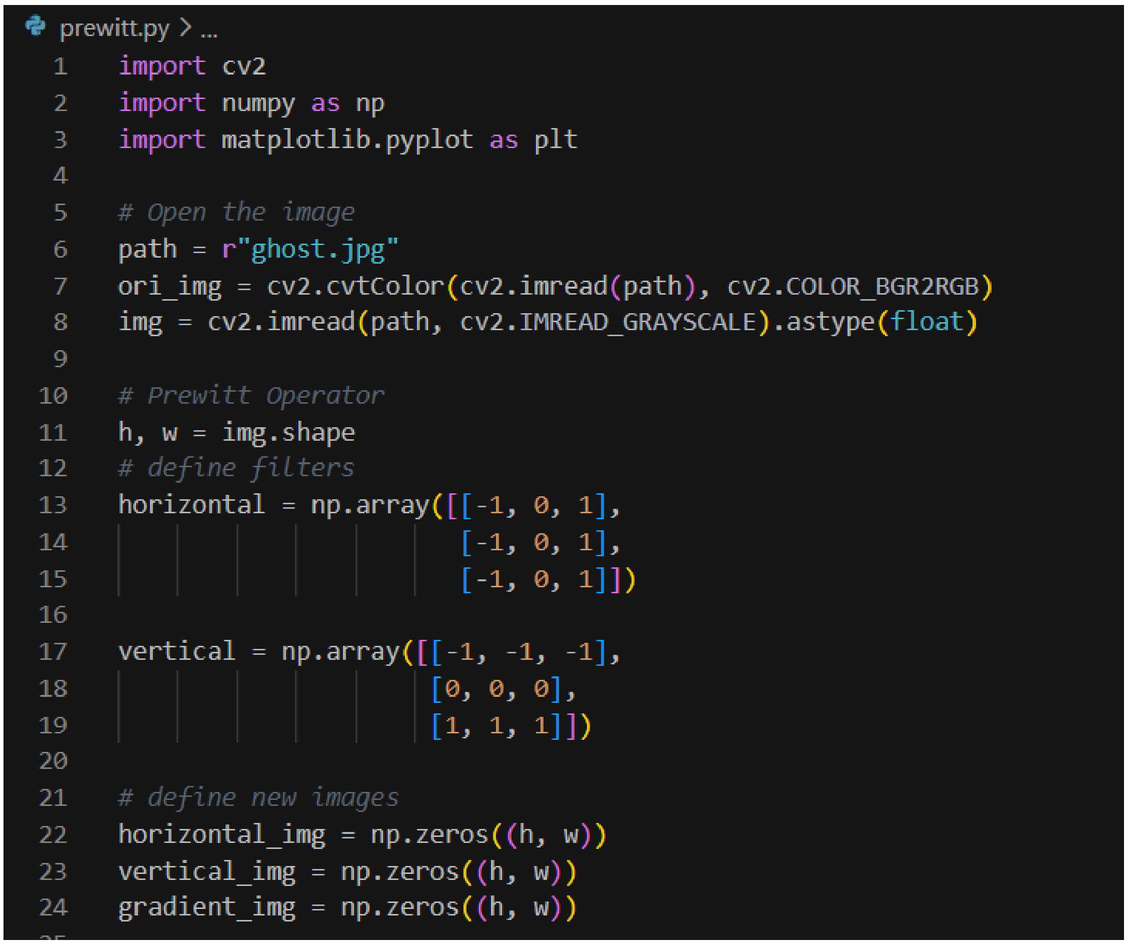
Assignment7

- Select two images and apply the Prewitt and Sobel operators for edge detection
- Select two images and apply Robert operator to the images
- Explain clearly about the horizontal derivative can well capture vertical edges and the vertical derivative can be used for detecting horizontal edges

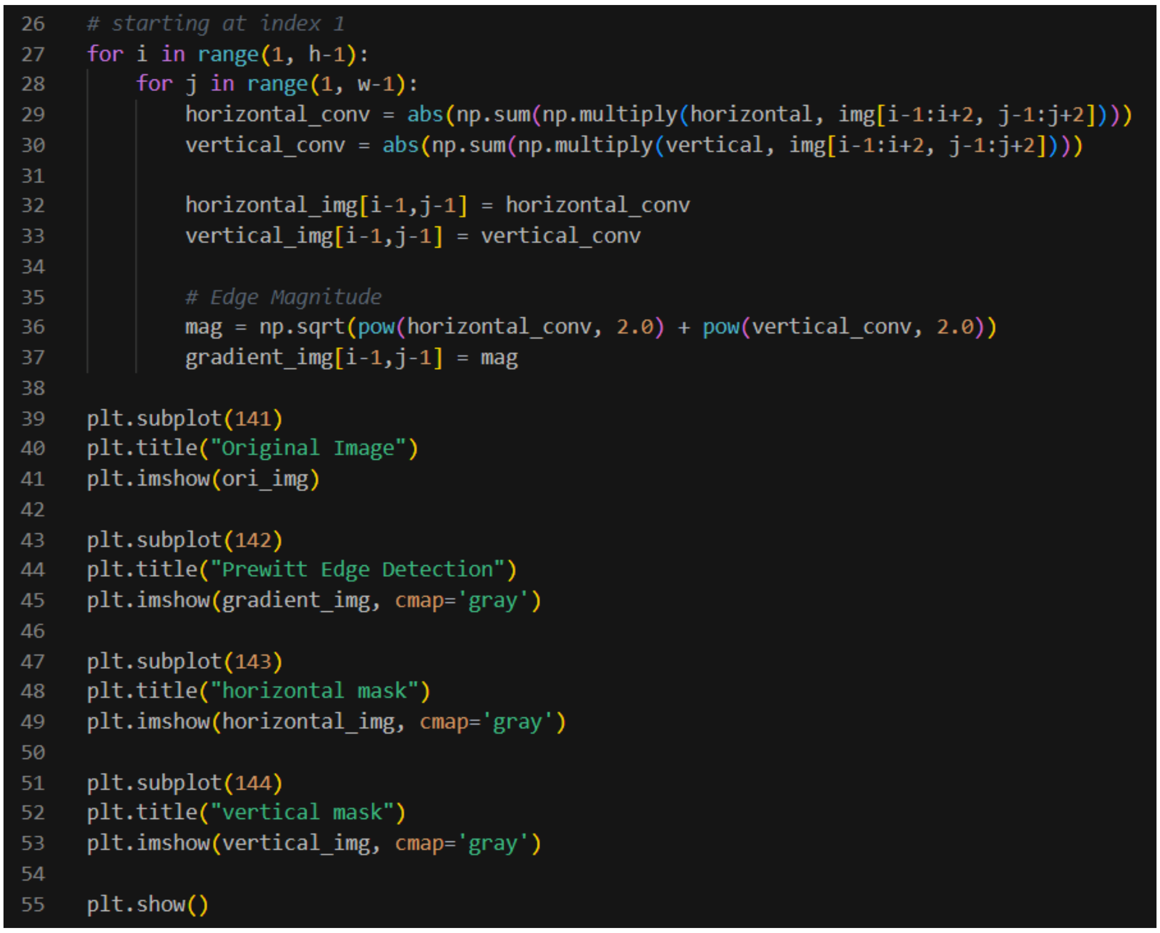
Prewitt Operator

Prewitt operator is one of edge detection methods which detects both horizontal and vertical edges in the image. We need to use a smoothing filter before finding a derivative filter for edge detection because it can reduce noise which can cause the edge's noise. For the smoothing filter, we use $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ in the horizontal axis and $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ in the vertical axis. Then, we need to convolve the smoothing filter in both axes to the image. Next, we need to find edge detection by using magnitude in gradient.

Coding part:

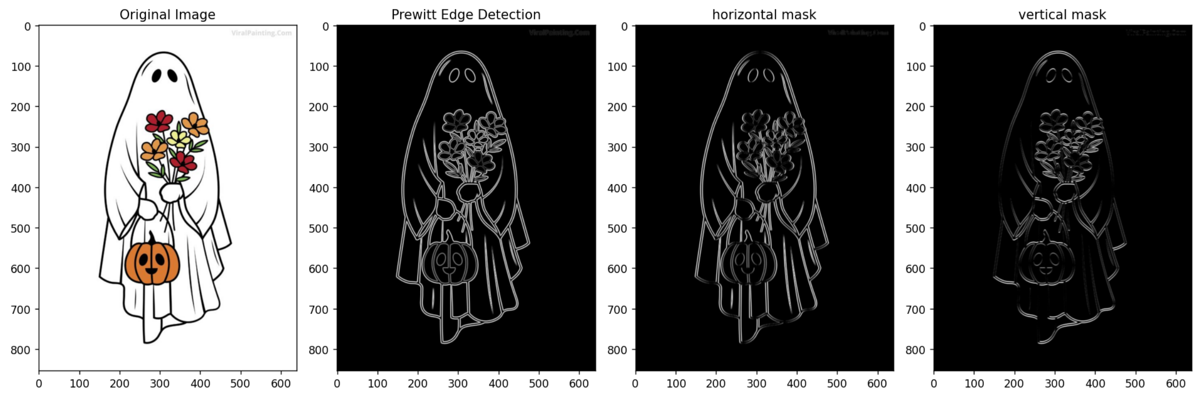


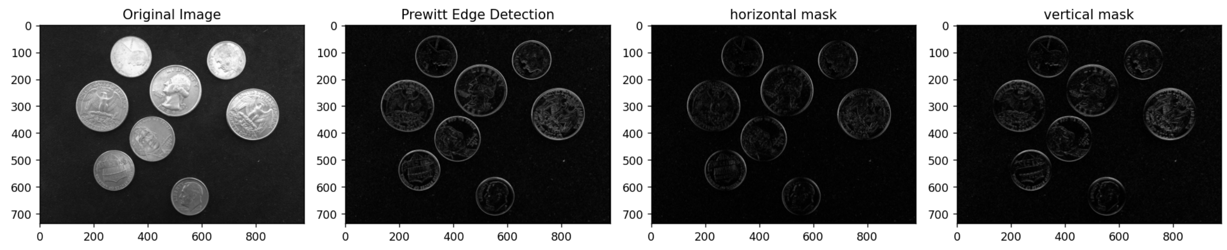
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Open the image
path = r"ghost.jpg"
ori_img = cv2.cvtColor(cv2.imread(path), cv2.COLOR_BGR2RGB)
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE).astype(float)

# Prewitt Operator
h, w = img.shape
# define filters
horizontal = np.array([[-1, 0, 1],
                       [-1, 0, 1],
                       [-1, 0, 1]])

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

# define new images
horizontal_img = np.zeros((h, w))
vertical_img = np.zeros((h, w))
gradient_img = np.zeros((h, w))
```



```python
# starting at index 1
for i in range(1, h-1):
    for j in range(1, w-1):
        horizontal_conv = abs(np.sum(np.multiply(horizontal, img[i-1:i+2, j-1:j+2])))
        vertical_conv = abs(np.sum(np.multiply(vertical, img[i-1:i+2, j-1:j+2])))

        horizontal_img[i-1,j-1] = horizontal_conv
        vertical_img[i-1,j-1] = vertical_conv

        # Edge Magnitude
        mag = np.sqrt(pow(horizontal_conv, 2.0) + pow(vertical_conv, 2.0))
        gradient_img[i-1,j-1] = mag

plt.subplot(141)
plt.title("Original Image")
plt.imshow(ori_img)

plt.subplot(142)
plt.title("Prewitt Edge Detection")
plt.imshow(gradient_img, cmap='gray')

plt.subplot(143)
plt.title("horizontal mask")
plt.imshow(horizontal_img, cmap='gray')

plt.subplot(144)
plt.title("vertical mask")
plt.imshow(vertical_img, cmap='gray')

plt.show()
```

The results:

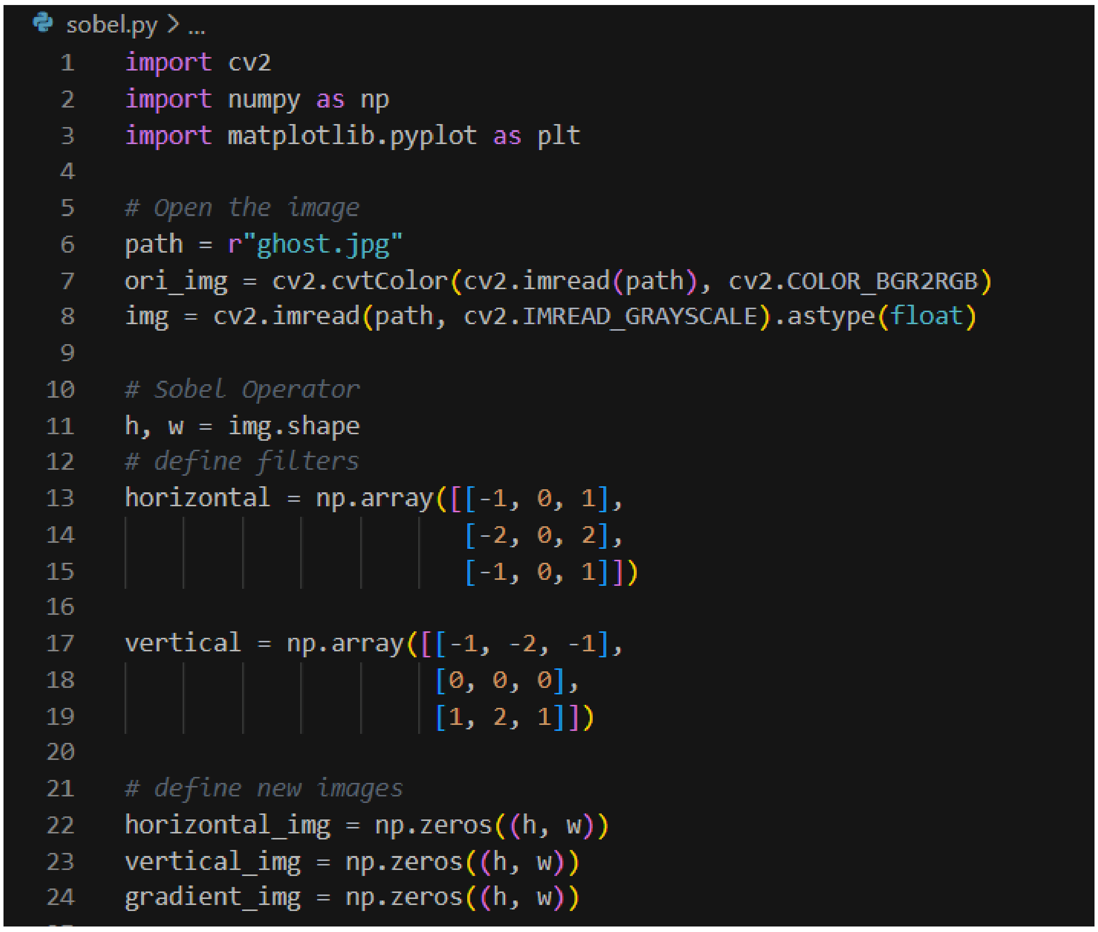




Sobel Operator

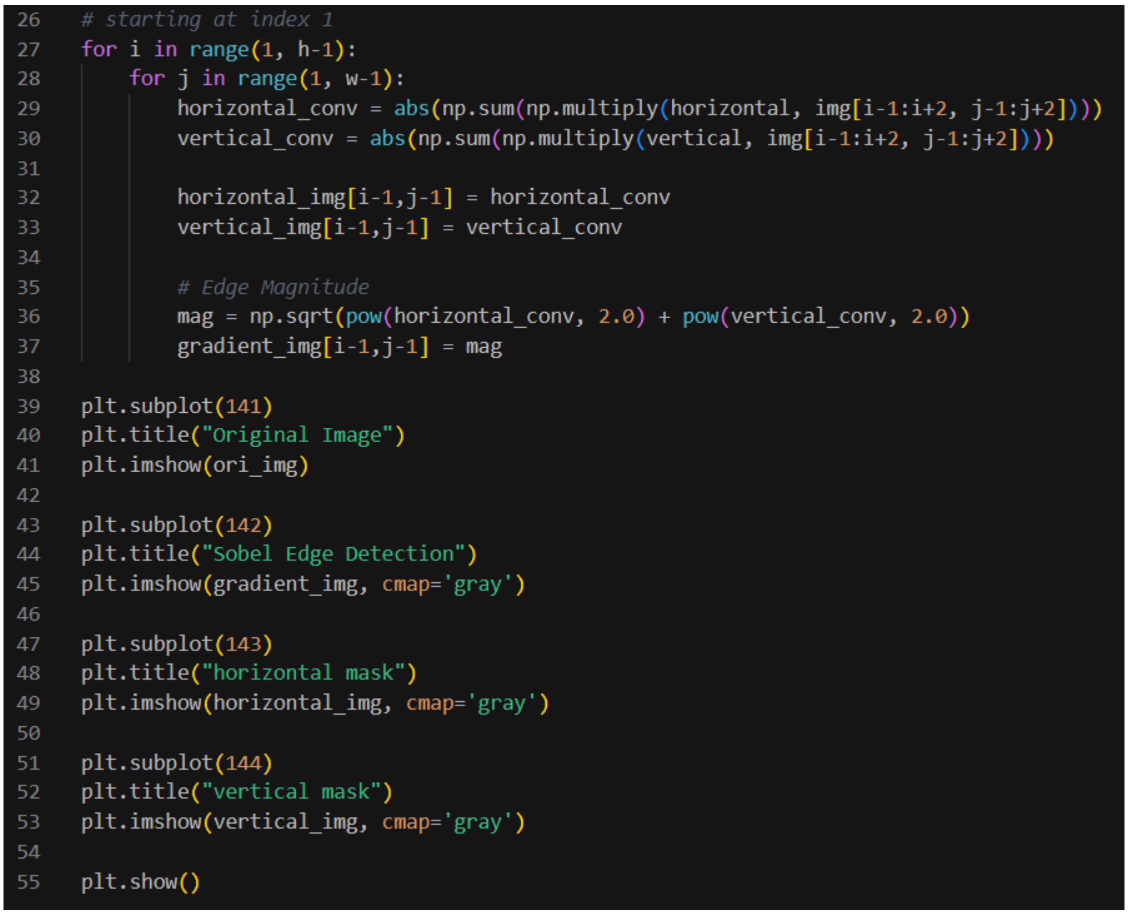
Sobel Operator is similar to the Prewitt operator but a little bit bigger for smoothing filters. The horizontal filter is $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ and the vertical filter is $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$.

The coding:



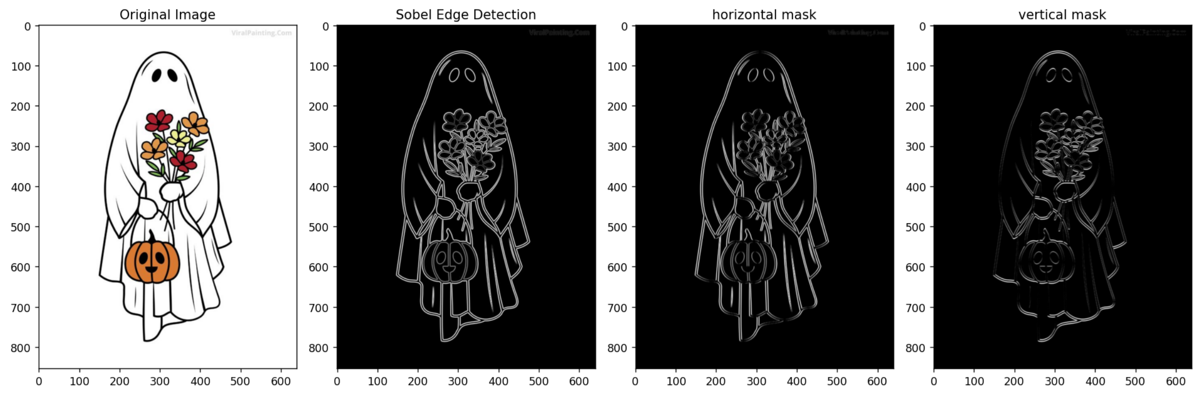
sobel.py > ...

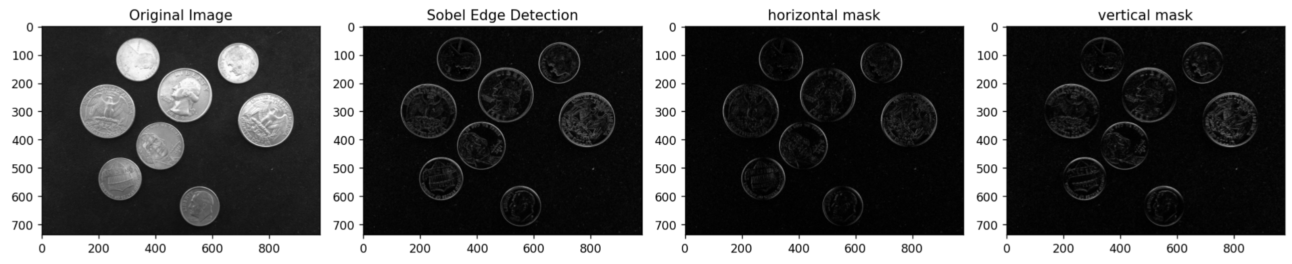
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Open the image
path = r"ghost.jpg"
ori_img = cv2.cvtColor(cv2.imread(path), cv2.COLOR_BGR2RGB)
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE).astype(float)

# Sobel Operator
h, w = img.shape
# define filters
horizontal = np.array([[-1, 0, 1],
                       [-2, 0, 2],
                       [-1, 0, 1]])

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

# define new images
horizontal_img = np.zeros((h, w))
vertical_img = np.zeros((h, w))
gradient_img = np.zeros((h, w))
```



```python
# starting at index 1
for i in range(1, h-1):
    for j in range(1, w-1):
        horizontal_conv = abs(np.sum(np.multiply(horizontal, img[i-1:i+2, j-1:j+2])))
        vertical_conv = abs(np.sum(np.multiply(vertical, img[i-1:i+2, j-1:j+2])))

        horizontal_img[i-1,j-1] = horizontal_conv
        vertical_img[i-1,j-1] = vertical_conv

        # Edge Magnitude
        mag = np.sqrt(pow(horizontal_conv, 2.0) + pow(vertical_conv, 2.0))
        gradient_img[i-1,j-1] = mag

plt.subplot(141)
plt.title("Original Image")
plt.imshow(ori_img)

plt.subplot(142)
plt.title("Sobel Edge Detection")
plt.imshow(gradient_img, cmap='gray')

plt.subplot(143)
plt.title("horizontal mask")
plt.imshow(horizontal_img, cmap='gray')

plt.subplot(144)
plt.title("vertical mask")
plt.imshow(vertical_img, cmap='gray')

plt.show()
```

The results:

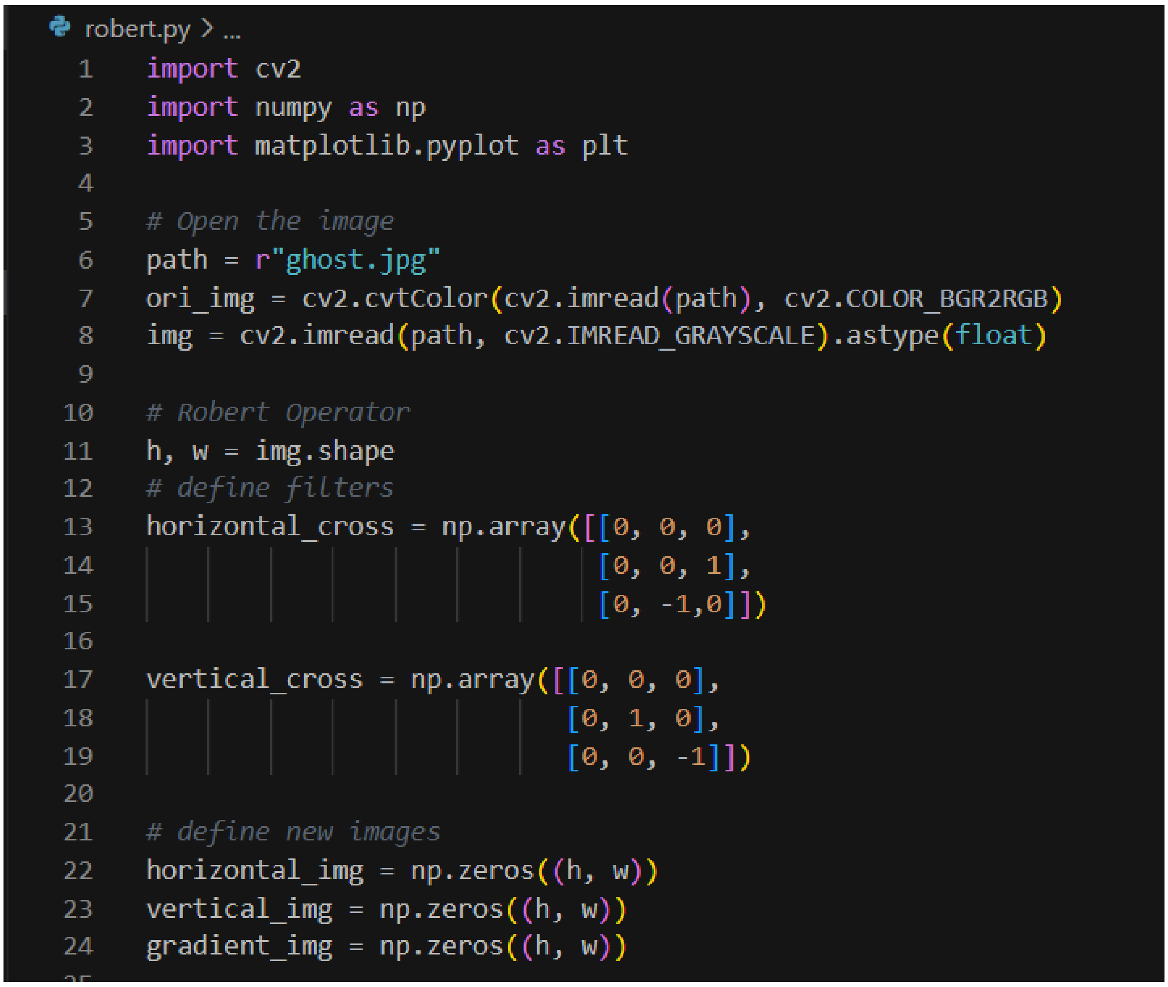




Robert Operator

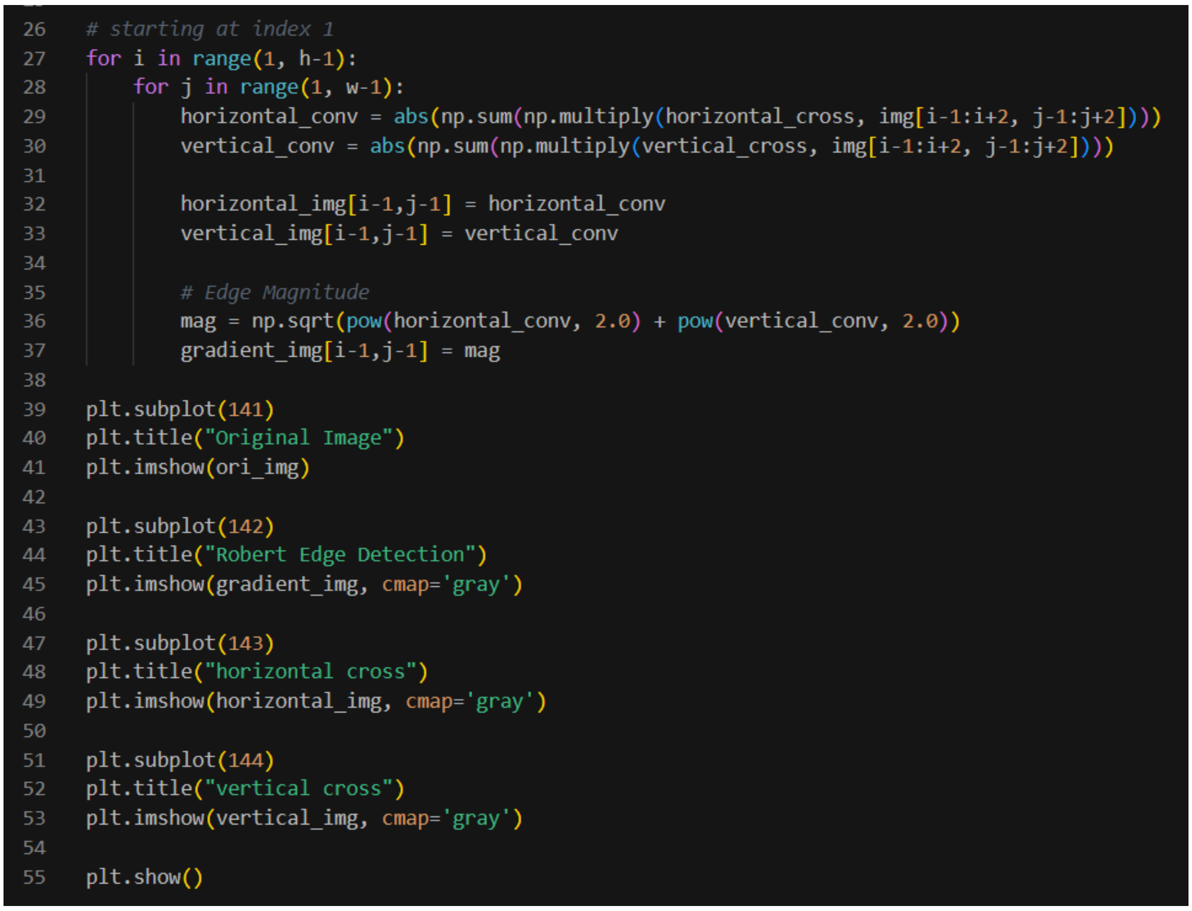
Rober operator also uses gradients to the image for edge detection as well, but uses diagonal filter masks along 2 different sides. There are $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$ for matrix 3 x 3. After filtering, we need to find the magnitude of the image in each iteration which will lead us to detect the edge in the image.

The coding:



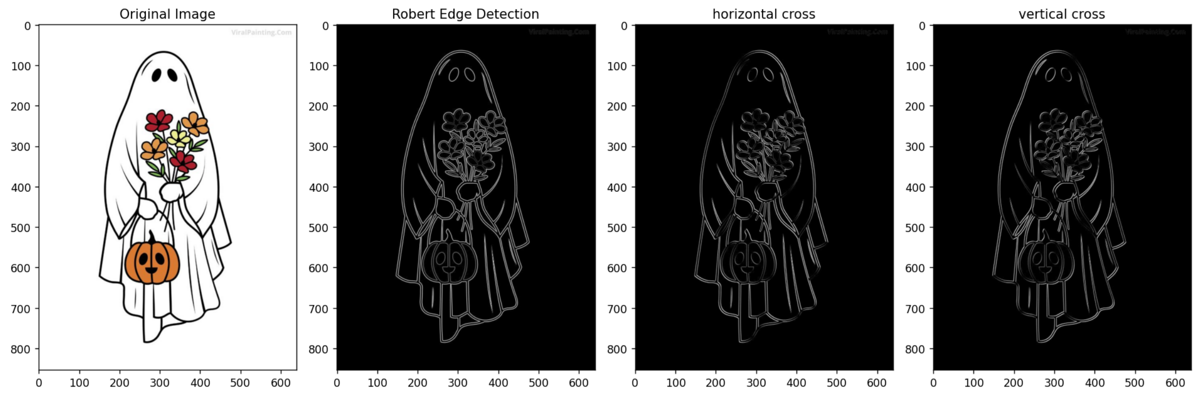
robert.py > ...

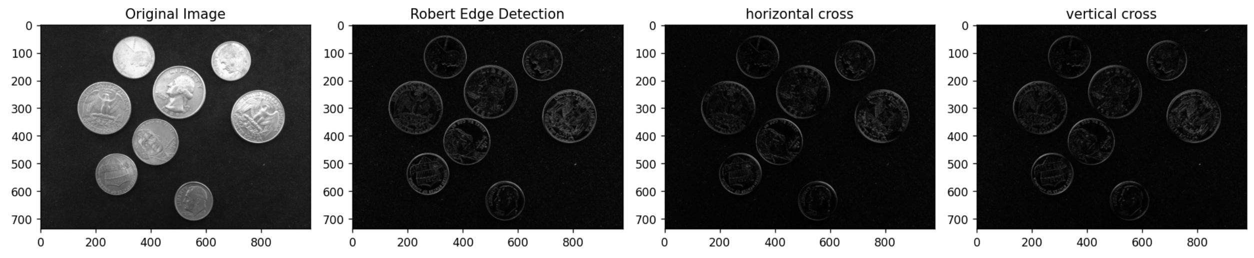
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Open the image
path = r"ghost.jpg"
ori_img = cv2.cvtColor(cv2.imread(path), cv2.COLOR_BGR2RGB)
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE).astype(float)

# Robert Operator
h, w = img.shape
# define filters
horizontal_cross = np.array([[0, 0, 0],
                             [0, 0, 1],
                             [0, -1,0]])

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

# define new images
horizontal_img = np.zeros((h, w))
vertical_img = np.zeros((h, w))
gradient_img = np.zeros((h, w))
```



```python
# starting at index 1
for i in range(1, h-1):
    for j in range(1, w-1):
        horizontal_conv = abs(np.sum(np.multiply(horizontal_cross, img[i-1:i+2, j-1:j+2])))
        vertical_conv = abs(np.sum(np.multiply(vertical_cross, img[i-1:i+2, j-1:j+2])))

        horizontal_img[i-1,j-1] = horizontal_conv
        vertical_img[i-1,j-1] = vertical_conv

        # Edge Magnitude
        mag = np.sqrt(pow(horizontal_conv, 2.0) + pow(vertical_conv, 2.0))
        gradient_img[i-1,j-1] = mag

plt.subplot(141)
plt.title("Original Image")
plt.imshow(ori_img)

plt.subplot(142)
plt.title("Robert Edge Detection")
plt.imshow(gradient_img, cmap='gray')

plt.subplot(143)
plt.title("horizontal cross")
plt.imshow(horizontal_img, cmap='gray')

plt.subplot(144)
plt.title("vertical cross")
plt.imshow(vertical_img, cmap='gray')

plt.show()
```

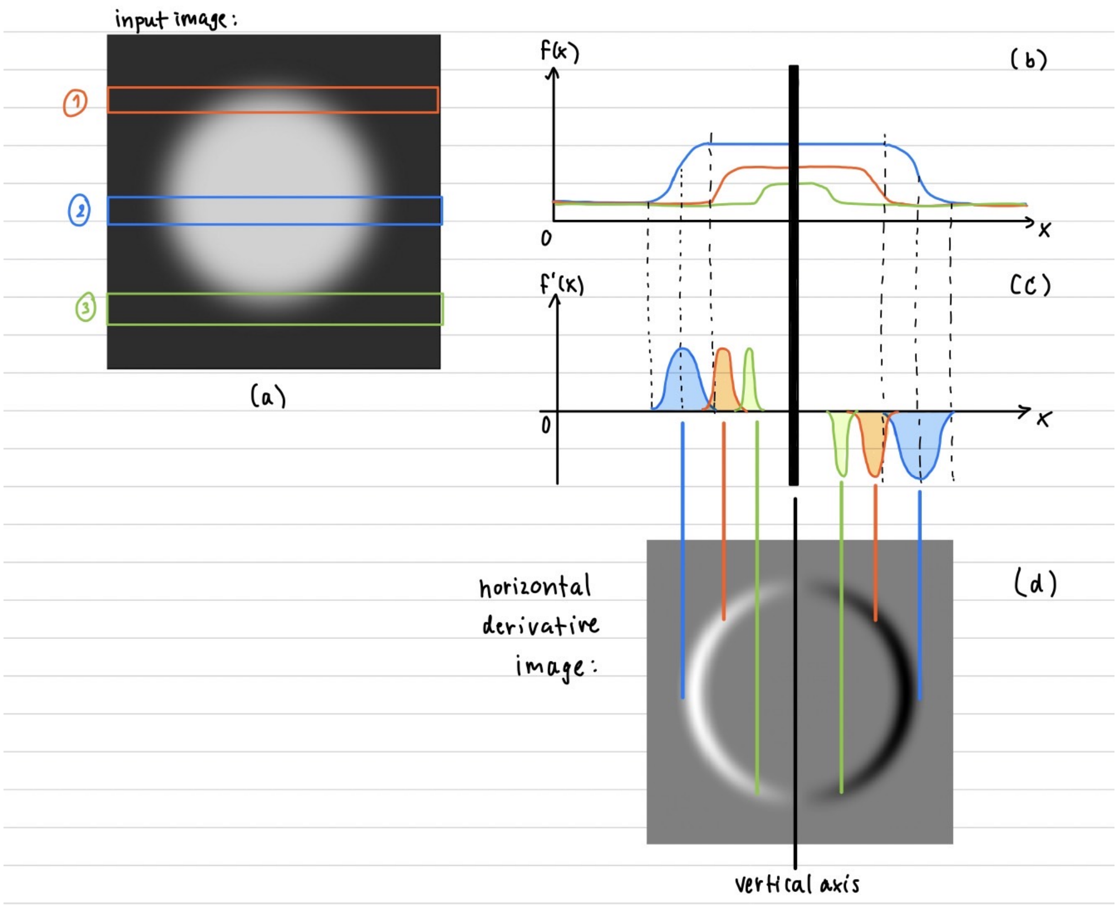
The results:





Horizontal derivative and vertical derivative

According to the results above, plotting the horizontal direction shows the vertical edges detection while plotting vertical direction shows the horizontal edges detection. It is because of the derivative intensity of the image by the interested axis. For example, we use a horizontal derivative to the image as figure below.



From (b), we plot the intensity from 3 horizontal masks in (a). Then, we plot the derivative of intensity from (b) to get the graph in (c). According to (c), we can see that there are zeros at the beginning, the middle and the ending. The between the middle and others are the values in distribution form which are considered as the edges. Therefore, we will get the result in (d). There is a vertical axis at the middle which makes left and right symmetry in different directions. To conclude, the horizontal derivative can capture the vertical edges and vice versa.