Assignment 2 – Open CV Image Filtering

Prepared by

Dylan Gartin

Course name -Computer Vision (SP 2022)

Course number - CAP4410.01

Date of submission – 2/4/2022

*NOW WITH PASTING! Proper Table of Content Listings For Headings! AND MORE! – Note we will be formatting the reports now with style, Due to major issues that occur even still with trying to implement the template. If you want something added to this template let us know.

Florida Polytechnic University

Table Of Contents

Introduction

Welcome back one and all to Open-CV with Dylan Gartin, now on to assignment two: Image Filtering. The Requirements for this assignment are the following items of which we will go through the Open CV versions first, followed by then my own counterpart:

"This assignment requires you to create and implement the following filters on a given image and discuss your observation for each filter. **Apply all filters with 3x3 and 5x5 image window sizes to all three images**

1. Box Filter (OpenCV)
2. Box Filter (do not use open cv built-in function, write your code to implement it)
3. Sobel Filter towards X-axis edges (do not use open cv built-in function, write your code to implement it)
4. Sobel filter towards Y-axis edges (do not use open cv built-in function, write your code to implement it)
5. Sobel filter with X-axis edges and Y-axis edges (do not use open cv built-in function, write your code to implement it)
6. Sobel Filter with X-axis edges and Y-axis edges (OpenCV)
7. Gaussian Filter (OpenCV)"

With the assignment items covered let's begin.

NOTE: An additional Image was used for testing the files before running final codes, testkirby.png, and is mainly needed to be noted for the challenges sections of the assignment for where it is used:

NOTE: The layout of the images for Box and Gaussian filters will always be in the following order:

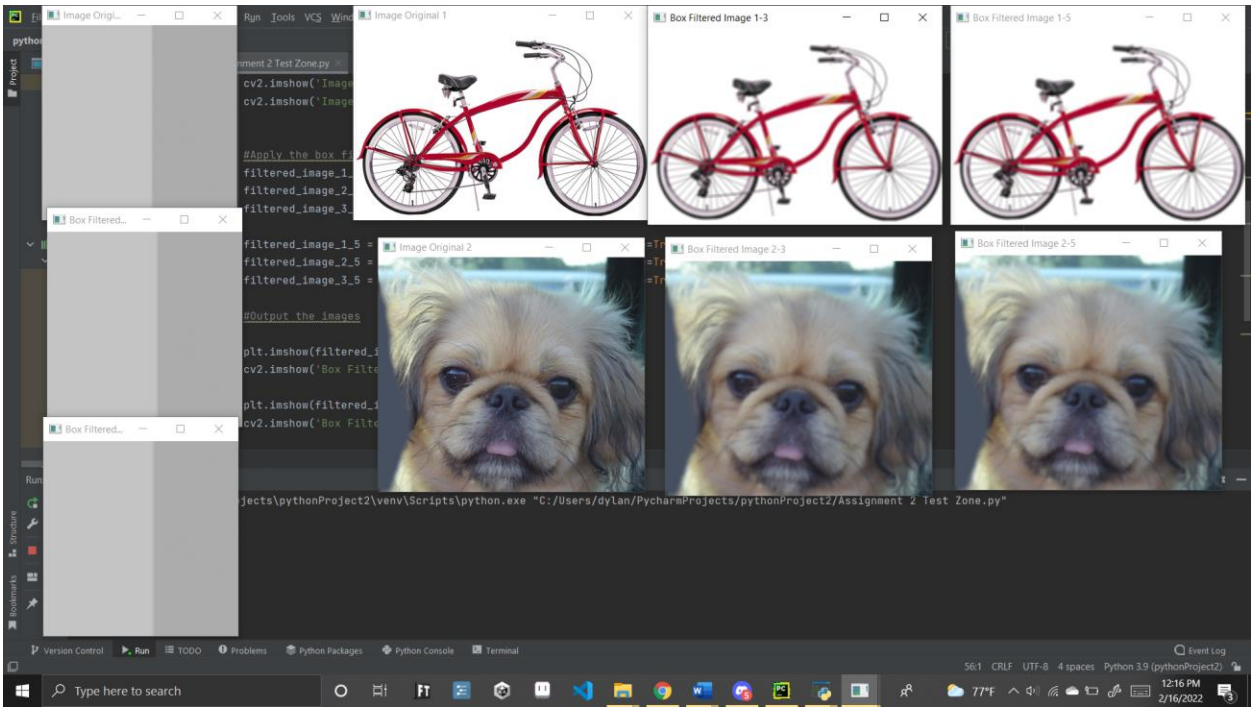


Figure 1: Image Example

Original image, followed by the 3 by 3 box filter, and then the 5 by 5 box filter. For images 1 and 2 this is done horizontally from right to left. For the third image with would be from the top of the image down. Please make sure to take note of this as the grey images in particular are hard to make out the names of.

Box Filter – (Open CV)

With Open CV there are various built-in functions for filters, this allows us to easily implement some filters that are used, an example of this would be the Box Filter, and its built-in function. cv2.boxFilter. We can find a lot of useful information about it from open cv itself https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html For how this works it is done by the following:

filtered_image_NAME = cv2.boxFilter(img_WE_WANT_FILTERED_, -1, (ARRAY_SIZE_X, ARRAY_SIZE_Y), normalize(True or False))

As such by using this function onto our images after loading it in we can very simply output a box filtered image out to the user without having to much trouble. The only important thing to note is to remember to change the array size properly or you might slip up there.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import numpy as np
#%matplotlib inline
```

```
# Read in the images
img1 = mpimg.imread('Test1.bmp')
img2 = mpimg.imread('Test2.bmp')
img3 = mpimg.imread('Test3.png')  # BECAUSE YOU JUST HAD TO MAKE EM ALL NOT
THE SAME TYPE ARGH... okay i'm good now

# Apply the color conversion to the image
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB) # also not technically needed
but it's here

# Display Original images
cv2.imshow('Image Original 1', img1)
cv2.imshow('Image Original 2', img2)
cv2.imshow('Image Original 3', img3)


#Apply the box filters to the images
filtered_image_1_3 = cv2.boxFilter(img1, -1, (3, 3), normalize=True)
filtered_image_2_3 = cv2.boxFilter(img2, -1, (3, 3), normalize=True)
filtered_image_3_3 = cv2.boxFilter(img3, -1, (3, 3), normalize=True)

filtered_image_1_5 = cv2.boxFilter(img1, -1, (5, 5), normalize=True)
filtered_image_2_5 = cv2.boxFilter(img2, -1, (5, 5), normalize=True)
filtered_image_3_5 = cv2.boxFilter(img3, -1, (5, 5), normalize=True)

#Output the images

plt.imshow(filtered_image_1_3, cmap='gray')
cv2.imshow('Box Filtered Image 1-3', filtered_image_1_3)

plt.imshow(filtered_image_1_5, cmap='gray')
cv2.imshow('Box Filtered Image 1-5', filtered_image_1_5)



plt.imshow(filtered_image_2_3, cmap='gray')
cv2.imshow('Box Filtered Image 2-3', filtered_image_2_3)

plt.imshow(filtered_image_2_5, cmap='gray')
cv2.imshow('Box Filtered Image 2-5', filtered_image_2_5)


plt.imshow(filtered_image_3_3, cmap='gray')
cv2.imshow('Box Filtered Image 3-3', filtered_image_3_3)

plt.imshow(filtered_image_3_5, cmap='gray')
cv2.imshow('Box Filtered Image 3-5', filtered_image_3_5)

cv2.waitKey(0)
```

Figure 2: Box Filter Code - Open CV Version

What came as the results for this was the following output:

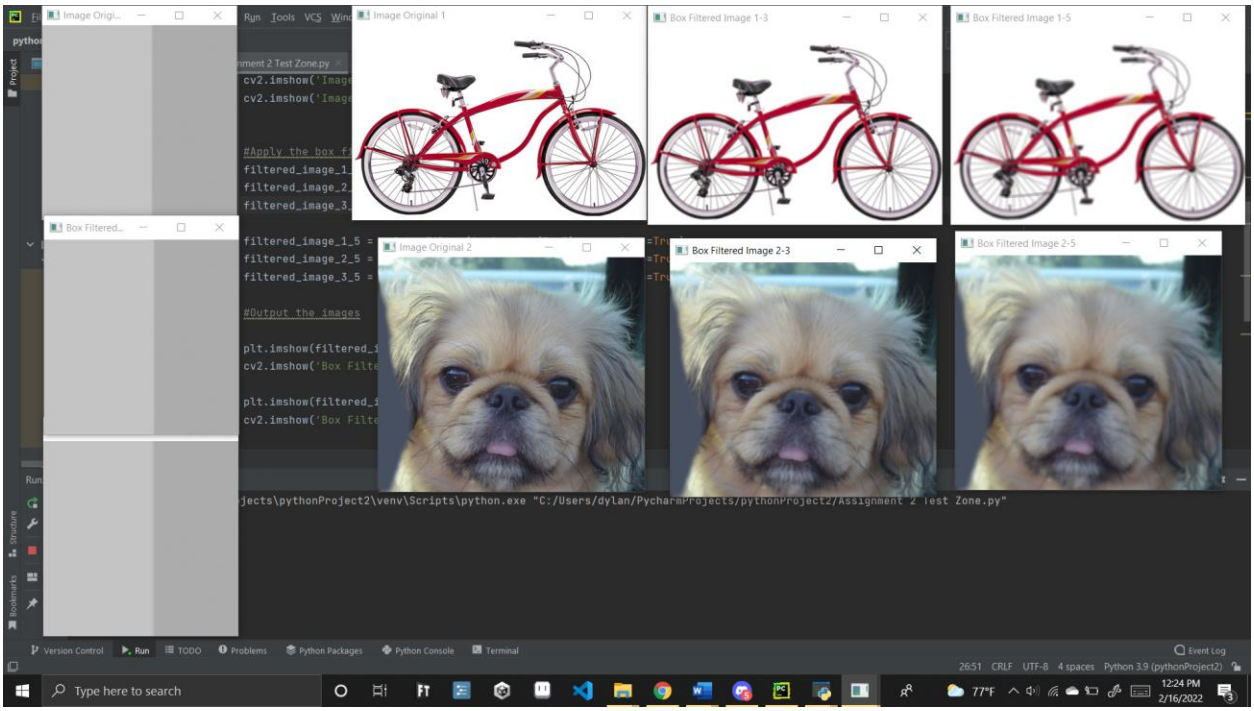


Figure 3: Box Filter Blur with OpenCV

Box Filter – (My Version)

For my Box filter I simply needed to apply the box filter matrix that we learned within class to all the images in a row, and then also check that the code works together when running it and does not have adverse effect on the other filters as well. Overall, there was no major issues in implementing it. The only important things to note is the use of the filters themselves. The box filter is done by simply using an array of 1's/ the number of ones in the array (9 for 3*3, 25 for 5*5). As such use this with the filter 2D command and you should be able output the images, the code for these filters is seen below, as well as the output.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import numpy as np
#%matplotlib inline

# Read in the images
img1 = mpimg.imread('Test1.bmp')
img2 = mpimg.imread('Test2.bmp')
img3 = mpimg.imread('Test3.png')  # BECAUSE YOU JUST HAD TO MAKE EM ALL NOT
THE SAME TYPE ARGH... okay i'm good now

# Apply the color conversion to the image
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
```

```
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB) # also not technically needed
but it's here

# Display Original images
cv2.imshow('Image Original 1', img1)
cv2.imshow('Image Original 2', img2)
cv2.imshow('Image Original 3', img3)

# The Box Array with only 3*3
box3 = np.array([[ 1, 1, 1],
                 [ 1, 1, 1],
                 [ 1, 1, 1]])

# The Box array as a 5*5
box5 = np.array([[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]])

#Apply the box filters to the images
filtered_image_1_3 = cv2.filter2D(img1, -1, box3/9)
filtered_image_2_3 = cv2.filter2D(img3, -1, box3/9)
filtered_image_3_3 = cv2.filter2D(img2, -1, box3/9)

filtered_image_1_5 = cv2.filter2D(img1, -1, box5/25)
filtered_image_2_5 = cv2.filter2D(img2, -1, box5/25)
filtered_image_3_5 = cv2.filter2D(img3, -1, box5/25)

#Output the images

plt.imshow(filtered_image_1_3, cmap='gray')
cv2.imshow('Box Filtered Image 1-3', filtered_image_1_3)

plt.imshow(filtered_image_1_5, cmap='gray')
cv2.imshow('Box Filtered Image 1-5', filtered_image_1_5)



plt.imshow(filtered_image_2_3, cmap='gray')
cv2.imshow('Box Filtered Image 2-3', filtered_image_2_3)

plt.imshow(filtered_image_2_5, cmap='gray')
cv2.imshow('Box Filtered Image 2-5', filtered_image_2_5)


plt.imshow(filtered_image_3_3, cmap='gray')
cv2.imshow('Box Filtered Image 3-3', filtered_image_3_3)

plt.imshow(filtered_image_3_5, cmap='gray')
cv2.imshow('Box Filtered Image 3-5', filtered_image_3_5)

cv2.waitKey(0)
```

Figure 4: Box Filter Code – My Version

And this resulted in the following image:

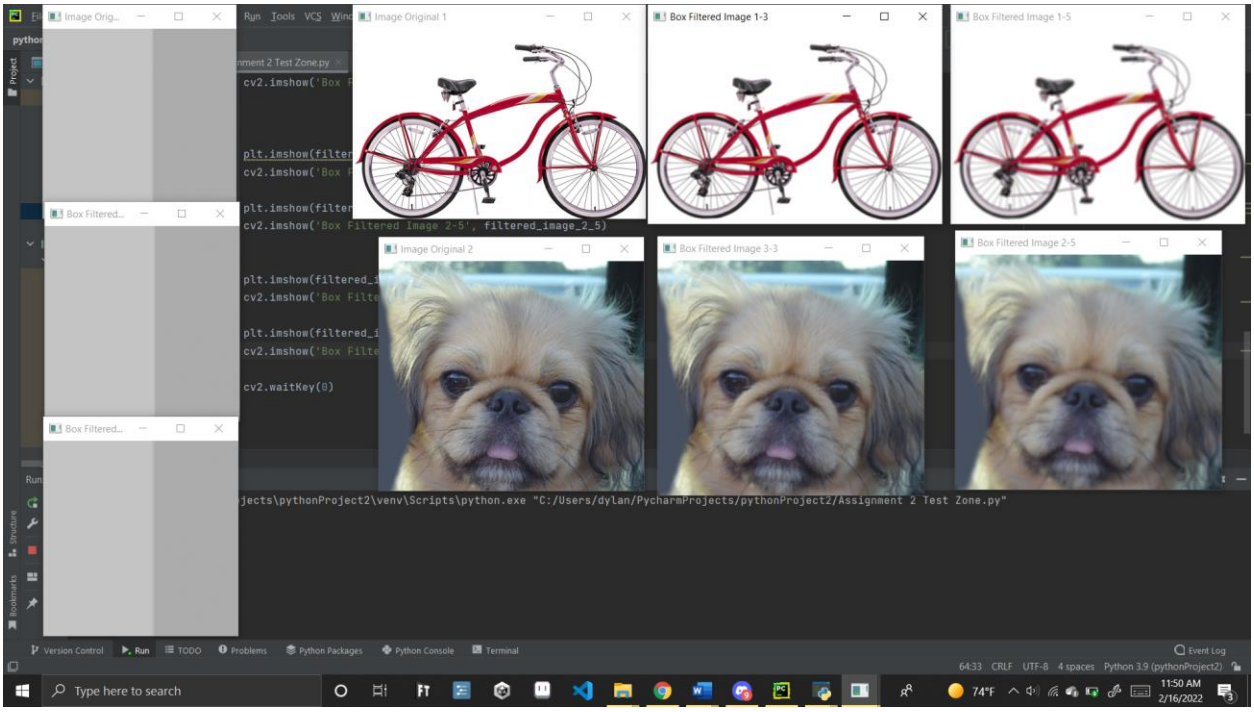


Figure 5: My Box Filter Blur Effect - My Version

Box Filter – (Comparisons)

Honestly between the two filters, I see no difference, I know my eye sight isn't the best but if I had to guess what was going on under the hood of the open cv box filter it would be the same as my code. as such I imagine the filter being built in is more so to save the programmer time then anything as it allows you to not have to build the arrays that are needed for the various sizes. Other than that, the main difference you can see is by the various sizes, of which the bigger the array the harder the blur effect which makes sense. Overall, though were happy with the results!

Sobel Filter (Open CV)

The Sobel filter is a bit more unique, mainly due to the fact it can be done with axis. Simply put for the Sobel filter we have the x axis and y axis that it can be applied to as well as both of them together. These build their own arrays that are built a bit differently than the box filter.

For our purposes, we will assume the Sobel Filters are built like the following items that were given by the professor:

(3*3) – Sobel filter array

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

(5*5) - Sobel filter array

+2	+2	+4	+2	+2
+1	+1	+2	+1	+1
0	0	0	0	0
-1	-1	-2	-1	-1
-2	-2	-4	-2	-2

Gx

+2	+1	0	-1	-2
+2	+1	0	-1	-2
+4	+2	0	-2	-4
+2	+1	0	-1	-2
+2	+1	0	-1	-2

Gy

Table 1: Sobel Array Templates Used

This information actually doesn't come into play though for the OpenCV code however which makes it likely we might get some different results due to such, but for that we'll have to wait and see.

While the assignment requested only the "x and y" version of the filter with open cv, we decided it would be better to do all the filters in order to show the effect, and quite frankly because it's actually easier to show them all as we have found a helpful resource, https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html . Using the info here as a base we were able to make a code that outputs all three of the images, provided you change the image that is passed in initially based on the note.

*NOTE FOR THIS CODE: Replace the "img = cv2.imread('testkirby.png')" with the following lines:

- img = cv2.imread('Test1.bmp')

- img = cv2.imread('Test2.bmp')
- img = cv2.imread('Test3.png')

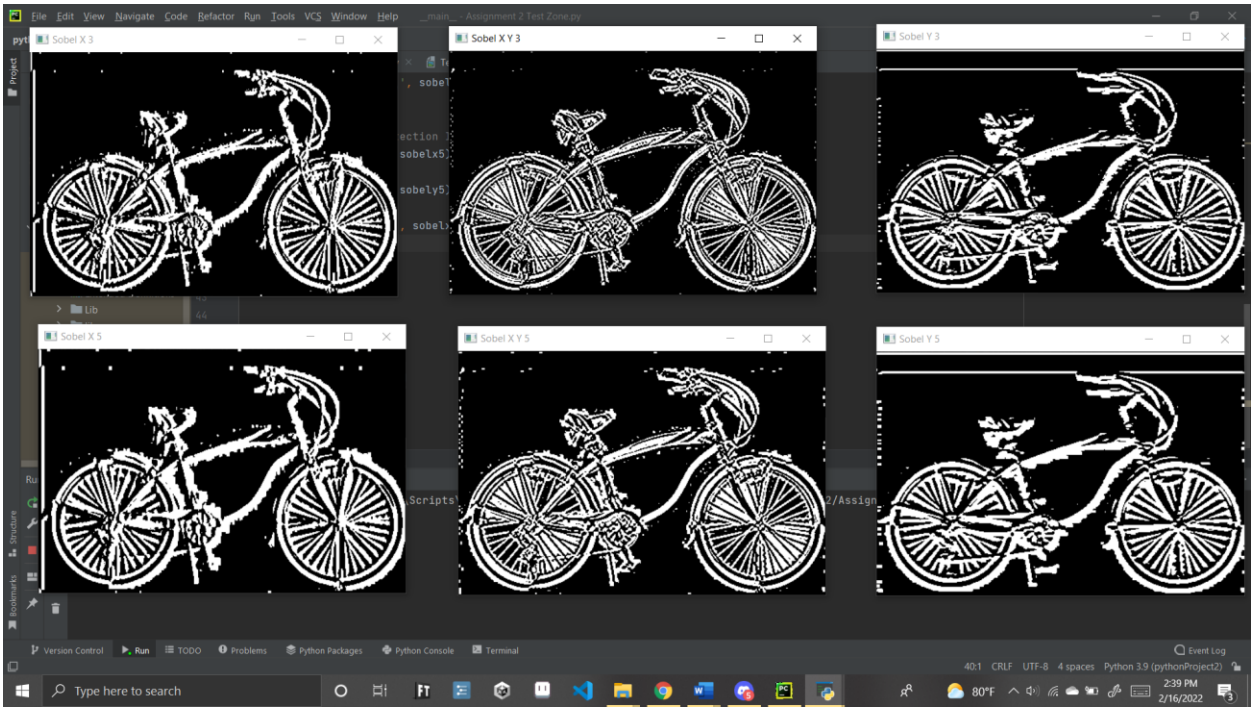
Each time you run the code to get the various images, otherwise you'll get a Kirby, poyo

```
import cv2

# Read the original image
# Read the original image You replace the below line with following
#img = cv2.imread('Test1.bmp')
#img = cv2.imread('Test2.bmp')
#img = cv2.imread('Test3.png')

img = cv2.imread('testkirby.png')

# Display original image we hide these in most of the photos
cv2.imshow('Original', img)


# Convert to graycsale to be able to detect edges
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# doing a Gauusian Blur, which will be done at the end of the document the
image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3, 3), 0)

# Sobel Edge Detection, fists for 3's then 5's
sobelx3 = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=3)  #
Sobel Edge Detection on the X axis 3
sobely3 = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=3)  #
Sobel Edge Detection on the Y axis 3
sobelxy3 = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=3)  #
Combined X and Y Sobel Edge Detection 3

sobelx5 = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5)  #
Sobel Edge Detection on the X axis 5
sobely5 = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5)  #
Sobel Edge Detection on the Y axis 5
sobelxy5 = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5)  #
Combined X and Y Sobel Edge Detection 5

# Display Sobel Edge Detection Images of 3 power array
cv2.imshow('Sobel X 3', sobelx3)

cv2.imshow('Sobel Y 3', sobely3)

cv2.imshow('Sobel X Y 3 ', sobelxy3)


# Display Sobel Edge Detection Images that are of 5 power array
cv2.imshow('Sobel X 5', sobelx5)

cv2.imshow('Sobel Y 5', sobely5)

cv2.imshow('Sobel X Y 5', sobelxy5)
```

```
cv2.waitKey(0) # 0 holds it in place when needed
cv2.destroyAllWindows()
```

Figure 6: Sobel Filter Code for “x”, “y,” and “x and y” Using Open CV

And the images that resulted from this are seen here:



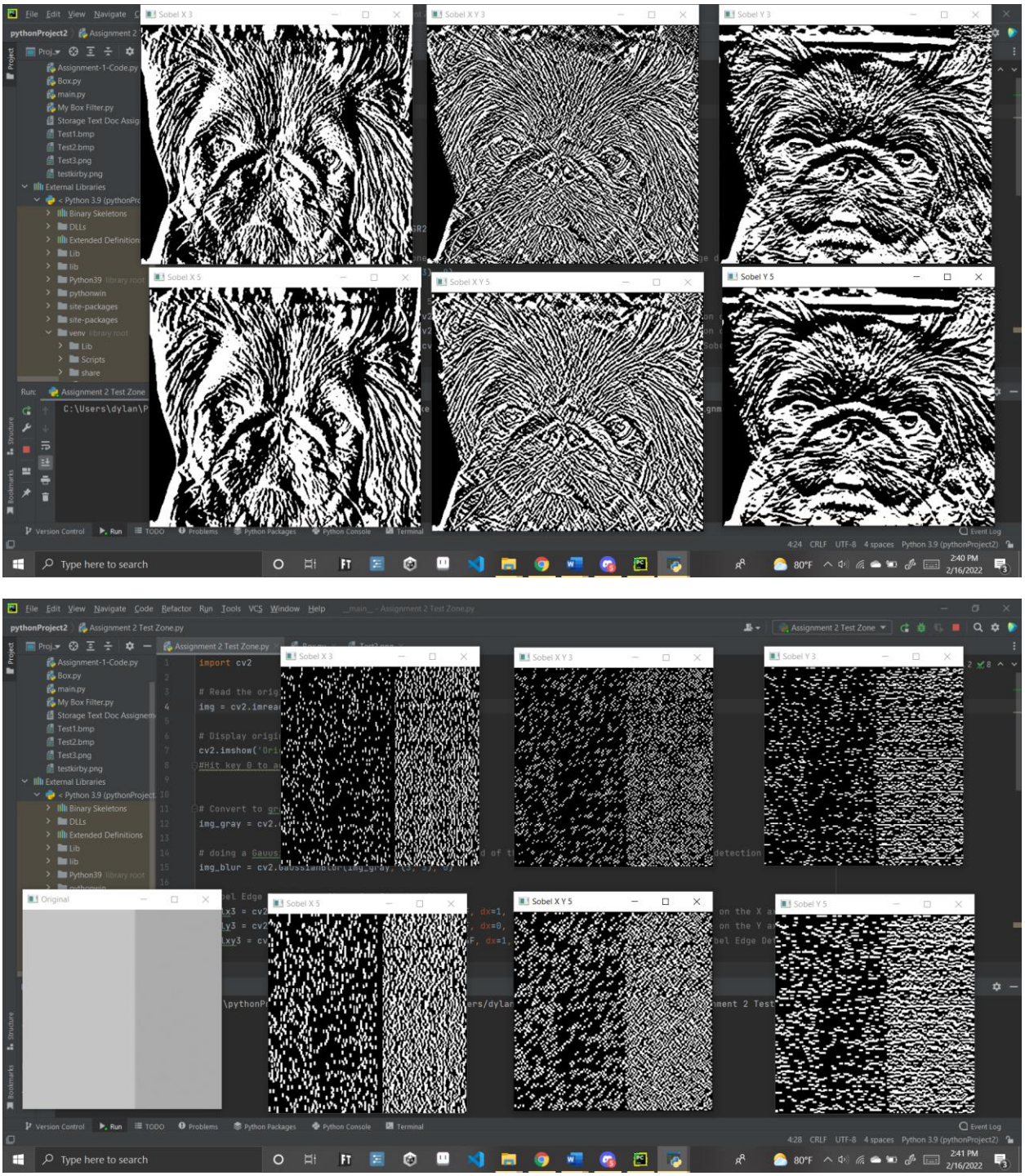


Figure 7: Sobel Filter's Image Outputs with Open CV

Sobel Filter (My Versions)

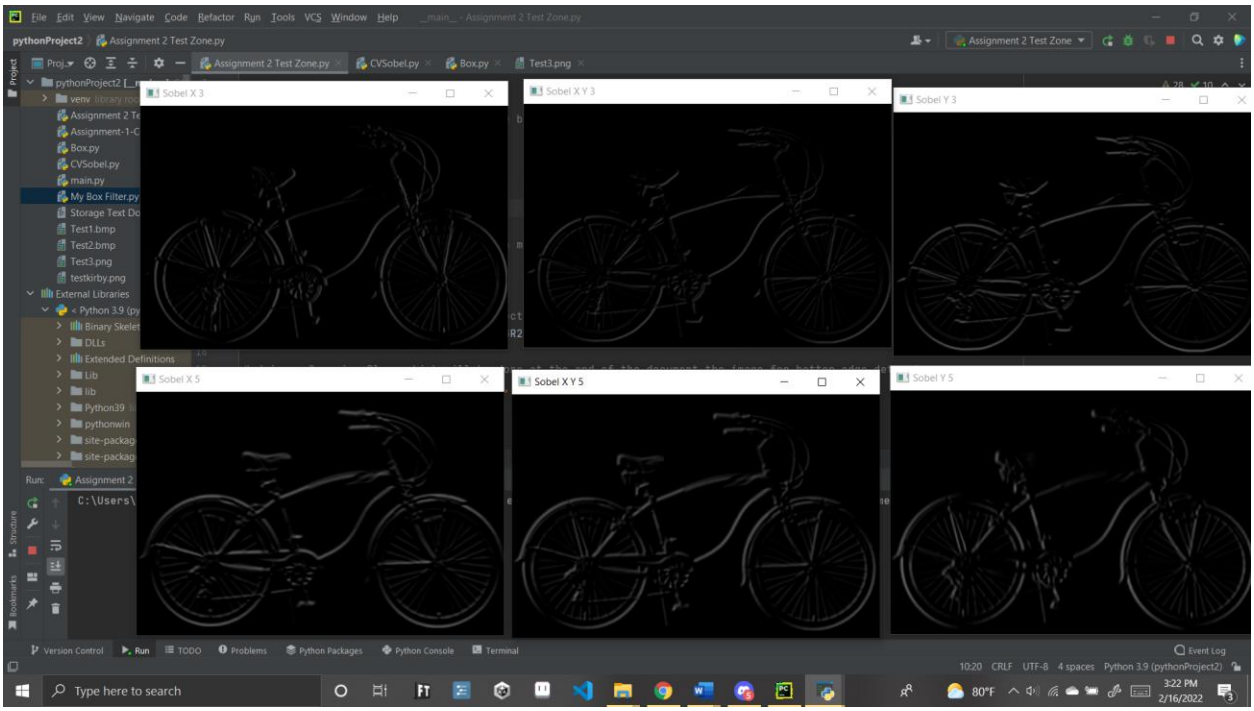
Now for my version of the code again, this is where the previous array figures will come into play. Simply put rather then using a box filter you can just use the actual arrays that correspond to the image. For the only X and Y one simply replacing the Box filter with the corresponding array works, as for the X and Y one this requires you to add the two together. After these are obtained simply make sure to add and divide by the total sum of absolute values just like with the box filter. Following these steps will result in code that looks like such:

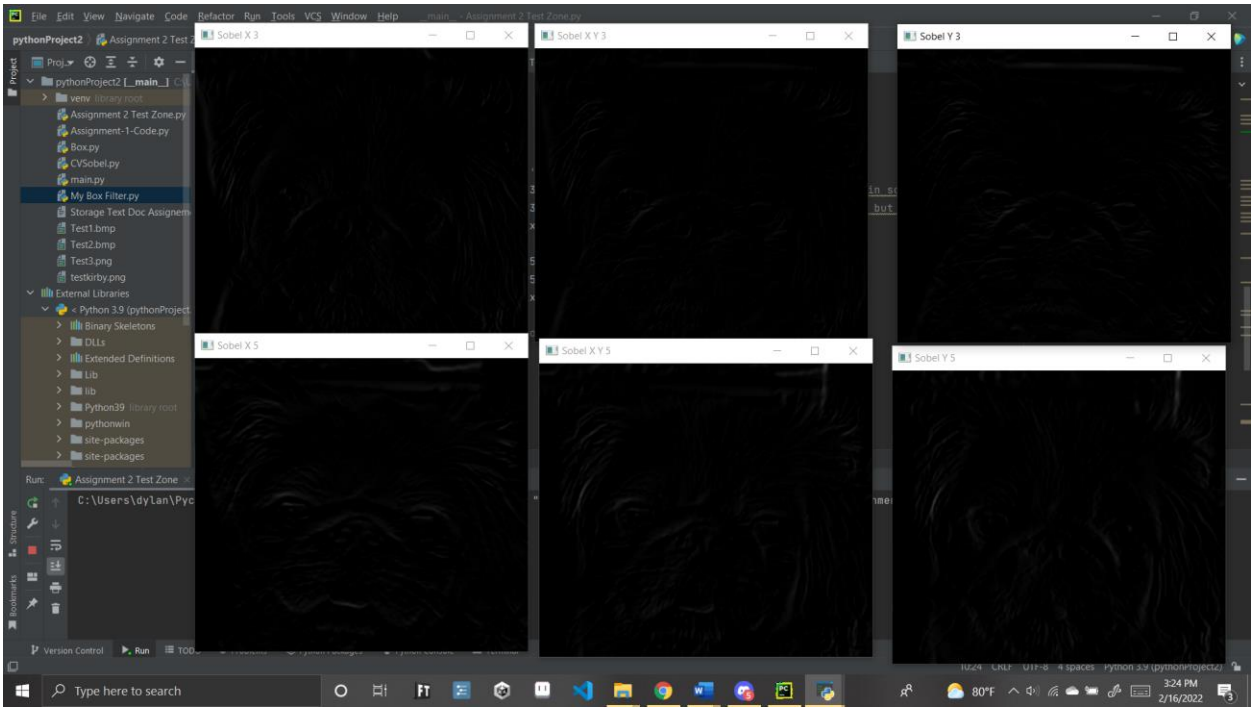
```
import cv2
import numpy as np # we need this for the array again

# Read the original image
# Read the original image You replace the below line with following
#img = cv2.imread('Test1.bmp')
#img = cv2.imread('Test2.bmp')
#img = cv2.imread('Test3.png')

img = cv2.imread('Test3.png')

# Display original image we hide these in most of the photos
cv2.imshow('Original', img)


# Convert to graycsale to be able to detect edges
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# doing a Gauusian Blur, which will be done at the end of the document the
image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3, 3), 0)

sobel_x3 = np.array([[ -1, 0, 1],
                     [ -2, 0, 2],
                     [ -1, 0, 1]])

sobel_y3 = np.array([[ -1, -2, -1],
                     [ 0, 0, 0],
                     [ 1, 2, 1]])

sobel_xy3= np.array([[ 0, -2, 0],
                     [ -2, 0, 2],
                     [ 0, 2, 0]])

# The Box array as a 5*5
sobel_x5  = np.array([  [2, 2, 4, 2, 2],
                        [1, 1, 2, 1, 1],
                        [0, 0, 0, 0, 0],
                        [-1, -1, -2, -1, -1],
                        [-2, -2, -4, -2, -2]])

sobel_y5  = np.array([  [2, 1, 0, -1, -2],
                        [2, 1, 0, -1, -2],
                        [4, 2, 0, -2, -4],
```

```
                          [2, 1, 0, -1, -2],
                          [2, 1, 0, -1, -2]])

sobel_xy5  = np.array([ [4, 3, 4, -1, 0],
                          [3, 2, 2, 0, -1],
                          [4, 2, 0, -2, -4],
                          [1, 0, -2, -2, -3],
                          [0, 1, -4, -3, -4]])

# Sobel Edge Detection, fists for 3's then 5's
sobelx3 = cv2.filter2D(img_blur, -1, sobel_x3/8) # Note: the divide by 8 does
change the effect in some intresting ways, we honestly perfer it removed
sobely3 = cv2.filter2D(img_blur, -1, sobel_y3/8) # without it there a bit
more noise in the data but overall brighter and more akin to original
sobelxy3 = cv2.filter2D(img_blur, -1, sobel_xy3/8)

sobelx5 = cv2.filter2D(img_blur, -1, sobel_x5/36) # there a total of 36 data
points here
sobely5 = cv2.filter2D(img_blur, -1, sobel_y5/36)
sobelxy5 = cv2.filter2D(img_blur, -1, sobel_xy5/52) # meanwhile 24+28

# Display Sobel Edge Detection Images of 3 power array
cv2.imshow('Sobel X 3', sobelx3)

cv2.imshow('Sobel Y 3', sobely3)

cv2.imshow('Sobel X Y 3 ', sobelxy3)


# Display Sobel Edge Detection Images that are of 5 power array
cv2.imshow('Sobel X 5', sobelx5)

cv2.imshow('Sobel Y 5', sobely5)

cv2.imshow('Sobel X Y 5', sobelxy5)

cv2.waitKey(0) # 0 holds it in place when needed
cv2.destroyAllWindows()
```

Figure 8: Sobel Filter Code for “x”, “y,” and “x and y” Using My Version

The images we got as outputs however don’t seem to line up with what we found using the built-in filter. As seen below:


Sobel X 3
Sobel X Y 3
Sobel Y 3
Sobel X 5
Sobel X Y 5
Sobel Y 5


Sobel X 3
Sobel X Y 3
Sobel Y 3
Sobel X 5
Sobel X Y 5
Sobel Y 5

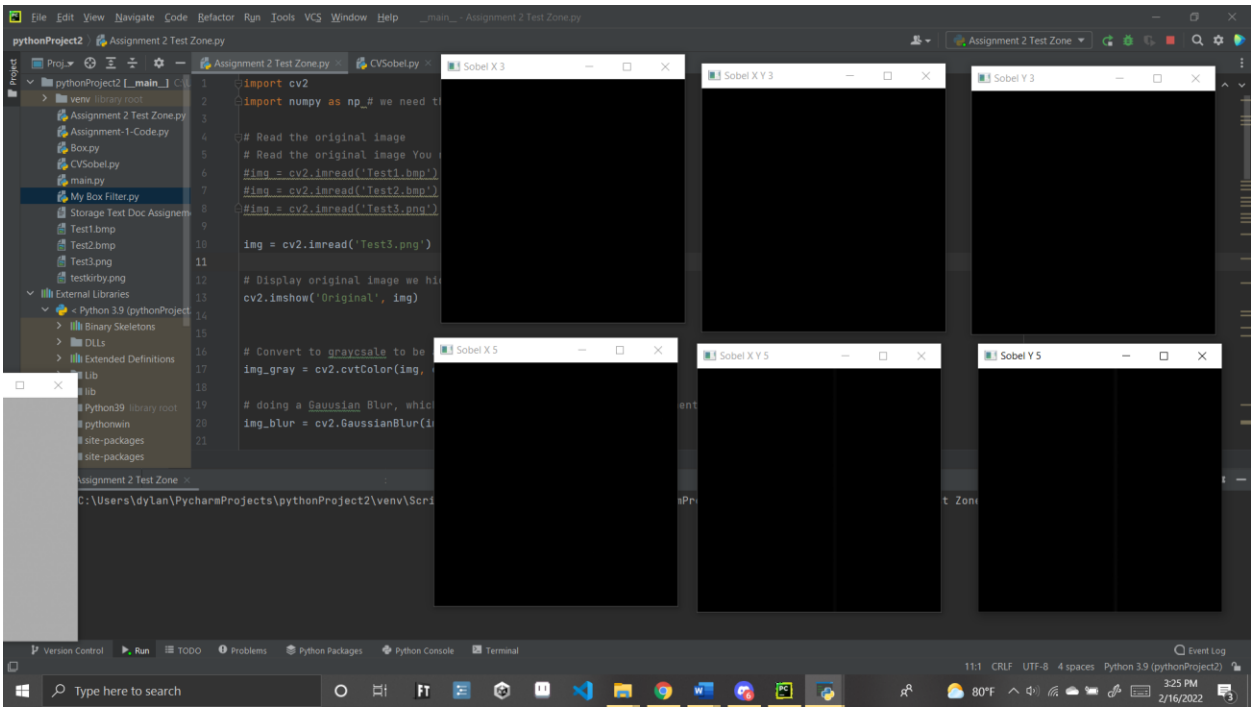
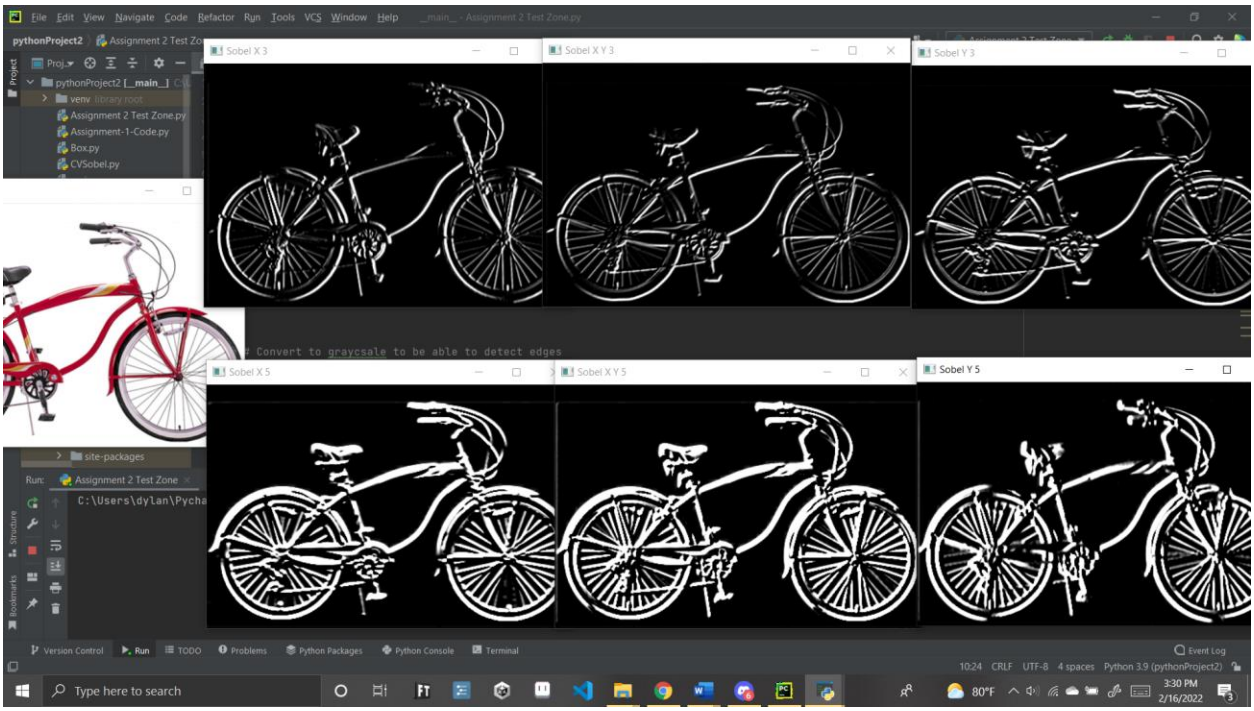


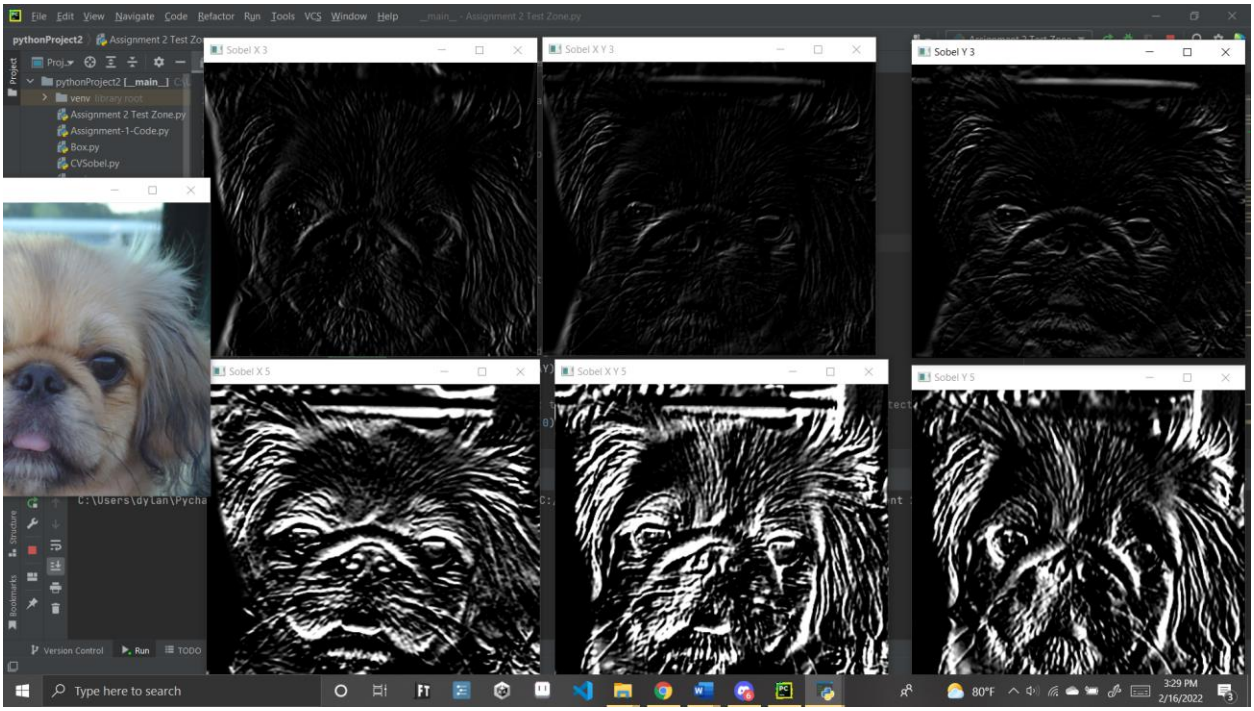
Figure 9: Sobel Filter's Image Outputs with My Version

So why is this? It's Likely a combination of two things of which we will go over in the comparison.

Sobel Filter (Comparison)

So, what's the scoop on why the Sobel filters came out so differently, well first and foremost is likely due to the divisions we see in the first code, by doing this it reduces the power of the filter to be more of an average, as such it caused the images to look darker. By removing this we get much better resulting images which are seen below:


Sobel X 3
Sobel X Y 3
Sobel Y 3
Sobel X 5
Sobel X Y 5
Sobel Y 5
Convert to grayscale to be able to detect edges
Assignment 2 Test Zone
Type here to search


Sobel X 3
Sobel X Y 3
Sobel Y 3
Sobel X 5
Sobel X Y 5
Sobel Y 5
Assignment 2 Test Zone
Type here to search

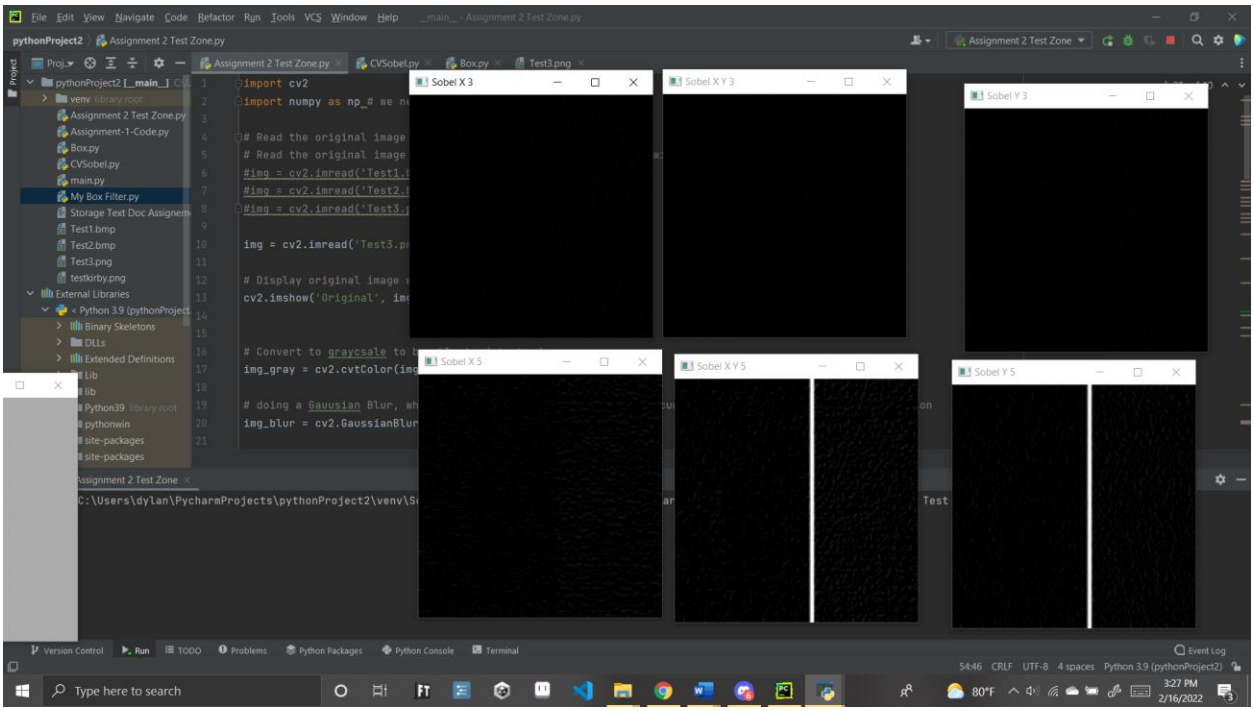


Figure 10: Sobel Filters Without Divisions Results

After that the other thing is likely under the hood calculations that we are simply unable to account for. Even if the values are slightly different for the arrays we'll have different filter results, so the only way to get a matching result would be to either pop open the hood and find open CV version of the code, or a very lucky guess. Overall, however the results are similar and there could even be merits to the other version of this code. Even just seeing the final figure of 10: we can see that my image simply has the dividing line, rather then a lot of the noise from the built-in function. Overall, I would say this version of code performs antiquatedly and in some ways better then OpenCV's built in plugin. The Code with divisions will also be shown below:

```
import cv2
import numpy as np # we need this for the array again

# Read the original image
# Read the original image You replace the below line with following
#img = cv2.imread('Test1.bmp')
#img = cv2.imread('Test2.bmp')
#img = cv2.imread('Test3.png')

img = cv2.imread('Test1.bmp')

# Display original image we hide these in most of the photos
cv2.imshow('Original', img)


# Convert to graycsale to be able to detect edges
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
# doing a Gauusian Blur, which will be done at the end of the document the
image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3, 3), 0)

sobel_x3 = np.array([[ -1, 0, 1],
                     [ -2, 0, 2],
                     [ -1, 0, 1]])

sobel_y3 = np.array([[ -1, -2, -1],
                     [ 0, 0, 0],
                     [ 1, 2, 1]])

sobel_xy3= np.array([[ 0, -2, 0],
                     [ -2, 0, 2],
                     [ 0, 2, 0]])

# The Box array as a 5*5
sobel_x5  = np.array([  [2, 2, 4, 2, 2],
                        [1, 1, 2, 1, 1],
                        [0, 0, 0, 0, 0],
                        [-1, -1, -2, -1, -1],
                        [-2, -2, -4, -2, -2]])

sobel_y5  = np.array([  [2, 1, 0, -1, -2],
                        [2, 1, 0, -1, -2],
                        [4, 2, 0, -2, -4],
                        [2, 1, 0, -1, -2],
                        [2, 1, 0, -1, -2]])

sobel_xy5  = np.array([ [4, 3, 4, -1, 0],
                        [3, 2, 2, 0, -1],
                        [4, 2, 0, -2, -4],
                        [1, 0, -2, -2, -3],
                        [0, 1, -4, -3, -4]])

# Sobel Edge Detection, fists for 3's then 5's
sobelx3 = cv2.filter2D(img_blur, -1, sobel_x3) # Note: the divide by 8 does
change the effect in some intresting ways, we honestly perfer it removed
sobely3 = cv2.filter2D(img_blur, -1, sobel_y3) # without it there a bit more
noise in the data but overall brighter and more akin to original
sobelxy3 = cv2.filter2D(img_blur, -1, sobel_xy3)

sobelx5 = cv2.filter2D(img_blur, -1, sobel_x5) # there a total of 36 data
points here
sobely5 = cv2.filter2D(img_blur, -1, sobel_y5)
sobelxy5 = cv2.filter2D(img_blur, -1, sobel_xy5) # meanwhile 24+28

# Display Sobel Edge Detection Images of 3 power array
cv2.imshow('Sobel X 3', sobelx3)

cv2.imshow('Sobel Y 3', sobely3)

cv2.imshow('Sobel X Y 3 ', sobelxy3)


# Display Sobel Edge Detection Images that are of 5 power array
```

```
cv2.imshow('Sobel X 5', sobelx5)

cv2.imshow('Sobel Y 5', sobely5)

cv2.imshow('Sobel X Y 5', sobelxy5)

cv2.waitKey(0) # 0 holds it in place when needed
cv2.destroyAllWindows()
```

Figure 11: Sobel Filters Without Divisions Code

Gaussian Filter (Open CV)

Another requested filter was the Gaussian filter, which at first is hard to really tell the effects of but it has a similar blurring effect as the box filters it seems, all be it a little weaker all round in my opinion. For the code fore this one though we simply needed to exchange the box filter code with the gaussian blur code in order to make this run. Overall, rather simple small change that causes a small change in the output.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import numpy as np
#%matplotlib inline

# Read in the images
img1 = mpimg.imread('Test1.bmp')
img2 = mpimg.imread('Test2.bmp')
img3 = mpimg.imread('Test3.png')  # BECAUSE YOU JUST HAD TO MAKE EM ALL NOT
THE SAME TYPE ARGH... okay i'm good now

# Apply the color conversion to the image
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB) # also not technically needed
but it's here

# Display Original images
cv2.imshow('Image Original 1', img1)
cv2.imshow('Image Original 2', img2)
cv2.imshow('Image Original 3', img3)


#Apply the box filters to the images
filtered_image_1_3 = cv2.GaussianBlur(img1,(3,3),0)
filtered_image_2_3 = cv2.GaussianBlur(img2,(3,3),0)
filtered_image_3_3 = cv2.GaussianBlur(img3,(3,3),0)

filtered_image_1_5 = cv2.GaussianBlur(img1,(5,5),0)
filtered_image_2_5 = cv2.GaussianBlur(img2,(5,5),0)
filtered_image_3_5 = cv2.GaussianBlur(img3,(5,5),0)

#Output the images
```

```
plt.imshow(filtered_image_1_3, cmap='gray')
cv2.imshow('Gau Filtered Image 1-3', filtered_image_1_3)

plt.imshow(filtered_image_1_5, cmap='gray')
cv2.imshow('Gau Filtered Image 1-5', filtered_image_1_5)


plt.imshow(filtered_image_2_3, cmap='gray')
cv2.imshow('Gau Filtered Image 2-3', filtered_image_2_3)

plt.imshow(filtered_image_2_5, cmap='gray')
cv2.imshow('Gau Filtered Image 2-5', filtered_image_2_5)


plt.imshow(filtered_image_3_3, cmap='gray')
cv2.imshow('Gau Filtered Image 3-3', filtered_image_3_3)

plt.imshow(filtered_image_3_5, cmap='gray')
cv2.imshow('Gau Filtered Image 3-5', filtered_image_3_5)

cv2.waitKey(0)
```

Figure 12: Gaussian Filter With Open CV Code

And this leads into the following filter results:

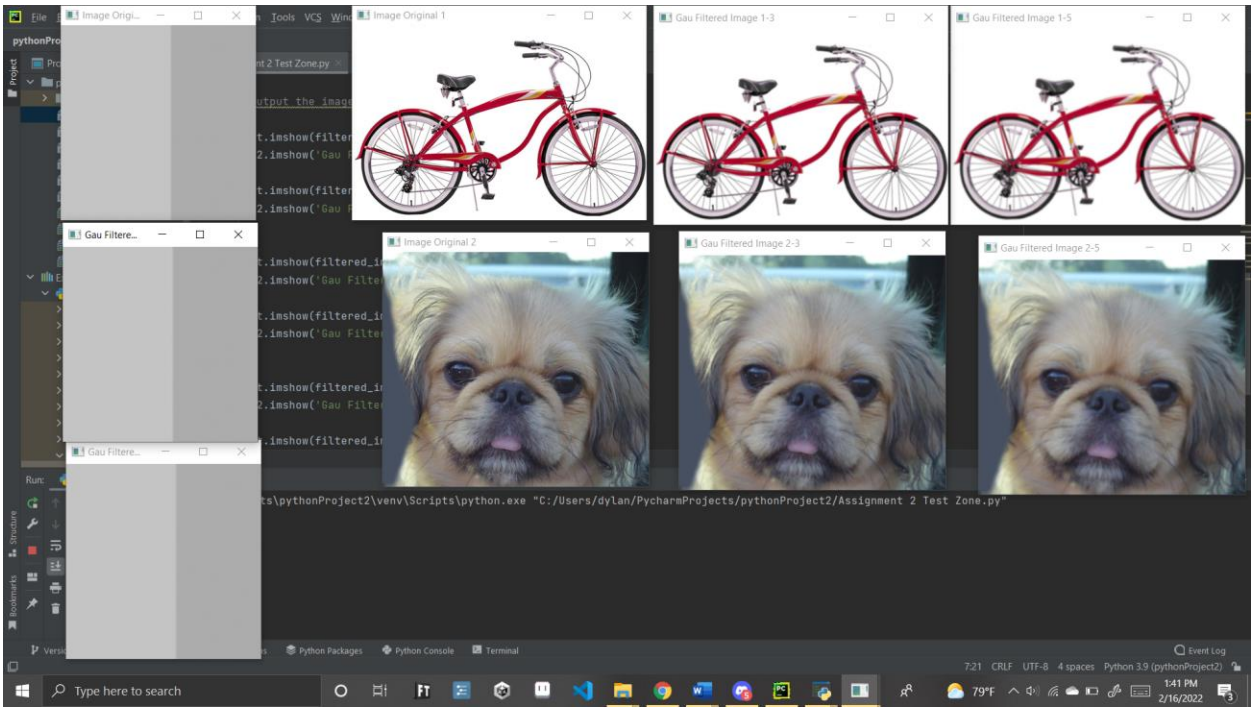


Figure 13: Gaussian Filter With Open CV Results

Challenges

There were not as much issues with this assignment as the first one, mainly due to there being better information on the topic the only noteworthy issue is the that OpenCV and the image output system have different type of color view, meaning that we had to find a line of code to change the color types so they would work properly. Luckily, we found this via https://stackoverflow.com/questions/39316447/opencv-giving-wrong-color-to-colored-images-on-loading this line of code which fixed the issue; img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) as such this was only a temporary issue.

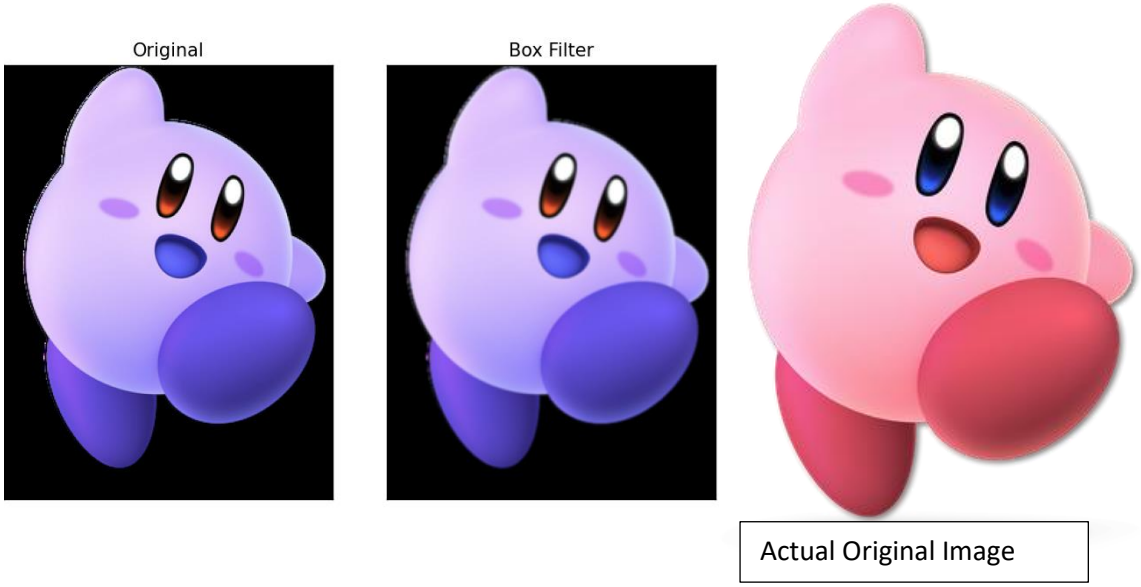


Figure 14: Image Issue – Color Channing Magic! (AKA: Goth Kirby)

Besides that, the only other major issue was just the template document for the report itself was not set up properly, as such we made our own template to do a more formal and professional looking paper.

Conclusion

In conclusion, Basic filters are not that hard to implement actually, and while results in filters may not be the exact same output useful data can be obtained even from versions of the filters that don't exactly match the guidelines. Overall, it's important to test and adjust the settings of your filters with various outputs to see if the results will be readable for your purposes. By doing such we should be able to get machines to detect objects easier.

References -

The following Sources were used within this document to help with making the code.

bholagabbarbholagabbar 3, et al. “OpenCV Giving Wrong Color to Colored Images on Loading.” *Stack Overflow*, 1 Aug. 1964, https://stackoverflow.com/questions/39316447/opencv-giving-wrong-color-to-colored-images-on-loading.

bholagabbarbholagabbar 3, et al. “OpenCV Giving Wrong Color to Colored Images on Loading.” *Stack Overflow*, 1 Aug. 1964, https://stackoverflow.com/questions/39316447/opencv-giving-wrong-color-to-colored-images-on-loading.

“Smoothing Images¶.” *OpenCV*, https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html.

“Sobel Derivatives.” *OpenCV*, https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html.