**EX.NO:6 DATE: 20.02.2025**

**CANNY EDGE DETECTION AND LoG & DoG**

**Aim:**  
To apply edge detection techniques including Gaussian Blur, Sobel Filter, Non-Maximum Suppression, Double Thresholding, Laplacian of Gaussian (LoG), and Difference of Gaussians (DoG).

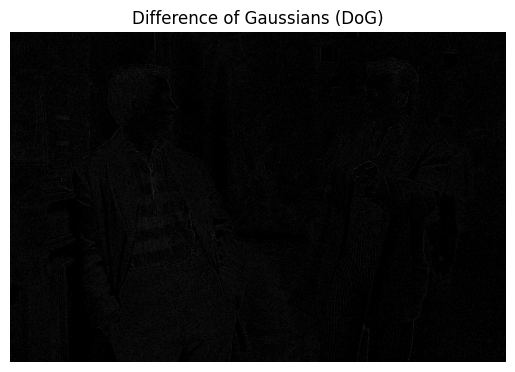
**Algorithm:**

1. Convert the image to grayscale.  
2. Apply Gaussian Blur for noise reduction.  
3. Compute image gradients using the Sobel operator.  
4. Apply Non-Maximum Suppression to refine edges.  
5. Use Double Thresholding to classify strong and weak edges.  
6. Compute the Laplacian of Gaussian (LoG) for edge detection.  
7. Compute the Difference of Gaussians (DoG) to enhance edges.  
8. Display the processed images.

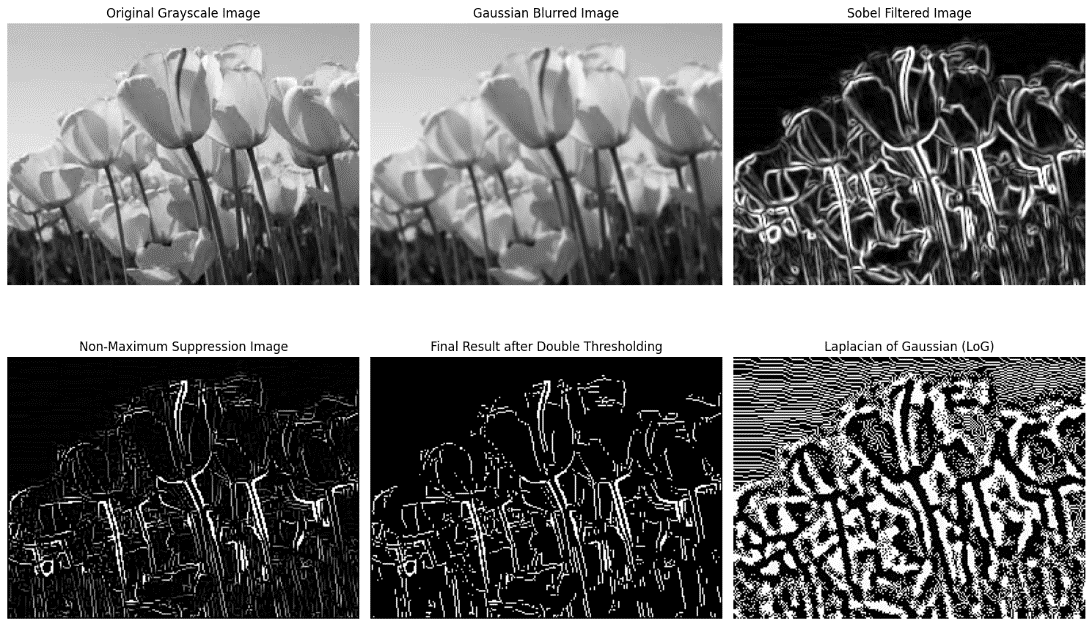
**Code**

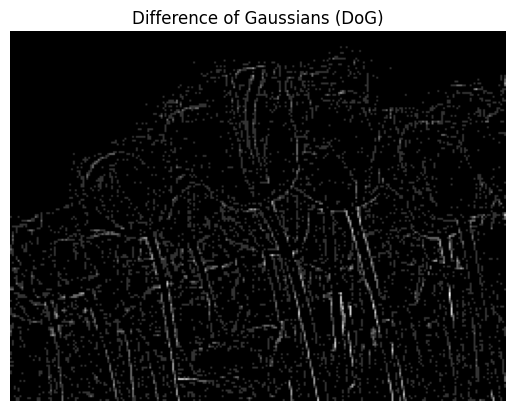
import cv2  
import matplotlib.pyplot as plt  
import numpy as np  
from scipy.ndimage import convolve  
  
def edge\_detection\_pipeline(gray\_image):  
 blurred\_gojo = cv2.GaussianBlur(gray\_image, (3, 3), 1)  
  
 def sobel\_filter(img):  
 kernelx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  
 kernely = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  
 gx = np.zeros\_like(img, dtype=np.int64)  
 gy = np.zeros\_like(img, dtype=np.int64)  
 padded\_img = cv2.copyMakeBorder(img, 1, 1, 1, 1, cv2.BORDER\_REFLECT)  
 for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 win = padded\_img[i:i+3, j:j+3]  
 gx[i, j] = np.sum(win \* kernelx)  
 gy[i, j] = np.sum(win \* kernely)  
 result\_img = np.hypot(gx, gy)  
 result\_img = np.clip(result\_img, 0, 255)  
 return result\_img.astype(np.uint8)  
  
 magnitude = sobel\_filter(blurred\_gojo)  
 grad\_x = np.zeros\_like(blurred\_gojo, dtype=np.float64)  
 grad\_y = np.zeros\_like(blurred\_gojo, dtype=np.float64)  
  
 padded\_img = cv2.copyMakeBorder(blurred\_gojo, 1, 1, 1, 1, cv2.BORDER\_REFLECT)  
 for i in range(blurred\_gojo.shape[0]):  
 for j in range(blurred\_gojo.shape[1]):  
 win = padded\_img[i:i+3, j:j+3]  
 grad\_x[i, j] = np.sum(win \* np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]))  
 grad\_y[i, j] = np.sum(win \* np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]))  
 direction = np.arctan2(grad\_y, grad\_x)  
 nms\_image = np.zeros\_like(magnitude)  
 angle = direction \* 180. / np.pi  
 angle[angle < 0] += 180  
  
 for i in range(1, magnitude.shape[0] - 1):  
 for j in range(1, magnitude.shape[1] - 1):  
 try:  
 q = 255  
 r = 255  
 if (0 <= angle[i, j] < 22.5) or (157.5 <= angle[i, j] <= 180):  
 q = magnitude[i, j + 1]  
 r = magnitude[i, j - 1]  
 elif 22.5 <= angle[i, j] < 67.5:  
 q = magnitude[i + 1, j - 1]  
 r = magnitude[i - 1, j + 1]  
 elif 67.5 <= angle[i, j] < 112.5:  
 q = magnitude[i + 1, j]  
 r = magnitude[i - 1, j]  
 elif 112.5 <= angle[i, j] < 157.5:  
 q = magnitude[i - 1, j - 1]  
 r = magnitude[i + 1, j + 1]  
 if (magnitude[i, j] >= q) and (magnitude[i, j] >= r):  
 nms\_image[i, j] = magnitude[i, j]  
 else:  
 nms\_image[i, j] = 0  
 except IndexError as e:  
 pass  
 strong\_edges = (nms\_image > 100)  
 weak\_edges = ((nms\_image >= 50) & (nms\_image <= 100))  
 result = np.zeros\_like(gray\_image)  
 strong\_i, strong\_j = np.where(strong\_edges)  
 zeros\_i, zeros\_j = np.where(nms\_image == 0)  
 result[strong\_i, strong\_j] = 255  
 result[zeros\_i, zeros\_j] = 0  
 def loG(image, sigma):  
 smoothed\_image = cv2.GaussianBlur(image, (0, 0), sigma)  
 lkernel = np.array([[0, 1, 0],  
 [1, -4, 1],  
 [0, 1, 0]])  
 result = convolve(smoothed\_image, lkernel)  
 return result.astype(np.uint8)  
  
 log\_image = loG(gray\_image, 2.5)  
 def doG(image, sigma):  
 sigma1 = sigma / np.sqrt(2)  
 sigma2 = np.sqrt(2) \* sigma  
 G1 = cv2.GaussianBlur(image, (3, 3), sigma1)  
 G2 = cv2.GaussianBlur(image, (3, 3), sigma2)  
 G = cv2.subtract(G1, G2)  
 return G.astype(np.uint8)  
  
 dog\_image = doG(gray\_image, 2)  
 return blurred\_gojo, magnitude, nms\_image, result, log\_image, dog\_image  
gojo = cv2.imread('edge.jpg')  
gojo = cv2.cvtColor(gojo, cv2.COLOR\_BGR2GRAY)  
blurred\_gojo, magnitude, nms\_image, result, log\_image, dog\_image = edge\_detection\_pipeline(gojo)  
fig, axes = plt.subplots(2, 3, figsize=(15, 10))  
axes[0, 0].imshow(gojo, cmap='gray')  
axes[0, 0].axis('off')  
axes[0, 0].set\_title('Original Grayscale Image')  
  
axes[0, 1].imshow(blurred\_gojo, cmap='gray')  
axes[0, 1].axis('off')  
axes[0, 1].set\_title('Gaussian Blurred Image')  
  
axes[0, 2].imshow(magnitude, cmap='gray')  
axes[0, 2].axis('off')  
axes[0, 2].set\_title('Sobel Filtered Image')  
  
axes[1, 0].imshow(nms\_image, cmap='gray')  
axes[1, 0].axis('off')  
axes[1, 0].set\_title('Non-Maximum Suppression Image')  
  
axes[1, 1].imshow(result, cmap='gray')  
axes[1, 1].axis('off')  
axes[1, 1].set\_title('Final Result after Double Thresholding')  
  
axes[1, 2].imshow(log\_image, cmap='gray')  
axes[1, 2].axis('off')  
axes[1, 2].set\_title('Laplacian of Gaussian (LoG)')  
  
plt.tight\_layout()  
plt.show()  
plt.figure()  
plt.imshow(dog\_image, cmap='gray')  
plt.axis('off')  
plt.title('Difference of Gaussians (DoG)')  
plt.show()





tulips = cv2.imread('download.jpeg') tulips = cv2.cvtColor(tulips, cv2.COLOR\_BGR2GRAY)  
  
blurred\_tulip, magnitude, nms\_image, result, log\_image, dog\_image = edge\_detection\_pipeline(tulips)  
  
fig, axes = plt.subplots(2, 3, figsize=(15, 10))  
  
axes[0, 0].imshow(tulips, cmap='gray')  
axes[0, 0].axis('off')  
axes[0, 0].set\_title('Original Grayscale Image')  
  
axes[0, 1].imshow(blurred\_tulip, cmap='gray')  
axes[0, 1].axis('off')  
axes[0, 1].set\_title('Gaussian Blurred Image')  
  
axes[0, 2].imshow(magnitude, cmap='gray')  
axes[0, 2].axis('off')  
axes[0, 2].set\_title('Sobel Filtered Image')  
  
axes[1, 0].imshow(nms\_image, cmap='gray')  
axes[1, 0].axis('off')  
axes[1, 0].set\_title('Non-Maximum Suppression Image')  
  
axes[1, 1].imshow(result, cmap='gray')  
axes[1, 1].axis('off')  
axes[1, 1].set\_title('Final Result after Double Thresholding')  
  
axes[1, 2].imshow(log\_image, cmap='gray')  
axes[1, 2].axis('off')  
axes[1, 2].set\_title('Laplacian of Gaussian (LoG)')  
  
plt.tight\_layout()  
plt.show()  
plt.figure()  
plt.imshow(dog\_image, cmap='gray')  
plt.axis('off')  
plt.title('Difference of Gaussians (DoG)')  
plt.show()





# Inference:

- Gaussian Blur reduces noise before edge detection.  
- Sobel Filter computes gradients in the x and y directions.  
- Non-Maximum Suppression refines edges by thinning them.  
- Double Thresholding classifies edges based on intensity.  
- LoG detects edges by applying a second derivative.  
- DoG highlights edges by subtracting blurred images.

**Result:**  
The applied edge detection techniques successfully enhance and detect edges in the input images, highlighting significant features and boundaries.