Image Processing: Working with the Open-Source Library OpenCV

Iman Abu-Khdair
Department of Computer Science and Engineering
University of South Florida *Tampa, Florida, USA*
iabukhdair@usf.edu

***Abstract*—An assignment showcasing eleven computer vision functions applied to either the entire chosen image or just specified area(s). The functions added implement histogram stretching, histogram equalization, Hough Transform for circles (with and without the Canny module), the Otsu method (alone, and in combination with histogram equalization, Gaussian Smoothing, and the last two combined with each other and Otsu), and finally QR code detection and decoding with and without Gaussian smoothing.**

Keywords—computer vision, histogram stretching, histogram equalization, Hough transform, Gaussian smoothing, Otsu algorithm, QR code detection.

I. INTRODUCTION

This assignment focuses on edge detection of grey-level and color images by using the Sobel Operator, Otsu algorithm, Gaussian blurring, and the Canny algorithm with and without the OpenCV library. This report is organized as follows. Section 2 details the intuition of the algorithms implemented for the added edge detection functions named “histostretch”, “hsitoequ”, “houghcir”, “cannyhough”, “smoothedge”, “otsu”, “otsuhisto”, “otsugau”, “combine”, and the additional functions “QRcode” and “QRcodepre”. Section 3 will display and explain the results for the assignment. Section 4 explains and compares the performances of edge detection on grey level and color images, utility of OpenCV, and the difference in performance between the Canny edge detector and the Sobel edge detector. In Section 5 we finalize the paper with a conclusion.

II. ALORITHMS USED

In this section, the intuition of the edge detection functions labeled *histostretch, histoequ*, *houghcir*, *cannyhough*, *smoothedge, otsu*, *otsuhisto*, *otsugau*, *combine*, and the additional functions *QRcode*, and *QRcodepre* are described. Images of the famous action movie actor Jackie Chan, the popular image used in image processing - Lenna, the beloved animated character Lord Farquaad from the animated film series Shrek, our favorite baboon, and a still life of a glass of wine and some berries are the subjects of manipulation to the added functions.

A. Histogram Stretching

In OpenCV, there is no obvious histogram stretching operation function, so to carry out this particular task it is necessary to combine a few given functions. To achieve this, we must first convert the given source image into a gray scale image by using the cvtColor() function with the parameters of the source matrix, target matrix, and the conversion name. In this instance, we use the COLOR_BGR2GRAY conversion to establish the grayscale value, and a temporary matrix to keep the grayed values of the source image. The next step was essentially the histogram stretching effect, which was performed by utilizing the built-in normalize() OpenCV function. This function takes on the grayed image, the final output or target image, the minimum value by which to normalize, the maximum value by which to normalize, and the standardized NORM_MINMAX value imposed. In our case, this function would appear as such: normalize(grayedImage, tgt, 0, 255, NORM_MINMAX); The results of this histogram stretching function are demonstrated by Figures 1 – 5 on various images. The difference in the gray values between the original grayed images and the stretched images is most visible between the images of Lenna, the Baboon, and the still life.

B. Histogram Equalization

Similar to the Histogram Stretching section, we must use a combination of given functions to achieve our desired goal of histogram equalization. To achieve this, we must first convert the given source image into a gray scale image by using the cvtColor() function with the parameters of the source matrix, target matrix, and the conversion name. In this instance, we use the COLOR_BGR2GRAY conversion to establish the grayscale value, and a temporary matrix to keep the grayed values of the source image. Thankfully, there is a built-in function just for equalization of images called equalizeHist(), but this must take in a gray scale image, so that is why our first step is to create a .pgm version of our original image. This simple equalizeHist() function takes on only two parameters – the source image, and the target or output image. The results of this histogram equalization function are demonstrated by Figures 6 – 10 on various images. The difference in the gray values between the original grayed images and the stretched images is visible between all the images.

C. Hough Transform on Circles

The Hough Transform on Circles task proved to be rather challenging as it required a longer process of planning. As the previous two functions, we must first convert the original image to a gray scale image by using the cvtColor() function with the parameters of the source matrix, target matrix, and the conversion name. Then we create a vector by which we store the circles we will create, use the HoughCircles() function in OpenCV, use the proper parameters, and then loop through the entire image trying to identify any circular objects that have a great enough contrast between the foreground and background regions of the image. This is implemented by a while loop that draws white circles around the objects that are detected by the program. The final step is the cloning of the produced image to the target or output file determined by the parameter file.

D. Hough Transform After Canny Edge Filter

The Hough Transform on Circles After Canny Edge Filter task proved to be less challenging as it required a similar process of planning, with just an extra step. As the previous functions, we must first convert the original image to a gray scale image by using the cvtColor() function with the parameters of the source matrix, target matrix, and the conversion name. Then we use the built in Canny() edge detection function with the parameters being the grayed image matrix, the temporary output matrix, and the two thresholds by which the detector should follow. After that, we must create a vector by which we store the circles we will create, use the HoughCircles() function in OpenCV, use the proper parameters, and then loop through the entire image trying to identify any circular objects that have a great enough contrast between the foreground and background regions of the image. This is implemented by a while loop that draws white circles around the objects that are detected by the program. The final step is the cloning of the produced image to the target or output file determined by the parameter file.

E. Edge Detection and Smoothing Filter

The Hough Transform on Circles After Gaussian Smoothing and Canny Edge Detection task proved to be less challenging as it required a similar process of planning, with just an extra step. As the previous functions, we must first convert the original image to a gray scale image by using the cvtColor() function with the parameters of the source matrix, target matrix, and the conversion name. Secondly, we perform Gaussian smoothing by implementing the built in GaussianBlur() function with the parameters of the grayed image matrix, the temporary output image matrix, the kernel size by which we blur (I chose a 5x5 window), and the standard deviations in the x and y for the kernel size (I chose to use the value zero for both parameters). Then we use the built in Canny() edge detection function with the parameters being the grayed image matrix, another temporary output matrix, and the two thresholds by which the detector should follow. After that, we must create a vector by which we store the circles we will create, use the HoughCircles() function in OpenCV, use the proper parameters, and then loop through the entire image trying to identify any circular objects that have a great enough contrast between the foreground and background regions of the image. This is implemented by a while loop that draws white circles around the objects that are detected by the program. The final step is the cloning of the produced image to the target or output file determined by the parameter file.

F. Otsu Algorithm

The Otsu thresholding algorithm, just as it sounds, applies a threshold to an edge detection method (typically the Sobel-Feldman operator, in which we use for this program). Essentially, the effect of this method is to first find the edges within the image with the Sobel() function using the desired kernel size, and then using that output image to create a gray level image that is binarized to black and white pixels depending on whether the pixel values are above or below the chosen threshold value. In this instance we are using the threshold() function which, in order, has the parameters input image, output image, 0, 255, and the threshold (which we use THRESH_OTSU). The values 0 and 255 are used because our input image will be a gray level image which ranges from pixel values 0 to 255.

G. Histogram Equalization and Otsu Algorithm

The Otsu thresholding algorithm with Histogram Equalization applies a threshold to an edge detection method (typically the Sobel-Feldman operator, in which we use for this program) after firstly performing histogram equalization. Essentially, the effect of this method is to first find the edges within the image with the Sobel() function using the desired kernel size, and then using that output image to create a gray level image that is binarized to black and white pixels depending on whether the pixel values are above or below the chosen threshold value. In this instance we are using the threshold() function which, in order, has the parameters input image, output image, 0, 255, and the threshold (which we use THRESH_OTSU). The values 0 and 255 are used because our input image will be a gray level image which ranges from pixel values 0 to 255. The effect this produces is not necessarily *drastically* different from the original Otsu method, but it is noticeable that there is much less noise and much greater detail on the more prominent areas of the images. The results that look best with this were Figures 21 and 23, Lord Farquaad and the still life.

H. Gaussian Smoothing and Otsu Algorithm

The Otsu thresholding algorithm with Gaussian Smoothing applies a threshold to an edge detection method (typically the Sobel-Feldman operator, in which we use for this program) after firstly performing Gaussian Smoothing. Essentially, the effect of this method is to first find the edges within the image with the Sobel() function using the desired kernel size, and then using that output image to create a gray level image that is binarized to black and white pixels depending on whether the pixel values are above or below the chosen threshold value. In this instance we are using the threshold() function which, in order, has the parameters input image, output image, 0, 255, and the threshold (which we use THRESH_OTSU). The values 0 and 255 are used because our input image will be a gray level image which ranges from pixel values 0 to 255. The effect this produces is slightly more different from the original Otsu method and the previous section, but it is noticeable that there is much less noise, much greater detail on the more prominent areas of the images, and thicker outlines on the important parts of the images. The results that look best with this were Figures 24, 25 and 28 – Jackie Chan confused, Lord Farquaad, and the still life.

I. Combining Gaussian Smoothing, Histogram Equalization, and Otsu Algorithm

The Otsu thresholding algorithm with Gaussian Smoothing and Histogram Equalization applies a threshold to an edge detection method (typically the Sobel-Feldman operator, in which we use for this program) after firstly performing Gaussian Smoothing and then histogram equalization. Essentially, the effect of this method is to first find the edges within the image with the Sobel() function using the desired kernel size, and then using that output image to create a gray level image that is binarized to black and white pixels depending on whether the pixel values are above or below the chosen threshold value. In this instance we are using the threshold() function which, in order, has the parameters input image, output image, 0, 255, and the threshold (which we use THRESH_OTSU). The values 0 and 255 are used because our input image will be a gray level image which ranges from pixel values 0 to 255. The effect this produces is VERY different from the original Otsu method, as it produces a comic book like images. There are large areas of white and black, smooth textures where they used to be more dotted or detailed. The results that look best or most polarizing with this were Figures 29, 31, and 32, Jackie Chan confused, Lord Farquaad, and the Baboon image.

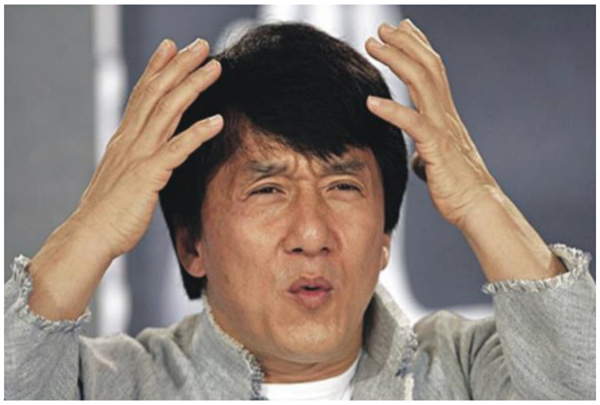
J. Extra Credit: QR Code Detection With and Without Preprocessing Algorithms

The extra credit section of this assignment was designed to take an image that contains a QR code placed somewhere randomly on it. There aren't many images available with such requirements, so it was necessary for me to create such an image myself. I chose to use the iconic Lenna image and a random QR code I found online. I converted the Lenna image to a pgm file to use for the preprocessed portion of this task, and kept the original, unprocessed image as a jpg file. This portion of the assignment was rather simple but took a while to implement correctly as the QRCodeDetector() function from OpenCV has had a few advances over the years and the parameters were difficult to actually comprehend and adapt to this task.

III. INPUTS AND THEIR RESULTS

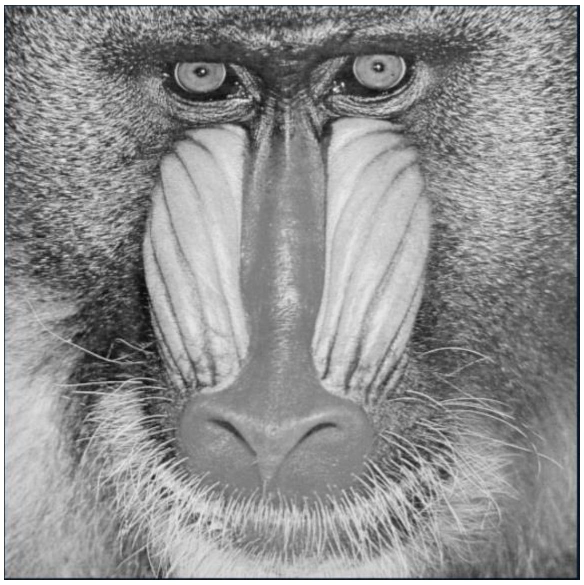
This section illustrates the results of the algorithm for *histostretch*, *histoequ*, houghcir, cannyhough, smoothedge, otsu, *otsuhisto*, *otsugau*, *combine*, and the additional functions *QRcode*, and *QRcodepre*. Changes to area size and location, and intensity are experimented with, showing variety and consistency of the functionality of the added code. Notice the variety in brightness, location, and area size between all the images of Jackie Chan confused, Lenna, Lord Farquaad from Shrek, our iconic Baboon image, and a still life of wine and berries shown below.

Original images (that have also been converted to .pgm files within the program), before applying any filters:

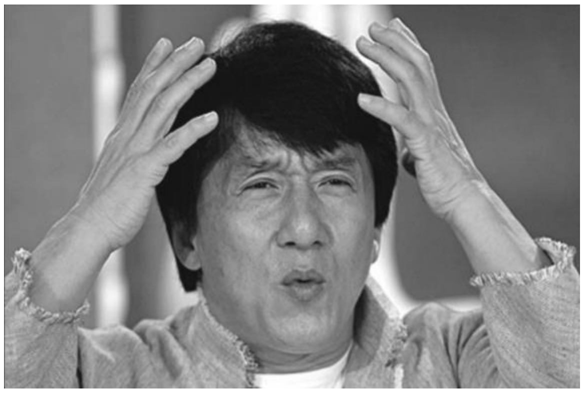

Original color image of Jackie Chan confused face, with no effects implemented.

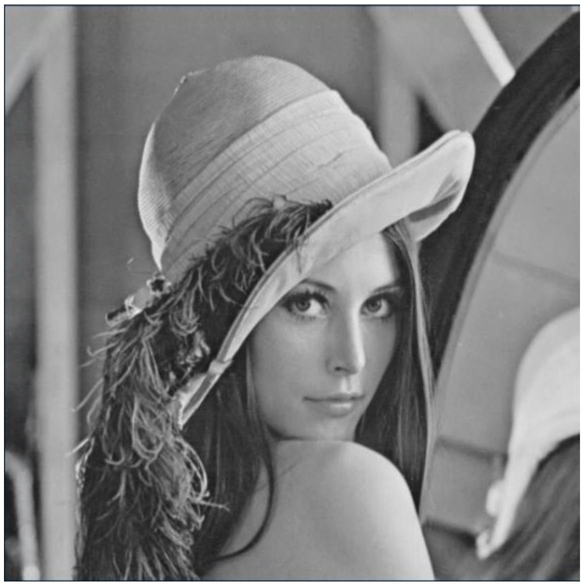
Original color image of Lenna, with no effects implemented.

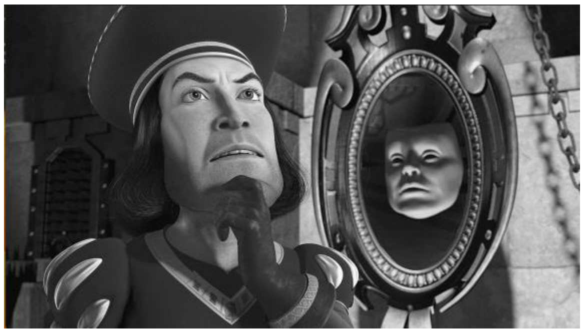
Original color image of Lord Farquaad from Shrek, with no effects implemented.

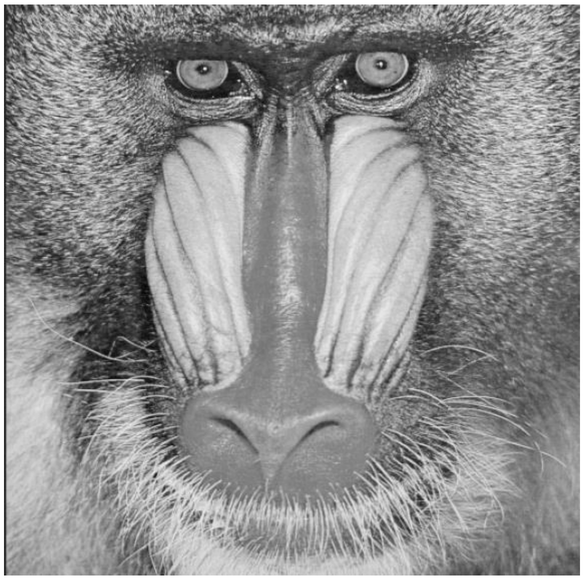

Original color image of Baboon, with no effects implemented.

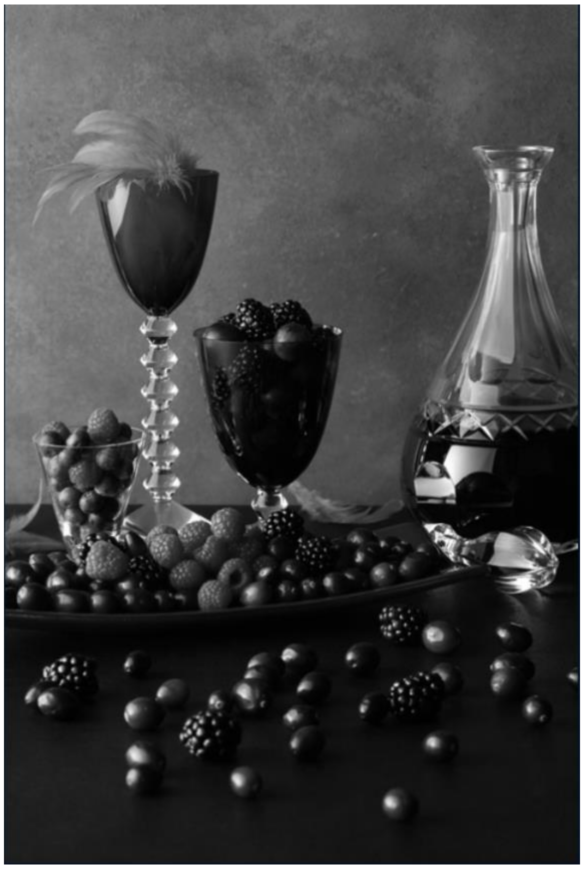
Original color image of a still life, with no effects implemented.


Original image of Jackie Chan confused with the gray filter applied to it.


Original image of Lenna with the gray filter applied to it.


Original image of Lord Farquaad with the gray filter applied to it.


Original image of Baboon with the gray filter applied to it.


Original image of a still life with the gray filter applied to it.

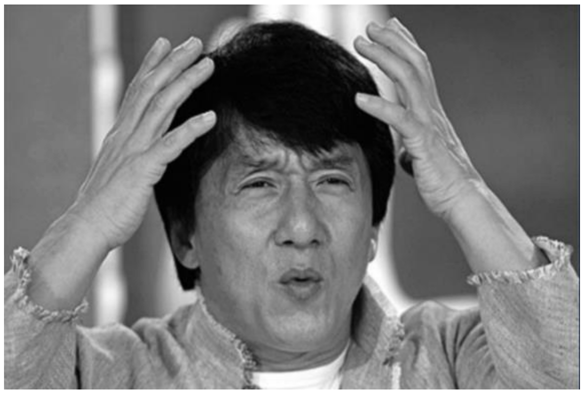


Figure 1: Image of Jackie Chan confused with histogram stretching applied.

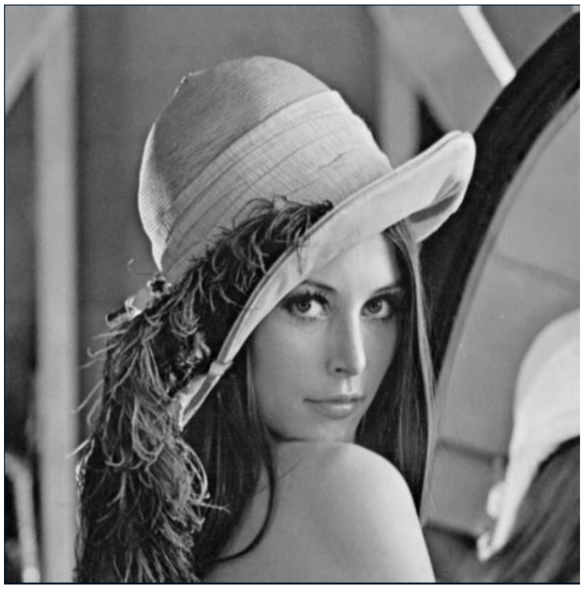


Figure 2: Image of Lenna with histogram stretching applied.

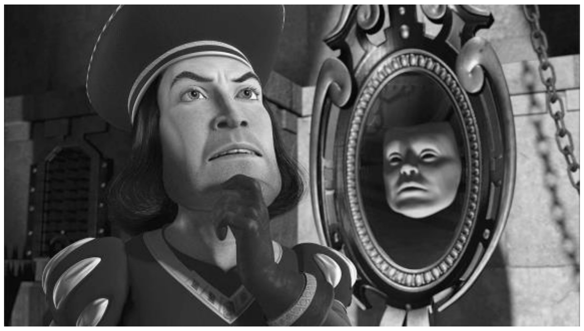


Figure 3: Image of Lord Farquaad with histogram stretching applied.

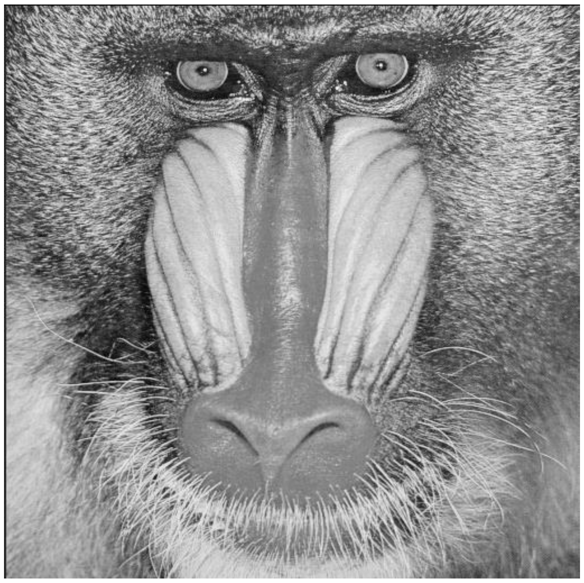


Figure 4: Image of Baboon with histogram stretching applied.

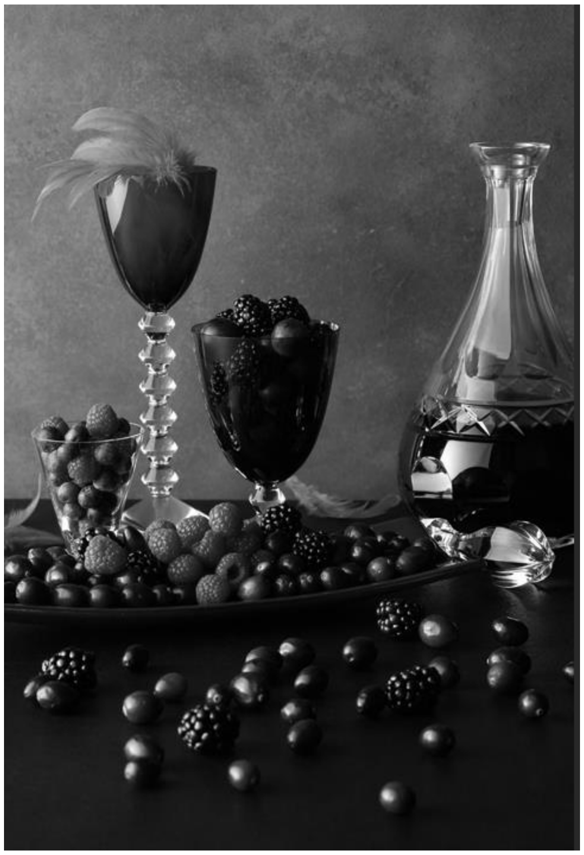


Figure 5: Image of a still life with histogram stretching applied.

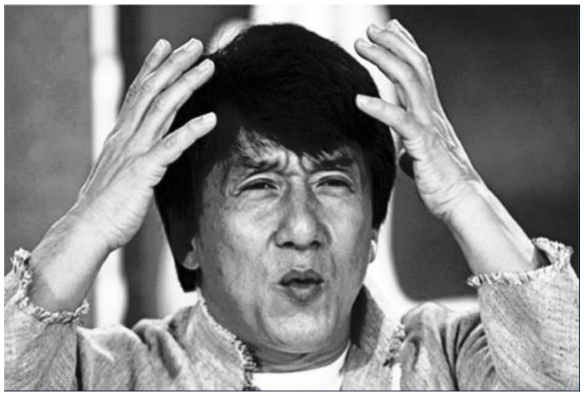


Figure 6: Histogram equalization applied to Lenna image.

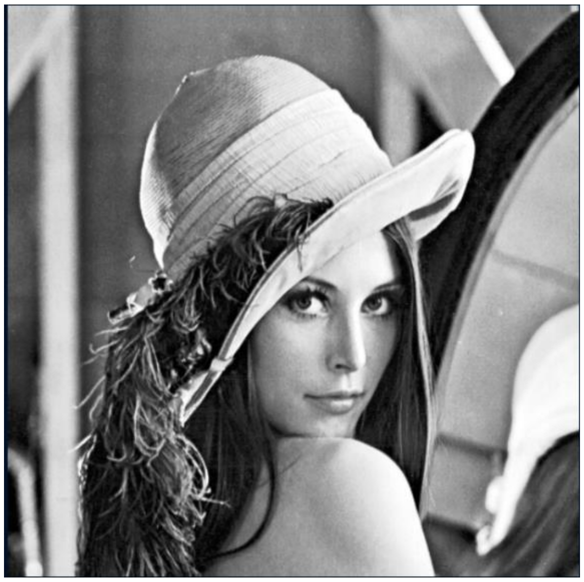


Figure 7: Histogram equalization applied to Lenna image.

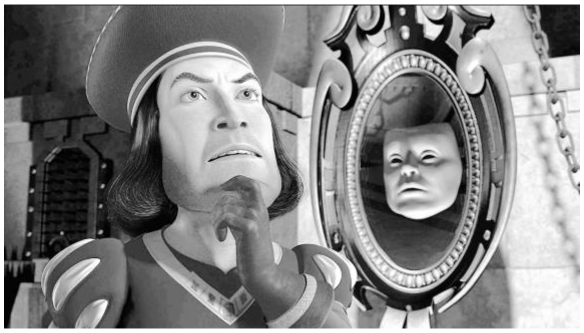


Figure 8: Histogram equalization applied to Lord Farquaad image.

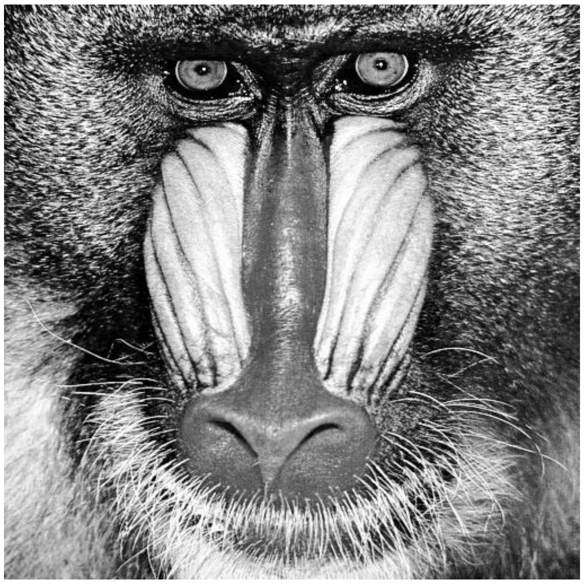


Figure 9: Histogram equalization applied to Baboon image.

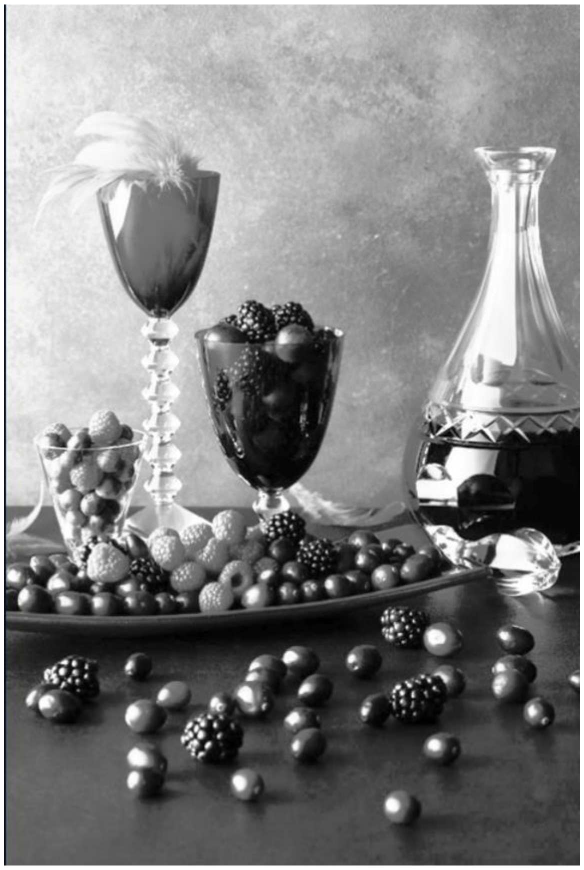


Figure 10: Histogram equalization applied to still life image.

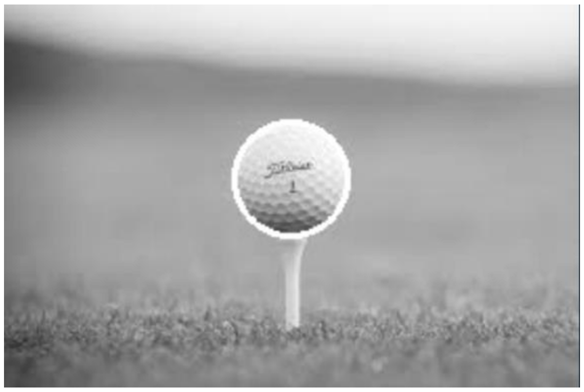


Figure 11: Hough Transform for circles on an image of a golf ball.

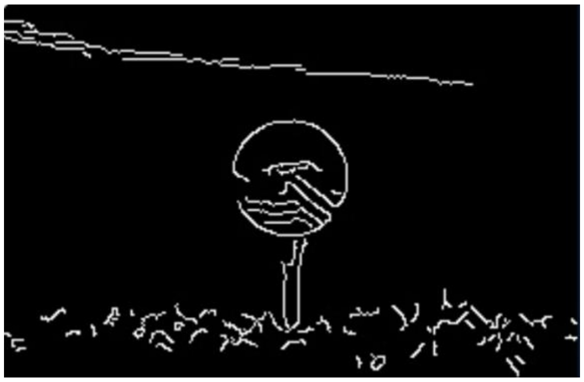


Figure 12: Hough Transform for circles with Canny module application on an image of a golf ball.

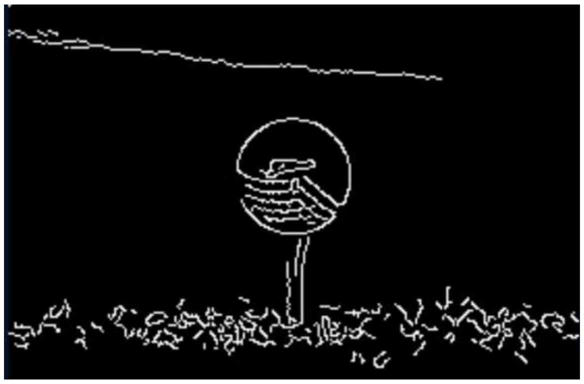


Figure 13: Hough Transform for circles with Gaussian smoothing and Canny edge detection on an image of a golf ball.

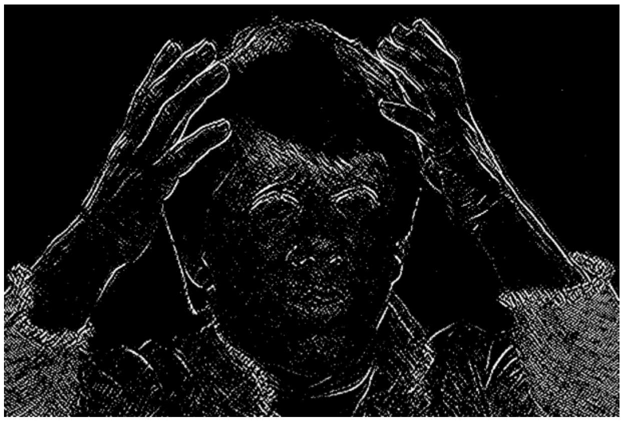


Figure 14: Otsu thresholding algorithm with Sobel kernel size 3x3 on the Jackie Chan confused face image.

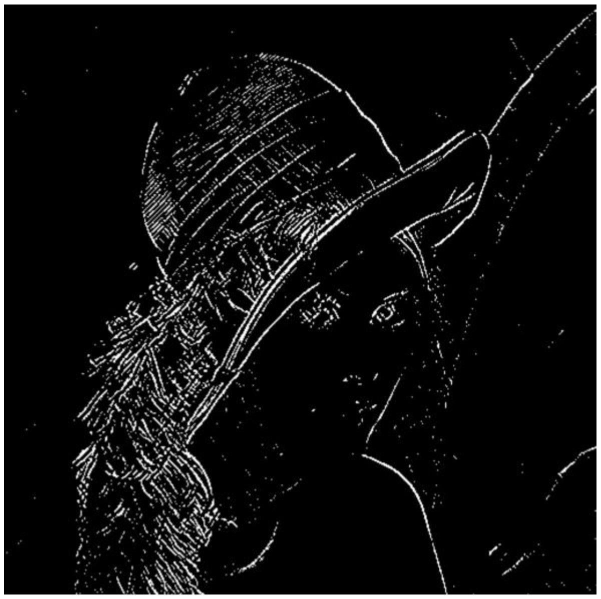


Figure 15: Otsu thresholding algorithm with Sobel kernel size 3x3 on Lenna image.

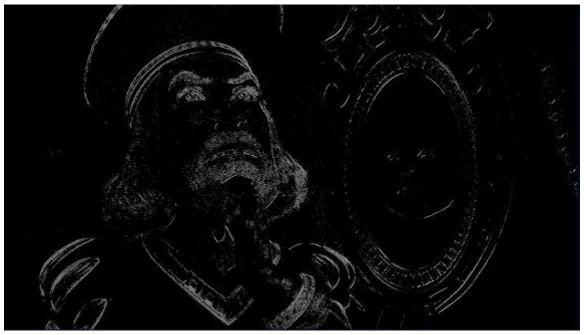


Figure 16: Otsu thresholding algorithm with Sobel kernel size 3x3 on image of Lord Farquaad.

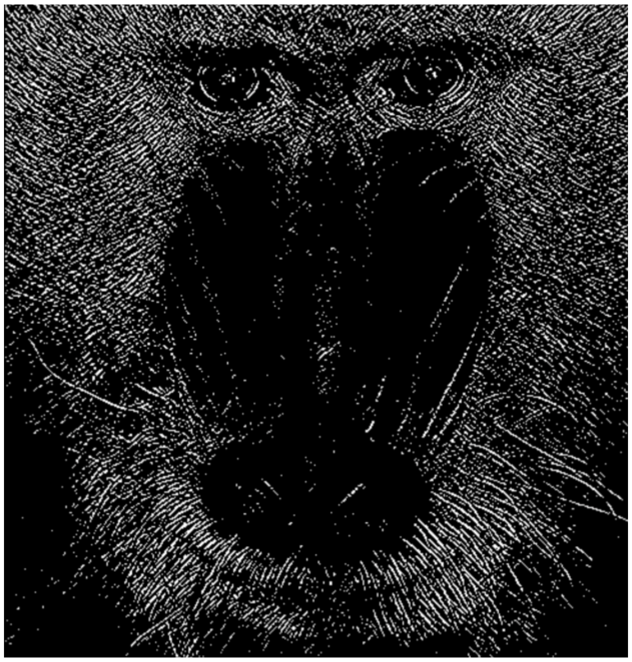


Figure 17: Otsu thresholding algorithm with Sobel kernel size 3x3 on the Baboon image.

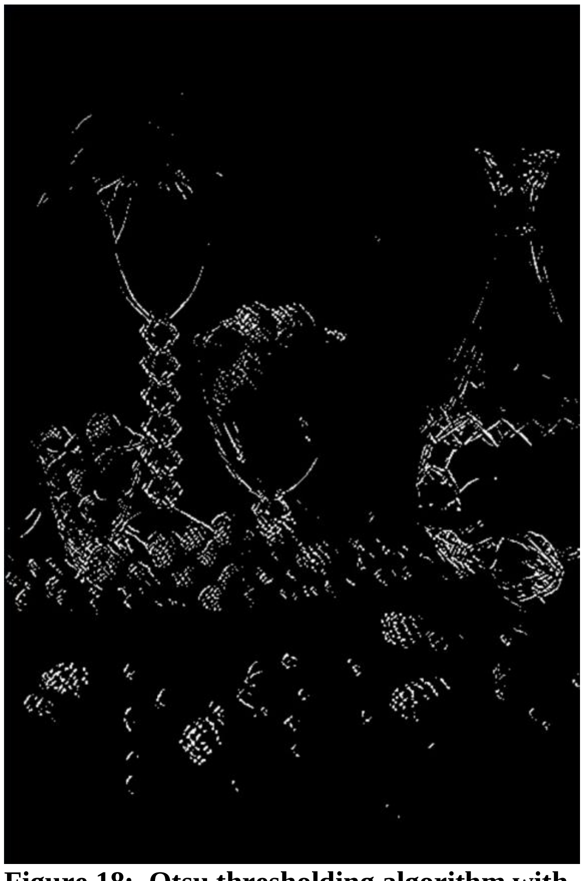


Figure 18: Otsu thresholding algorithm with Sobel kernel size 3x3 on the still life image.

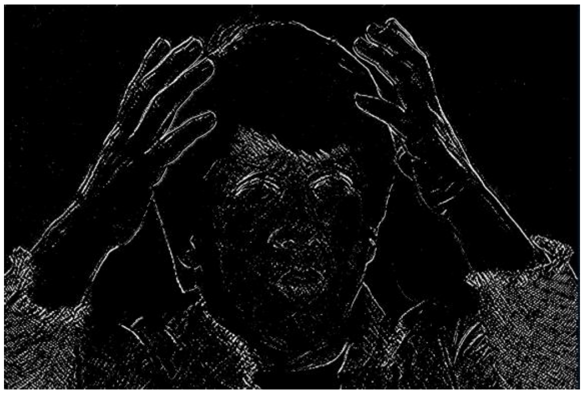


Figure 19: Histogram Equalization followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the Jackie Chan confused face image.

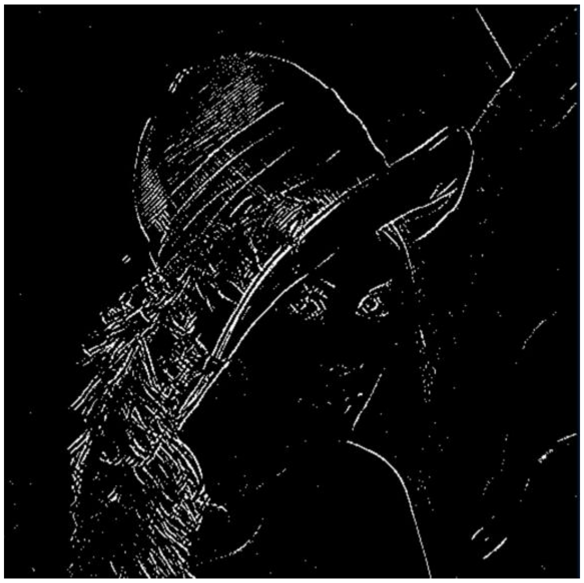


Figure 20: Histogram Equalization followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on Lenna image.

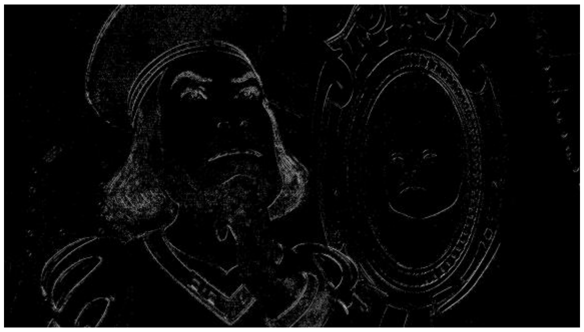


Figure 21: Histogram Equalization followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on image of Lord Farquaad.

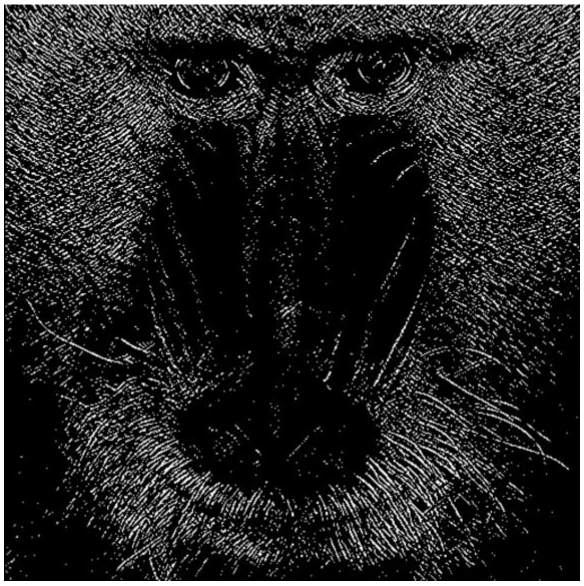


Figure 22: Histogram Equalization followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the Baboon image.

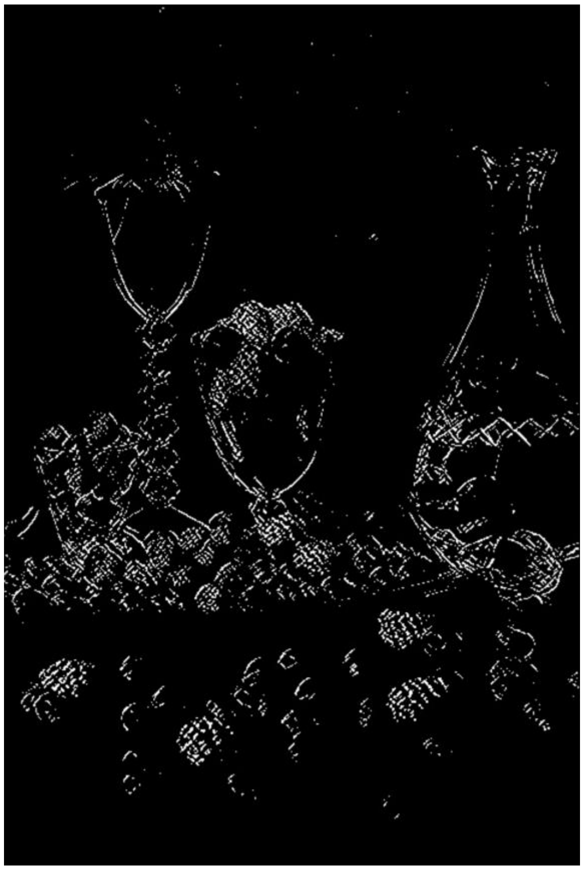


Figure 23: Histogram Equalization followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the still life image.

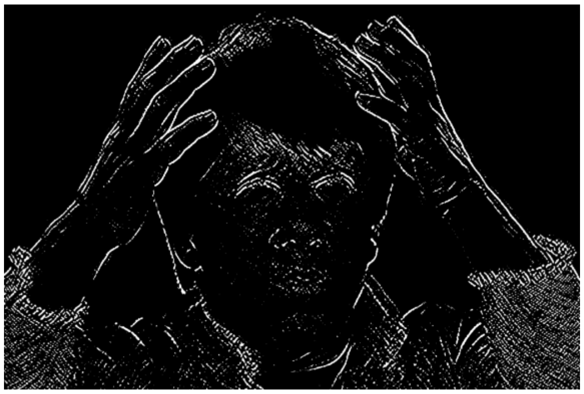


Figure 24: Gaussian Smoothing followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the Jackie Chan confused face image.

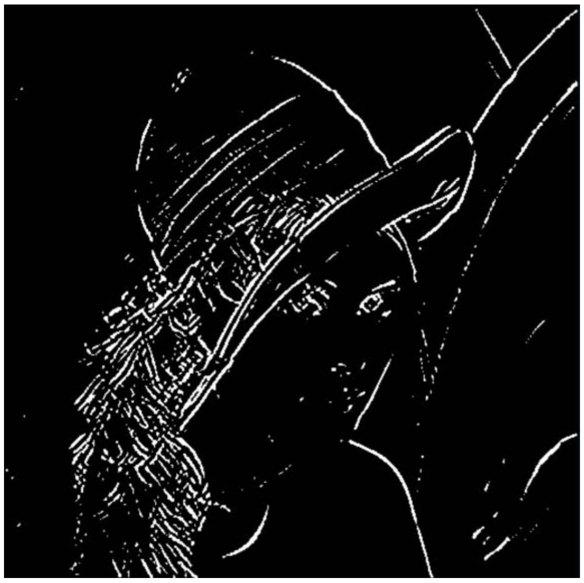


Figure 25: Gaussian Smoothing followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on Lenna image.

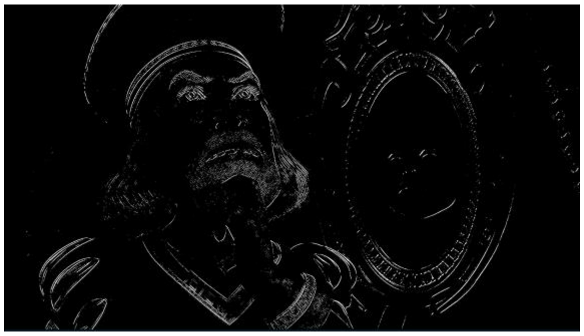


Figure 26: Gaussian Smoothing followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on image of Lord Farquaad.

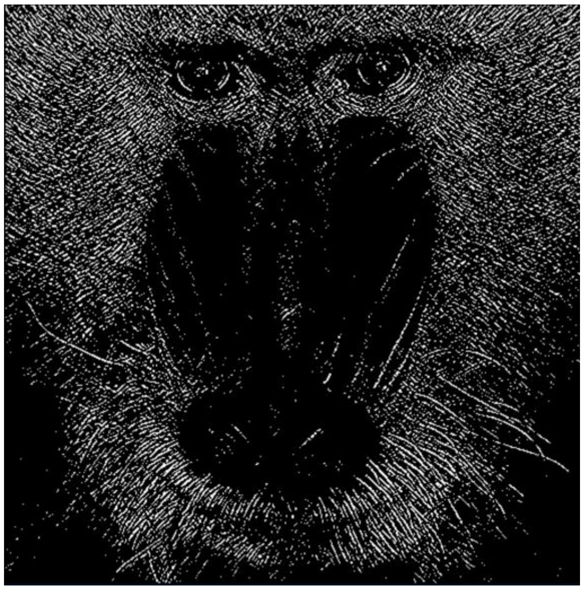


Figure 27: Gaussian Smoothing followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the Baboon image.

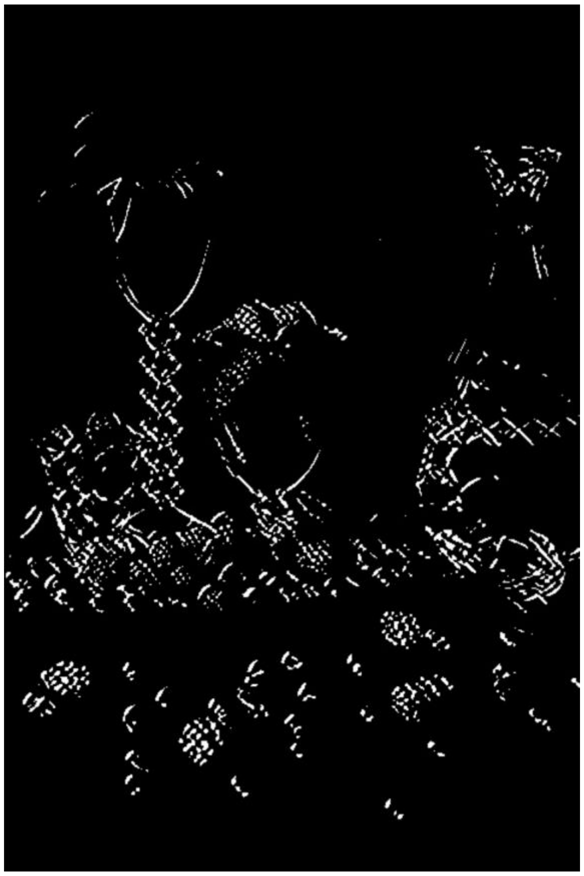


Figure 28: Gaussian Smoothing followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the still life image.

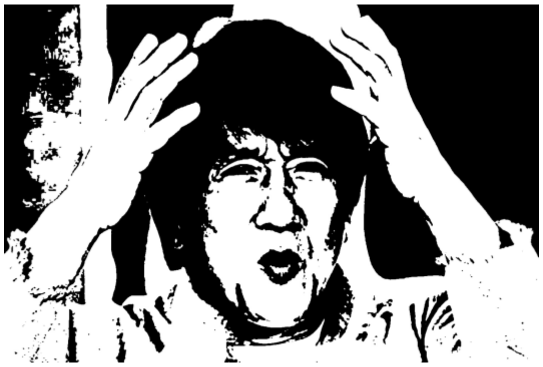


Figure 29: Gaussian Smoothing followed by Histogram Equalization, followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the Jackie Chan confused face image.

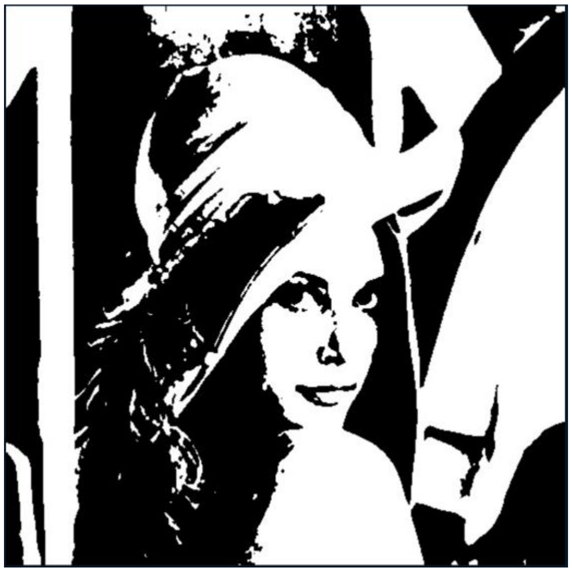


Figure 30: Gaussian Smoothing followed by Histogram Equalization, followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on Lenna image.

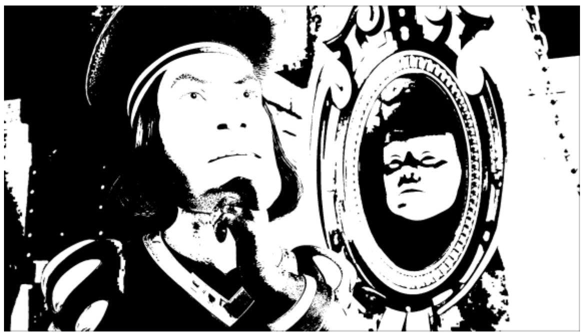


Figure 31: Gaussian Smoothing followed by Histogram Equalization, followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on image of Lord Farquaad.

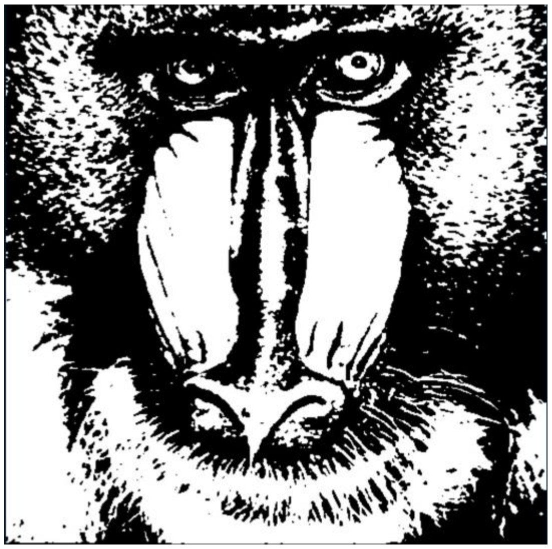


Figure 32: Gaussian Smoothing followed by Histogram Equalization, followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the Baboon image.

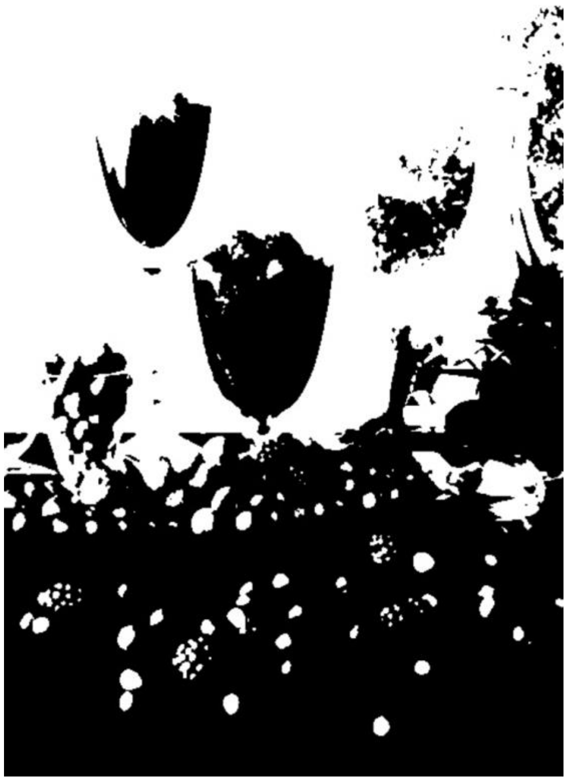


Figure 33: Gaussian Smoothing followed by Histogram Equalization, followed by Otsu thresholding algorithm with Sobel kernel size 3x3 on the still life image.

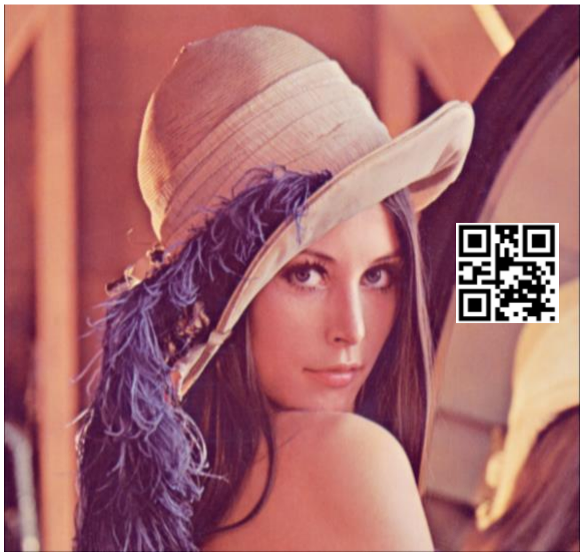


Figure 34: QR code detection and decoder on an unprocessed image of Lenna.

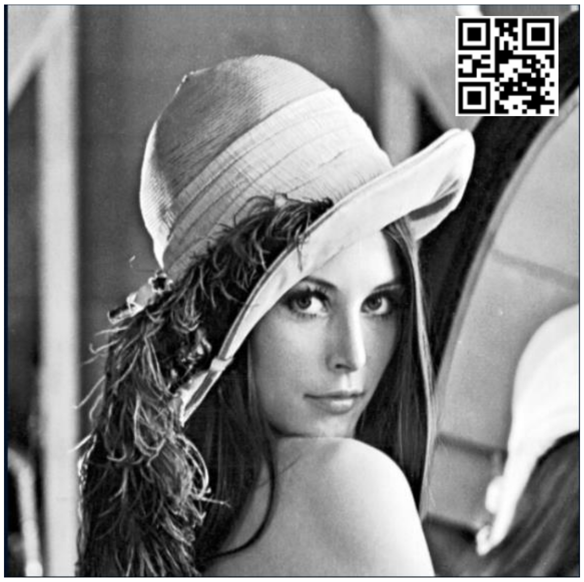


Figure 34: QR code detection and decoder on a preprocessed image of Lenna.

IV. PERFORMANCES OF OPERATORS

Edge detection seemed to have performed better with color images rather than gray level images. Unfortunately, there are some problems with the Sobel operator such as the amount of noise that it does not ignore, affecting its signal-to-noise ratio to be lower than it should. That is why the Canny edge detection technique is more favorable. However, the Canny technique does take longer to compute as it has a larger computational complexity compared to the Sobel operator. The Sobel operator is computational simple since it has a great gradient calculation approximation, but it also does have the factor of depth playing a part in its performance so it does not serve the same purpose as other edge detection techniques. Since OpenCV is an open-source library that makes image processing and computer vision as a whole more convenient for programmers, it requires less code and a more optimized time and computational complexity than if we were to write our own functions. OpenCV is a handy tool that expedites the image processing functions that are commonly used helping to improve overall performance of computer vision programs.

V. CONCLUSION

In this assignment, image smoothing or blurring, and edge detection were analyzed and performed with and without the C++ open-source computer vision library OpenCV. The program I created implemented simple edge detection of grey level images within user-specified ROIs using the Sobel operator (with both 3x3 and 5x5 kernel sizes), fixed thresholding gradients provided by the user, and generating binary edge images that are derived from the amplitude of the image’s gradient, as well as implemented Gaussian smoothing with OpenCV. Functions that fully operate on the OpenCV library are more advanced edge detection techniques such as the Sobel operator, the Otsu thresholding algorithm, the Canny technique, as well as an extra color space conversion and Canny technique combination.