Computer Vision

Home Work 9

Name – ROHIT DAS Student ID – NTNU_61047086s

Project – Edge Detection.

Language and library used: Python, OpenCV, Numpy.

Description: This program will perform the following functions while executing lena.bmp image file:

- Convert image to Grayscale
- Apply Prewitt Edge Detection
- Apply Robert Edge Detection
- Apply Sobel Edge Detection
- Apply Frei Chen Edge Detection
- Apply Robinson Compass Image
- Apply Krisch Edge Detection
- Apply Nevatia Babu Image
- All with their respective threshold

Parameters: None. Please Copy-paste the image path inside the program.

Algorithms Used –

Part 1: Convert Image to Grayscale

Principal Code:

```
image_file = r"F:\Fall 2021 NTNU\Computer Vision NTU\Chapter-7\HomeWork-
9\lena.bmp"
gray_image = cv2.imread(image_file,0)
```

Part 2: Prewitt Edge Detection

Principal Code:

```
def prewitt(g_image,threshold):
    img = np.copy(g_image)
    prewittImage = np.copy(g_image)
    # Horizontal_kernel
    k1 = np.array([
        [-1, 0, 1],
        [-1, 0, 1],
        [-1, 0, 1]
    ])
    # Vertical_kernel
    k2 = np.array([
        [-1, -1, -1],
        [0, 0, 0],
        [1, 1, 1]
    ])
    # Avoid Out of Index Control
    index_control = np.zeros((img.shape[0] - 2, img.shape[1] - 2), np.int)
    for h_row in range(index_control.shape[0]):
        for w_col in range(index_control.shape[1]):
            p1 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k1)
            p2 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k2)
            # Calculate Gradient Magnitude
            G = np.sqrt(p1 ** 2 + p2 ** 2)
            if G >= threshold:
                prewittImage[h_row,w_col] = 0
            else:
                prewittImage[h_row,w_col] = 255

    return prewittImage

def convolution(img, kernel):
    row_a, col_a = img.shape
    rb, col_b = kernel.shape
    res = 0
    for h_row in range(row_a):
        for w_col in range(col_a):
            if 0 <= row_a - h_row - 1 < rb and 0 <= col_a - w_col - 1 <
col_b:
                res += img[h_row, w_col] * kernel[row_a - h_row - 1, col_a
- w_col - 1]
    return res
```

Part 3: Robert Edge Detection

Principal Code –

```
def robert(g_image,threshold):
    img = np.copy(g_image)
    RobertImage = np.copy(g_image)
    # Horizontal_kernel
    k1 = np.array([
        [1, 0],
        [0, -1]
    ])
    # Vertical_kernel
    k2 = np.array([
        [0, 1],
        [-1, 0]
    ])
    # avoid out of image range
    index_control = np.zeros((img.shape[0] - 2, img.shape[1] - 2),
dtype='int32')
    for h_row in range(index_control.shape[0]):
        for w_col in range(index_control.shape[1]):
            # calculate Gx and Gy of Robert
            p1 = convolution(img[h_row:h_row + 2, w_col:w_col + 2], k1)
            p2 = convolution(img[h_row:h_row + 2, w_col:w_col + 2], k2)
            # Calculate Gradient Magnitude
            G = np.sqrt(p1 ** 2 + p2 ** 2)
            if G >= threshold:
                RobertImage[h_row, w_col] = 0
            else:
                RobertImage[h_row, w_col] = 255
    return RobertImage
def convolution(img, kernel):
    row_a, col_a = img.shape
    rb, col_b = kernel.shape
    res = 0
    for h_row in range(row_a):
        for w_col in range(col_a):
            if 0 <= row_a - h_row - 1 < rb and 0 <= col_a - w_col - 1 <
col_b:
                res += img[h_row, w_col] * kernel[row_a - h_row - 1, col_a
- w_col - 1]
    return res
```

Part 4: Sobel Edge Detection

Principal Code-

```
def sobel(g_image,threshold):
    img = np.copy(g_image)
    SobelImage = np.copy(g_image)
    # Horizontal kernel
    k1 = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ])
    # Vertical Kernel
    k2 = np.array([
        [-1, -2, -1],
        [0, 0, 0],
        [1, 2, 1]
    ])
    # Avoid Index Out of Control
    index_control = np.zeros((img.shape[0] - 2, img.shape[1] - 2),
dtype='int32')
    for h_row in range(index_control.shape[0]):
        for w_col in range(index_control.shape[1]):
            # calculate Gx and Gy of Sobel
            p1 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k1)
            p2 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k2)
            # Calculate Gradient Magnitude
            G = np.sqrt(p1 ** 2 + p2 ** 2)
            if G >= threshold:
                SobelImage[h_row, w_col] = 0
            else:
                SobelImage[h_row, w_col] = 255

    return SobelImage

def convolution(img, kernel):
    row_a, col_a = img.shape
    rb, col_b = kernel.shape
    res = 0
    for h_row in range(row_a):
        for w_col in range(col_a):
            if 0 <= row_a - h_row - 1 < rb and 0 <= col_a - w_col - 1 <
col_b:
                res += img[h_row, w_col] * kernel[row_a - h_row - 1, col_a
- w_col - 1]
    return res
```

Part 5: Frei and Chen Gradient Operator Edge Detection.

Principal Code-

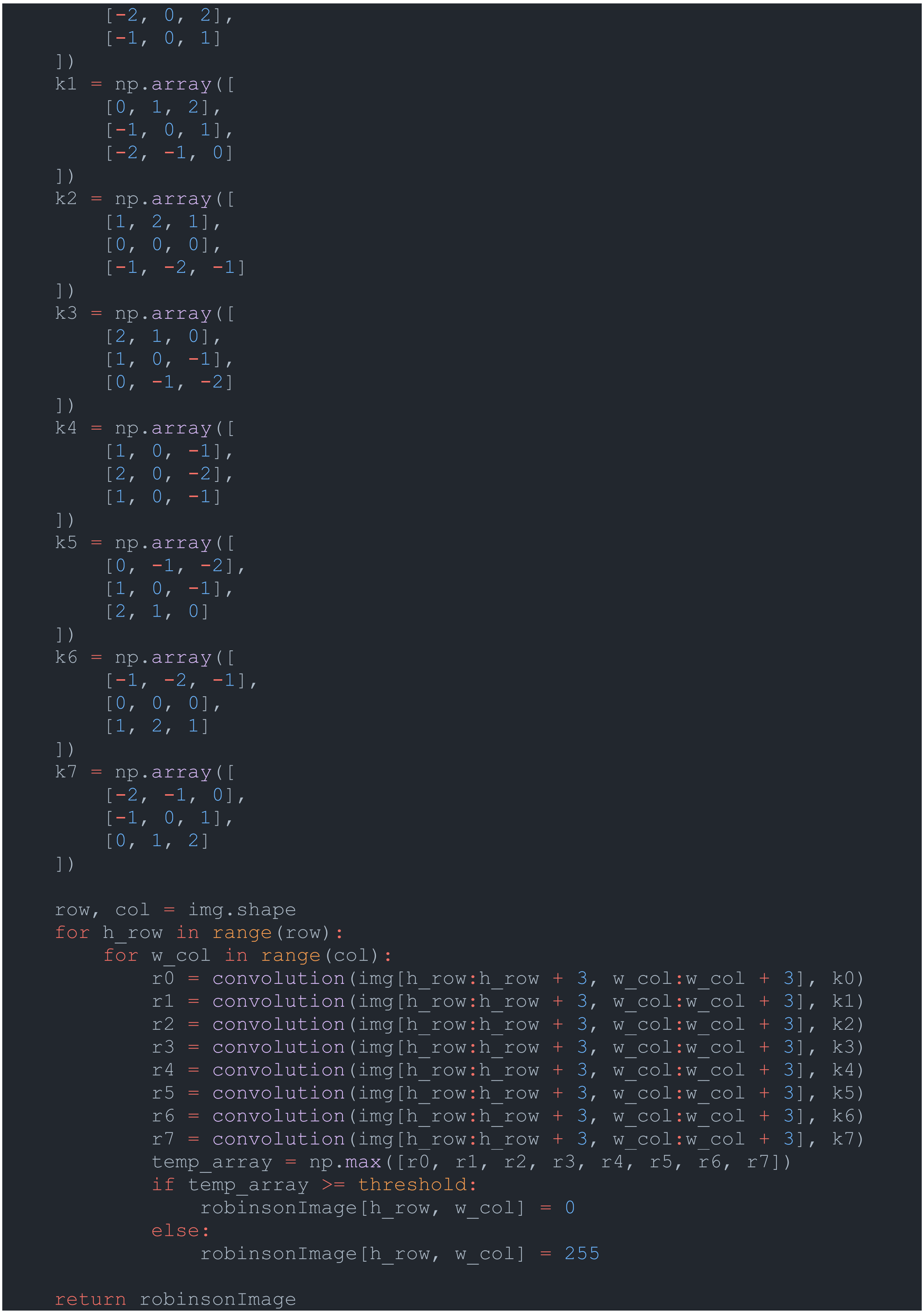
```
def Frei_and_Chen_Gradient_operator(g_image,threshold):
    img = np.copy(g_image)
    Frei_Chen_Image = np.copy(g_image)
    # Horizontal kernel
    k1 = np.array([
        [-1, -np.sqrt(2), -1],
        [0, 0, 0],
        [1, np.sqrt(2), 1]
    ])
    # Vertical Kernel
    k2 = np.array([
        [-1, 0, 1],
        [-np.sqrt(2), 0, np.sqrt(2)],
        [-1, 0, 1]
    ])
    # Avoid Index Out of Control
    index_control = np.zeros((img.shape[0] - 2, img.shape[1] - 2),
dtype='int32')
    for h_row in range(index_control.shape[0]):
        for w_col in range(index_control.shape[1]):
            # calculate Gx and Gy of Frei and Chen
            p1 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k1)
            p2 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k2)
            # Calculate Gradient Magnitude
            G = np.sqrt(p1 ** 2 + p2 ** 2)
            if G >= threshold:
                Frei_Chen_Image[h_row, w_col] = 0
            else:
                Frei_Chen_Image[h_row, w_col] = 255
    return Frei_Chen_Image

def convolution(img, kernel):
    row_a, col_a = img.shape
    rb, col_b = kernel.shape
    res = 0
    for h_row in range(row_a):
        for w_col in range(col_a):
            if 0 <= row_a - h_row - 1 < rb and 0 <= col_a - w_col - 1 <
col_b:
                res += img[h_row, w_col] * kernel[row_a - h_row - 1, col_a
- w_col - 1]
    return res
```

Part 6 : Krisch Operator Edge Detection

```
def Krisch(g_image,threshold):
    img = np.copy(g_image)
    KrischImage = np.copy(g_image)
    k0 = np.array([
        [-3, -3, 5],
        [-3, 0, 5],
        [-3, -3, 5]
    ])
    k1 = np.array([
        [-3, 5, 5],
        [-3, 0, 5],
        [-3, -3, -3]
    ])
    k2 = np.array([
        [5, 5, 5],
        [-3, 0, -3],
        [-3, -3, -3]
    ])
    k3 = np.array([
        [5, 5, -3],
        [5, 0, -3],
        [-3, -3, -3]
    ])
    k4 = np.array([
        [5, -3, -3],
        [5, 0, -3],
        [5, -3, -3]
    ])
    k5 = np.array([
        [-3, -3, -3],
        [5, 0, -3],
        [5, 5, -3]
    ])
    k6 = np.array([
        [-3, -3, -3],
        [-3, 0, -3],
        [5, 5, 5]
    ])
    k7 = np.array([
        [-3, -3, -3],
        [-3, 0, 5],
        [-3, 5, 5]
    ])

    row, col = img.shape
    for h_row in range(row):
        for w_col in range(col):
            r0 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k0)
            r1 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k1)
            r2 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k2)
            r3 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k3)
            r4 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k4)
            r5 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k5)
            r6 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k6)
            r7 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k7)
            temp_array = np.max([r0, r1, r2, r3, r4, r5, r6, r7])
            if temp_array >= threshold:
                KrischImage[h_row, w_col] = 0
            else:
                KrischImage[h_row, w_col] = 255

    return KrischImage
```

Part 7 : Robinson Compass Operator Edge Detection

```
def robinson_compass_operator(g_image,threshold):
    img = np.copy(g_image)
    robinsonImage = np.copy(g_image)
    k0 = np.array([
        [-1, 0, 1],
```



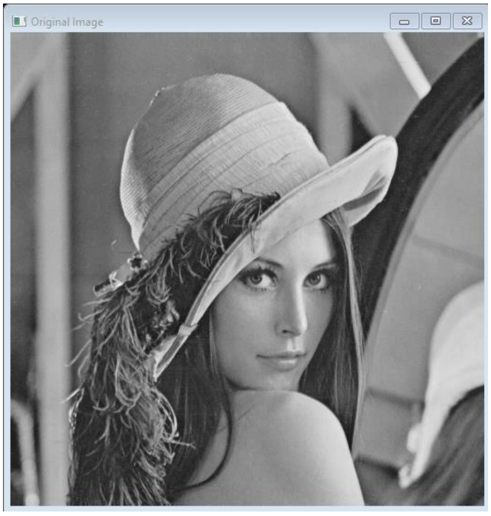
```
        [-2, 0, 2],
        [-1, 0, 1]
    ])
    k1 = np.array([
        [0, 1, 2],
        [-1, 0, 1],
        [-2, -1, 0]
    ])
    k2 = np.array([
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1]
    ])
    k3 = np.array([
        [2, 1, 0],
        [1, 0, -1],
        [0, -1, -2]
    ])
    k4 = np.array([
        [1, 0, -1],
        [2, 0, -2],
        [1, 0, -1]
    ])
    k5 = np.array([
        [0, -1, -2],
        [1, 0, -1],
        [2, 1, 0]
    ])
    k6 = np.array([
        [-1, -2, -1],
        [0, 0, 0],
        [1, 2, 1]
    ])
    k7 = np.array([
        [-2, -1, 0],
        [-1, 0, 1],
        [0, 1, 2]
    ])

    row, col = img.shape
    for h_row in range(row):
        for w_col in range(col):
            r0 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k0)
            r1 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k1)
            r2 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k2)
            r3 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k3)
            r4 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k4)
            r5 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k5)
            r6 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k6)
            r7 = convolution(img[h_row:h_row + 3, w_col:w_col + 3], k7)
            temp_array = np.max([r0, r1, r2, r3, r4, r5, r6, r7])
            if temp_array >= threshold:
                robinsonImage[h_row, w_col] = 0
            else:
                robinsonImage[h_row, w_col] = 255

    return robinsonImage
```

Part 8: Nevatia Babu Operator Edge Detection

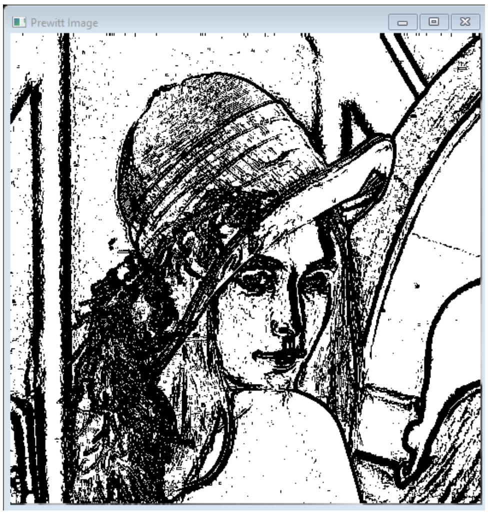
```
def nevatia_babu_operator(g_image,threshold):
    # Nevatia Babu 5*5 Operator
    img = np.copy(g_image)
    nevatia_babu_Image = np.copy(g_image)
    k0 = np.array([
        [100, 100, 100, 100, 100],
        [100, 100, 100, 100, 100],
        [0, 0, 0, 0, 0],
        [-100, -100, -100, -100, -100],
        [-100, -100, -100, -100, -100],
    ])
    k1 = np.array([
        [100, 100, 100, 100, 100],
        [100, 100, 100, 78, -32],
        [100, 92, 0, -92, -100],
        [32, -78, -100, -100, -100],
        [-100, -100, -100, -100, -100]
    ])
    k2 = np.array([
        [100, 100, 100, 32, -100],
        [100, 100, 92, -78, -100],
        [100, 100, 0, -100, -100],
        [100, 78, -92, -100, -100],
        [100, -32, -100, -100, -100]
    ])
    k3 = np.array([
        [-100, -100, 0, 100, 100],
        [-100, -100, 0, 100, 100],
        [-100, -100, 0, 100, 100],
        [-100, -100, 0, 100, 100],
        [-100, -100, 0, 100, 100]
    ])
    k4 = np.array([
        [-100, 32, 100, 100, 100],
        [-100, -78, 92, 100, 100],
        [-100, -100, 0, 100, 100],
        [-100, -100, -92, 78, 100],
        [-100, -100, -100, -32, 100]
    ])
    k5 = np.array([
        [100, 100, 100, 100, 100],
        [-32, 78, 100, 100, 100],
        [-100, -92, 0, 92, 100],
        [-100, -100, -100, -78, 32],
        [-100, -100, -100, -100, -100]
    ])
    # Avoid Index out of Bound
    index_control = np.zeros((img.shape[0] - 4, img.shape[1] - 4), np.int)
    for h_row in range(index_control.shape[0]):
        for w_col in range(index_control.shape[1]):
            n0 = np.sum(img[h_row: h_row + 5, w_col: w_col + 5] * k0)
            n1 = np.sum(img[h_row: h_row + 5, w_col: w_col + 5] * k1)
            n2 = np.sum(img[h_row: h_row + 5, w_col: w_col + 5] * k2)
            n3 = np.sum(img[h_row: h_row + 5, w_col: w_col + 5] * k3)
            n4 = np.sum(img[h_row: h_row + 5, w_col: w_col + 5] * k4)
            n5 = np.sum(img[h_row: h_row + 5, w_col: w_col + 5] * k5)
            temp_array = np.max([n0, n1, n2, n3, n4, n5])
            if temp_array >= threshold:
                nevatia_babu_Image[h_row, w_col] = 0
            else:
                nevatia_babu_Image[h_row, w_col] = 255

    return nevatia_babu_Image
```

Example:

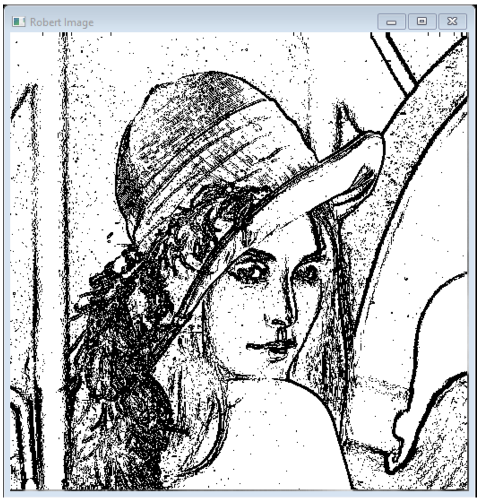
- Original image



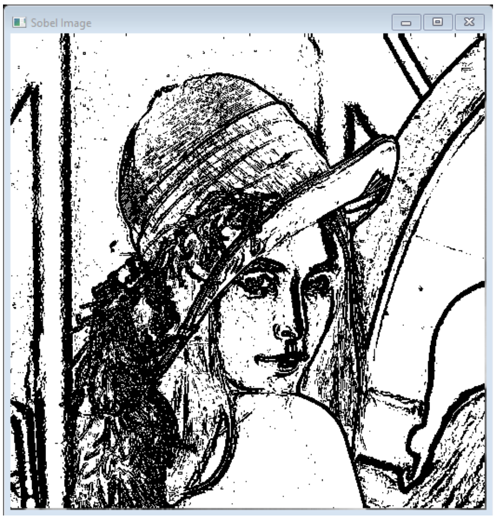
- Apply Prewitt Edge Detection



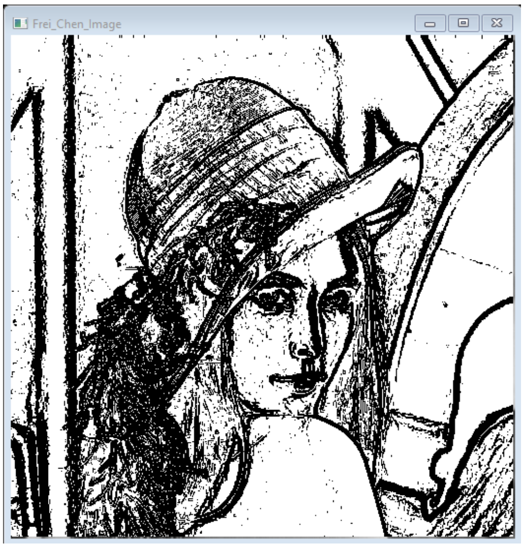
- Apply Robert Edge Detection



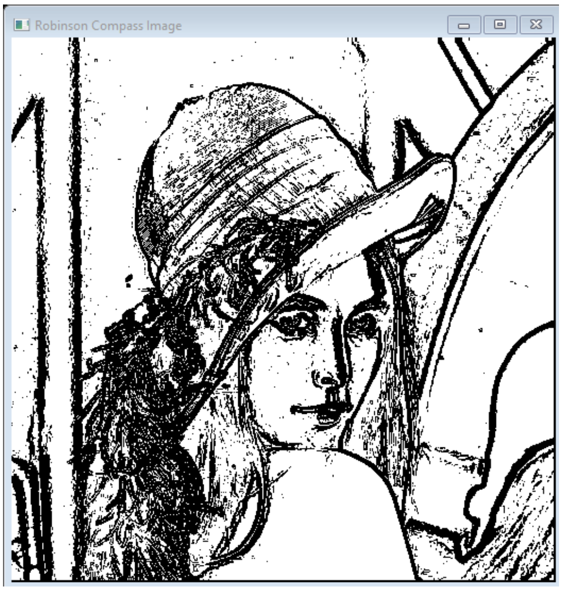
- Apply Sobel Edge Detection



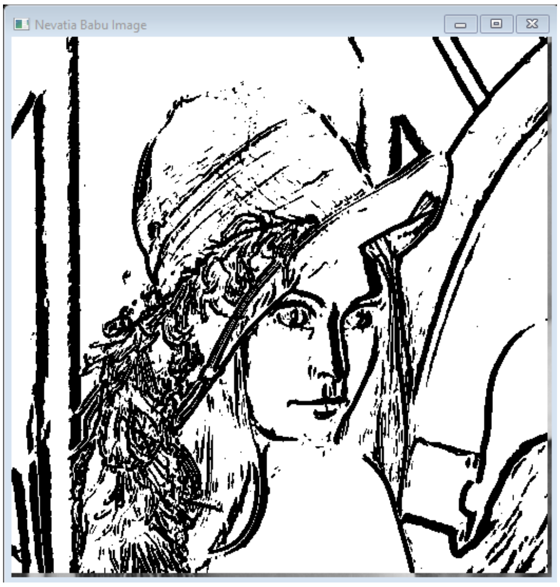
- Apply Frei Chen Edge Detection



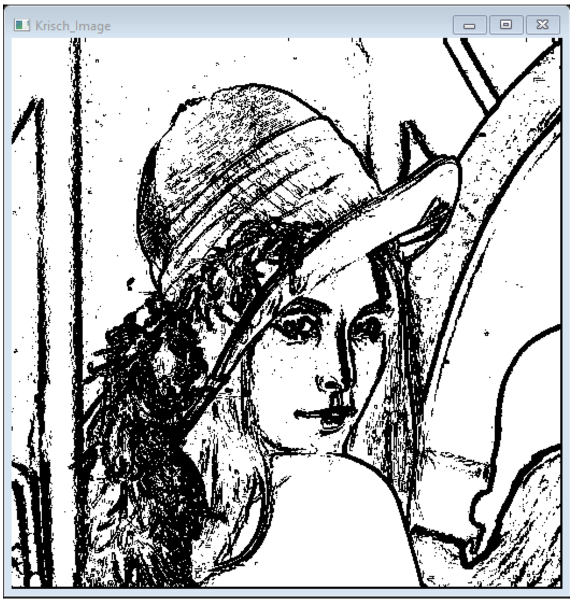
- Apply Robinson Compass Image



- Apply Nevatia Babu Image



- Apply Krisch Edge Detection


Krisch_Image