Assignment 3: Edge Detection and OpenCV Utilization.

KEJVI CUPA
Department of Computer Science and Engineering
University of South Florida, Tampa, Florida, USA

1. Introduction and overall description

This assignment focuses on the utilization of OpenCV and more specifically edge detection. Using opencv we will apply multiple methods of edge detection. These methods will be applied on both grayscale and color images. Specifically, Sobel and Canny will be utilized for edge detection. To improve the results of these edge detection methods thresholding will be also utilized ranging from fixed thresholding to more advanced forms such as the Otsu algorithm and Histogram Equalization. The methods are used specifically to improve the feature detection in the original images before applying edge detection methods. Regions of Interest (ROIs) are implemented in order to allow for different regions in an image to be tested and try different user input combinations. Of course, this helps with the computation side considering that it will be less computationally expensive to perform an operation in the ROI rather than the full image. Overall, there will be thirteen new functions that will be implemented: sobel3, sobel5, coloredge3, coloredge5, binaryedge, direction_binaryedge, canny, histoeq, cannyhistoeq, otsucv, otsuhisto, QRcode, QRcodequl. A more detailed description of each function will be provided in this report.

2. Description of Algorithm

The first algorithm to be implemented is “sobel3”. This algorithm is used to perform edge detection on a grayscale image. The function is used using OpenCV and a kernel size of 3 is utilized in the computation of Sobel. In order to compute edge detection using sobel, initially the gradient with respect to the x direction and y direction is calculated again using OpenCV. Once these gradients are calculated they are converted to 8U format. After that the two gradients are added together to compute the gradient in that pixel location. Then that gradient is used as the value to update the existing pixel value, and that is how we are able to perform edge detection using Sobel with kernel size 3. In this algorithm up to 3 ROI are specified which are user defined.

The second algorithm to be implemented is “sobel5”.This algorithm is used to perform edge detection on a grayscale image. The function is used using OpenCV and a kernel size of 5 is utilized in the computation of Sobel. In order to compute edge detection using sobel, initially the gradient with respect to the x direction and y direction is calculated again using OpenCV. Once these gradients are calculated they are converted to 8U format. After that the two gradients are added together to compute the gradient in that pixel location. Then that gradient is used as the value to update the existing pixel value, and that is how we are able to perform edge detection using Sobel with kernel size 3. In this algorithm up to 3 ROI are specified which are user defined.

The third algorithm to be implemented is “coloredge3”. This algorithm builds upon the “sobel3” algorithm. In this one we are performing edge detection using sobel on color images. The color image in the RGB/BGR color space is initially converted to HSV color space and then sobel with kernel size is applied to the V component of the color image. After all the changes have been performed, the V component is updated in the original image. Then the original image is converted back to HSV for output. The computation of sobel is the same as before and it is described in the algorithm above and won’t be repeated here again to avoid repetitiveness. In this algorithm up to 3 ROI are specified which are user defined.

The fourth algorithm to be implemented is “coloredge5”. This algorithm builds upon the “sobel5” algorithm. In this one we are performing edge detection using sobel on color images. The color image in the RGB/BGR color space is initially converted to HSV color space and then sobel with kernel size is applied to the V component of the color image. After all the changes have been performed, the V component is updated in the original image. Then the original image is converted back to HSV for output. The computation of sobel is the same as before and it is described in the algorithm above and won’t be repeated here again to avoid repetitiveness. In this algorithm up to 3 ROI are specified which are user defined.

The fifth algorithm to be implemented is “binaryedge”. This algorithm builds upon the sobel algorithms by adding thresholding. The sobel logic is the same as described in the previous algorithms. After the sobel has been applied, the updated pixel values are compared to the built-in threshold and if the pixel value is larger than the threshold the pixel intensity is set to 255, and if it is less it set to 0. This allows for a better distinction in the edge detection output. In this algorithm up to 3 ROI are specified which are user defined.

The sixth algorithm to be implemented is “direction_binaryedge”. This algorithm builds upon the sobel algorithms by adding thresholding using direction. The sobel logic is the same as described in the previous algorithms. The direction is calculated using the formula: angle = atan(gradient_y / gradient_x). After the sobel has been applied, the updated pixel values are compared to the angle and if the pixel value is larger than the angle the pixel intensity is set to 255, and if it is less it set to 0. This allows for a customization of the edge detection, by showing only edges that meet a certain criteria that we set. In this algorithm up to 3 ROI are specified which are user defined.

The seventh algorithm to be implemented is “canny”. The user will provide the ROI parameters and up to three ROIs will be implemented. Canny is used now instead of Sobel and the utilization is performed via OpenCV. However, Canny is performed when there is noise or some other degradation in the image and it is a more advanced method of edge detection. Initially, after the image has been converted to grayscale 1 channel, blurring is performed. After that, we apply the Canny operation using the OpenCV method and we apply double thresholding. The low threshold is set internally and the higher threshold is usually set as lower threshold * 3, so that is also the value set. So whenever we have some noise or certain types of degradation in the image, we utilize Canny.

The eighth algorithm to be implemented is “histoeq”. The user will provide the ROI parameters. This algorithm applies histogram equalization via the OpenCV method on the ROI of the image specified by the user. With histogram equalization we can get the intensity level distribution of the ROI and generate a uniform distribution of the gray levels across the range and basically provide better contrast in low contrast images.

The ninth algorithm to be implemented is “cannyhistoeq”. The user provides the ROI parameters and it will be up to 3 ROIs. This algorithm combines both histogram equalization and Canny. Initially, histogram equalization is performed in the image and then Canny operation for edge detection is performed. The goal of this algorithm is to apply histogram equalization to improve image contrast and make Canny edge detection easier.

The tenth algorithm to be implemented is “otsucv”. The user provides the ROI parameters and it will be up to 3 ROIs. This algorithm applies the Otsu thresholding algorithm via OpenCV to determine the best threshold level for segmentation and perform it. This algorithm is a brute force algorithm as it will test each possible threshold level and distribute the pixels into background and foreground pixels and then compute the sum of variances. Where the variance is the smallest that is where the best threshold is and chooses that level. The Otsu algorithm provides good results for thresholding.

The eleventh algorithm to be implemented is “otsuhisto”. The user provides the ROI parameters and it will be up to 3 ROIs. This algorithm builds upon the previous algorithm by implementing histogram equalization which will be applied only on the background pixels as determined by the Otsu threshold algorithm. The algorithm aims to improve the segmentation of the image using histogram equalization.

The twelfth algorithm to be implemented is “QRcode”. This algorithm is implemented using the OpenCV function, and it is used to read the QR portion in an image. It is able to return the located QR code in the image and also read its content which can usually be in the form of a link. This algorithm is applied to the entire image and the image is searched for that QR code.

The thirteenth algorithm to be implemented is “QRcodequl”. This algorithm is implemented by combining the QRcode function and Histogram equalization functions of OpenCV. This algorithm aims to build on the previous algorithm, by utilizing histogram equalization to improve the contrast in the original image if it has poor contrast and the QR code is difficult to identify and interpret. The histogram equalization component is used to help with the localization of the QR code component in the image. Again, this algorithm is applied on the entire original image.

3. Results

Below are shown the results (images) with different combinations of the respective function parameters.

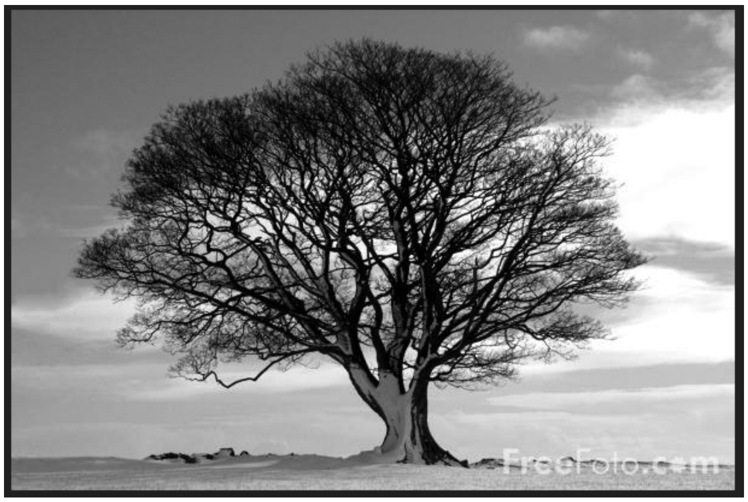

Figure 1: Original Image

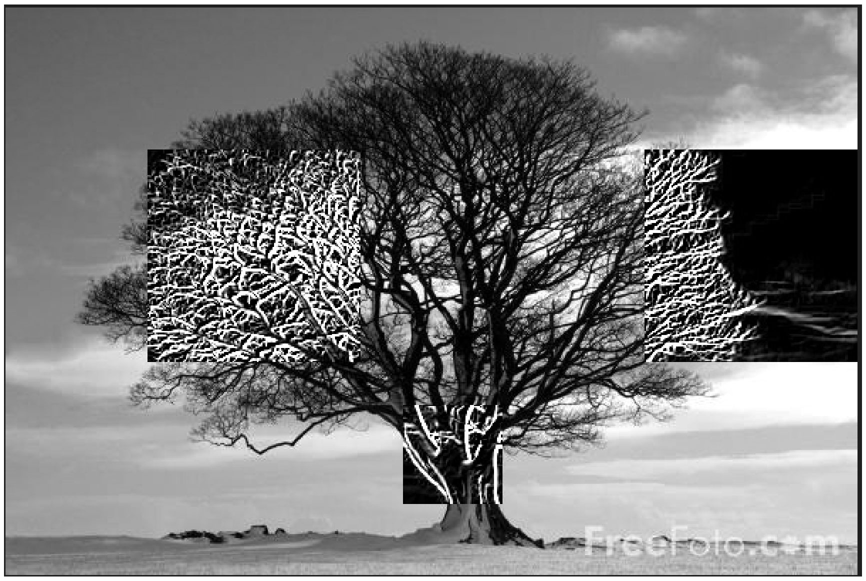

Figure 2: “sobel3” function. Applying Sobel Edge detection with kernel size 3x3 on the 3 used ROIs. The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70

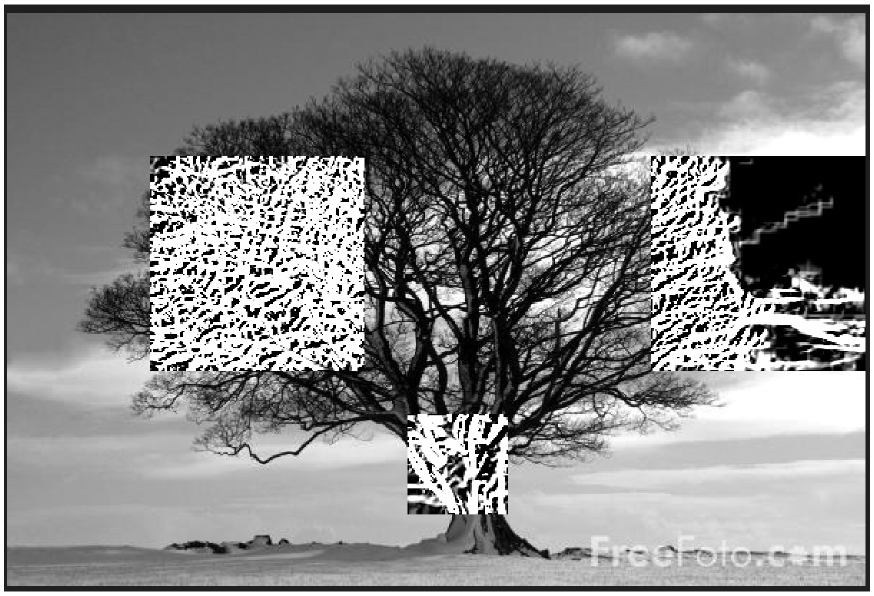

Figure 3: “sobel5” function. Applying Sobel Edge detection with kernel size 5x5 on the 3 used ROIs. The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70

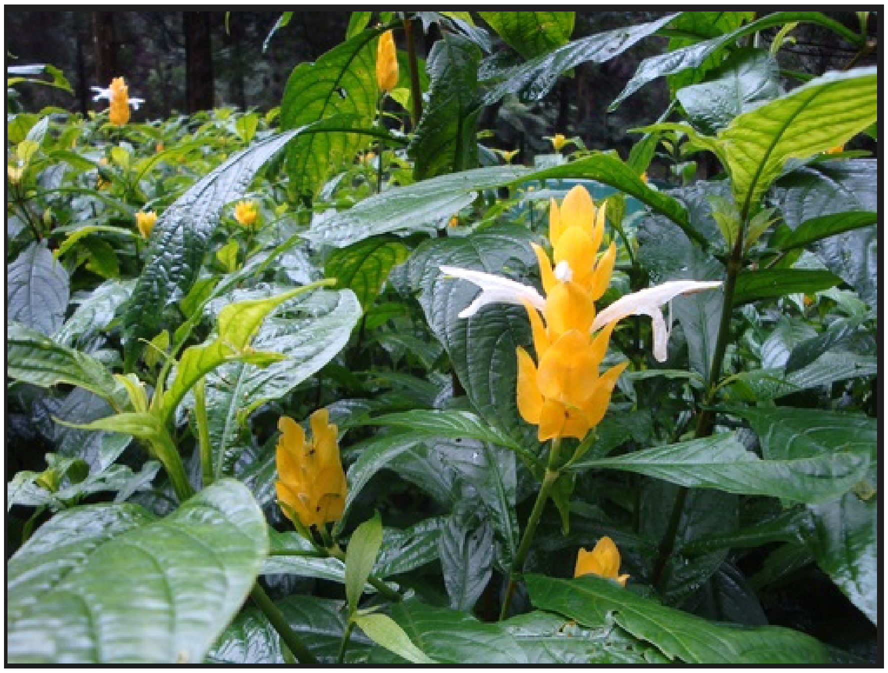

Figure 4: Original Color Image

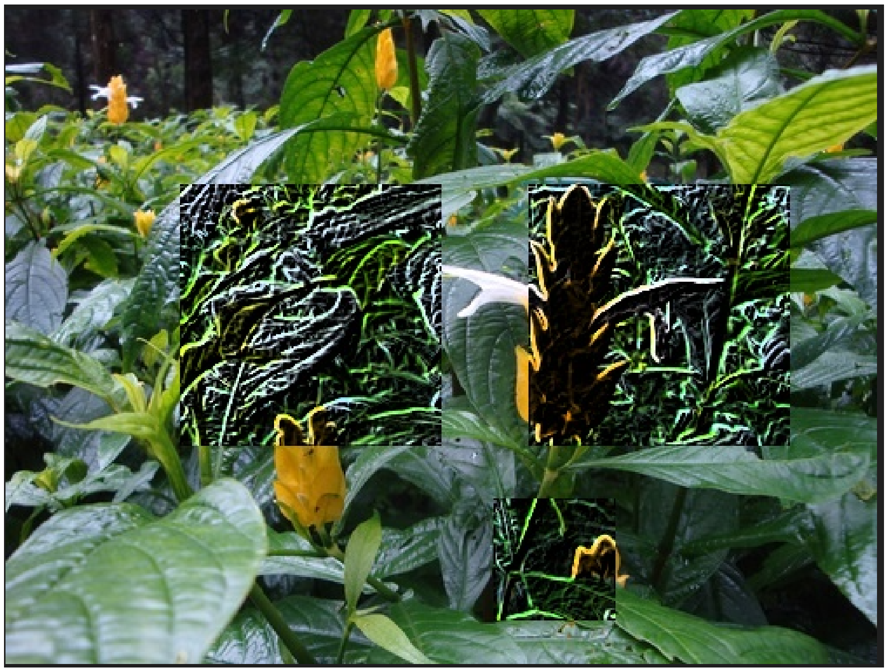

Figure 5: “coloredge3” function. Applies Sobel edge detection with kernel size 3x3 on 3 ROIs on the V component of the color image which was converted from RGB to HSV color space. The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 300, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70

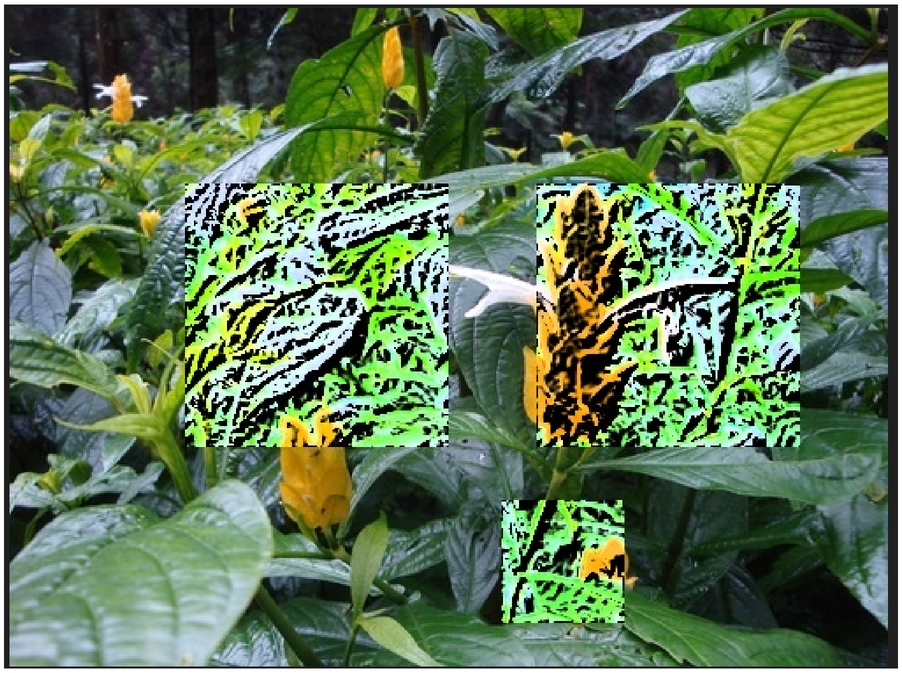

Figure 6: “coloredge5” function. Applies Sobel edge detection with kernel size 5x5 on 3 ROIs on the V component of the color image which was converted from RGB to HSV color space. The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 300, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70

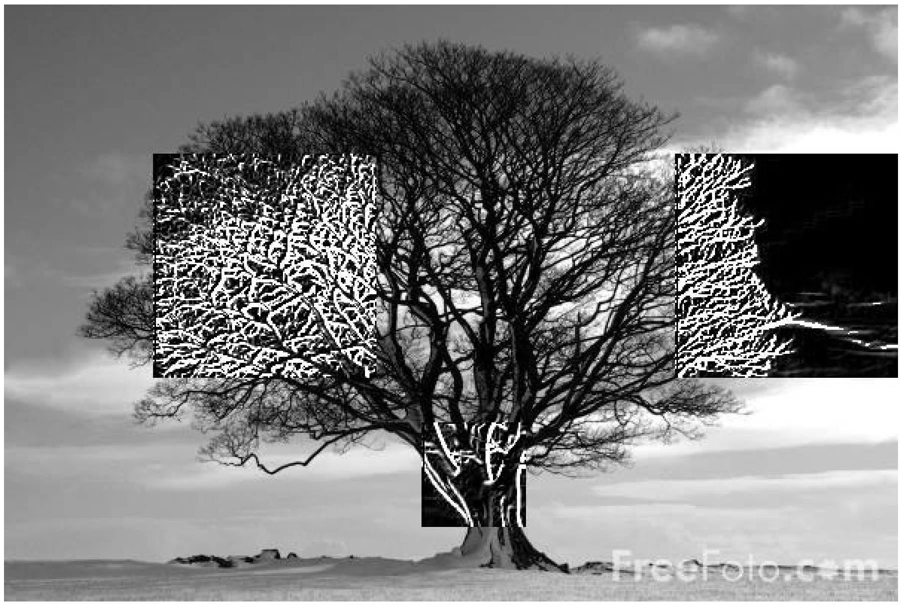

Figure 7: “binaryedge” function. Binarification of the image up to 3 ROIs after Sobel with kernel size 3x3 has been applied using internal thresholding. The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70

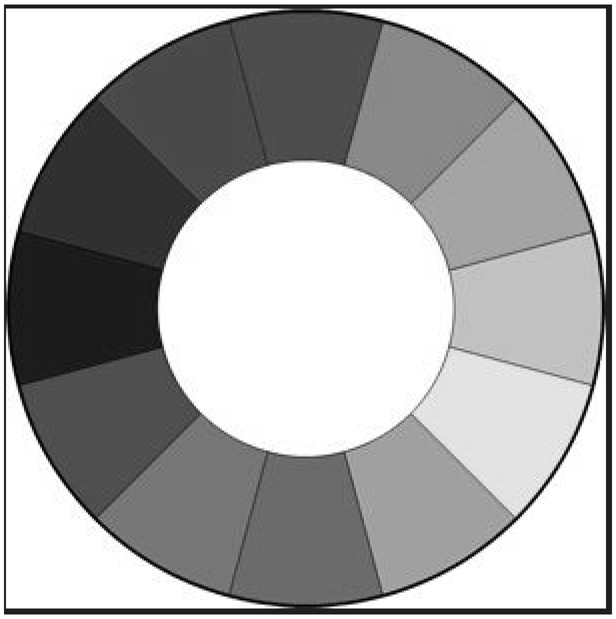
Figure 8: Original Image

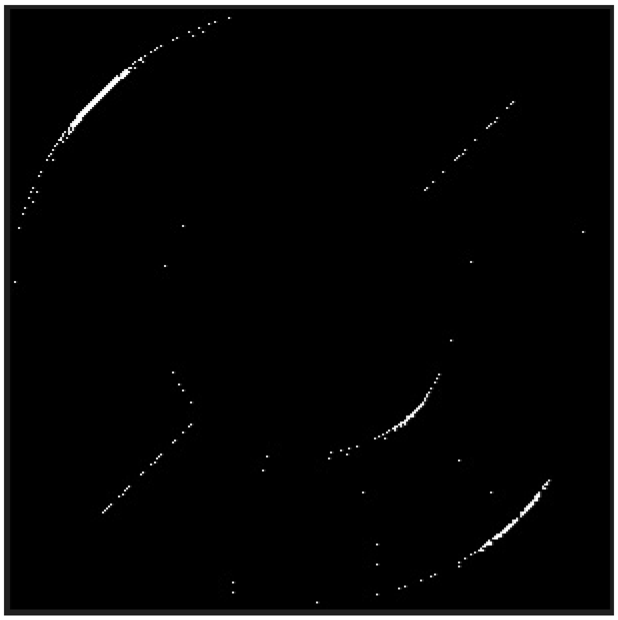

Figure 9: “direction_binaryedge” function. Binarification of the image after Sobel with kernel size 3x3 has been applied using internal thresholding by using pixel direction and internal pixel intensity thresholding. Angle range 35-55 degrees. (45 degrees). The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70

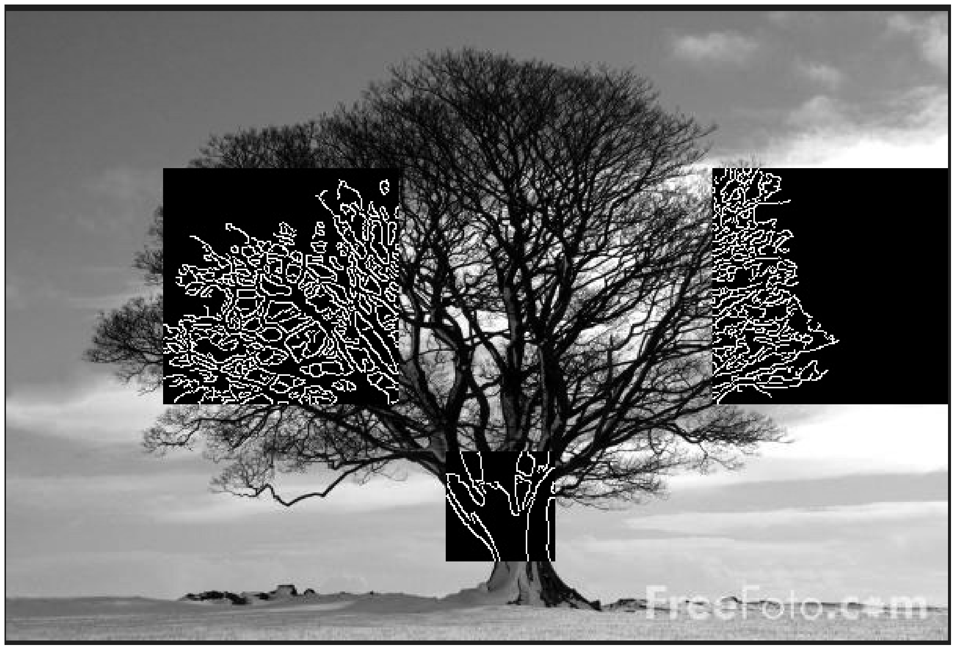

Figure 10: “cannycv” function. Applying Canny Edge detection with kernel size 3x3 on the 3 used ROIs. Canny also utilizes blurring, edge thinning and double thresholding after performing Sobel. It is a multi-stage algorithm. The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70.

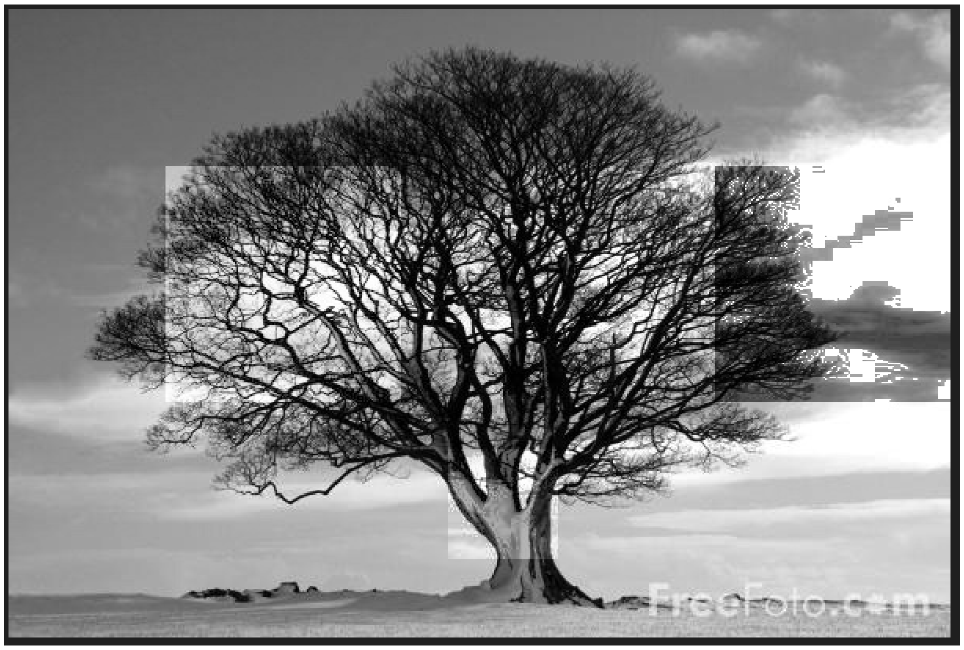

Figure 11: “histoeq” function. Perform Histogram Equalization on the ROI to enhance contrast. Up to 3 ROIs.The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70.

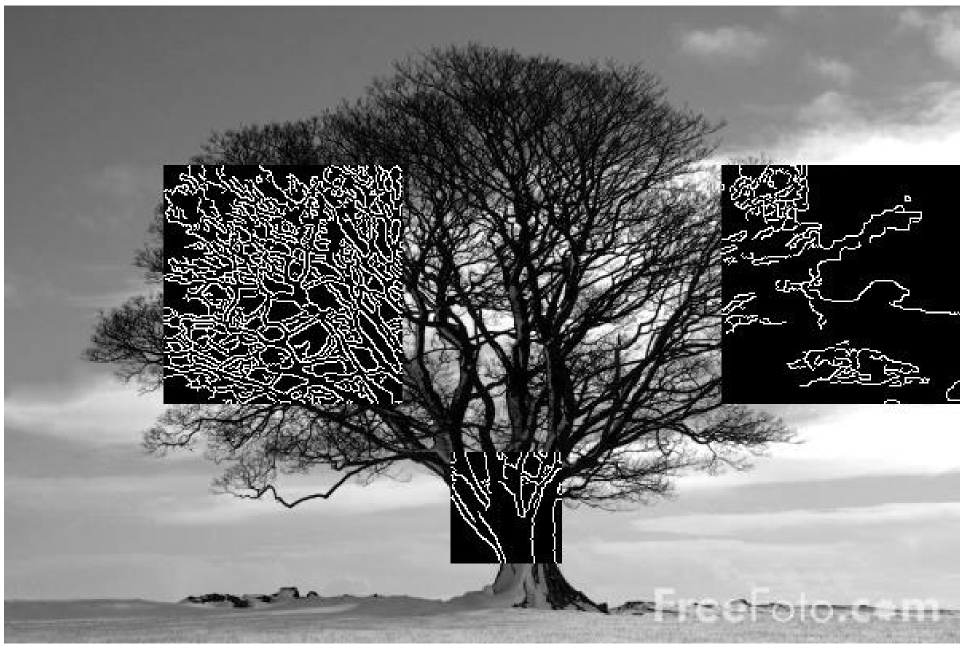

Figure 12: “cannyhistoeq” function. Apply Canny edge detection on ROI after it has been initially Histogram Equalized to improve edge detection. Up to 3 ROIs.The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70.

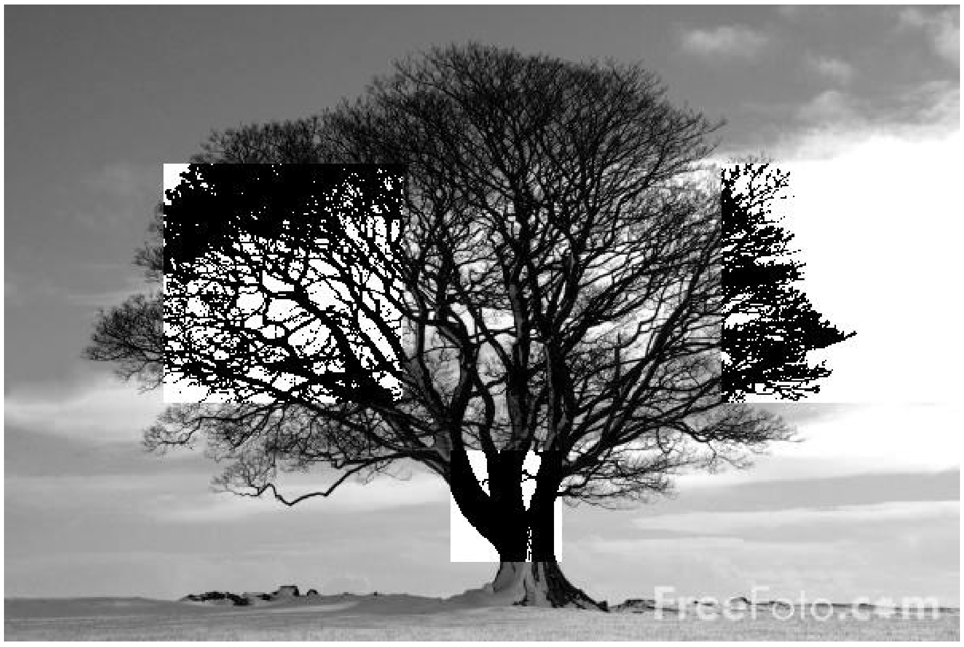

Figure 13: “otsucv” function. Apply the Otsu Thresholding Algorithm on ROI. Otsu will find the best threshold level in a brute-force way. Up to 3 ROIs.The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70.

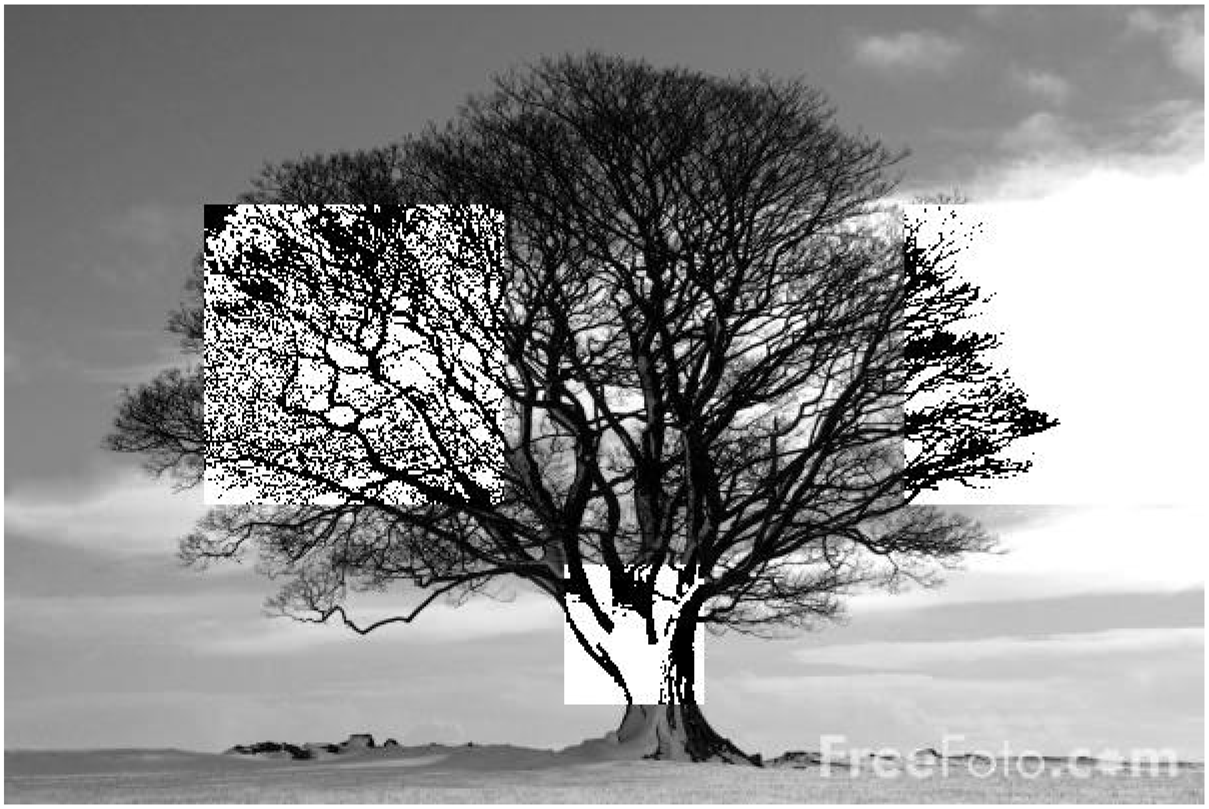

Figure 14: “otsuhisto” function. Apply the Otsu Thresholding Algorithm on ROI. Otsu will find the best threshold level in a brute-force way. Up to 3 ROIs.The used parameter configuration is: #roi = 3, x1= 100, y1= 100, Sx1= 150, Sy1= 150, x2 = 450, y2 = 100, Sx2 = 150, Sy2 = 150, x3 = 280, y3 = 280, Sx3 = 70, Sy3 = 70.

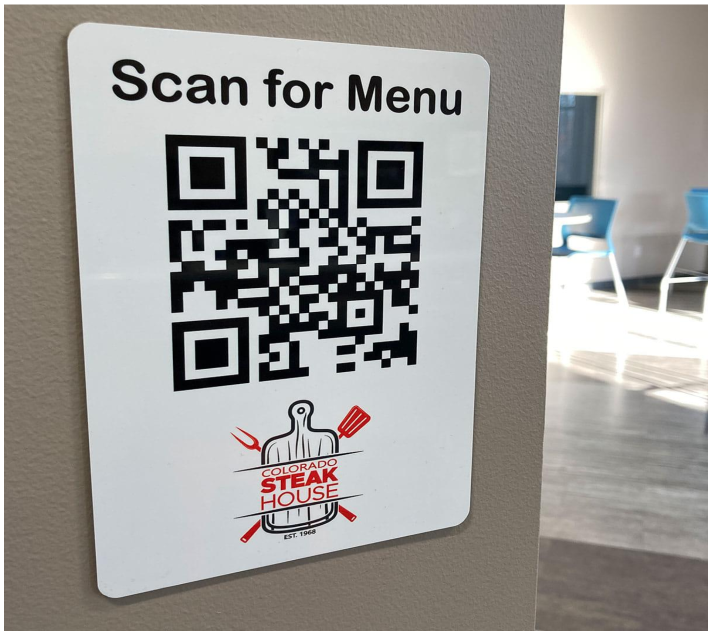

Figure 15: Original Image that contains QR code.

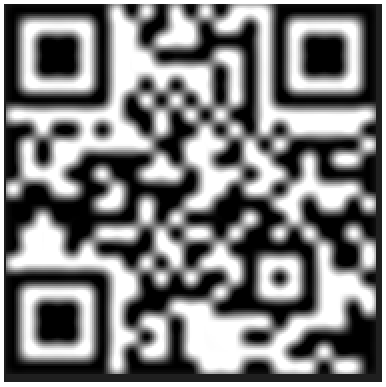

Figure 16: “QRcode” function. QRCodeDetection from OpenCV is utilized to detect and decode the QR code in an image. The function returns the rectified QR image which is the QR code that was detected in Figure 15 in a rectified format. The data returned from the function is the content of the QR code after it has been decoded. It can be a message/link or some content. In this case a “link” is printed in the console. Link : “*https://www.napnameplates.com/*”

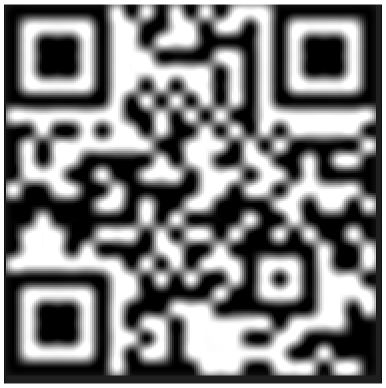

Figure 16: “QRcodequl” function. Perform Histogram Equalization on the image first and then apply the QRCodeDetection from OpenCV to detect and decode the QR code in an image. The image output is the rectified QR code and the console prints out the decoded content which in this case is a link “ *https://www.napnameplates.com/*”.

4. Discussion of Results

We were able to perform edge detection on both gray level and color images but utilizing OpenCV methods. The two main methods of edge detection applied were Sobel and Canny. For Sobel we used the Sobel operator with kernel size 3 and 5. Canny is a more advanced method of edge detection considering that it is a multi-stage algorithm that performs blurring and thresholding built-in. As mentioned, above edge detection was performed on both gray level and color images. Based on the results, we saw that these edge detection methods were rather efficient and successful at finding the edges.

To apply edge detection on color images, we had to first convert the image from RGB color space to HSV, and then apply Sobel on the V component. Computationally, applying edge detection on color images is more intensive than on gray level images, and it takes slightly longer. As far as the results go, edge detection seems a bit more logical to apply on gray level images to more accurately depict the edges. When applying to color images, perhaps it would be best to first convert the color image to gray level first. Regardless, edge detection did work properly on color images, but the results seemed better on gray level images.

Additionally, binary edge images were generated. We tested both using an internal threshold level and then adjusting the pixel values accordingly. Set to 255 if pixel value is greater than threshold and to 0 if it is smaller/equal. Then, in order to make it even more adaptive and customize what type of edges we want we also implemented binary edge image generation using the direction of the pixel as a threshold. The direction angle was computed using the formula angle = atan(gradient_y / gradient_x). Based on the angle we can choose which pixels to display in the output image.

In addition to Sobel, Canny edge detection was also implemented on gray level images. As mentioned above, Canny is a more advanced form of edge detection, because it is a multi-step algorithm after all. We apply blurring, and then use Canny from OpenCV. Canny does use Sobel but also performs edge thinning and thresholding, which is also why the Canny output displays edges better. So based on the results, Canny was able to perform better than Sobel in edge detection. The edges are much more clear with Canny. However, sometimes Canny can miss some edges due to the blurring and double thresholding that are also utilized internally. So in certain cases Sobel will perform better than Canny and sometimes Canny will perform better. That reminds me of the “No free lunch theorem”, that there is no single algorithm that performs best in all cases. In our context, it is important to know when to use Sobel and when to use Canny.

We also were able to perform Histogram Equalization using the OpenCV method. In the previous assignment we implemented Histogram Stretching. One downside of Histogram Stretching is that if the image's pixels are already covering the entire range 0 - 255, then Histogram Stretching wouldn't do anything. However, with Histogram Equalization that is not the case because the goal of Histogram Equalization is to create a uniform distribution of pixels across gray levels. This can also lead to some levels being combined together, therefore Histogram Equalization does not suffer from the same limitation of Histogram Stretching. We use Histogram Equalization to improve the image contrast to make it easier to identify edges potentially.

We tested that theory by applying Histogram Equalization to the image, and then applying Canny on it. It is important to understand what type of image it is best to use Histogram Equalization on. In our case, the effect of Histogram Equalization can be visualized in our output. The image/roi selection and the fact that we kept the same image was intentional because it makes it easier for us to compare the results between different algorithms. In this case, we can see that Histogram Equalization was able to improve our results in one ROI and somewhat worsen them in another. Canny edge detection wasn't able to detect all edges in one ROI of our image as compared to Sobel, however after applying Histogram Equalization, it was able to detect more edges a lot more accurately. However, in another ROI where the background was mixed with the foreground (object of interest), Histogram Equalization did alter the image a bit too much, making Canny identify edges that shouldn't be there. So, Histogram Equalization can certainly help and improve Canny's results, but our ROI selection is important as the results will vary on poor ROI selections.

The Otsu Thresholding was another algorithm that we were able to implement/test using OpenCV. We specify the range we are interested in and in our case it was 0 - 255. The Otsu algorithm is much more advanced than the Fixed Thresholding we have used in the past. Firstly, we use thresholding in image segmentation. Using a fixed threshold value is not computationally expensive and it is fast. However, finding that best level to set as the threshold level can be quite difficult and the results will vary afterwards. The Otsu algorithm is a brute-force algorithm that basically tests each gray level as a threshold level, and separates the pixel into background and foreground. It then computes the variances and finds where the sum of variances is minimal for which level. Then that level is selected as the best thresholding level. As you can see it is much more complicated, and computationally expensive, however it yields much better results than Fixed thresholding.

In another algorithm, to boost the result of the Otsu thresholding, Histogram Equalization was applied in the background pixels determined by Otsu. So after we isolated the background pixels by using the threshold level returned by Otsu, we performed Histogram Equalization on them. Then the background pixels that we just changed due to Histogram Equalization, were used to update the old corresponding pixels in the original image. Then we proceeded to apply Otsu Thresholding using OpenCV. The result is a sharper, thinner edge which is clear. There was improvement after using Histogram Equalization.

The last portion was to implement a QR code detection algorithm. For this algorithm OpenCV was utilized as well. The OpenCV method of QRCodeDetector, which was detectAndDecode makes it very simple to simply pass an image and then the algorithm will scan it to determine the QR code. Then it will output a rectified image of the QR code and also output the result/data/content that the QR code holds. Considering that the image that contains the QR code may not be under proper lighting conditions, or perhaps the contrast is poor, we also implemented Histogram Equalization. Since Histogram Equalization is great for contrast enhancement it can improve the contrast of the image and more specifically, the QR code portion. So with Histogram Equalization, it will be easier for the QRCodeDetector to detect the QR in certain images and be able to decode the content. Of course, it is important to use Histogram Equalization properly in the correct images that could need it so as to prevent any significant alteration of the image features which could impact the QR code portion and making it impossible to decode it. But overall, Histogram Equalization will improve image contrast and QR code readability in images that were taken under poor lighting conditions. Otherwise, Histogram Equalization won't make much of an improvement.

All of the previous algorithms were implemented using OpenCV. OpenCV is a great framework that certainly makes it much easier and more efficient to test different and complex algorithms very quickly. We have implemented a few of these algorithms manually before and it was challenging and the code we may have implemented may not have been properly optimized and fine-tuned. So overall OpenCV is a great tool for us that allows us to implement and test different algorithms very fast and efficiently.

5. Conclusion

Overall, in this assignment we were able to experiment with edge detection and the OpenCV utilization. We tested different edge detection methods such as Sobel and Canny with different kernel sizes 3, 5. Furthermore, we tried to complement these methods with Histogram Equalization, and thresholding internally with fixed value and by direction. We were able to test advanced algorithms such as Histogram Equalization and the Otsu Thresholding quickly using OpenCV, and were able to see the difference in results compared to simpler algorithms implemented in the past. Everything was implemented with OpenCV, which made the implementation of these algorithms fast and efficient. These algorithms were applied on both Gray level images and color images, and similar to previous assignments up to 3 Regions of Interest (ROI) were utilized. We were able to see how these algorithms, as advanced as they are, can still be improved and complemented by other algorithms such as Histogram Equalization. So we can always make out algorithms even more adaptive to suit our needs. There was a learning curve with OpenCV, but it is truly a great tool for testing image processing techniques quickly and efficiently. We saw how fast it was to implement a QRCodeDetector with OpenCV. Ultimately, this assignment provided a productive way to further improve our knowledge in Image Processing.