Experiment 9

Shashwat Shah
60004220126
BE Comps C22

stify - Prewitt & Sobel filters work good on noisy images.

ory: Prewitt and sobel filters are first order derivaty filters used in edge detection for image processing. These filters highlight regions of high spatial frequency. which often correspond to edge in an image. Both filters are designed to approximate the gradient of the image intensity ction, helping to identify areas where there is a sharp change in intensity i.e edges.

ustification -

hen applying the Prewitt and Sobel filters to nosy Images, the following observations can be made.

Sensitivity to Noise

Both prewitt and sobel filters are sensitive to noise, especially high frequency noise such as Gaussian noise This is because noise introduces abrupt intensity changes across the image, which the filters may incorrectly identify as edge.

Performance in noisy conditions

In noisy images, the output of prewitt and sobel filters become cluttered with fake edge cause by noise This leads to poor performance in detecting the actual structure of the image.

Conclusion : Using the Prewitt and Sobel filters on noi
does not yield good result. These filters are desi
to highlight edges based on intensity gradients, bu
noise introduces gradients that are mistaken for e

NAME: Shashwat Shah SAP ID: 60004220126 DIV/BATCH:C22 DATE: 14/10/24

DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 09

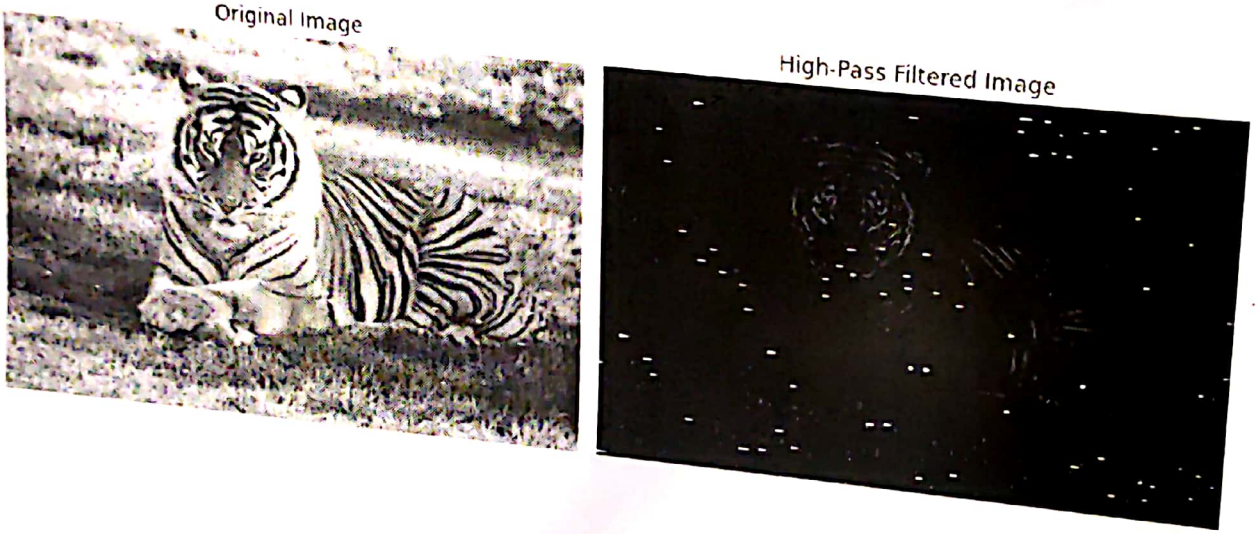
AIM: To implement Edge detection using Sobel & Perwitt masks.

CODE:
SOBEL

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

# Load the uploaded image
image_path = 'image2.jpg'  # Replace with the actual path if needed image = cv2.imread(image_path)

# Step 1: Display the original image plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image') plt.axis('off')

# Step 2: Convert the image to grayscale and display it gray_image = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY) plt.subplot(3, 3, 2), plt.imshow(gray_image, cmap='gray'),
plt.title('Grayscale Image')
plt.axis('off')

# Step 3: Apply Sobel X (fx) on grayscale image sobel_x_kernel = np.array([[-1, 0,
1],
                                    [-2, 0, 2],
                                    [-1, 0, 1]])

r1 = cv2.filter2D(gray_image, -1, sobel_x_kernel)  # Applying Sobel X plt.subplot(3, 3, 3),
plt.imshow(r1, cmap='gray'), plt.title('Sobel X (r1)')
plt.axis('off')

        # Step 4: Apply Sobel Y (fy) on  grayscale image
        sobel_y_kernel = np.array([[-1,  -2, -1],
                                   [0,  0, 0],
                                   [1,  2, 1]])

r2 = cv2.filter2D(gray_image, -1, sobel_y_kernel)  # Applying Sobel Y
```

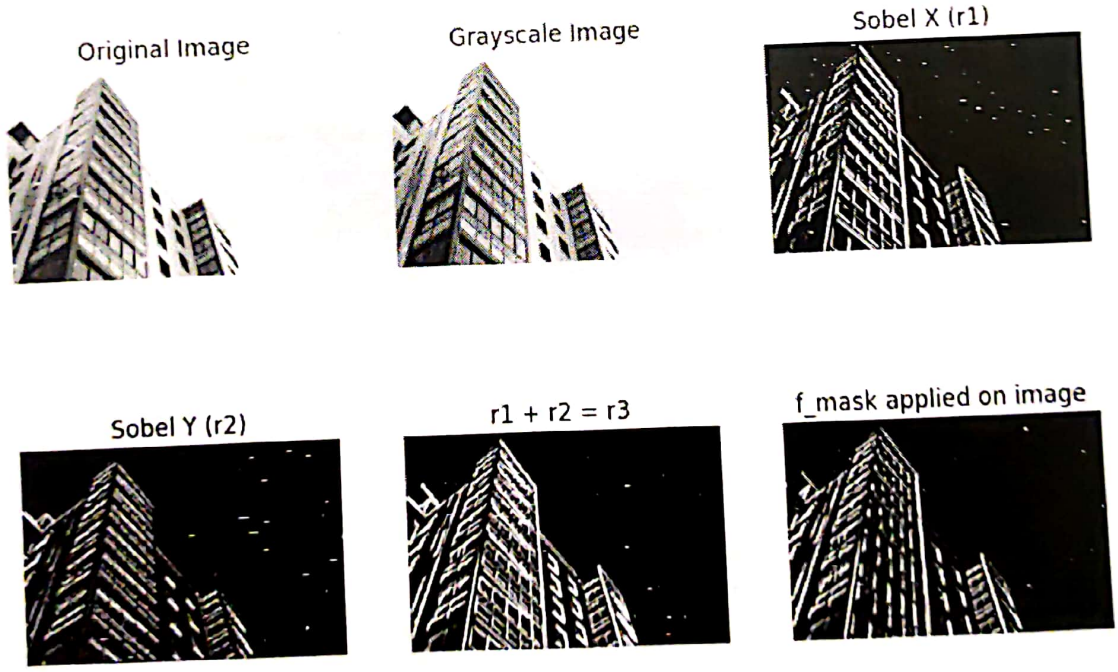
```
# High-pass filtered image
plt.subplot(1, 3, 2)
plt.imshow(high_pass)
plt.title('High-Pass Filtered Image')
plt.axis('off')

plt.tight_layout()
plt.show()
```

OUTPUT:



```
lt.subplot(3, 3, 4), plt.imshow(r2, cmap='gray'), plt.title('Sobel Y (r2)')
lt.axis('off')

 Step 5: Add r1 and r2 to obtain r3 and display
3 = cv2.add(np.abs(r1), np.abs(r2))  # Adding r1 and r2 plt.subplot(3, 3, 5), plt.imshow(r3,
:map='gray'), plt.title('r1 + r2 = r3')
plt.axis('off')

# Step 6: Add fx and fy to obtain f_mask
f_mask = sobel_x_kernel + sobel_y_kernel  # Adding Sobel X and Y kernels
# plt.subplot(3, 3, 6), plt.imshow(f_mask, cmap='gray'), plt.title('f_mask = fx + fy')
plt.axis('off')

# Step 7: Apply f_mask on the grayscale image and display f_mask_applied =
cv2.filter2D(gray_image, -1, f_mask)  # Applying combined mask
plt.subplot(3, 3, 6), plt.imshow(f_mask_applied, cmap='gray'), plt.title('f_mask applied on image')
plt.axis('off')

# Show all the steps in one figure plt.show()
```

OUTPUT:



PERWITT

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

# Step 1: Load and display the original image image =
cv2.imread('image2.jpg') plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image') plt.axis('off')

# Step 2: Convert the image to grayscale and display it gray_image = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY) plt.subplot(3, 3, 2), plt.imshow(gray_image, cmap='gray'),
plt.title('Grayscale Image')
plt.axis('off')

# Define Prewitt kernels prewitt_x = np.array([[ -
1, 0, 1],
                                    [ -1, 0, 1],
                                    [ -1, 0, 1]])

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

# Step 3: Apply Prewitt operator (derivative in x direction) and display the result (r1)
r1 = cv2.filter2D(gray_image, cv2.CV_64F, prewitt_x)
r1 = np.abs(r1)  # Absolute values to handle negative edges plt.subplot(3, 3, 3), plt.imshow(r1,
cmap='gray'), plt.title('Prewitt X (r1)')
plt.axis('off')

# Step 4: Apply Prewitt operator (derivative in y direction) and display the result (r2)
r2 = cv2.filter2D(gray_image, cv2.CV_64F, prewitt_y)
r2 = np.abs(r2)  # Absolute values to handle negative edges plt.subplot(3, 3, 4), plt.imshow(r2,
cmap='gray'), plt.title('Prewitt Y (r2)')
plt.axis('off')

# Step 5: Add r1 and r2 to get r3, and display r3 r3 = cv2.add(r1, r2)
plt.subplot(3, 3, 5), plt.imshow(r3, cmap='gray'), plt.title('r1 + r2 = r3')
plt.axis('off')
```

```
# Step 6: Add Prewitt X and Prewitt Y to obtain the filter mask
f_mask = prewitt_x + prewitt_y
plt.axis('off')
f_mask_applied = cv2.filter2D(gray_image, cv2.CV_64F, f_mask)

# Display the result of applying f_mask on the grayscale image
plt.subplot(3, 3, 6), plt.imshow(np.abs(f_mask_applied), cmap='gray'),
plt.title('f_mask applied on Image')
plt.axis('off')


plt.tight_layout()
plt.show()
```

OUTPUT: