# Assignment 2

**Abigail Stucki**
Computer Science Undergraduate Student, Florida Polytechnic University

A report on computer vision focused on the creation and implementation of filters upon imagery.

Department of Computer Science
Florida Polytechnic University, Florida, USA

February 2024

## ABSTRACT

The premise of this project was to apply knowledge of the mathematical and logical components of image filtration via code. The entire foundation upon which computer vision is built encompasses the analysis and data extraction of images or similar forms of information being passed to a computer and then parsed to be further assessed. Image filtering and the ability to process images as matrices to apply mathematical functions is key to such ordeals.

Numerous image filters were expanded upon through the educational setting in a presentation detailing the mathematical premise behind image filtering and the physical changes caused by these functions. The Box Filter or Mean Filter, the Sobel Filter, and the Gaussian Blur Filter were centered constructs.

Programmed implementation of these features was developed using both manual manners as well as with an external library to provide comparison and further comprehension of how image filtering is accessible. The following documentation will detail the application of these concepts in Python code for image analysis and manipulation.

# CONTENTS

ABSTRACT . . . . . . . . . . ii

CONTENTS . . . . . . . . . . iii

LIST OF FIGURES . . . . . . . . . . iv

1 INTRODUCTION . . . . . . . . . . 1

1.1 BOX FILTER . . . . . . . . . . 1

1.2 BOX FILTER (OPENCV) . . . . . . . . . . 4

1.3 X-EDGE SOBEL FILTER . . . . . . . . . . 5

1.4 Y-EDGE SOBEL FILTER. . . . . . . . . . . 6

1.5 COMBINATION SOBEL FILTER . . . . . . . . . . 7

1.6 SOBEL FILTER (OPENCV) . . . . . . . . . . 9

1.7 GAUSSIAN BLUR . . . . . . . . . . 10

1.8 FINAL CODE OUTPUT . . . . . . . . . . 12

REFERENCES . . . . . . . . . . 13

# LIST OF FIGURES

Figure 1.1 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 2
Figure 1.2 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 2
Figure 1.3 Code calculating the average value within a size x size matrix. . . . . . . . . 2
Figure 1.4 Code implementing a manual Box Filter. . . . . . . . . . . . . . . . . . . . . 3
Figure 1.5 Visual example of Box Filter. . . . . . . . . . . . . . . . . . . . . . . . . . . 3
Figure 1.6 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 4
Figure 1.7 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 4
Figure 1.8 Code demonstrating OpenCV's Box Filter. . . . . . . . . . . . . . . . . . . . 4
Figure 1.9 Visual example of Box Filter using OpenCV. . . . . . . . . . . . . . . . . . . 5
Figure 1.10 Visual comparison of Box Filter manual and using OpenCV. . . . . . . . . . 5
Figure 1.11 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 6
Figure 1.12 Code demonstrating manual implementation of an X-edge Sobel Filter. . . . 6
Figure 1.13 Visual example of Sobel Filter on X-edge. . . . . . . . . . . . . . . . . . . . 6
Figure 1.14 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 7
Figure 1.15 Code demonstrating manual implementation of a Y-edge Sobel Filter. . . . . 7
Figure 1.16 Visual example of Sobel Filter on Y-edge. . . . . . . . . . . . . . . . . . . . 7
Figure 1.17 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 8
Figure 1.18 Code demonstrating manual implementation of a combined Sobel Filter. . . 8
Figure 1.19 Visual example of a combined Sobel Filter. . . . . . . . . . . . . . . . . . . 8
Figure 1.20 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 9
Figure 1.21 Code demonstrating manual implementation of a combined Sobel Filter. . . 9
Figure 1.22 Visual example of OpenCV's Sobel Filter. . . . . . . . . . . . . . . . . . . . 9
Figure 1.23 Visual comparison of the varying Sobel Filters. . . . . . . . . . . . . . . . . 10
Figure 1.24 Code demonstrating display of altered image in a subplot. . . . . . . . . . . 10
Figure 1.25 Code demonstrating OpenCV's Gaussian Blur Filter. . . . . . . . . . . . . . 10
Figure 1.26 Visual example of OpenCV's Gaussian Blur Filter. . . . . . . . . . . . . . . . 11
Figure 1.27 Image of final code output. . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

# 1 INTRODUCTION

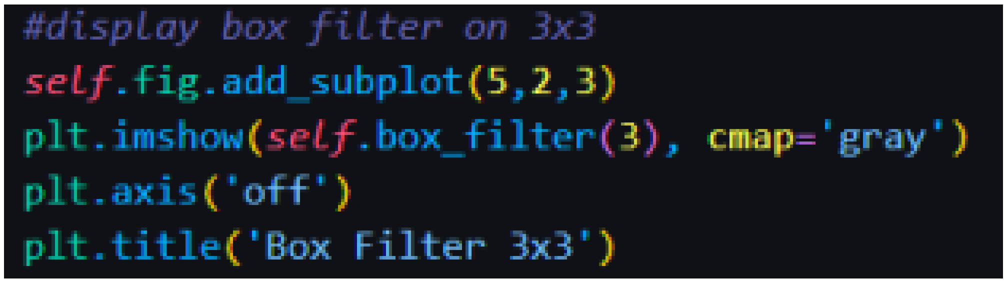
This document entails the progression and demonstration of using the Python programming language to apply computer vision techniques focused on image filtration. Along with manual implementation, these functions accessed external libraries including the following: OpenCV-Python library (Bradski 2000), Python OS module, NumPy library (Harris et al. 2020), and Matplotlib library (Hunter 2007).

The essential tasks to be executed in the program included:

1. Manual Box Filter implementation.
2. Box Filter implementation via the OpenCV library.
3. Manual Sobel Filter implementation with a focus on X-axis edges.
4. Manual Sobel Filter implementation with a focus on Y-axis edges.
5. Manual Sobel Filter implementation combining both X-axis and Y-axis edges.
6. Sobel Filter implementation via the OpenCV library.
7. Gaussian Filter implementation via the OpenCV library.

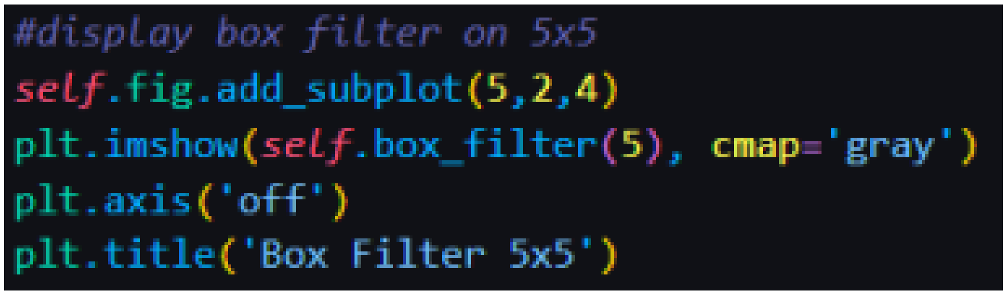
## 1.1 BOX FILTER

A Box Filter, otherwise referred to as a Mean Filter, replaces pixels of a predetermined range (3x3, 5x5, etc.) with the average pixel value of the surrounding values within the aforementioned range. The library Matplotlib was used in displaying the resulting image. In order to implement this filter, an understanding of linear algebra and the mechanics of matrices was vital. The filter traverses the image matrix and analyses smaller chunks in the given dimension size x size to calculate the average value, replacing the original value. The image is then returned and displayed in a visually appealing manner for user analysis and comprehension.



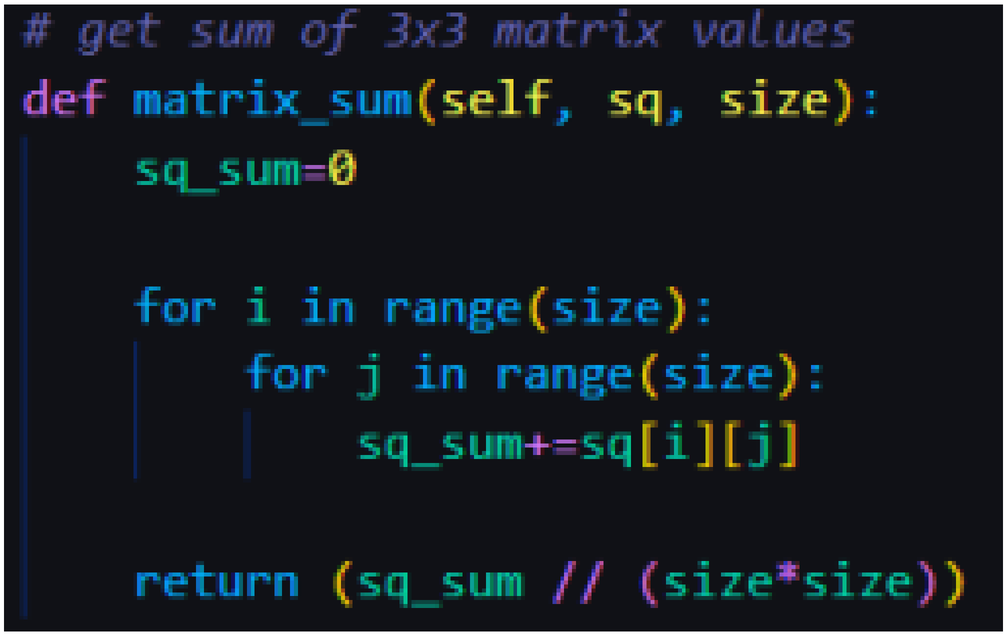
```python
#display box filter on 3x3
self.fig.add_subplot(5,2,3)
plt.imshow(self.box_filter(3), cmap='gray')
plt.axis('off')
plt.title('Box Filter 3x3')
```

**Figure 1.1.** This code calls upon the self.box_filter() function with an input of 3, initiating the creation and eventual display of a 3x3 Box Filter.



```python
#display box filter on 5x5
self.fig.add_subplot(5,2,4)
plt.imshow(self.box_filter(5), cmap='gray')
plt.axis('off')
plt.title('Box Filter 5x5')
```

**Figure 1.2.** This code calls upon the self.box_filter() function with an input of 5, initiating the creation and eventual display of a 5x5 Box Filter.

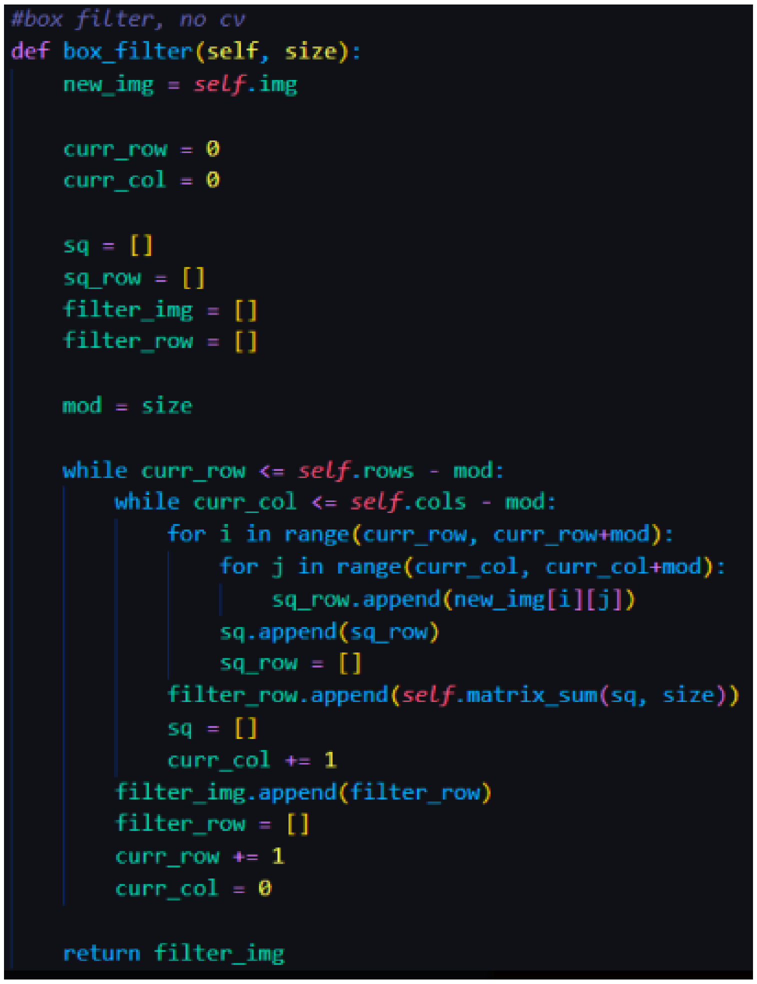


```python
# get sum of 3x3 matrix values
def matrix_sum(self, sq, size):
    sq_sum=0

    for i in range(size):
        for j in range(size):
            sq_sum+=sq[i][j]

    return (sq_sum // (size*size))
```

**Figure 1.3.** This code calculates the average or mean value within a given matrix of dimensions size x size to be used in the application of a Box Filter.



```
#box filter, no cv
def box_filter(self, size):
    new_img = self.img

    curr_row = 0
    curr_col = 0

    sq = []
    sq_row = []
    filter_img = []
    filter_row = []

    mod = size

    while curr_row <= self.rows - mod:
        while curr_col <= self.cols - mod:
            for i in range(curr_row, curr_row+mod):
                for j in range(curr_col, curr_col+mod):
                    sq_row.append(new_img[i][j])
                sq.append(sq_row)
                sq_row = []
            filter_row.append(self.matrix_sum(sq, size))
            sq = []
            curr_col += 1
        filter_img.append(filter_row)
        filter_row = []
        curr_row += 1
        curr_col = 0

    return filter_img
```

**Figure 1.4.** This code applies a box filter to the image by parsing the image matrices and calculating the averages, then returning the altered image. The while and for loops within the function traverse all of the image to ensure the Box Filter completely applies and overwrites the pre-existing image.

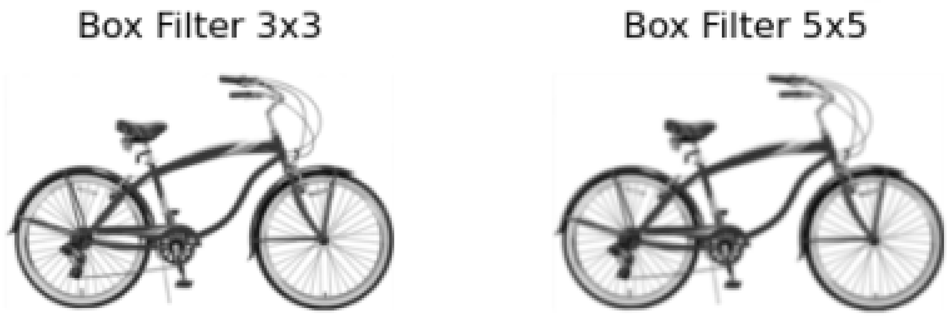
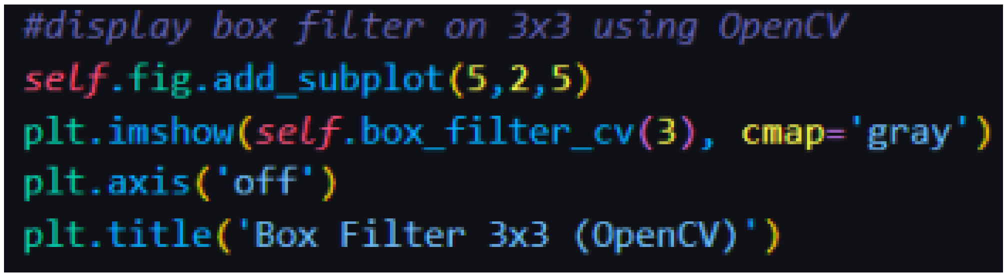


**Figure 1.5.** Displayed image from the manual Box Filter.

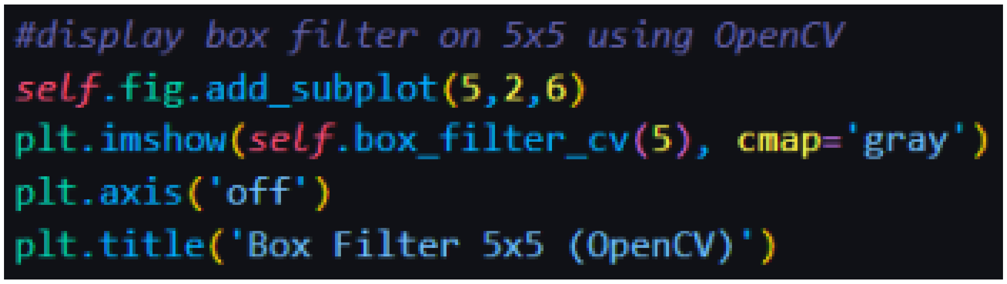
## 1.2 BOX FILTER (OPENCV)

After manual implementation of a Box Filter, another Box Filter function was applied using the OpenCV library and its existing capabilities.



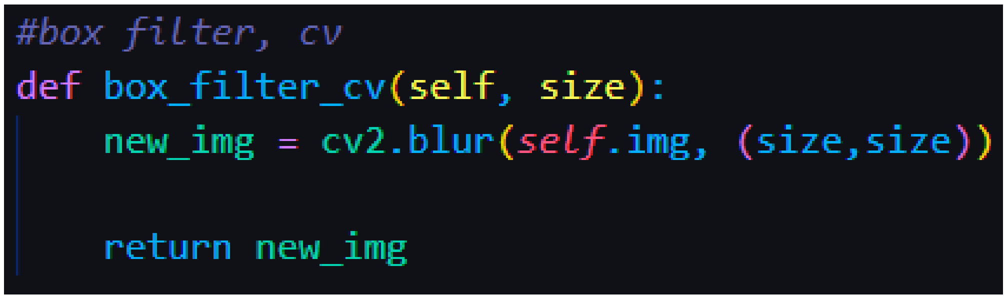
```python
#display box filter on 3x3 using OpenCV
self.fig.add_subplot(5,2,5)
plt.imshow(self.box_filter_cv(3), cmap='gray')
plt.axis('off')
plt.title('Box Filter 3x3 (OpenCV)')
```

**Figure 1.6.** This code calls upon the self.box_filter_cv() function with an input of 3, initiating the creation and eventual display of a 3x3 Box Filter.



```python
#display box filter on 5x5 using OpenCV
self.fig.add_subplot(5,2,6)
plt.imshow(self.box_filter_cv(5), cmap='gray')
plt.axis('off')
plt.title('Box Filter 5x5 (OpenCV)')
```

**Figure 1.7.** This code calls upon the self.box_filter_cv() function with an input of 5, initiating the creation and eventual display of a 5x5 Box Filter.



```python
#box filter, cv
def box_filter_cv(self, size):
    new_img = cv2.blur(self.img, (size,size))

    return new_img
```

**Figure 1.8.** This code demonstrates OpenCV's Box Filter and returns the created image of a given dimension size x size.

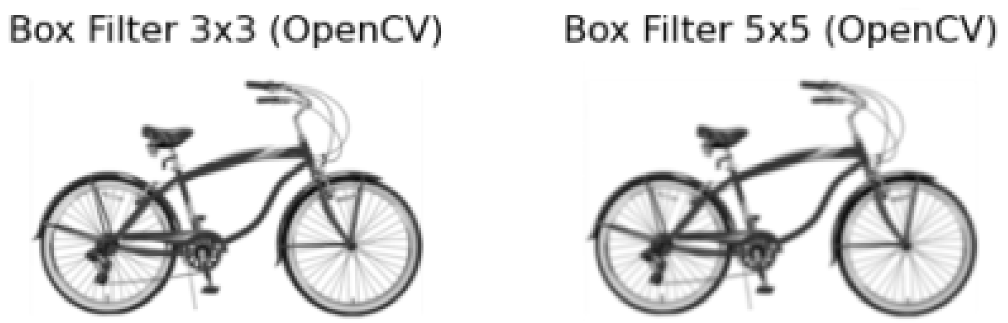


**Figure 1.9.** Displayed image from the application of OpenCV's Box Filter.

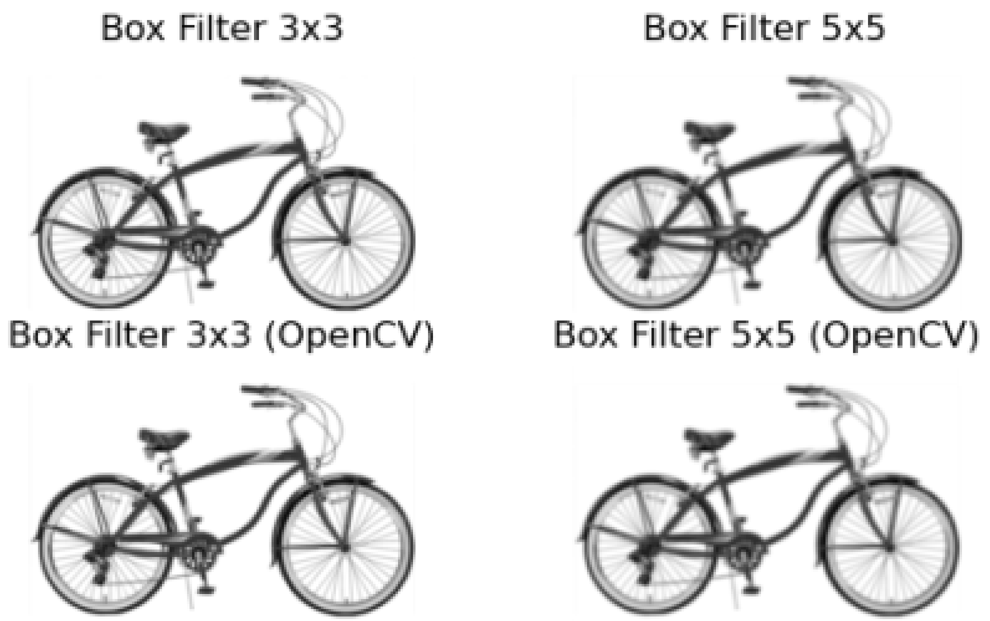
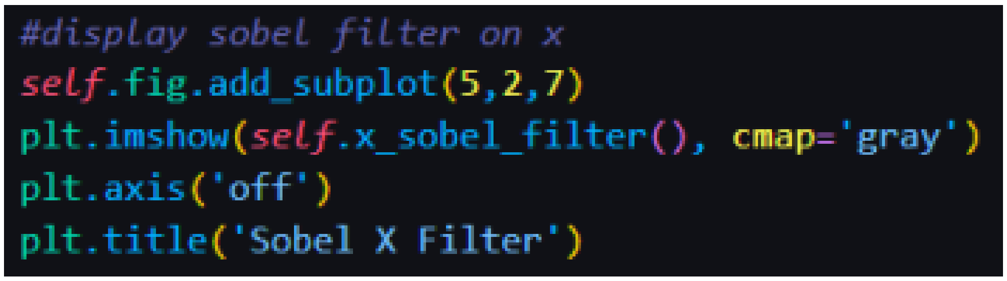


**Figure 1.10.** Displayed images from the various implementations of Box Filter. There is minimal to no appearing differences between the filter quality in resulting images.

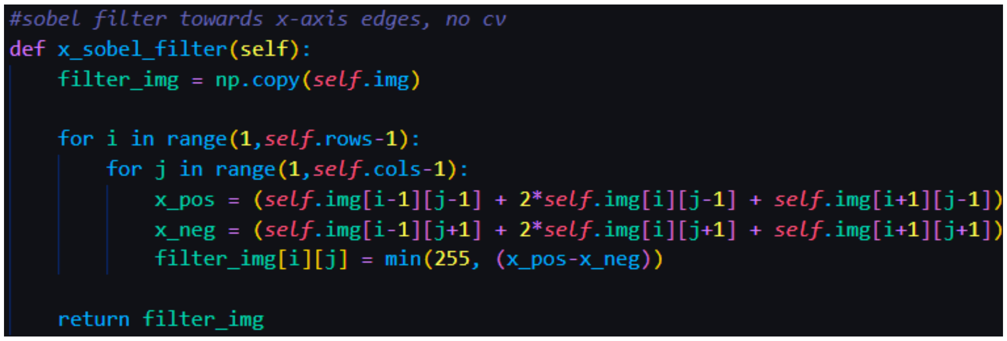
## 1.3 X-EDGE SOBEL FILTER

The next feature to be implemented was a manual Sobel filter focused on the X-edges. The X-edge matrix must be multiplied to each grid and then the resulting sum replaces the center value. This was replicated through the code using loops to traverse the image and apply the filter.



```
#display sobel filter on x
self.fig.add_subplot(5,2,7)
plt.imshow(self.x_sobel_filter(), cmap='gray')
plt.axis('off')
plt.title('Sobel X Filter')
```

**Figure 1.11.** This code calls upon the self.x_sobel_filter() function, initiating the creation and eventual display of an X-edge focused Sobel Filter.



```
#sobel filter towards x-axis edges, no cv
def x_sobel_filter(self):
    filter_img = np.copy(self.img)

    for i in range(1,self.rows-1):
        for j in range(1,self.cols-1):
            x_pos = (self.img[i-1][j-1] + 2*self.img[i][j-1] + self.img[i+1][j-1])
            x_neg = (self.img[i-1][j+1] + 2*self.img[i][j+1] + self.img[i+1][j+1])
            filter_img[i][j] = min(255, (x_pos-x_neg))

    return filter_img
```

**Figure 1.12.** This code demonstrates the use of matrices to apply a Sobel Filter with a focus on the X-edge.

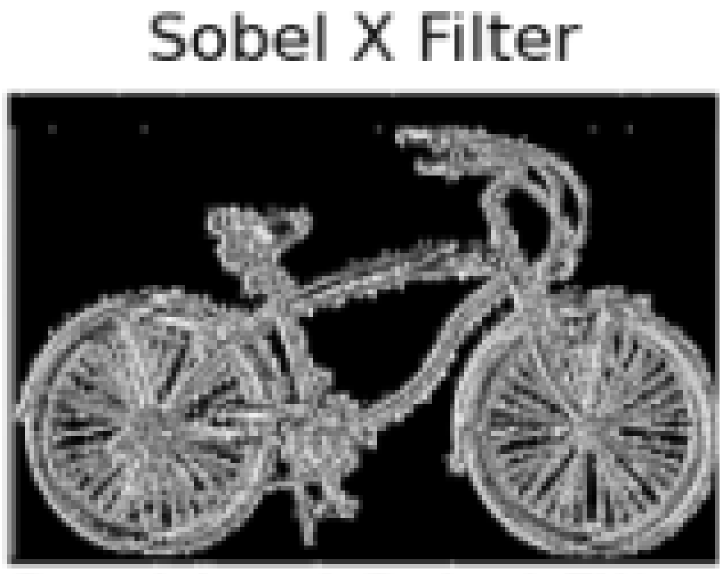
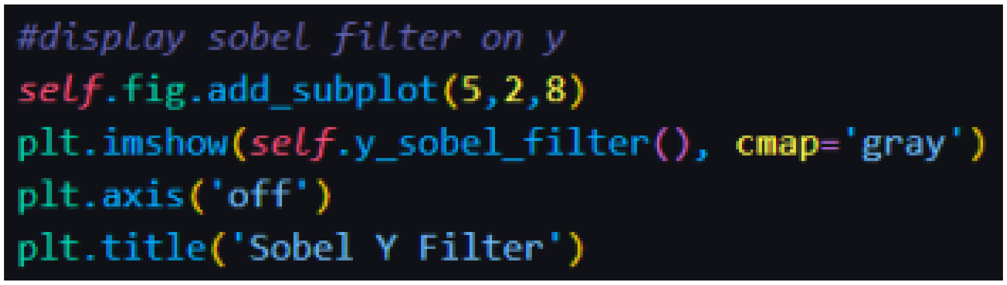


**Figure 1.13.** Displayed image from the manual Sobel Filter on X-edge.

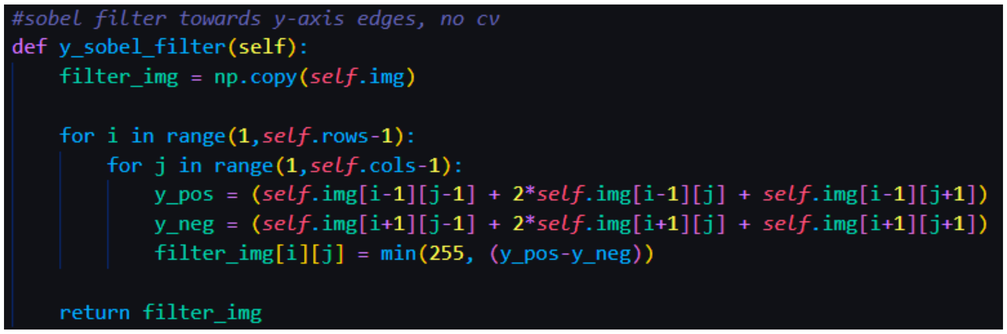
## 1.4 Y-EDGE SOBEL FILTER

The Y-Edge Sobel Filter works similarly to the X-edge implementation, with a slight difference in the matrix used to alter the image in multiplication and summation calculations.



```
#display sobel filter on y
self.fig.add_subplot(5,2,8)
plt.imshow(self.y_sobel_filter(), cmap='gray')
plt.axis('off')
plt.title('Sobel Y Filter')
```

**Figure 1.14.** This code calls upon the self.y_sobel_filter() function, initiating the creation and eventual display of an Y-edge focused Sobel Filter.



```
#sobel filter towards y-axis edges, no cv
def y_sobel_filter(self):
    filter_img = np.copy(self.img)

    for i in range(1,self.rows-1):
        for j in range(1,self.cols-1):
            y_pos = (self.img[i-1][j-1] + 2*self.img[i-1][j] + self.img[i-1][j+1])
            y_neg = (self.img[i+1][j-1] + 2*self.img[i+1][j] + self.img[i+1][j+1])
            filter_img[i][j] = min(255, (y_pos-y_neg))

    return filter_img
```

**Figure 1.15.** This code demonstrates the use of matrices to apply a Sobel Filter with a focus on the Y-edge.

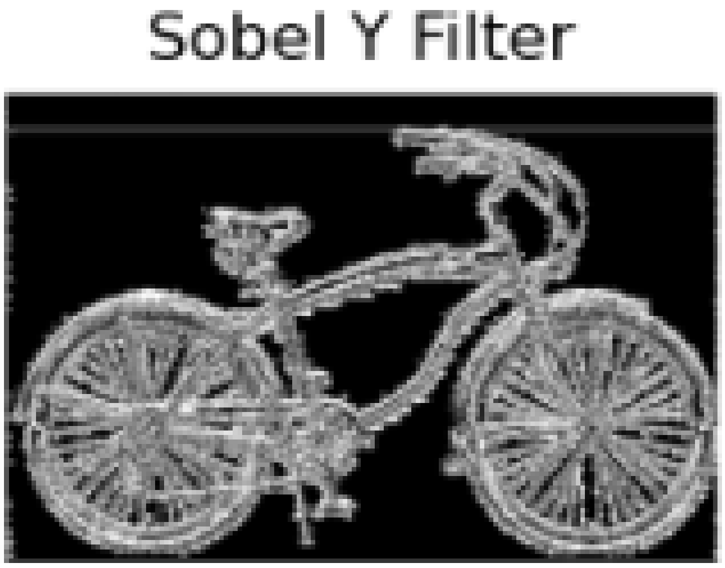
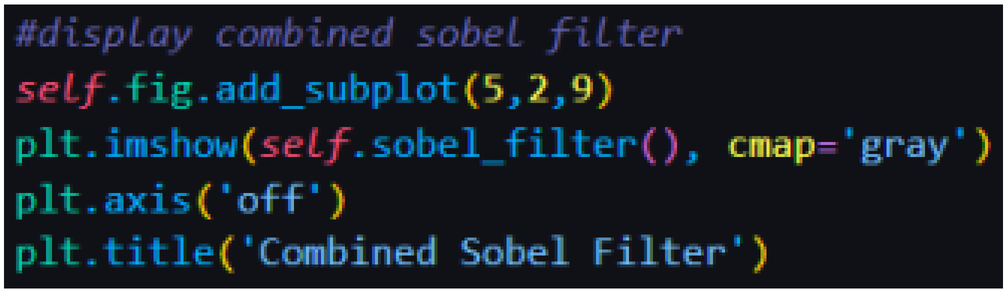


**Figure 1.16.** Displayed image from the manual Sobel Filter on Y-edge.

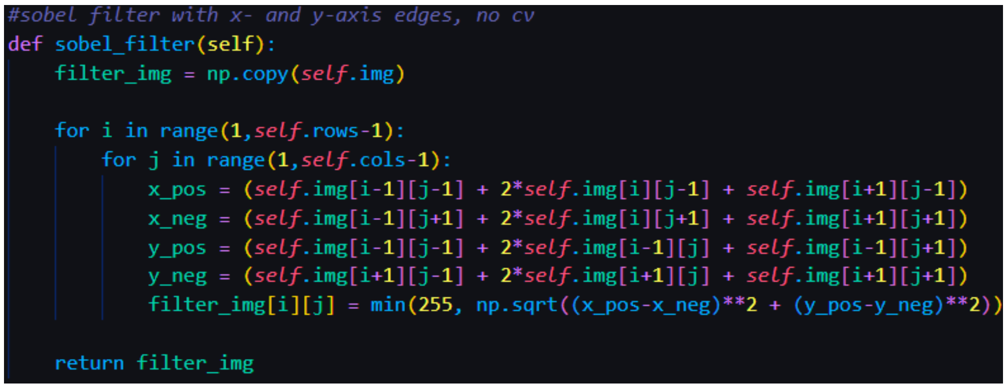
## 1.5 COMBINATION SOBEL FILTER

A combination Sobel Filter was implemented by combining the results of the X-edge and Y-edge Sobel Filters and mathematically balancing the changes to optimize the resulting image.



```
#display combined sobel filter
self.fig.add_subplot(5,2,9)
plt.imshow(self.sobel_filter(), cmap='gray')
plt.axis('off')
plt.title('Combined Sobel Filter')
```

**Figure 1.17.** This code calls upon the self.sobel_filter() function, initiating the creation and eventual display of a combined X- and Y-edge Sobel Filter.



```
#sobel filter with x- and y-axis edges, no cv
def sobel_filter(self):
    filter_img = np.copy(self.img)

    for i in range(1,self.rows-1):
        for j in range(1,self.cols-1):
            x_pos = (self.img[i-1][j-1] + 2*self.img[i][j-1] + self.img[i+1][j-1])
            x_neg = (self.img[i-1][j+1] + 2*self.img[i][j+1] + self.img[i+1][j+1])
            y_pos = (self.img[i-1][j-1] + 2*self.img[i-1][j] + self.img[i-1][j+1])
            y_neg = (self.img[i+1][j-1] + 2*self.img[i+1][j] + self.img[i+1][j+1])
            filter_img[i][j] = min(255, np.sqrt((x_pos-x_neg)**2 + (y_pos-y_neg)**2))

    return filter_img
```

**Figure 1.18.** This code demonstrates the use of matrices to apply a Sobel Filter combining both the X-edges and Y-edges for an optimal output.

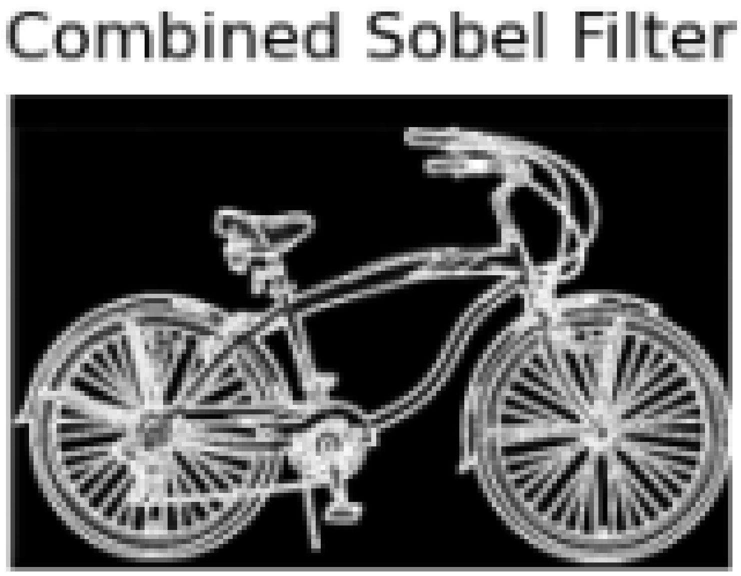
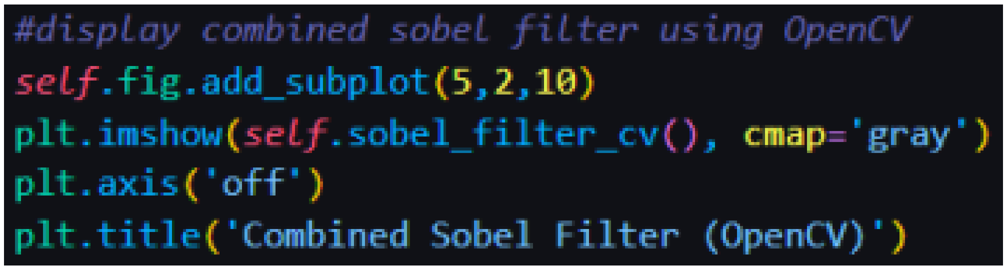


**Figure 1.19.** Displayed image from the manual combined Sobel Filter.

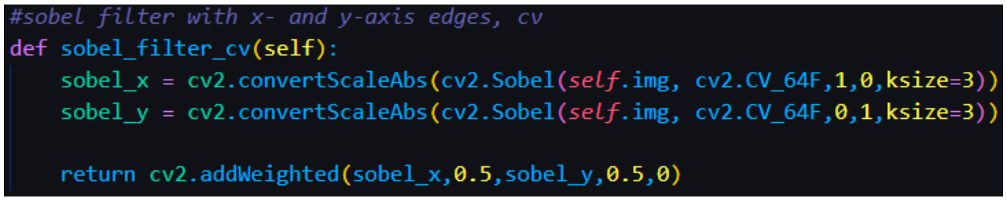
## 1.6 SOBEL FILTER (OPENCV)

After manual implementation of the Sobel Filter, another Sobel Filter function was applied using the OpenCV library and its existing capabilities.



```python
#display combined sobel filter using OpenCV
self.fig.add_subplot(5,2,10)
plt.imshow(self.sobel_filter_cv(), cmap='gray')
plt.axis('off')
plt.title('Combined Sobel Filter (OpenCV)')
```

**Figure 1.20.** This code calls upon the self.sobel_filter_cv() function, initiating the creation and eventual display of OpenCV's Sobel Filter.



```python
#sobel filter with x- and y-axis edges, cv
def sobel_filter_cv(self):
    sobel_x = cv2.convertScaleAbs(cv2.Sobel(self.img, cv2.CV_64F,1,0,ksize=3))
    sobel_y = cv2.convertScaleAbs(cv2.Sobel(self.img, cv2.CV_64F,0,1,ksize=3))

    return cv2.addWeighted(sobel_x,0.5,sobel_y,0.5,0)
```

**Figure 1.21.** This code demonstrates the use of OpenCV's built-in Sobel Filter function.

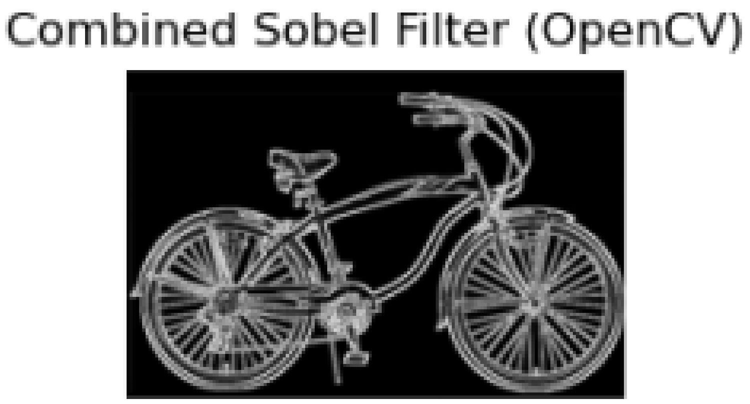


**Figure 1.22.** Displayed image from OpenCV's Sobel Filter.

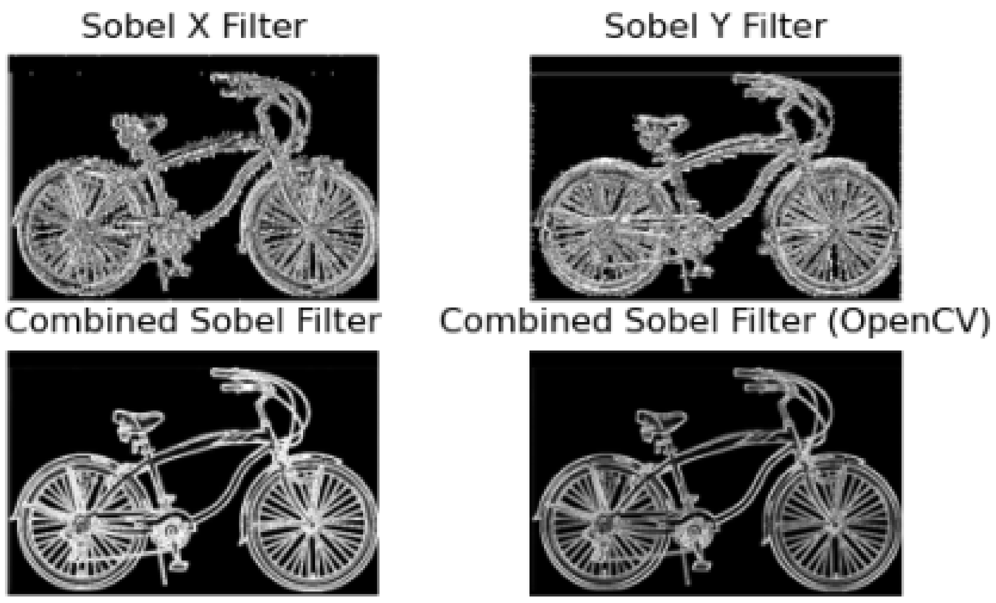
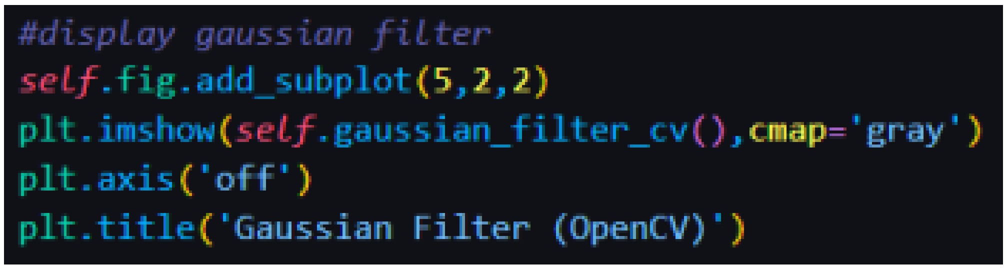


**Figure 1.23.** Displayed image depicting different Sobel Filters (manual and OpenCV) side by side.

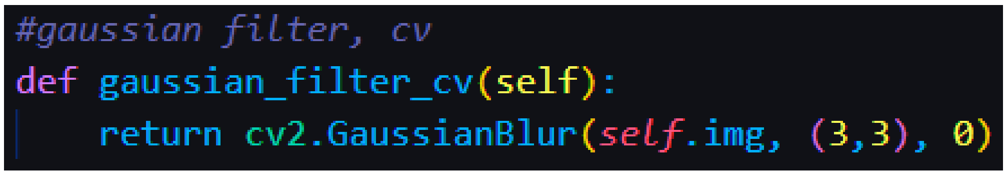
## 1.7 GAUSSIAN BLUR

The final filter to be implemented was the Gaussian Blur, using OpenCV's pre-existing function and a 3x3 dimension for the filtration.



```
#display gaussian filter
self.fig.add_subplot(5,2,2)
plt.imshow(self.gaussian_filter_cv(),cmap='gray')
plt.axis('off')
plt.title('Gaussian Filter (OpenCV)')
```

**Figure 1.24.** This code calls upon the self.gaussian_filter_cv() function, initiating the creation and eventual display of OpenCV's Gaussian Blur Filter.



```
#gaussian filter, cv
def gaussian_filter_cv(self):
    return cv2.GaussianBlur(self.img, (3,3), 0)
```

**Figure 1.25.** This code demonstrates the implementation of OpenCV's built-in Gaussian Blur Filter function.

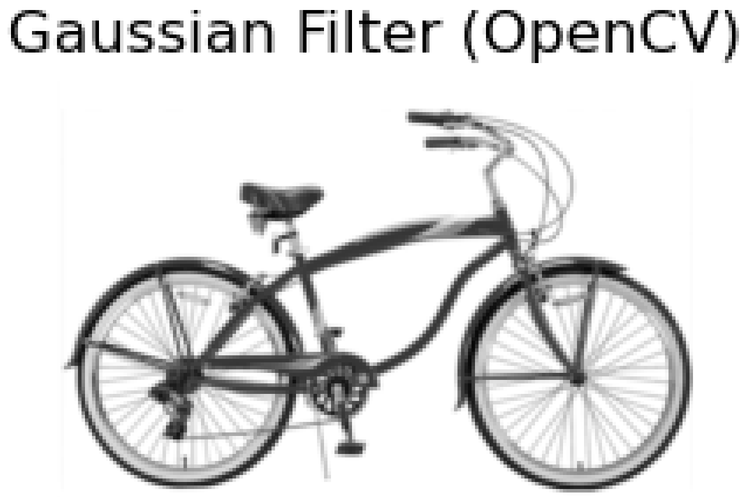


**Figure 1.26.** Displayed image from OpenCV's Gaussian Blur Filter.

## 1.8 FINAL CODE OUTPUT

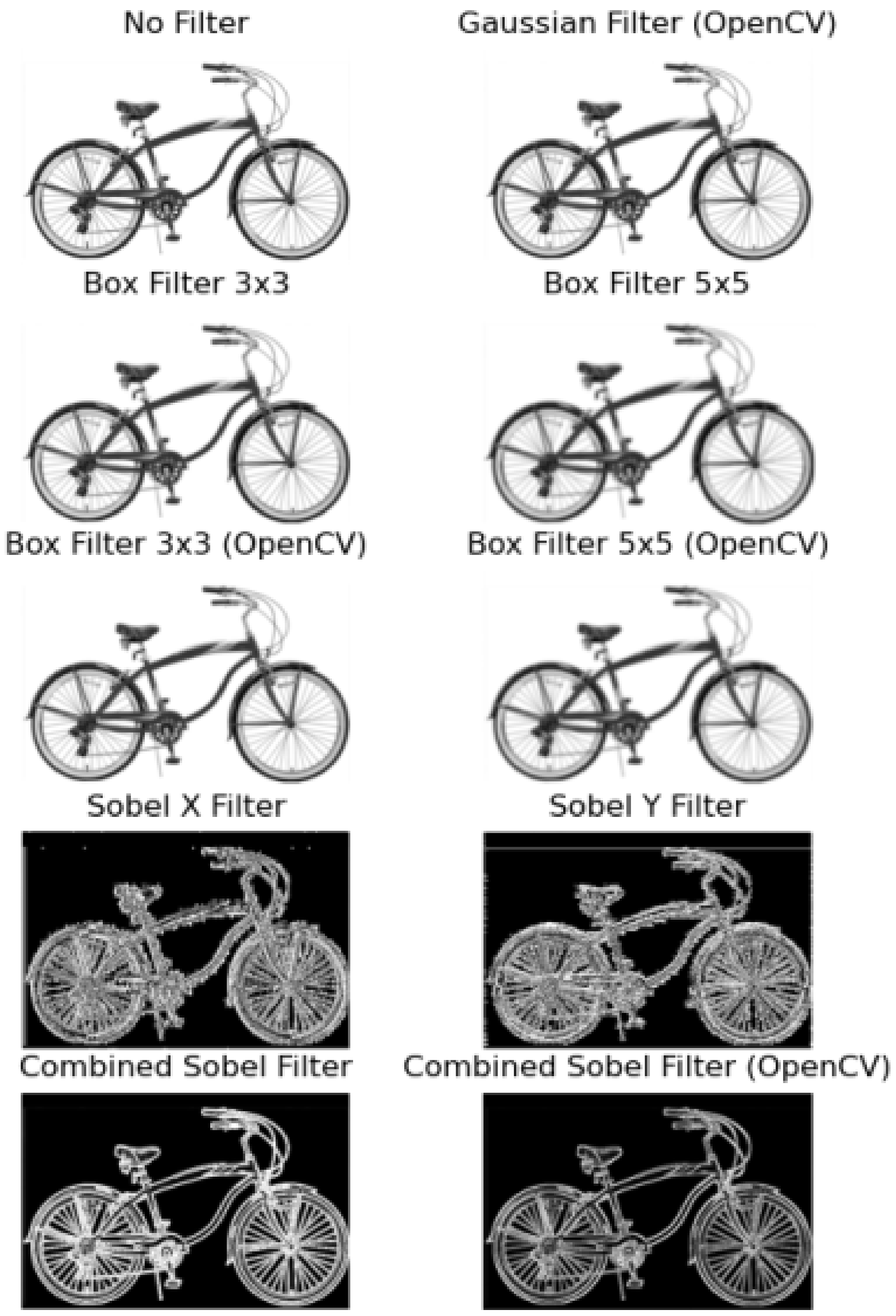


**Figure 1.27.** Displayed image from program execution.

## REFERENCES

Bradski, G. (2000). “The OpenCV Library.” *Dr. Dobb’s Journal of Software Tools*.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). “Array programming with NumPy.” *Nature*, 585(7825), 357–362.

Hunter, J. D. (2007). “Matplotlib: A 2d graphics environment.” *Computing in Science & Engineering*, 9(3), 90–95.