|  |
| --- |
|  |
| Computer Vision  Assignment 01  BSAI-6A |
| |  |  |  | | --- | --- | --- | | **02-136221-034** | **Dawood Shahzad** | **Miss Reema Qaiser** | |

Contents

[**Introduction** 2](#_Toc180875915)

[**Task 1: Gaussian Pyramid and Sobel Edge Detection** 2](#_Toc180875916)

[1.1 Template: 2](#_Toc180875917)

[1.2 Cluttered: 2](#_Toc180875918)

[1.3 **Output:** 3](#_Toc180875919)

[1.4 **Code:** 3](#_Toc180875920)

[**Task 2: Laplacian Pyramid and Prewitt Edge Detection** 6](#_Toc180875921)

[2.1 Template: 6](#_Toc180875922)

[2.2 Cluttered: 6](#_Toc180875923)

[2.3 **Output:** 7](#_Toc180875924)

[2.4 **Code:** 8](#_Toc180875925)

[**Conclusion** 11](#_Toc180875926)

[**References** 11](#_Toc180875927)

# **Introduction**

I explored two different techniques for detecting specific objects in cluttered images, using a Gaussian pyramid with Sobel edge detection, and a Laplacian pyramid with Prewitt edge detection. The goal was to accurately locate a template object within a more complex, cluttered scene.

# **Task 1: Gaussian Pyramid and Sobel Edge Detection**

## 1.1 Template:

A close up of a goat's face

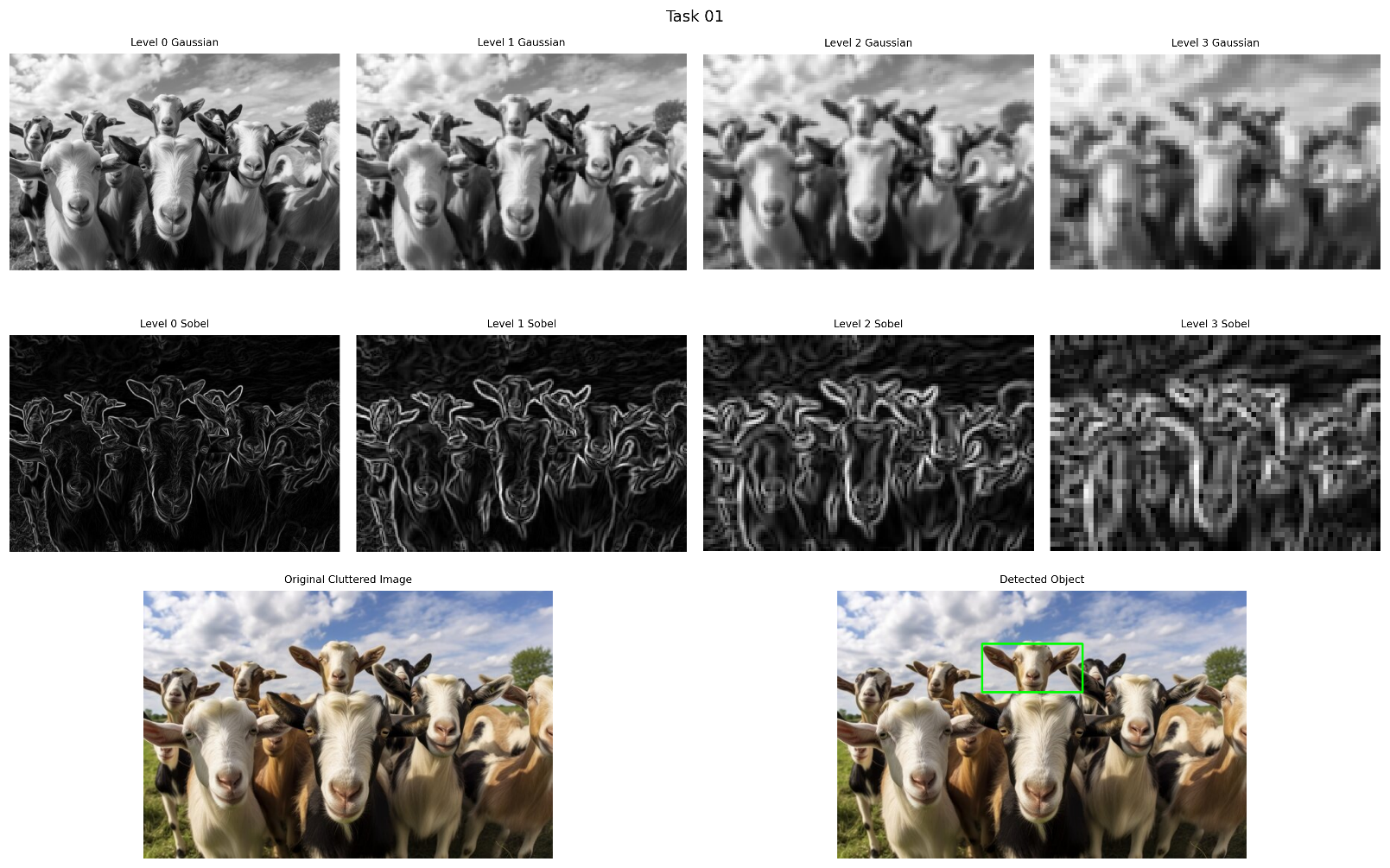
Description automatically generated

## 1.2 Cluttered:

A group of goats looking at the camera

Description automatically generated

## 1.3 **Output:**



## 1.4 **Code:**

import cv2

import matplotlib.pyplot as plt

def create\_gaussian\_pyramid(image, levels):

    """

    Create a Gaussian pyramid of the specified number of levels.

    """

    pyramid = [image]

    for \_ in range(levels - 1):

        pyramid.append(cv2.pyrDown(pyramid[-1]))

    return pyramid

def sobel\_edges(image):

    """

    Apply Sobel edge detection to an image. As mentioned in the slide.

    """

    sobel\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3) # Detect edges in x direction

    sobel\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3) # Detect edges in y direction

    # Combining the gradients

    magnitude = cv2.magnitude(sobel\_x, sobel\_y)

    return cv2.normalize(magnitude, None, 0, 255, cv2.NORM\_MINMAX).astype('uint8')

def template\_match\_pyramid\_visualize(cluttered\_path, template\_path, pyramid\_levels, match\_threshold):

    """

    Perform template matching with visualization of gaussian pyramid and Sobel levels along with the final result.

    """

    # Load images

    cluttered\_image = cv2.imread(cluttered\_path)

    template = cv2.imread(template\_path)

    final\_detection = cluttered\_image.copy()

    if cluttered\_image is None or template is None:

        raise ValueError("Failed to load one or both images")

    # Convert to grayscale

    gray\_cluttered = cv2.cvtColor(cluttered\_image, cv2.COLOR\_BGR2GRAY)

    gray\_template = cv2.cvtColor(template, cv2.COLOR\_BGR2GRAY)

    # Create pyramids

    pyramid\_original = create\_gaussian\_pyramid(gray\_cluttered, pyramid\_levels)

    pyramid\_edges = [sobel\_edges(level) for level in pyramid\_original]

    # Process template

    edge\_template = sobel\_edges(gray\_template)

    # Store results for visualization

    matching\_results = []

    # Process each pyramid level

    current\_template = edge\_template  # Start with the original edge template

    for i, edge\_level in enumerate(pyramid\_edges):

        if i > 0:

            # Downscale the template to match the current pyramid level

            current\_template = cv2.pyrDown(current\_template)

        # Template matching

        """Matching method: Correlation Coefficient Normalized(-1,1)"""

        result = cv2.matchTemplate(edge\_level, current\_template, cv2.TM\_CCOEFF\_NORMED)

        matching\_results.append(result)

        # Find best match location

        min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

        # Apply edited detection logic with max\_val threshold check

        if max\_val > match\_threshold:

            # Draw detection on the original image at the finest level

            h, w = current\_template.shape

            top\_left = max\_loc

            bottom\_right = (top\_left[0] + w, top\_left[1] + h)

            cv2.rectangle(final\_detection, top\_left, bottom\_right, (0, 255, 0), 2)

            break  # Exit loop once a match is found above the threshold

    # Create visualization

    fig = plt.figure(figsize=(15, 10))  # Adjusted figure size

    # Calculate grid parameters

    n\_rows = 3  # Gaussian, Sobel, plus row for the final detection

    n\_cols = pyramid\_levels

    # Create gridspec for better control of subplot layout

    gs = fig.add\_gridspec(n\_rows, n\_cols, hspace=0.05, wspace=0.05)

    # Plot pyramid levels (Gaussian and Sobel)

    for i in range(pyramid\_levels):

        # Original image at each level (Gaussian)

        ax = fig.add\_subplot(gs[0, i])

        ax.imshow(pyramid\_original[i], cmap='gray')

        ax.set\_title(f'Level {i} Gaussian', fontsize=8)

        ax.axis('off')

        # Edge detection results (Sobel)

        ax = fig.add\_subplot(gs[1, i])

        ax.imshow(pyramid\_edges[i], cmap='gray')

        ax.set\_title(f'Level {i} Sobel', fontsize=8)

        ax.axis('off')

    # Plot the original cluttered image in the first column of the last row

    ax = fig.add\_subplot(gs[2, :2])

    ax.imshow(cv2.cvtColor(cluttered\_image, cv2.COLOR\_BGR2RGB))

    ax.set\_title('Original Cluttered Image', fontsize=8)

    ax.axis('off')

    # Plot the final detected result in the second column of the last row

    ax = fig.add\_subplot(gs[2, 2:])

    ax.imshow(cv2.cvtColor(final\_detection, cv2.COLOR\_BGR2RGB))

    ax.set\_title('Detected Object', fontsize=8)

    ax.axis('off')

    # Add a main title

    plt.suptitle('Task 01', fontsize=12, y=0.92)

    # Adjust subplots to reduce extra padding around the figure

    plt.subplots\_adjust(left=0.01, right=0.99, top=0.9, bottom=0.01)

    return fig

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

    try:

        # Perform template matching with visualization

        fig = template\_match\_pyramid\_visualize(

            'cluttered.jpg',

            'template.png',

            pyramid\_levels=4,

            match\_threshold=0.7

        )

        # Show the plot

        plt.show()

        # Save the visualization with higher DPI for better quality

        fig.savefig('pyramid\_visualization.png', dpi=300, bbox\_inches='tight')

    except Exception as e:

        print(f"Error: {str(e)}")

# **Task 2: Laplacian Pyramid and Prewitt Edge Detection**

## 2.1 Template:



## 2.2 Cluttered:



## A collage of a group of cats wearing sunglasses Description automatically generated2.3 **Output:**

## 2.4 **Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def create\_laplacian\_pyramid(image, levels):

    """

    Create a Laplacian pyramid using Gaussian upsampling for the specified number of levels. As mentioned in the slide.

    """

    # Gaussian Pyramid

    gaussian\_pyramid = [image]

    for i in range(levels - 1):

        down = cv2.pyrDown(gaussian\_pyramid[-1])

        gaussian\_pyramid.append(down)

    # Laplacian Pyramid

    laplacian\_pyramid = []

    for i in range(levels - 1):

        # Upsample the next level in Gaussian Pyramid

        gaussian\_expanded = cv2.pyrUp(gaussian\_pyramid[i + 1], dstsize=(gaussian\_pyramid[i].shape[1], gaussian\_pyramid[i].shape[0]))

        # Laplacian = current Gaussian level - upsampled Gaussian level

        laplacian = cv2.subtract(gaussian\_pyramid[i], gaussian\_expanded)

        laplacian\_pyramid.append(laplacian)

    # Add the top level of the Gaussian pyramid to the Laplacian pyramid

    laplacian\_pyramid.append(gaussian\_pyramid[-1])

    return laplacian\_pyramid

def prewitt\_edge\_detection(image):

    """

    Apply Prewitt edge detection to an image.

    """

    # Applying prewitt operator manually

    kernel\_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype = int)

    kernel\_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype = int)

    #aplying filter and convert to float32

    edge\_x = cv2.filter2D(image, cv2.CV\_32F, kernel\_x)

    edge\_y = cv2.filter2D(image, cv2.CV\_32F, kernel\_y)

    #combining the edge\_x and edge\_y to get the final edge

    magnitude = cv2.magnitude(edge\_x, edge\_y)

    return cv2.normalize(magnitude, None, 0, 255, cv2.NORM\_MINMAX).astype('uint8')

def template\_match\_laplacian\_visualize(cluttered\_path, template\_path, pyramid\_levels, match\_threshold):

    """

    Perform template matchimg with visualization of Laplacian pyramid and prewitt levels, along with the final result.

    """

    # Load images

    cluttered\_image = cv2.imread(cluttered\_path)

    template = cv2.imread(template\_path)

    final\_detection = cluttered\_image.copy()

    if cluttered\_image is None or template is None:

        raise ValueError("Failed to load one or both images")

    # Convert to grayscale

    gray\_cluttered = cv2.cvtColor(cluttered\_image, cv2.COLOR\_BGR2GRAY)

    gray\_template = cv2.cvtColor(template, cv2.COLOR\_BGR2GRAY)

    # Create pyramids

    laplacian\_pyramid = create\_laplacian\_pyramid(gray\_cluttered, pyramid\_levels)

    prewitt\_edges = [prewitt\_edge\_detection(level) for level in laplacian\_pyramid]

    # Process template

    edge\_template = prewitt\_edge\_detection(gray\_template)

    # Store results for visualization

    matching\_results = []

    detected\_locations = []

    # Process each pyramid level

    current\_template = edge\_template  # Start with the original edge template

    for i, edge\_level in enumerate(prewitt\_edges):

        if i > 0:

            # Downscale the template to match the current pyramid level

            current\_template = cv2.pyrDown(current\_template)

        # Template matching

        """Matching method: Correlation Coefficient Normalized(-1,1)"""

        result = cv2.matchTemplate(edge\_level, current\_template, cv2.TM\_CCOEFF\_NORMED)

        matching\_results.append(result)

        # Find best match location

        min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

        detected\_locations.append((max\_loc, max\_val, current\_template.shape))

        # Apply edited detection logic with max\_val threshold check

        if max\_val > match\_threshold:

            # Draw detection on the original image at the finest level

            h, w = current\_template.shape

            top\_left = max\_loc

            bottom\_right = (top\_left[0] + w, top\_left[1] + h)

            cv2.rectangle(final\_detection, top\_left, bottom\_right, (0, 255, 0), 2)

            break  # Exit loop once a match is found above the threshold

    # Create visualization

    fig = plt.figure(figsize=(15, 10))  # Adjusted figure size

    # Calculate grid parameters

    n\_rows = 3  # Laplacian, Prewitt, plus row for the final detection

    n\_cols = pyramid\_levels

    # Create gridspec for better control of subplot layout

    gs = fig.add\_gridspec(n\_rows, n\_cols, hspace=0.05, wspace=0.05)

    # Plot pyramid levels (Laplacian and Prewitt)

    for i in range(pyramid\_levels):

        # Original image at each level (Laplacian)

        ax = fig.add\_subplot(gs[0, i])

        ax.imshow(laplacian\_pyramid[i], cmap='gray')

        ax.set\_title(f'Level {i} Laplacian', fontsize=8)

        ax.axis('off')

        # Edge detection results (Prewitt)

        ax = fig.add\_subplot(gs[1, i])

        ax.imshow(prewitt\_edges[i], cmap='gray')

        ax.set\_title(f'Level {i} Prewitt', fontsize=8)

        ax.axis('off')

    # Plot the original cluttered image in the first column of the last row

    ax = fig.add\_subplot(gs[2, :2])

    ax.imshow(cv2.cvtColor(cluttered\_image, cv2.COLOR\_BGR2RGB))

    ax.set\_title('Original Cluttered Image', fontsize=8)

    ax.axis('off')

    # Plot the final detected result in the second column of the last row

    ax = fig.add\_subplot(gs[2, 2:])

    ax.imshow(cv2.cvtColor(final\_detection, cv2.COLOR\_BGR2RGB))

    ax.set\_title('Detected Object', fontsize=8)

    ax.axis('off')

    # Add a main title

    plt.suptitle('Task 02', fontsize=12, y=0.92)

    # Adjust subplots to reduce extra padding around the figure

    plt.subplots\_adjust(left=0.01, right=0.99, top=0.9, bottom=0.01)

    return fig

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

    try:

        # Perform template matching with visualization

        fig = template\_match\_laplacian\_visualize(

            'cluttered2.jpg',

            'template2.png',

            pyramid\_levels=4,

            match\_threshold=0.6

        )

        # Show the plot

        plt.show()

        # Save the visualization with higher DPI for better quality

        fig.savefig('laplacian\_pyramid\_visualization.png', dpi=300, bbox\_inches='tight')

    except Exception as e:

        print(f"Error: {str(e)}")

# **Conclusion**

This assignment taught me about how we can find objects in complex images. The combination of image pyramids and edge detection really helps in making the search more accurate. If template has **multiple** occurrences, then Gaussian Pyramid with Sobel edge detection gave slightly better results for my specific images.

# **References**

* Lectures (mostly for techniques)
* [OpenCV Template Matching Documentation](https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html) (mostly for cv2)
* Chatbot (mostly for matplotlib)