the pixels that lie under the area of the chosen kernel

size. The kernel size that was used for our method is a 3x3 matrix. The results of this method can be seen in Figure 115 – 117.

Figure 115. Median (HSV)

Figure 116. Median (HSL)

Figure 117. Median (grey)

Previously when we applied the median blur by itself we indicated that the median blur is process intensive, which was true given that for adequate results we had to use an 11x11 kernel size, however, given that our image has been processed with other methods, using a 3x3 matrix as we did here did not result in the same slow down as we experienced using an 11x11 matrix.

Now that we have applied all the image smoothing methods we needed, we move onto using morphological transformation methods. We will initially apply the dilate method to increase the area of the detected shapes in the image, which in our case is the white line mainly.

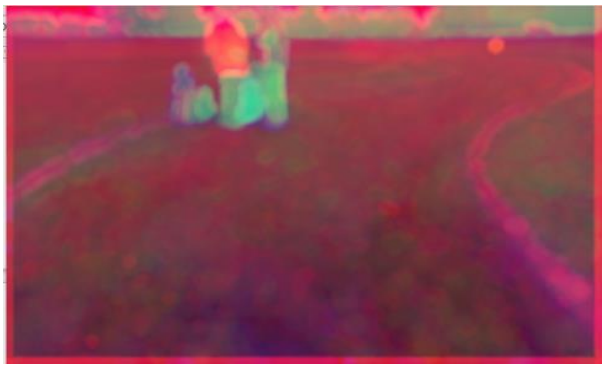

Figure 118. Dilate (HSV)

Figure 119. Dilate (HSL)

Figure 120. Dilate (grey)

As we showed in section 4.4 where we compared the histogram plots in tabular form. From the numerical values that we had recorded in table 2 indicate that performing dilation followed by erosion increases the pixels with the white color value due to dilation then results in a more continuous line after the application of erosion. Since our goal is to detect the white lines, we applied the steps based on conclusion made from the histograms' tabulated data.

For best results, we used a 5x5 kernel size and increased and repeated the process twice. As shown in Figure 118 – 120, we can that areas of the white line and the surrounding area have increased in size, which was the expected outcome.

Next we applied the erosion method to erode any value that does not equate to 1 under the kernel, in which case the pixels will be eroded. As stated above, the method works similar to the 2-D convolution method where this is applied to each pixel. Figures 121 – 123 show the result of erosion.

As we have concluded from the tabulated data in section 4.4 table 2 that dilation increases the white line area, while erosion will remove all the surrounding pixels if they do not equate to 1 under the

kernel. From Figures 121 – 123 and as we have suspected that upon the application of the dilation followed by erosion we will obtain a more continuous white line.

Now we can apply the next method called thresholding. The thresholding method returns our image from the video feed with colors that fall within the range that we set between 0 - 255. In our case it returned the range between 127 – 255. Which in our case for HSV the main color returned was red, for HSL was green, and for grey was white.

Figure 121. Erosion (HSV)

Figure 122. Erosion (HSL)

Figure 123. Erosion (grey)

From the histogram plots that we have seen in the previous sections, the HSV and HSL color spaces had their peaks below 100, while for grey the peak range was from 75 to 150. Upon the application of the threshold method, we isolated the peaks that were attributed to the grassy area between the white lines. This method is one of the easiest methods applied to isolate a color objects for object tracking, which in our case was white lines. Given the results from shown in Figures 124 – 126, we can now apply the edge detection method for white line detection.

The final step to our method is the application of edge detection. We applied the Canny edge detection method since it produced better results than the Sobel and Laplacian method as we have seen in the previous sections. Given the results of the threshold method in Figures 124 - 126, we expect that the HSL and Grey will have better results than the HSV since more red was retrieved using the HSV color space.

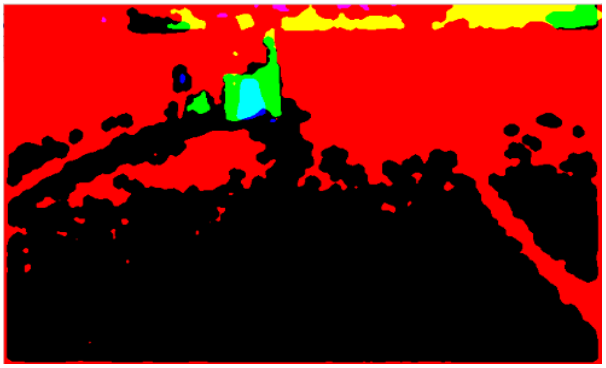

Figure 124. Threshold (HSV)

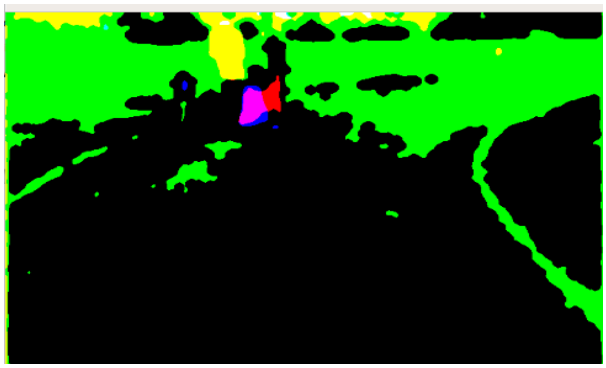

Figure 125. Threshold (HSL)

Figure 126. Threshold (grey)

From the results of the Canny edge method shown in Figures 127 - 129, we can see that the HSV has more edges retrieved in the grassy area in comparison with the HSL and grey color spaces. While we also can see that the grey has even less noise than the HSL. Therefore, the HSV and HSL color spaces will be abandoned, and we only used the grey color space to finalize our project.

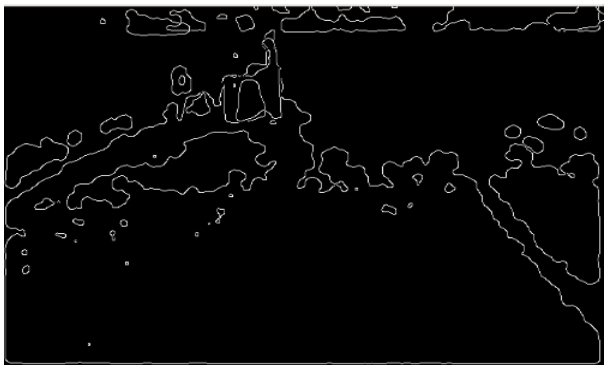

Figure 127. Canny Edge (HSV)

Figure 128. Canny Edge (HSL)

Figure 129. Canny Edge (grey)

Given the choice of the grey color space, we applied the dilation method to increase the white line area for its detection. The dilation method concludes our experiment with satisfactory results for the purpose of the white line detection for our autonomous vehicle. The final test to this method was during the competition, where we tested see our algorithm's performance at the onsite lighting conditions and any adjustments to the algorithm will have to be done onsite. Upon testing it on site, the results were satisfactory from the filtering of the image and the visibility of the white line and no further adjustments were made.

CHAPTER 5. RESULTS

From the forgoing, it is evident that computer vision is a complicated subject that is dependent on many external and internal (e.g. sensory devices) conditions to obtain desirable results. Conditions change dynamically and along with equipment quality and sensitivity to various conditions make the task more difficult, much trial and error and testing are necessary in order to arrive at a viable conclusion for a specific application. Therefore, depending on the application of computer vision and external conditions, various algorithms should be developed to work best for a particular condition.

In the previous competition (IVGC 2018), the method used was only applied to HSV color space without an extensive consideration to the other color spaces as we have done during this trial. In our final method we have applied multiple iterations of the 2-D convolution and Gaussian blurring, dilation, edge detection, and followed by more dilation for increase of while line area. Upon the arrival to this method, also much trial and error was needed. When the algorithm was tested in the pre-qualification runs during the competition, some tweaks to the code were necessary for conditions that were not encountered during development on test videos from previous competitions. From the results obtained during the competition runs it was evident that an improvement to the method was necessary, although the results still gave somewhat satisfactory results and we were able to complete the white line detection task.

In attempt to improve our results, the methods as presented in this paper were re-examined, tested, and then implemented. The methods used for re-examination included during this improvement project for image processing were image smoothing, morphological transformation, color masking, thresholding, and edge detection.

We initially attempted the masking method, since it is the easiest method applied for color object tracking, and since our goal was to detect the white line, we used the method to mask green colored pixels in the HSL color space, red colored pixels in HSV. However, color masking proved to be very sensitive to mask the range using both color spaces and resulted in a high removal of the white lines, which quickly proved to be inadequate for our application.

Given the inadequacy of the masking for our application, we have applied 5 different image smoothing methods (2-D convolution, averaging, gaussian, median, and bilateral) for noise removal. For each method we plotted the image histograms to understand the methods ability for noise removal and

pixel convergence. The Gaussian method had the best results not only reducing singleton high peaks but performed well blending pixel values across the color range 0 – 255. However, the Gaussian method alone could not result in the degree of noise removal desired. Therefore, in addition to the Gaussian we used the 2-D convolution method and the median at a smaller scale by reducing the matrix size used for the computation of the average to ensure that processing intensity is reduced as well.

Following the image smoothing methods used, we implemented the morphological transformation methods which include (erosion, dilation, opening, closing, and gradient). By plotting the image histograms for each of the methods and color spaces and recording distinct features in tabular form enabled us to see how pixel values were being altered by those method. Such an approach ultimately showed that the application of the dilation followed by erosion resulted in an increase of white colored pixels. Therefore, that led us to the final conclusion of using the dilation followed by erosion method, rather than the contrary which initially was believed to be the correct approach. The final result of the applied Morphological Transformation method resulted in a more continuous and saturated white line than what we commenced with.

Upon the results obtained from the morphological transformation methods, we applied the binary thresholding method, which ultimately resulted in the removal (conversion to 0) of all pixel values below a set value which in our case was 127. The application of this method resulted in a maximum removal of the grassy area without much loss of the white line. As seen in the images above that upon the application of the Binary Thresholding method, the area closer to the robot was processed to an adequate degree, while further away from the image contained the colored pixels that we attempted to mask, however, that did not hinder our results since as the robot traversed the course, the area closest to the robot removed most of the colored pixels.

At last we applied the Canny edge detection method which merely converted the image retrieved form the binary thresholding to lines that delineated the edges of the remained pixels. This ensured to remove all color from the image, for us only to detected edges comprised of white pixels. However, given that the algorithm for our autonomous vehicle searched for white lines between it was required to traverse the course, we applied the dilation method from the morphological transformation to increase the

thickness of the white line as well as edges of obstacles that were scattered throughout the course for easier detection.

Using the method above, we ran the autonomous vehicle during the pre-competition trial runs to determine if any adjustments were necessary prior to the actual runs during the competition. The live video feed from the ZED camera was processed and returned frames with distinct white lines and reduced noise for the autonomous vehicle to detect and for use for the path mapping algorithm. We visually evaluated the results. From this, it was deemed that the method produced satisfactory results and no adjustments were necessary.

CHAPTER 6. CONCLUSION AND FUTURE WORK

The methods used that were available to us in OpenCV-Python enabled us to remove a large portion of the noise retrieved from the video feed, however, some of the noise remained which we were not able to remove.

When the algorithm was processed on the video feed rather than a stationary image, there were locations of the white line that were lost, which is not a desirable effect. However, even when the image was processed to a minimum the same outcome was true, which indicates that the white line itself was drawn with varying degrees of density and therefore hinders adequate detection.

Given that, some locations resulted in an undetectable white line or it was detectable to a minimum degree upon which the attempt to remove noise, removed some of the white line. It was important to note that the application of the white line must be consistent, especially on a grassy field where various noise is introduced due to varying lighting conditions as well as varying shades of the grass.

So far, the results were satisfactory, most of the noise was successfully removed, while maintaining ability to detect the white lines' edges that delineated the course. However, until the algorithm is tested on site, during the competition with the weather conditions that we will have at the time, the results remain inconclusive and therefore, modifications to the code may still be necessary.

Future work considerations that are worth looking into include probabilistic Hough line transform for straight line detection, and Kalman filtering [19], and feature matching methods. The Hough line transform works to detect straight line segments by the application of multiple sinusoidal curves, and if intersected, then a line is detected by finding the number of intersections, which indicates that the more intersections occur, more points define the detected line. Threshold can be set to indicate minimum number of intersections required for consideration of a line. While the Kalman filtering method is an algorithm that uses existing measurements of previously obtained data to predict an estimate of future data which can potentially help in predicting the curvature of the white lines that delineate the course path. Feature matching can be used to compare an image of a line segment with the real-time video feed retrieved from our camera, if the image is matched to the video feed, then a white line is detected.

REFERENCES

[1] "Electromagnetic Spectrum - Introduction", *Imagine.gsfc.nasa.gov*, 2013. [Online]. Available: https://imagine.gsfc.nasa.gov/science/toolbox/emspectrum1.html.

[2] "RGB color space", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/RGB_color_space.

[3] "HSL and HSV", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV.

[4] T. Huang, "Computer Vision : Evolution And Promise", *CERN Accelerating Science (Illinois U., Urbana),* 2009. [Online]. Available: http://cds.cern.ch/record/400313.

[5] H. Palus, “Representations of colour images in different colour spaces,” in *The Colour Image Processing Handbook*, Boston, MA: Springer US, 1998, pp. 67–90.

[6] R. Serway, J. Jewett and V. Peroomian, *Physics for scientists and engineers with modern physics*, 9th ed. Boston, MA: Brooks/Cole, Cengage Learning, 2014.

[7] X. Wang, R. Hansch, L. Ma and O. Hellwich, "Comparison of Different Color Spaces for Image Segmentation using Graph-cut", *IEEE*, vol. 1, pp. 301-308, 2014.

[8] Ahmad, I. Moon and S. Shin, "Color-to-Grayscale Algorithms effect on Edge Detection – A Comparative Study", *IEEE*, 2018. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.ndsu.nodak.edu/stamp/stamp.jsp?tp=&arnumber=8330719.

[9] S. Dunn and J. Brown, *Radio Frequency Interference*. MIT, 2003. [Online]. Available: https://www.haystack.mit.edu/edu/pcr/RFI/RFI%20unit%20final%20AR3000A.pdf.

[10] L. Dong, J. Zhou and T. Dai, "Color Image Noise Covariance Estimation with Cross-Channel Image Noise Modeling", *2018 IEEE International Conference on Multimedia and Expo (ICME)*, 2018. Available: 10.1109/icme.2018.8486558.

[11] F. Bounini, D. Gingras, V. Lapointe and H. Pollart, "Autonomous Vehicle and Real Time Road Lanes Detection and Tracking", *2015 IEEE Vehicle Power and Propulsion Conference (VPPC)*, 2015. Available: 10.1109/vppc.2015.7352903.

[12] E. Perumal and P. Arulandhu, "Multilevel morphological fuzzy edge detection for color images (MMFED)", *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, 2017. Available: 10.1109/iceeccot.2017.8284680.

[13] N. Hayashi, T. Tomizawa, T. Suehiro and S. Kudoh, "A robust white line detection technique for double circular operator", *2012 IEEE International Conference on Mechatronics and Automation*, 2012. Available: 10.1109/icma.2012.6285120.

[14] Y. Zhang, X. Han, H. Zhang and L. Zhao, "Edge detection algorithm of image fusion based on improved Sobel operator", *2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2017. Available: 10.1109/itoec.2017.8122336.

[15] D. Patel and S. More, "Edge detection technique by fuzzy logic and Cellular Learning Automata using fuzzy image processing", *2013 International Conference on Computer Communication and Informatics*, 2013. Available: 10.1109/iccci.2013.6466130.

[16] S. Gupta and S. Mazumdar, "Sobel Edge Detection Algorithm", *International Journal of Computer Science and Management Research*, vol. 2, no. 2, 2013.

[17] C. Busch and M. Eberle, "Morphological Operations for Color-Coded Images", *Computer Graphics Forum*, vol. 14, no. 3, pp. 193-204, 1995. Available: 10.1111/j.1467-8659.1995.cgf143_0193.x

[18] K. Sreedhar, "Enhancement of Images Using Morphological Transformations", *International Journal of Computer Science and Information Technology*, vol. 4, no. 1, pp. 33-50, 2012. Available: 10.5121/ijcsit.2012.4103.

[19] M. Enjat Munajat, D. Widyantoro and R. Munir, "Vehicle detection and tracking based on corner and lines adjacent detection features", *2016 2nd International Conference on Science in Information Technology (ICSITech)*, 2016. Available: 10.1109/icsitech.2016.7852641.

[20] U. Sinha, "The Sobel and Laplacian Edge Detectors - AI Shack", *Aishack.in*, 2019. [Online]. Available: http://www.aishack.in/tutorials/sobel-laplacian-edge-detectors/.

[21] Belongie, S., Malik, J. and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4), pp.509-522.

[22] Hariharan, B., Arbelaez, P., Bourdev, L., Maji, S. and Malik, J. (2011). Semantic contours from inverse detectors. *2011 International Conference on Computer Vision*.

[23] OpenCV: Introduction to OpenCV-Python Tutorials, 2019. [Online]. Available: https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

[24] Numpy – Numpy, 2019. [Online]. Available: https://www.numpy.org

[25] Matplotlib: Python plotting – Matplotlib 3.1.1 documentation, 2019. [Online]. Available: https://matplotlib.org

[26] General Python FAQ – Python 3.7.4 documentation, 2019. [Online]. Available: https://docs.python.org/3/faq/general.html