|  |  |
| --- | --- |
| **EXERCISE NO:6**  **DATE:10/02/2025** | **CANNY EDGE DETECTION AND**  **LOG AND DOG FILTERS** |

**AIM:**

to an image and applying edge detection and log filters and dog filters and various methods in a step-by-step process.

**HOUSE PIC**

**ALOGRITHM :**

1. Apply Filter
   * Algorithm: Apply a convolutional filter to an image.
2. Gradient Image
   * Algorithm: Compute the gradient magnitude of an image using Sobel operators.
3. Discrete Derivative
   * Algorithm: Compute the discrete derivative of an image using differences.
4. Gaussian Smoothing
   * Algorithm: Apply Gaussian smoothing to an image to reduce noise.
5. Edge Detection
   * Algorithm: Perform edge detection using Sobel, Roberts, and Canny operators.
6. Show Edge Detection
   * Algorithm: Visualize edge detection results for different methods.

Log filters and dog filters:

1. apply\_log\_filter:
   * Algorithm: First, apply a Gaussian blur to the image with a specified kernel size and sigma value. Then, apply a Laplacian filter to the blurred image to detect edges.
2. apply\_dog\_filter:
   * Algorithm: Apply two Gaussian blurs to the image with different sigma values. Subtract the second blurred image from the first to highlight edges and details.
3. display\_image:
   * Algorithm: Use Matplotlib to display an image with a title in grayscale.
4. main:
   * Algorithm: Load an image in grayscale, check for errors in loading, apply the specified filters, and display the filtered images along with relevant annotations.

**Function:**

1. apply\_filter(image, kernel): Applies a convolutional filter to an image.

2. gradient\_image(image): Computes the gradient magnitude of an image using Sobel operators.

3. discrete\_derivative(image): Calculates the discrete derivative of an image using differences.

4. gaussian\_smoothing(image, kernel\_size=5, sigma=1.0): Applies Gaussian smoothing to reduce noise in an image.

5. edge\_detection(image): Performs edge detection using Sobel, Roberts, and Canny operators.

6. show\_edge\_detection(image): Visualizes the results of various edge detection methods on an image.

7 apply\_log\_filter(image, sigma=1.0, ksize=(5, 5)): Applies a Laplacian of Gaussian (LoG) filter to an image.

8.apply\_dog\_filter(image, sigma1=1.0, sigma2=2.0, ksize=(5, 5)): Applies a Difference of Gaussian (DoG) filter to an image.

8. display\_image(image, title): Displays an image with a given title using Matplotlib.

9. main(): Reads an image, applies various filters, and displays the results.

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def apply\_filter(image, kernel):

k\_size = kernel.shape[0] # Get kernel size

pad = k\_size // 2

padded\_image = np.pad(image.astype(np.float32), pad, mode='edge')

filtered\_image = np.zeros\_like(image, dtype=np.float32)

for i in range(image.shape[0]):

for j in range(image.shape[1]):

region = padded\_image[i:i+k\_size, j:j+k\_size]

filtered\_image[i, j] = np.sum(region \* kernel)

return np.clip(filtered\_image, 0, 255).astype(np.uint8)

def gradient\_image(image):

Gx = np.array([[-1, 1]], dtype=np.float32)

Gy = np.array([[-1], [1]], dtype=np.float32)

grad\_x = apply\_filter(image, Gx)

grad\_y = apply\_filter(image, Gy)

gradient\_magnitude = np.sqrt(grad\_x.astype(np.float32)\*\*2 + grad\_y.astype(np.float32)\*\*2)

return np.clip(gradient\_magnitude, 0, 255).astype(np.uint8)

def discrete\_derivative(image):

dx = np.diff(image, axis=1, append=image[:, -1:])

dy = np.diff(image, axis=0, append=image[-1:, :])

gradient\_magnitude = np.sqrt(dx.astype(np.float32)\*\*2 + dy.astype(np.float32)\*\*2)

return np.clip(gradient\_magnitude, 0, 255).astype(np.uint8)

def gaussian\_smoothing(image, kernel\_size=5, sigma=1.0):

ax = np.linspace(-(kernel\_size // 2), kernel\_size // 2, kernel\_size)

gauss = np.exp(-0.5 \* np.square(ax) / np.square(sigma))

kernel = np.outer(gauss, gauss)

kernel /= np.sum(kernel)

return apply\_filter(image, kernel)

def edge\_detection(image):

sobel\_3x3 = sobel\_operator(image, 3)

sobel\_5x5 = sobel\_operator(image, 5)

roberts\_kernel\_x = np.array([[1, 0], [0, -1]], dtype=np.float32)

roberts\_kernel\_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

roberts\_x = apply\_filter(image, roberts\_kernel\_x)

roberts\_y = apply\_filter(image, roberts\_kernel\_y)

roberts = np.sqrt(roberts\_x.astype(np.float32)\*\*2 + roberts\_y.astype(np.float32)\*\*2)

canny = cv2.Canny(image.astype(np.uint8), 100, 200)

return sobel\_3x3, sobel\_5x5, np.clip(roberts, 0, 255).astype(np.uint8), canny

def show\_edge\_detection(image):

grad\_image = gradient\_image(image)

discrete\_deriv = discrete\_derivative(image)

smoothed\_image = gaussian\_smoothing(image)

sobel\_3x3, sobel\_5x5, roberts, canny = edge\_detection(image)

fig, axes = plt.subplots(2, 4, figsize=(15, 8))

axes[0, 0].imshow(image, cmap='gray')

axes[0, 0].set\_title("Original")

axes[0, 1].imshow(grad\_image, cmap='gray')

axes[0, 1].set\_title("Gradient Image")

axes[0, 2].imshow(discrete\_deriv, cmap='gray')

axes[0, 2].set\_title("Discrete Derivative")

axes[0, 3].imshow(smoothed\_image, cmap='gray')

axes[0, 3].set\_title("Gaussian Smoothing")

axes[1, 0].imshow(sobel\_3x3, cmap='gray')

axes[1, 0].set\_title("Sobel 3x3")

axes[1, 1].imshow(sobel\_5x5, cmap='gray')

axes[1, 1].set\_title("Sobel 5x5")

axes[1, 2].imshow(roberts, cmap='gray')

axes[1, 2].set\_title("Roberts")

axes[1, 3].imshow(canny, cmap='gray')

axes[1, 3].set\_title("Canny Edge Detection")

for ax in axes.ravel():

ax.axis("off")

plt.tight\_layout()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

image = cv2.imread("Downloads/naruto .jpeg", cv2.IMREAD\_GRAYSCALE)

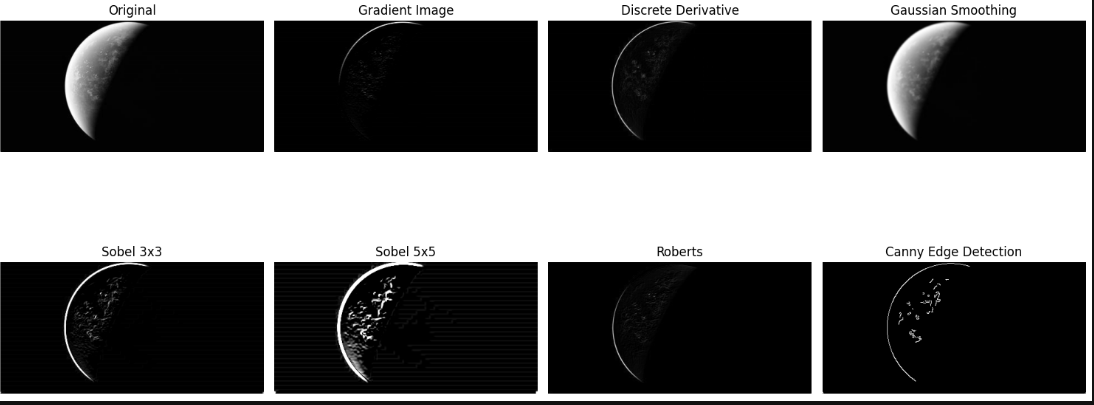
if image is None:

raise ValueError("Error loading image. Check file path.")

image = image.astype(np.float32) # Ensure image is float before processing

show\_edge\_detection(image)

**output:**



**Code;**

import cv2

import numpy as np

from matplotlib import pyplot as plt

def apply\_log\_filter(image, sigma=1.0, ksize=(5, 5)):

gaussian\_blur = cv2.GaussianBlur(image, ksize, sigma)

log = cv2.Laplacian(gaussian\_blur, cv2.CV\_64F)

return log

def apply\_dog\_filter(image, sigma1=1.0, sigma2=2.0, ksize=(5, 5)):

gaussian1 = cv2.GaussianBlur(image, ksize, sigma1)

gaussian2 = cv2.GaussianBlur(image, ksize, sigma2)

dog = gaussian1 - gaussian2

return dog

def display\_image(image, title):

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

plt.title(title)

plt.show()

def main():

pdf\_path = "Downloads/LOG\_DOG.pdf"

output\_directory = "Downloads"

# Read the input image

image\_path = "Downloads/bilioniare.jpg"

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

if image is None:

print("Error: Unable to read image.")

return

# Topics Covered

print("Lecture 11: LoG and DoG Filters")

# Today's Topics

print("Today's Topics:")

print("- LoG Filter")

print("- DoG Filter")

# Recall: First Derivative Filters

print("Recall: First Derivative Filters")

# First derivative in x direction

dx = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

display\_image(dx, 'First Derivative (dx)')

# Second-Derivative Filters

print("Second-Derivative Filters")

# Second derivative in x direction

dxx = cv2.Sobel(image, cv2.CV\_64F, 2, 0, ksize=3)

display\_image(dxx, 'Second Derivative (dxx)')

# Numerical Derivatives

print("Numerical Derivatives")

# Using central difference for second derivative

dxx\_num = cv2.Laplacian(image, cv2.CV\_64F)

display\_image(dxx\_num, 'Numerical Second Derivative')

# Example: Second Derivatives

print("Example: Second Derivatives")

# Applying second derivatives

dyy = cv2.Sobel(image, cv2.CV\_64F, 0, 2, ksize=3)

display\_image(dyy, 'Second Derivative (dyy)')

# Finding Zero-Crossings

print("Finding Zero-Crossings")

log\_filtered\_image = apply\_log\_filter(image, sigma=1.5, ksize=(7, 7))

zero\_crossings = np.where(np.abs(log\_filtered\_image) < 1e-5, 255, 0)

display\_image(zero\_crossings, 'Zero-Crossings')

# Edge Detection Summary

print("Edge Detection Summary")

edges = cv2.Canny(image, 100, 200)

display\_image(edges, 'Edge Detection')

# Finite Difference Laplacian

print("Finite Difference Laplacian")

laplacian = cv2.Laplacian(image, cv2.CV\_64F)

display\_image(laplacian, 'Laplacian')

# Example: Laplacian

print("Example: Laplacian")

# Apply Laplacian filter

laplacian\_filtered = cv2.Laplacian(image, cv2.CV\_64F)

display\_image(laplacian\_filtered, 'Laplacian Filtered')

# Notes about the Laplacian

print("Notes about the Laplacian")

# Display Laplacian details

print("Laplacian filter detects edges by calculating second-order derivatives.")

# LoG Filter

print("LoG Filter")

log\_filtered\_image = apply\_log\_filter(image, sigma=1.5, ksize=(7, 7))

display\_image(log\_filtered\_image, 'LoG Filter')

# 1D Gaussian and Derivatives

print("1D Gaussian and Derivatives")

x = np.linspace(-10, 10, 100)

gauss\_1d = np.exp(-x\*\*2 / 2)

plt.plot(x, gauss\_1d)

plt.title('1D Gaussian')

plt.show()

# Second Derivative of a Gaussian

print("Second Derivative of a Gaussian")

gauss\_2nd\_derivative = -x \* np.exp(-x\*\*2 / 2)

plt.plot(x, gauss\_2nd\_derivative)

plt.title('Second Derivative of a Gaussian')

plt.show()

# Effect of LoG Operator

print("Effect of LoG Operator")

display\_image(log\_filtered\_image, 'Effect of LoG Filter')

# Zero-Crossings as an Edge Detector

print("Zero-Crossings as an Edge Detector")

display\_image(zero\_crossings, 'Zero-Crossings Edge Detection')

# Note: Closed Contours

print("Note: Closed Contours")

print("Zero-crossings often form closed contours similar to topo maps.")

# Other uses of LoG: Blob Detection

print("Other uses of LoG: Blob Detection")

blobs = cv2.dilate(log\_filtered\_image, None)

display\_image(blobs, 'Blob Detection')

# Pause to Think for a Moment

print("Pause to Think for a Moment")

print("How can an edge finder also be used to find blobs in an image?")

# Example: LoG Extrema

print("Example: LoG Extrema")

extrema = cv2.erode(log\_filtered\_image, None)

display\_image(extrema, 'LoG Extrema')

# LoG Blob Finding

print("LoG Blob Finding")

log\_blobs = cv2.dilate(log\_filtered\_image, None)

display\_image(log\_blobs, 'LoG Blob Finding')

# Efficient Implementation Approximating LoG with DoG

print("Efficient Implementation Approximating LoG with DoG")

dog\_filtered\_image = apply\_dog\_filter(image, sigma1=1.0, sigma2=2.0, ksize=(7, 7))

display\_image(dog\_filtered\_image, 'DoG Filter')

# Efficient Implementation

print("Efficient Implementation")

print("Approximating LoG using DoG can save computational resources.")

# Back to Blob Detection

print("Back to Blob Detection")

display\_image(log\_blobs, 'LoG Blobs Revisited')

# Other uses of LoG: Blob Detection

print("Other uses of LoG: Blob Detection")

display\_image(blobs, 'Blob Detection')

# Other uses for LoG: Image Coding

print("Other uses for LoG: Image Coding")

compressed\_image = cv2.pyrDown(log\_filtered\_image)

display\_image(compressed\_image, 'Image Coding with LoG')

# Other uses for LoG: Image Coding

print("Other uses for LoG: Image Coding (Repeated)")

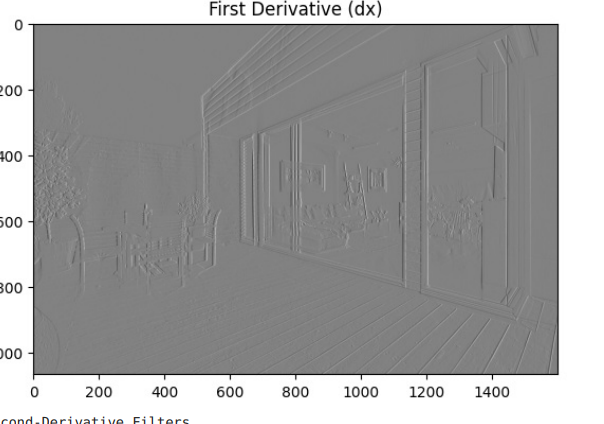
more\_compressed\_image = cv2.pyrDown(compressed\_image)

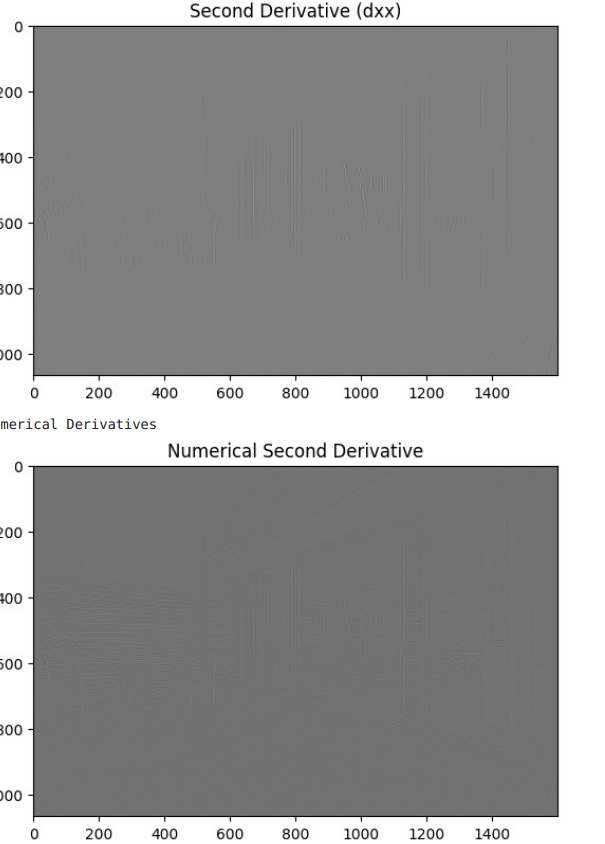
display\_image(more\_compressed\_image, 'Further Compressed Image')

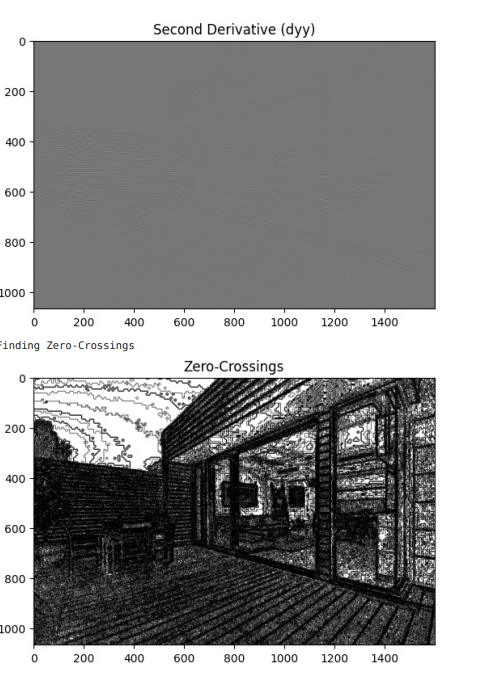
if \_\_name\_\_ == "\_\_main\_\_":

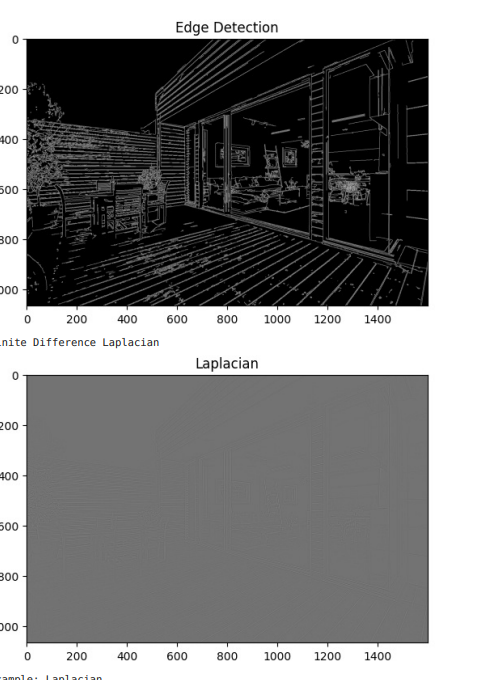
main()

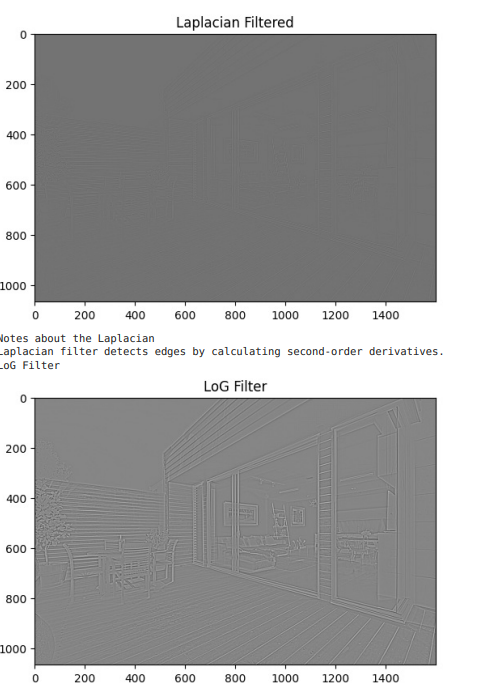
**output:**

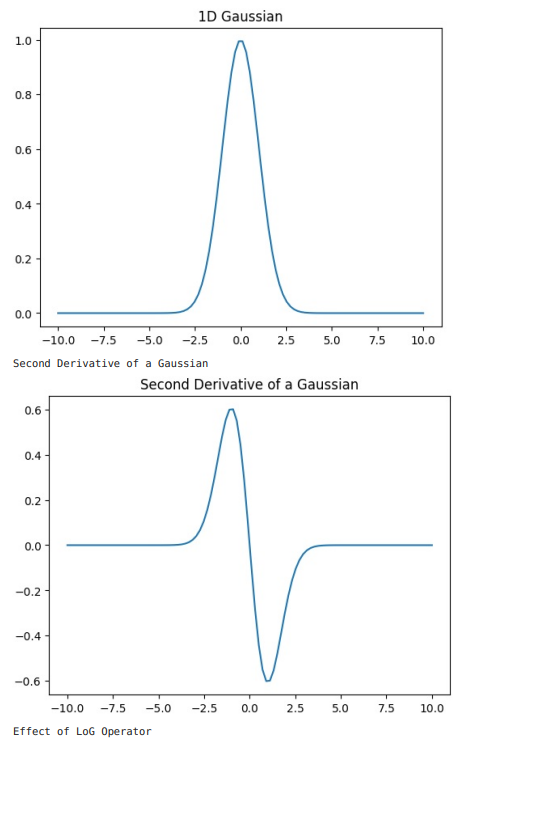
****

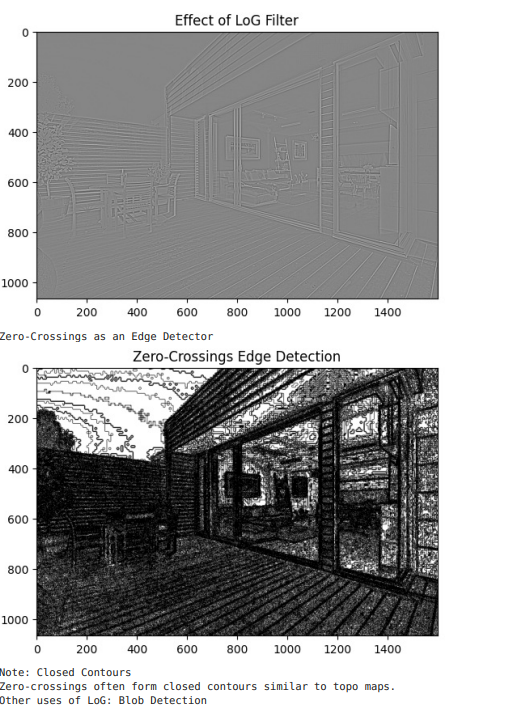
****

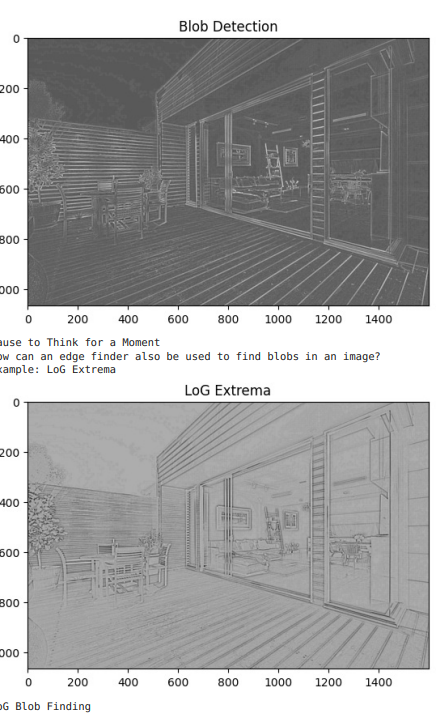
****

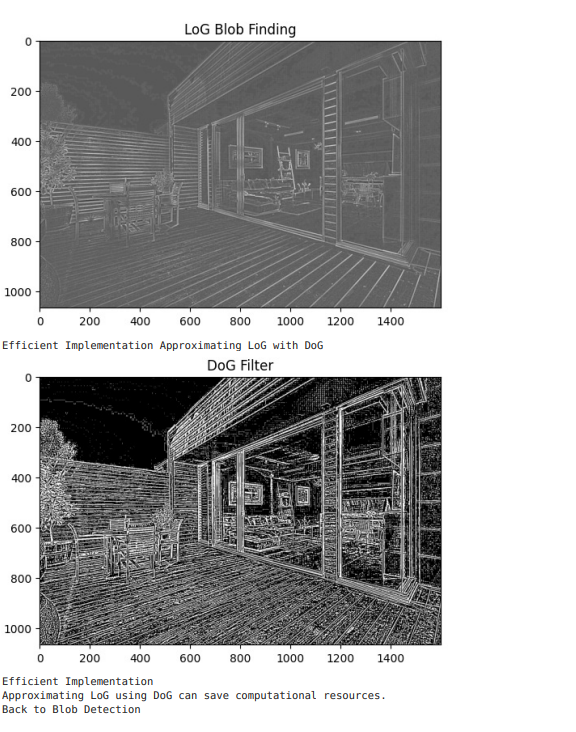
****

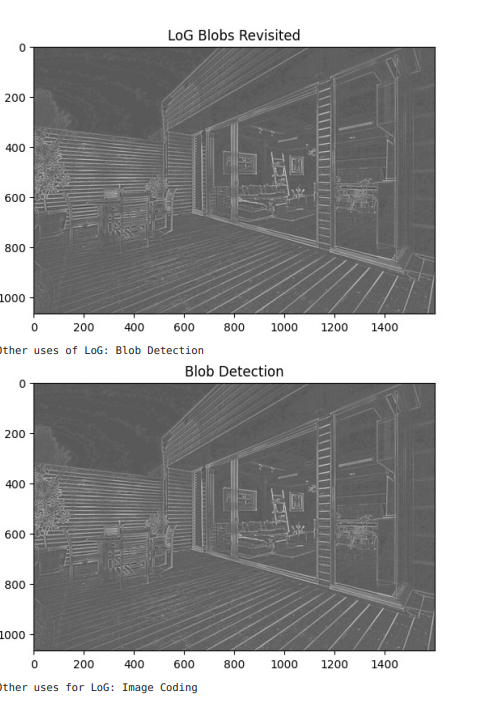
****

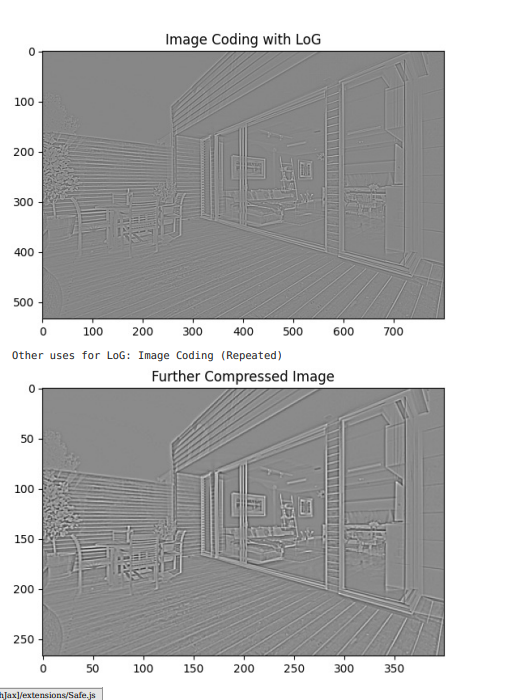
****

****

****

****

****

****

**RESULTS ;**

WE SUCCESSFULLY IMPLEMENTED AND EXCUTED OF THE CANNY EDGE DETECTION AND L0G AND DOG FILTERS: