Assignment 2

Author 1, Ethan Eisenga
Affiliation: Florida Polytechnic University

A report on assignment 2 which consists of blurring and detecting edges with various methods of OpenCV and manually.

Report GIRS-2024-23-01

Department of Computer Science
Florida Polytechnic University, Florida, USA

February 2024

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The original image of a bicycle was blurred using Box Filters and Gaussian both with manually written code and using OpenCV. Additionally, Sobel was applied to detect edges of the image with 1st derivative kernels in both the X-Axis and Y-Axis, as well as the XY-Axis by applying Pythagorean's Theorem. Kernels of different diameters were applied to determine how various diameters affected the amount of blurring was applied and how the edge detection was affected.

2 PROGRAM

An in depth analysis of each partition in the program.

2.1 PRE-PROCESSING

OpenCV and Numpy are the only libraries that were imported. Some constants were defined for Window Creation and Filter Application below. No explicit global constants were needed, however. The original image was loaded initially as 'image' to be used by all filters. See Figure 2.1.

```
1  import cv2 as cv
2  import numpy as np
3
4  # Load Image
5  image = cv.imread('bicycle.bmp')
```

Figure 2.1. Lines 1-5: Pre-Processing which includes importing libraries and loading the original image

2.2 METHODS

Methods used in the program that were not imported by another library, although inspired by the cited references. Includes work done for manual filter application.

2.2.1 create_kernel()

This method is called whenever a kernel is needed, particularly for Filter Application. kernel_diameter is inputted such that a diameter of 5 corresponds to a 5x5 kernel filled with 1's. This is used for manual box filter as it uses all 1's in the kernel to find an unweighted average. See Figure 2.2.

```python
7  def create_kernel(kernel_diameter):
8    # Confirm diameter is a positive odd integer
9    if (kernel_diameter%2!=1 or kernel_diameter<=0):
10     raise ValueError('create_kernel() kernel_diameter must be a positive odd integer')
11
12   return [[1] * kernel_diameter for _ in range(kernel_diameter)] #diameter of 3 = 3x3 kernel
```

Figure 2.2. Lines 7-12: create_kernel() method which creates a kernel of specified diameter

2.2.2 convolution()

This method is called whenever convolution needs to be applied to an image. The parameters required are the image, and a kernel. This is manual convolution as opposed to OpenCV-convolution. The Kernel radius is derived from the diameter as it represents the number of pixels on all 4 sides that are not able to be included in the convolution and need to be manually set as black pixels. The condition on line 30 is what determines if the current pixel is outside of the border or not. Line 32 creates an array that represents the pixels in the current mask, which are then averaged and stored in the current pixel. See Figure 2.3.

```python
14 # Convolution manually written
15 def convolution(image, kernel):
16   kernel_diameter = len(kernel[0]) # get diameter of kernel
17   kernel_radius = (kernel_diameter-1)//2 #i.e. 3x3 has radius of 1, the center pixel is not counted as part of the radius.
18   #// instead of / to ensure its outputted as an integer type
19
20   # Get the dimensions of the image
21   height, width, channels = image.shape
22
23   # Set the size of new image, will have kernel_radius black pixels on each side
24   average = np.zeros((height, width, channels), dtype=np.uint8) # dtype=np.uint8 seems to be necessary otherwise the image gets turned into intermittent pink pixels
25
26   # Iterate through each pixel
27   for y in range(height):
28     for x in range(width):
29       for c in range(channels):
30         if x<=kernel_radius or y<=kernel_radius or x>=(width-kernel_radius) or y>=(height-kernel_radius): #if in the border where kernel can't touch, fill as black
31           average[y, x, c] = 0
32           continue
33         window = image[y-kernel_radius:y+kernel_radius+1, x-kernel_radius:x+kernel_radius+1, c]
34         average[y, x, c] = np.sum(window * kernel)/kernel_diameter**2
35
36   return average
```

Figure 2.3. Lines 14-36: convolution() method which performs convolution on an image with a specified kernel

2.2.3 apply_threshold()

This method applies a given threshold in a range of [0, 1] to a given image. The threshold represents the % distance from the minimum pixel to the maximum pixel's magnitude. If a threshold is given outside the specified range a Value Error is raised. The image is converted from BGR to Grayscale with the following weights on line 54 [.144, .587, .299]. The min and max pixel magnitude first need to be found in lines 56-65. Afterwards, a final iteration occurs where pixels that meet the threshold are black, and pixels that don't are white. See Figure 2.4.

```python
38 # Apply a threshold to an image by outputting white if it does not meet the threshold,
39 # black if it does. Threshold must range from [0-1]
40 def apply_threshold(image, threshold):
41     #threshold refers to the % distance from min towards max, from 0 to 100% [0, 1]
42     #threshold of .5 refers to the midpoint between min and max
43     #threshold of 1 refers to the max value
44     #threshold of 0 refers to the min value
45 
46     if threshold<0 or threshold>1:
47         raise ValueError('apply_threshold() threshold must be in range [0,1]')
48 
49     # Get the dimensions of the image
50     height, width, channels = image.shape
51 
52     # Set the size of new image, will have kernel_radius black pixels on each side
53     thresholded_image = np.zeros((height, width, channels), dtype=np.uint8) # dtype=np.uint8 seems to be necessary otherwise the image gets turned into intermitten pink pixels
54     channel_weights = [.144, .587, .299] #bgr 0.299 · Red + 0.587 · Green + 0.114 · Blue
55 
56     # Determine min/max for normalization
57     min = np.inf
58     max = -np.inf
59     for y in range(height):
60         for x in range(width):
61             for c in range(channels):
62                 if image[y, x, c]<min: #new min
63                     min = image[y, x, c]
64                 elif image[y, x, c]>max: #new max
65                     max = image[y, x, c]
66 
67     # Set pixels to either 0 or 255 if it meets the threshold
68     for y in range(height):
69         for x in range(width):
70             # average value between channels
71             sum = 0
72             for c in range(channels):
73                 sum = sum + image[y, x, c]*channel_weights[c] #*channel_weights to convert to grayscale
74             average = sum//3 #round down to stay as integer
75             if average>(max-min)*threshold+min: #if the weighted average of all 3 channels meets the threshold
76                 thresholded_image[y, x, 0] = 0 #black if >thresh
77                 thresholded_image[y, x, 1] = 0 #black if >thresh
78                 thresholded_image[y, x, 2] = 0 #black if >thresh
79             else:
80                 thresholded_image[y, x, 0] = 255 #white if <thresh
81                 thresholded_image[y, x, 1] = 255 #white if <thresh
82                 thresholded_image[y, x, 2] = 255 #white if <thresh
83 
84     return thresholded_image
```

Figure 2.4. Lines 38-84: apply_threshold() method which applies a filter to a specified image

2.3 WINDOW CREATION

All windows are created for each blurring and filter technique, both manual and opencv, of unspecified kernel diameter. The windows are shifted horizontally and vertically by the width and height of the image, respectively. See Figure 2.5.

2.4 FILTER APPLICATION

All filters and blurring techniques are applied with inputted kernel_diameter of either 3 or 5. The original image is loaded without changes. Box filter is applied by OpenCV and manually. Guassian is applied by OpenCV. Sobel is applied for X, Y, and XYAxis manually, and XyAxis with OpenCV. For Sobel manuals, the kernels are also inputted manually as copies found from the cited sources instead of computing it on-site. As such, the kernel_diameter must be either 3 or 5 in order for Sobel manual to function correctly. For Sobel Manual, the image is convolved using convolution(), then thresholded using apply_threshold(). For XY-Axis, the X axis and Y axis are computed into XY-Axis using (x**2+y**2)**.5. See Figure 2.6.

```
87   # Create Windows
88   # Set dimensions for window separation
89   height, width, channel = image.shape # Get height/width quickly for window size and offset
90   height=height+30 # Add 28 pixels of height for the window tab size
91
92   # Original
93   cv.namedWindow('Original')
94   cv.moveWindow('Original', width*2, height*1)
95
96   # Box Filter
97   cv.namedWindow('BoxFilter')
98   cv.moveWindow('BoxFilter', width*0, height*0)
99
100  # Box Filter Manual
101  cv.namedWindow('BoxFilterManual')
102  cv.moveWindow('BoxFilterManual', width*0, height*1)
103
104  # Gaussian
105  cv.namedWindow('Gaussian')
106  cv.moveWindow('Gaussian', width*0, height*2)
107
108  # Sobel
109  cv.namedWindow('Sobelmanualxaxis')
110  cv.moveWindow('Sobelmanualxaxis', width*1, height*0)
111  cv.namedWindow('Sobelmanualyaxis')
112  cv.moveWindow('Sobelmanualyaxis', width*1, height*1)
113  cv.namedWindow('Sobelmanualxyaxis')
114  cv.moveWindow('Sobelmanualxyaxis', width*1, height*2)
115  cv.namedWindow('Sobelxyaxis')
116  cv.moveWindow('Sobelxyaxis', width*2, height*2)
```

Figure 2.5. Lines 87-116: Creation of all windows

2.5 SHOW IMAGES

The images for original, boxfilter, gaussian, and sobel for both manual and opencv are displayed to the screen at once with specified kernel_diameter. After any key is pressed, the windows are closed. See Figure 2.7.

```
120 # Create Filtered images
121 kernel_diameter = 5
122 # Original
123 image = image #no effect
124 
125 # Box Filter
126 box = cv.boxFilter(image, -1, (kernel_diameter,kernel_diameter))
127 
128 # Box Filter Manual
129 boxManual = convolution(image, create_kernel(kernel_diameter))
130 
131 # Gauss
132 gauss = cv.GaussianBlur(image,(kernel_diameter,kernel_diameter),0)
133 
134 # Sobel
135 # Set kernel and threshold
136 if kernel_diameter==3:
137   vert_kernel = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]] #vertical edge detector (product of deriv and gaussian filters)
138   threshold = .2 #halved for xyaxis
139 elif kernel_diameter==5:
140   vert_kernel = [[1,2,0,-2,-1],
141                  [4,8,0,-8,-4],
142                  [6,12,0,-12,-6],
143                  [4,8,0,-8,-4],
144                  [1,2,0,-2,-1]]
145   threshold = .15 #halved for xyaxis
146 else:
147   raise ValueError('Sobel manual kernel_diameter must be 3 or 5')
148 horiz_kernel = np.transpose(vert_kernel) #horizontal edge detector
149 
150 sobelmanual_xaxis = apply_threshold(convolution(image, vert_kernel), threshold)
151 sobelmanual_yaxis = apply_threshold(convolution(image, horiz_kernel), threshold)
152 sobelmanual_xyaxis = apply_threshold((convolution(image, vert_kernel)**2 + convolution(image, horiz_kernel)**2)**.5, threshold)  #sqrt(x^2+y^2)
153 
154 sobel_xyaxis = cv.cvtColor(image, cv.COLOR_BGR2GRAY) # convert from bgr to grayscale automatically
155 sobel_xyaxis = cv.GaussianBlur(sobel_xyaxis,(kernel_diameter,kernel_diameter), sigmaX=0, sigmaY=0)
156 sobel_xyaxis = cv.Sobel(src=sobel_xyaxis, ddepth=cv.CV_64F, dx=1, dy=1, ksize=kernel_diameter)
```

Figure 2.6. Lines 120-156: Filter Application

```
159 # Show Images
160 # Original
161 cv.imshow('Original', image)
162 
163 # BoxFilter
164 cv.imshow('BoxFilter', box)
165 cv.imshow('BoxFilterManual', boxManual)
166 
167 # Gaussian
168 cv.imshow('Gaussian', gauss)
169 
170 # Sobel
171 cv.imshow('Sobelmanualxaxis', sobelmanual_xaxis)
172 cv.imshow('Sobelmanualyaxis', sobelmanual_yaxis)
173 cv.imshow('Sobelmanualxyaxis', sobelmanual_xyaxis)
174 cv.imshow('Sobelxyaxis', sobel_xyaxis)
175 
176 # Wait to close
177 cv.waitKey(0)
178 cv.destroyAllWindows()
```

Figure 2.7. Lines 159-178: Show Images

3 FUNCTIONALITY

3.1 ORIGINAL

The original image is displayed without any blurring or filters applied. Original 3.1.

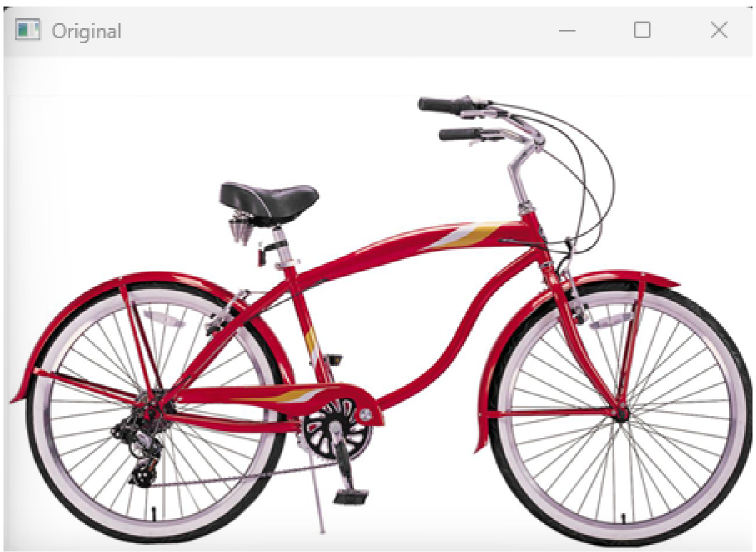

Figure 3.1. Original image without blurring/filtering

3.2 BOX FILTERS

The manual box filter is computed by calling convolution() with a 1-filled kernel using create_kernel(), then displayed to the screen. OpenCV box filter is also created and displayed. Both are displayed for both 3x3 and 5x5 kernels.

3.2.1 Manual

3X3 3.4.

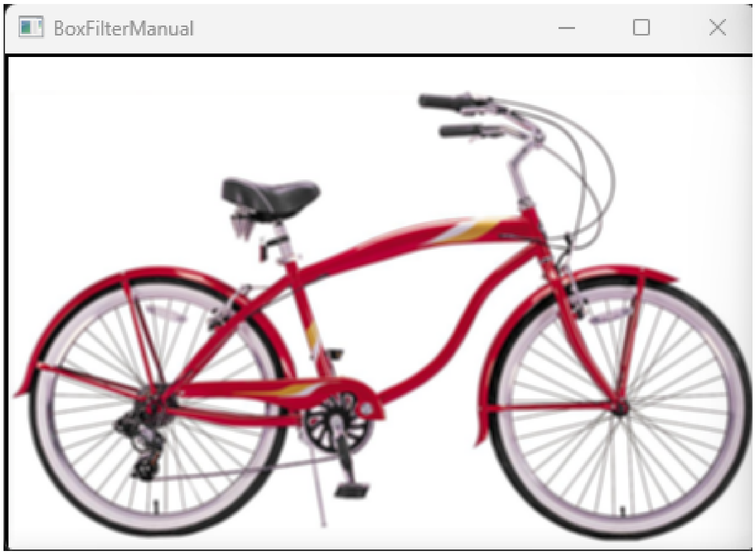

Figure 3.2. Box Filter 3x3 Manually computed

5X5 3.5.

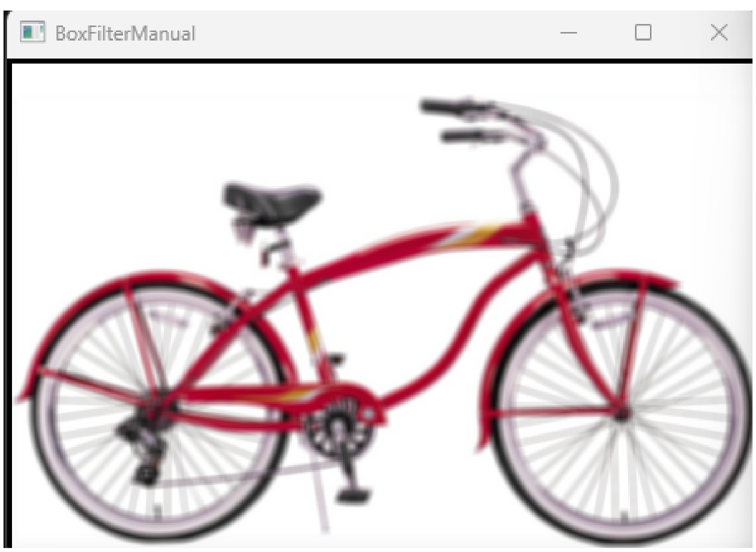

Figure 3.3. Box Filter 5x5 Manually computed

3.2.2 OpenCV

3X3 3.4.

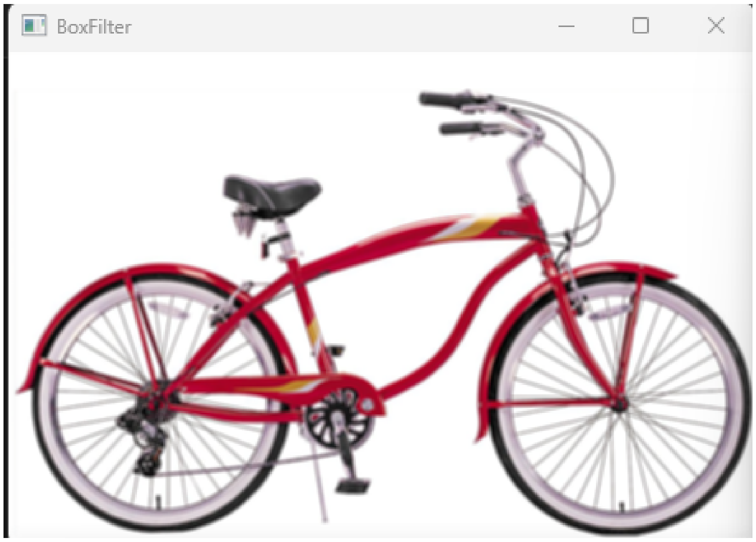

Figure 3.4. Box Filter 3x3 using OpenCV library

5X5 3.5.

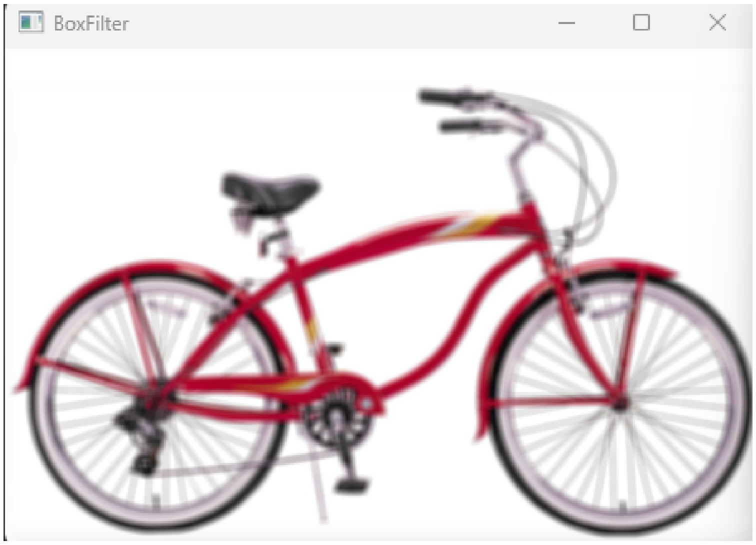

Figure 3.5. Box Filter 5x5 using OpenCV library

3.3 GAUSSIAN

Gaussian blurring is applied strictly with OpenCV and not manually.

3.3.1 OpenCV

3X3 3.6.

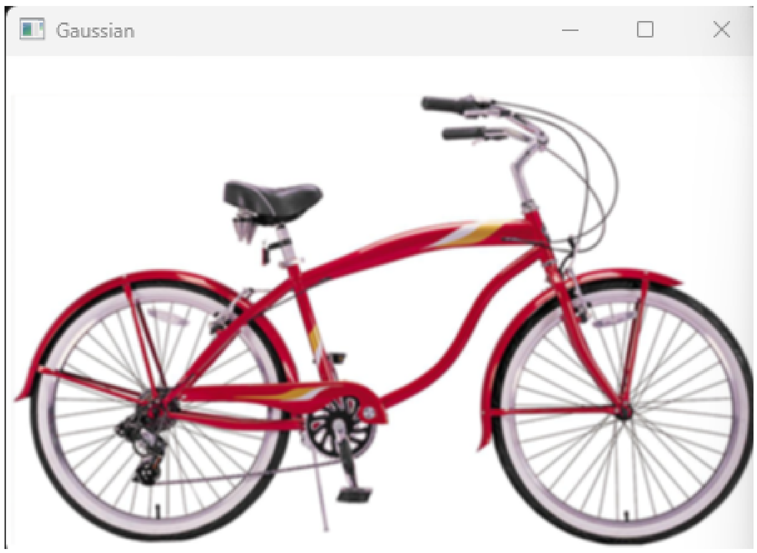

Figure 3.6. Gaussian 3x3 using OpenCV libary

5X5 3.7.

3.4 SOBEL

Sobel filters are applied manually and with OpenCV for X-Axis, Y-Axis, and XY-Axis. The XY-Axis referred to simply equates to (X**2+Y**2)**.5, or rather, Pythagorean's Theorem applied to the X-Axis and Y-Axis either manually or with OpenCV.

3.4.1 X-Axis

Manual

3X3 3.8.

5X5 3.9.

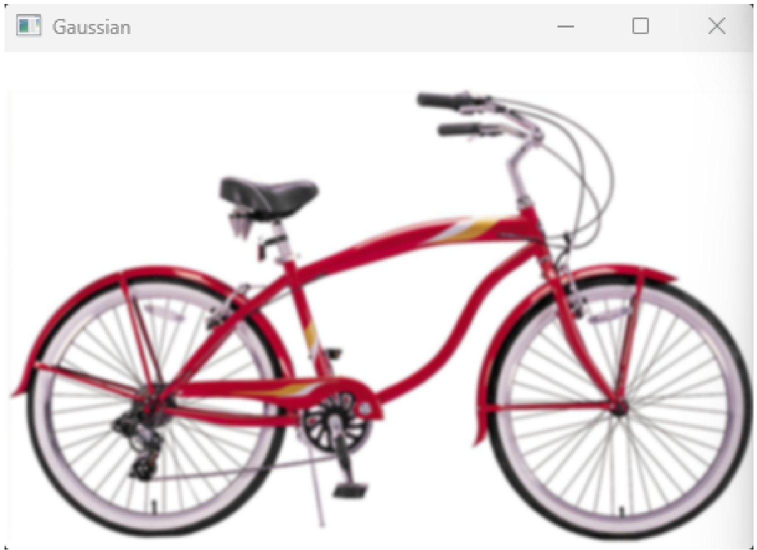

Figure 3.7. Gaussian 5x5 using OpenCV libary

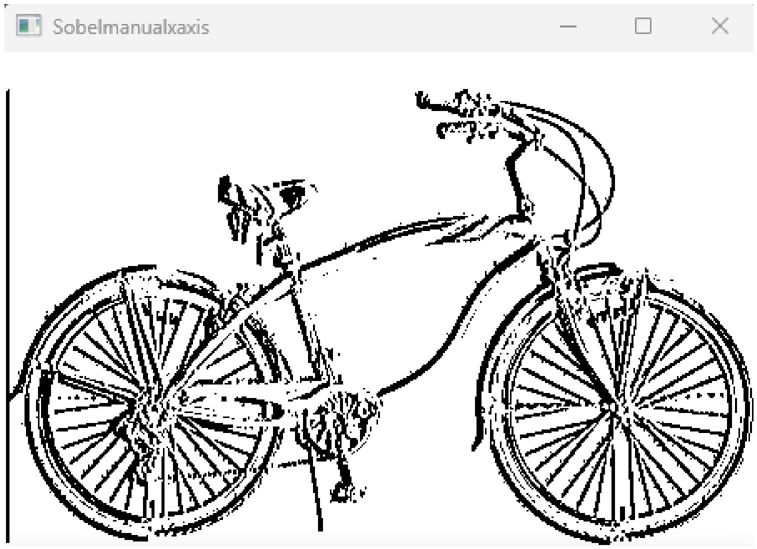

Figure 3.8. Sobel 3x3 X-Axis Manually computed

3.4.2 Y-Axis

Manual

3X3 3.10.

5X5 3.11.

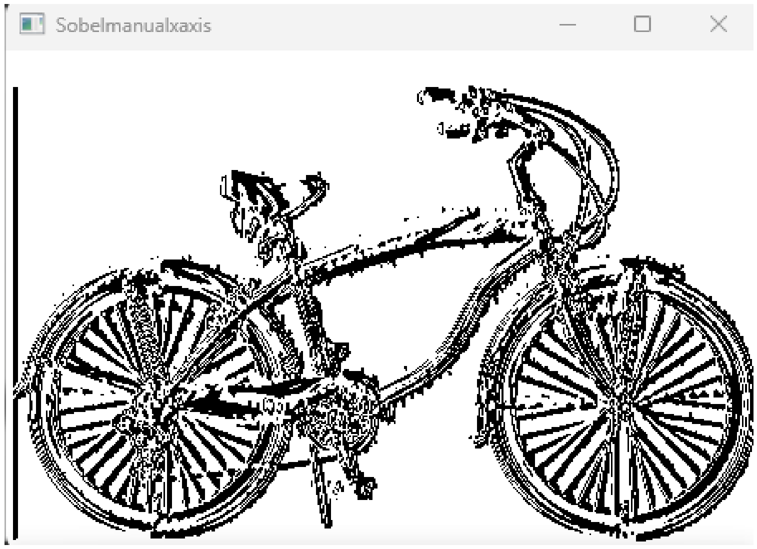

Figure 3.9. Sobel 5x5 X-Axis Manually computed

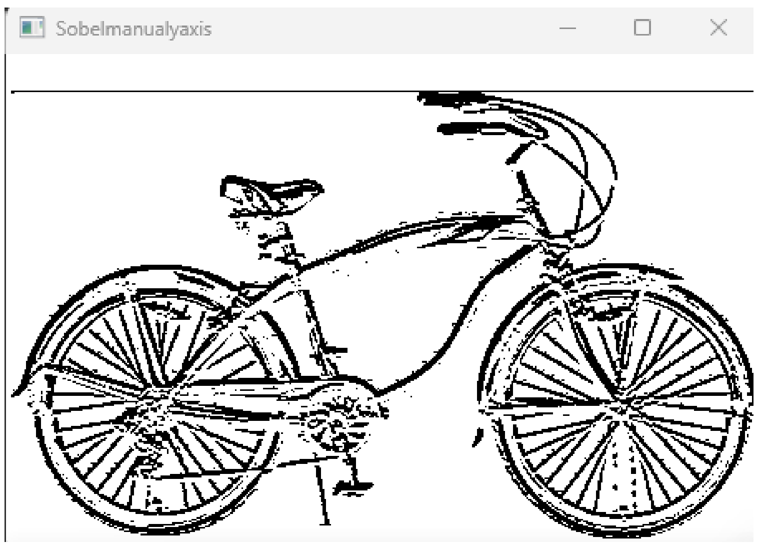

Figure 3.10. Sobel 3x3 Y-Axis Manually computed

3.4.3 XY-Axis

Manual

3X3 3.12.

5X5 3.13.

OpenCV

3X3 3.14.

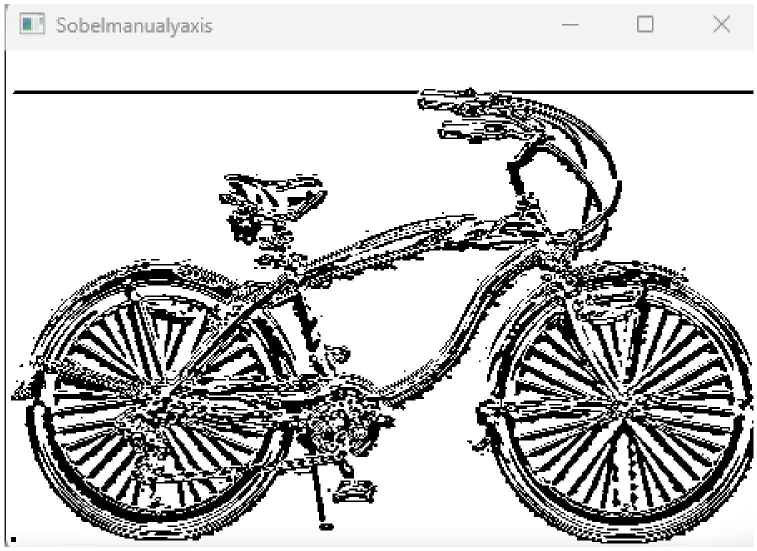

Figure 3.11. Sobel 5x5 Y-Axis Manually computed

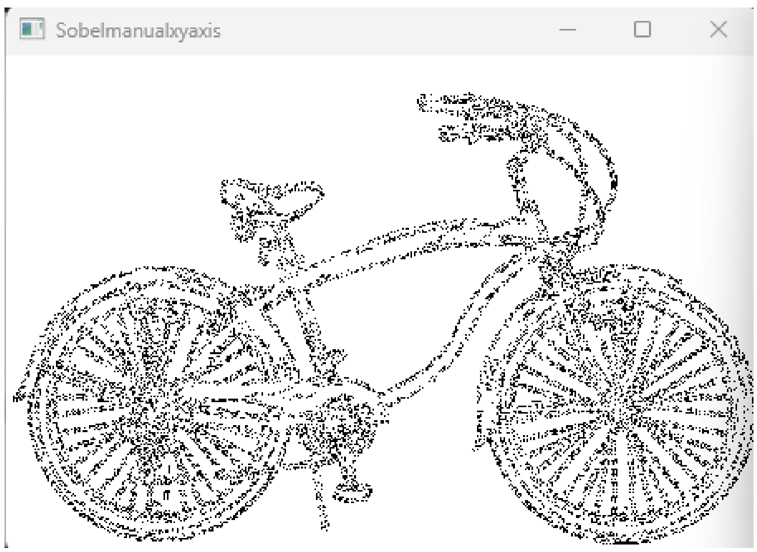

Figure 3.12. Sobel 3x3 XY-Axis Manually computed

5X5 3.15.

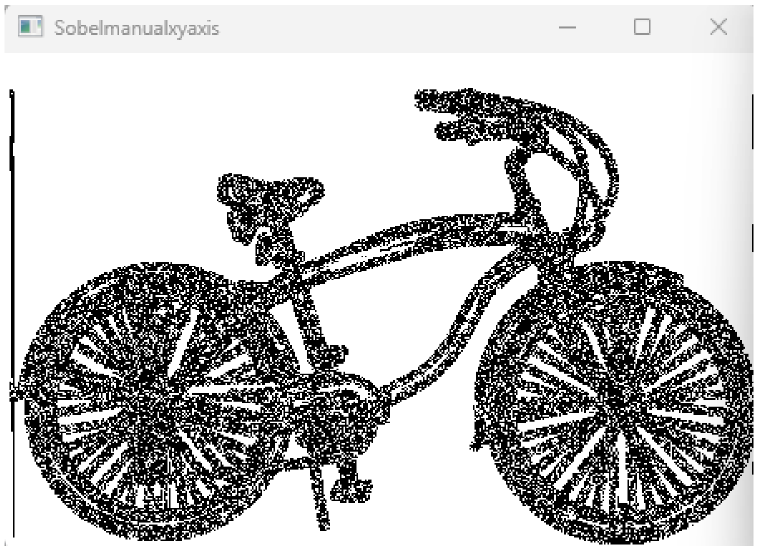

Figure 3.13. Sobel 5x5 XY-Axis Manually computed

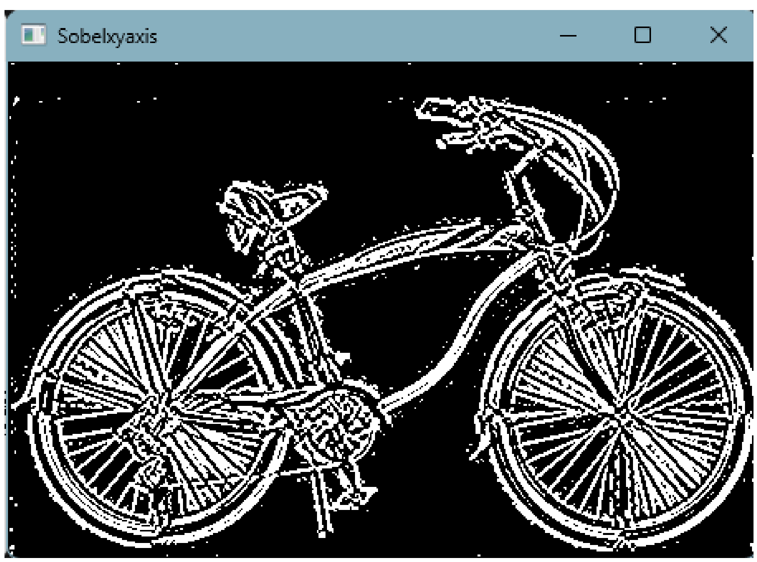

Figure 3.14. Sobel 3x3 XY-Axis using OpenCV

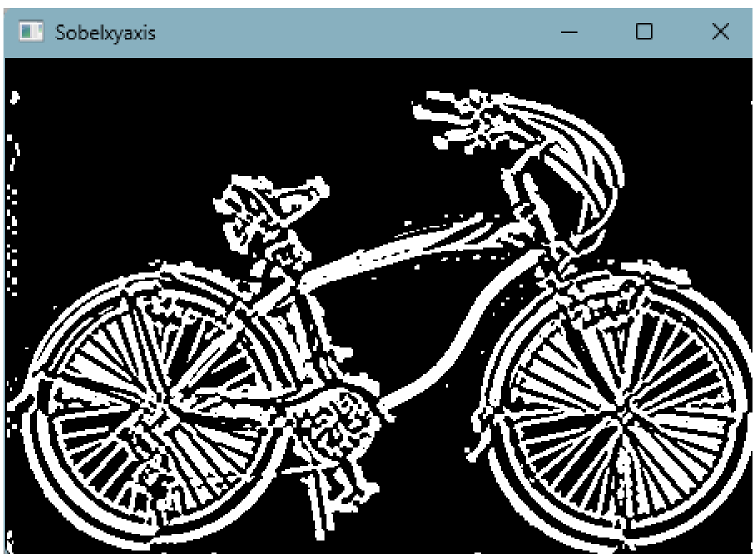

Figure 3.15. Sobel 5x5 XY-Axis using OpenCV

4 DISCUSSION

When applying the filter to images in BGR instead of Grayscale, some channels would meet the threshold while others weren't. As a result, one pixel could be outputted as [255, 0, 0]. Even though the pixels are outputted as either 0 or 255, the outputted pixel is blue using these values. All combinations were created, such as [255, 255, 0], [0, 255, 0], etc. The apply_threshold() method was adjusted first by calculating the average value between the 3 channels. Later, it was learned that gray scale can be computed as just a weighted average of the 3 channels. This weighted average was used in place of the unweighted average, and the BGR image, to represent the image in Grayscale.

There were difficulties making the apply_threshold work simply. In its current form, the threshold parameter represents the % distance from the min to the max gray-scale value in the image, that way the threshold is based on a normalized image. However, it seems that min and max aren't great values to base it off, as most pixels that aren't the background share very similar grayscale values. If this were to be redone, the threshold would instead be related to the # of z-scores from the mean, with negative values being below the mean, and positive values above the mean. Even this doesn't seem like it would detect edges nearly as well as the cv-built in Sobel method.

Additionally, there were issues getting the cv-Sobel to have the same formatting as all other images. The color of the edges and the background are inverted. An attempt was made to replace 'sobel' with '255 – sobel' to invert the colors, but the resulting image was just a white background with no detected images. The solution was never found, and the cv-Sobel was left as incorrectly inverted colors.

Lastly, the manual xy-axis Sobel may be calculated from the x-axis Sobel and the y-axis Sobel incorrectly. After applying (x**2+y**2)**.5, the output seems fragmented. Especially for the 5x5 kernel. Different thresholds were attempted to get around the fragmentation, but did not have any significant benefit.

REFERENCES

[1] Blur Image using cv2.blur(), indianaiproduction, 2024, https://indianaiproduction.com

[2] Edge Detection using OpenCV, learnopencv, 2024, https://learnopencv.com

[3] Image Blurring, OpenCV, 2024, docs.opencv.org