EXPERIMENT-

**AIM:** Write a program to generate an artificial black and white image and apply first order, second order derivative operator and difference of guassian.

**SOFTWARE**: Python(spyder)

**THEORY:**

**1st Order Derivative Filter (Sobel Filter):**

Most edge detection methods work on the assumption that the edge occurs where there is a discontinuity in the intensity function or a very steep intensity gradient in the image. Using this assumption, if one take the derivative of the intensity value across the image and find points where the derivative is maximum then the edge could be located. The gradient is a vector, whose components measure how rapid pixel value are changing with distance in the x and y direction. Thus, the components of the gradient found by using the following equations (1) & (2). 𝜕𝑓(𝑥, 𝑦) 𝜕𝑥 = Δ𝑥 = 𝑓(𝑥 + 𝑑𝑥, 𝑦) − 𝑓(𝑥, 𝑦) 𝑑𝑥 … … . (1)

𝜕𝑓(𝑥, 𝑦) 𝜕𝑦 = Δ𝑦 = 𝑓(𝑥, 𝑦 + 𝑑𝑦) − 𝑓(𝑥, 𝑦) 𝑑𝑦 … … . (2)

Where dx & dy measure distance along the x and y directions respectively. In discrete images, one can consider dx & dy in terms of numbers of pixel between two points. dx = dy = 1 (pixel spacing) is the point at which pixel coordinates are(i, j) thus, the value of (Δ𝑥 𝑎𝑛𝑑 Δ𝑦) can calculated by equations (3) & (4).

Δ𝑥 = 𝑓(𝑖 + 1,𝑗) − 𝑓(𝑖,𝑗) … … … (3)

Δ𝑦 = 𝑓(𝑖,𝑗 + 1) − 𝑓(𝑖,𝑗) … … …(4)

In order to detect the presence of a gradient discontinuity, one could calculate the change in the gradient at (i, j) .This can be done by finding the following magnitude measure and the gradient direction θ is given by the equation (5).

∅ = tan−1 [ Δ𝑦 /Δ ] …… . . (5)

The Sobel operator is an example of the gradient method. It is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function .

Δ𝑥 = [ −1 1 0 0 ],

Δ𝑦 = [ −1 0 1 0 ]

An advantage of using a larger mask size is that the errors due to the effects of noise are reduced by local averaging within the neighborhood of the mask. An advantage of using a mask of odd size is that the operators are centered and can therefore provide an estimate that is based on a center pixel (i,j). One important edge operator of this type is the Sobel edge operator.

Δ𝑥 = [ −1 0 1 −2 0 2 −1 0 1 ],

Δ𝑦 = [ 1 2 1 0 0 0 −1 −1 −1 ]

**2nd Order Derivative Operators (Laplacian Filter) :**

The Laplacian is a 2-D measure of the 2 nd derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise. The operator normally takes a single gray level image as input and produces another binary image as output. The zero crossing detector looks for places in the Laplacian of an image where the value of the Laplacian passes through zero i.e. points where the Laplacian changes sign. Such points often occur at edges in images i.e. points where the intensity of theimage changes rapidly, but they also occur at places that are not as easy to associate with edges. It is best to think of the zero crossing detector as some sort of feature detector rather than as a specific edge detector. Zero crossings always lie on closed contours, and so the output from the zero crossing detectors is usually a binary image with single pixel thickness lines showing the positions of the zero crossing points.

Δ 2𝑓 = 𝜕 2𝑓 /𝜕𝑥2 + 𝜕 2𝑓/ 𝜕𝑦2 …..(6)

For X-direction, 𝜕 2𝑓 /𝜕𝑥2 = 𝑓(𝑥 + 1, 𝑦) + 𝑓(𝑥 − 1, 𝑦) − 2𝑓(𝑥, 𝑦)….(7)

For Y-direction, 𝜕 2𝑓/ 𝜕𝑦2 = 𝑓(𝑥, 𝑦 + 1) + 𝑓(𝑥, 𝑦 − 1) − 2𝑓(𝑥, 𝑦)…..(8)

By substituting, Equations (7) and (8) in (6), we obtain the equation (9)

Δ 2𝑓(𝑥, 𝑦) = 𝑓(𝑥 + 1, 𝑦) + 𝑓(𝑥 − 1, 𝑦) + 𝑓(𝑥, 𝑦 + 1) + 𝑓(𝑥, 𝑦 − 1) − 4𝑓(𝑥, 𝑦) …(9)

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic black and white image with blocks

image\_size = (200, 200)

synthetic\_image = np.zeros(image\_size, dtype=np.uint8)

synthetic\_image[::20, ::20] = 255

synthetic\_image[10::20, 10::20] = 255

# Apply first-order derivatives (Sobel)

sobel\_x = cv2.Sobel(synthetic\_image, cv2.CV\_64F, 1, 0, ksize=3)

sobel\_y = cv2.Sobel(synthetic\_image, cv2.CV\_64F, 0, 1, ksize=3)

# Apply second-order derivative (Laplacian)

laplacian = cv2.Laplacian(synthetic\_image, cv2.CV\_64F)

# Apply Gaussian filters with different kernel sizes

kernel\_sizes = [3, 5, 9]

gaussian\_results = [cv2.GaussianBlur(synthetic\_image, (k, k), 0) for k in kernel\_sizes]

# Display all images in one figure

plt.figure(figsize=(12, 8))

# Original Image

plt.subplot(2, 3, 1)

plt.imshow(synthetic\_image, cmap='gray')

plt.title('Original Image')

# Sobel X

plt.subplot(2, 3, 2)

plt.imshow(sobel\_x, cmap='gray')

plt.title('Sobel X')

# Sobel Y

plt.subplot(2, 3, 3)

plt.imshow(sobel\_y, cmap='gray')

plt.title('Sobel Y')

# Laplacian

plt.subplot(2, 3, 4)

plt.imshow(laplacian, cmap='gray')

plt.title('Laplacian')

# Combined Gaussian Filters

combined\_gaussian = np.hstack(gaussian\_results)

plt.subplot(2, 3, 5)

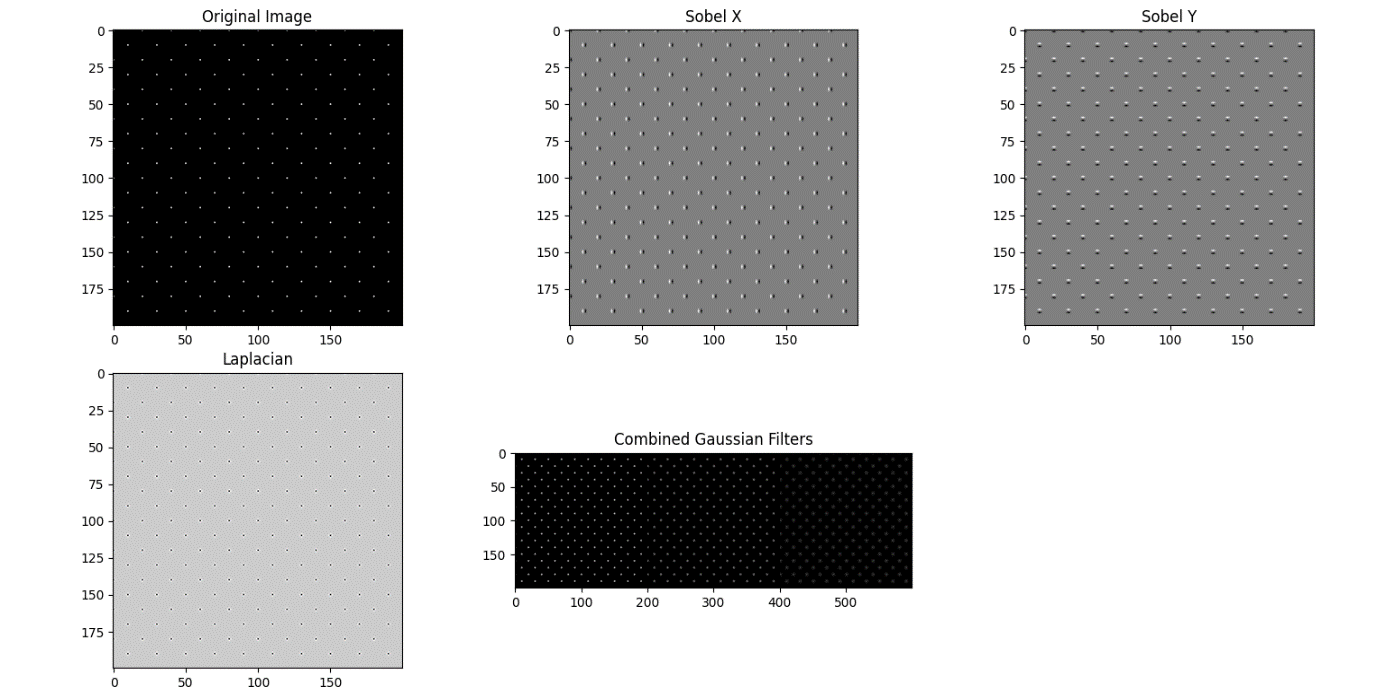
plt.imshow(combined\_gaussian, cmap='gray')

plt.title('Combined Gaussian Filters')

plt.tight\_layout()

plt.show()

**output:**

****