Name: Phạm Lê Gia Kiệt **IRN:** 2131200037	**WEEK 8 - CSE457**

Final Review

Project 1: Image Enhancement and Filtering

Description:
Develop a tool that takes an input image and applies various enhancement techniques such as histogram equalization, noise reduction (using median filter), and edge enhancement.

Techniques Used:

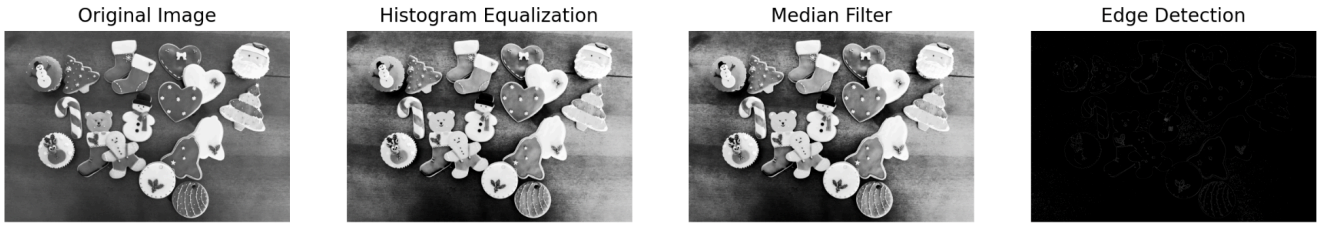
Histogram processing
Spatial filtering
Noise reduction algorithms

Steps:
Load an input image.
Implement histogram equalization to enhance contrast.
Apply a median filter to reduce noise.
Use edge detection algorithms to highlight edges.
Provide a comparison of the original and processed images.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    """ Load an image from a file path. """
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image not found or unable to load.")
    return image
```

```python
def histogram_equalization(image):
    """ Apply histogram equalization to enhance contrast. """
    return cv2.equalizeHist(image)

def median_filter(image, ksize=5):
    """ Apply median filter to reduce noise. """
    return cv2.medianBlur(image, ksize)

def edge_detection(image):
    """ Apply Canny edge detection to highlight edges. """
    edges = cv2.Canny(image, 100, 200)
    return edges

def plot_comparison(original, processed_images, titles):
    """ Plot the original and processed images for comparison. """
    plt.figure(figsize=(15, 5))

    plt.subplot(1, len(processed_images) + 1, 1)
    plt.imshow(original, cmap='gray')
    plt.title("Original Image")
    plt.axis('off')

    for i, (image, title) in enumerate(zip(processed_images, titles), start=2):
        plt.subplot(1, len(processed_images) + 1, i)
        plt.imshow(image, cmap='gray')
        plt.title(title)
        plt.axis('off')

    plt.show()

def main(image_path):
    original_image = load_image(image_path)

    # Step 1: Histogram Equalization
    hist_eq_image = histogram_equalization(original_image)

    # Step 2: Median Filter
    median_filtered_image = median_filter(hist_eq_image)

    # Step 3: Edge Detection
    edges_image = edge_detection(median_filtered_image)
```

```
    # Plot the comparison
    processed_images = [hist_eq_image, median_filtered_image, edges_image]
    titles = ["Histogram Equalization", "Median Filter", "Edge Detection"]
    plot_comparison(original_image, processed_images, titles)

if __name__ == "__main__":
    image_path = '1.jpg'  # Replace with your image path
    main(image_path)
```



Project 2: Image Segmentation

Description:
Create a program that segments an image into its constituent parts using different segmentation techniques like thresholding, region-based segmentation, and edge detection.

Techniques Used:

Thresholding
Region-based segmentation
Edge detection
Steps:

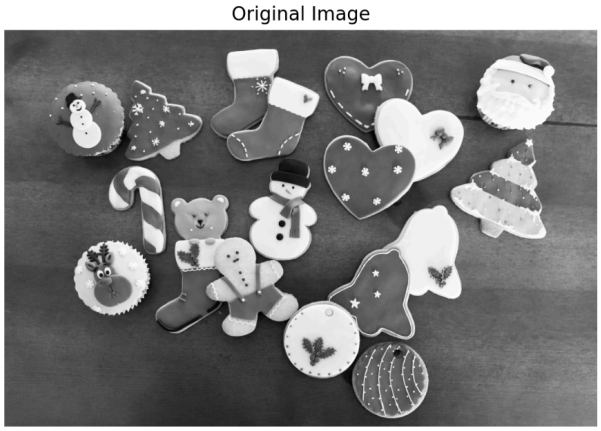
Load an input image.
Apply Otsu's thresholding method to separate foreground and background.
Implement region-based segmentation to identify distinct objects.
Use edge detection to highlight boundaries.
Visualize the segmented output.

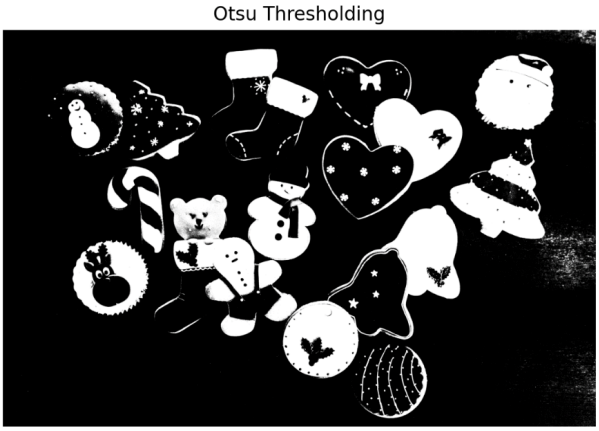
```
import cv2
```

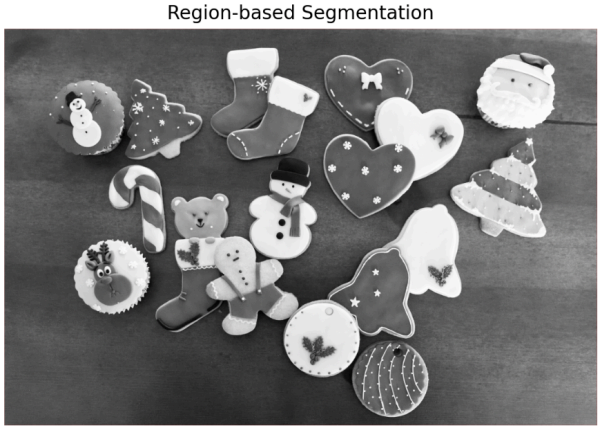
```
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    """Load an image from a file path."""
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image not found or unable to load.")
    return image

def otsu_thresholding(image):
    """Apply Otsu's thresholding method to separate foreground and background."""
    _, thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return thresh

def region_based_segmentation(image):
    """Implement region-based segmentation using watershed algorithm."""
    # Noise removal
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel, iterations=2)

    # Sure background area
    sure_bg = cv2.dilate(opening, kernel, iterations=3)

    # Finding sure foreground area
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    _, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

    # Finding unknown region
    sure_fg = np.uint8(sure_fg)
    unknown = cv2.subtract(sure_bg, sure_fg)

    # Marker labelling
    _, markers = cv2.connectedComponents(sure_fg)

    # Add one to all labels so that sure background is not 0, but 1
    markers = markers + 1

    # Now, mark the region of unknown with zero
    markers[unknown == 0] = 0
```

```
    image_color = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
    markers = cv2.watershed(image_color, markers)
    image_color[markers == -1] = [255, 0, 0]

    return image_color

def edge_detection(image):
    """Apply Canny edge detection to highlight edges."""
    edges = cv2.Canny(image, 100, 200)
    return edges

def plot_comparison(original, processed_images, titles):
    """Plot the original and processed images for comparison."""
    plt.figure(figsize=(15, 10))

    plt.subplot(2, 2, 1)
    plt.imshow(original, cmap='gray')
    plt.title("Original Image")
    plt.axis('off')

    for i, (image, title) in enumerate(zip(processed_images, titles), start=2):
        plt.subplot(2, 2, i)
        if len(image.shape) == 2:
            plt.imshow(image, cmap='gray')
        else:
            plt.imshow(image)
        plt.title(title)
        plt.axis('off')

    plt.show()

def main(image_path):
    original_image = load_image(image_path)

    # Step 1: Otsu's Thresholding
    otsu_image = otsu_thresholding(original_image)

    # Step 2: Region-based Segmentation
    region_segmented_image = region_based_segmentation(original_image)

    # Step 3: Edge Detection
```

```
    edges_image = edge_detection(original_image)

    # Plot the comparison
    processed_images = [otsu_image, region_segmented_image, edges_image]
    titles = ["Otsu Thresholding", "Region-based Segmentation", "Edge Detection"]
    plot_comparison(original_image, processed_images, titles)

if __name__ == "__main__":
    image_path = '1.jpg'  # Replace with your image path
    main(image_path)
```







Project 3: Morphological Image Processing

Description:
Build a project that applies morphological operations to an image to demonstrate effects like dilation, erosion, opening, and closing.

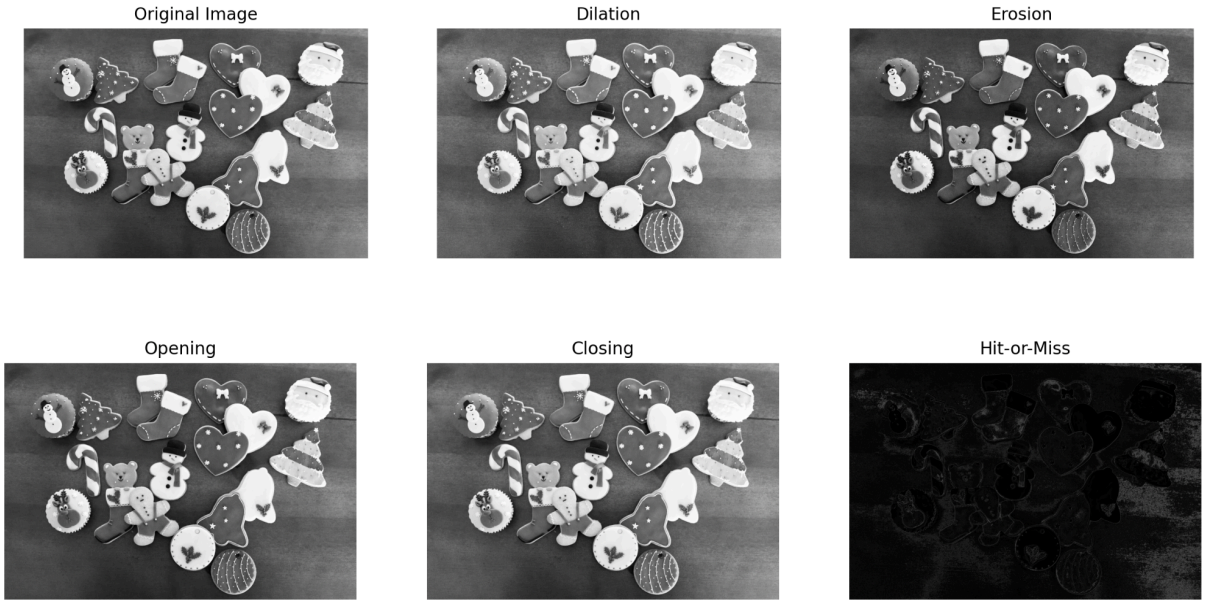
Techniques Used:
Dilation and erosion
Opening and closing
Hit-or-Miss transform
Steps:

Load an input image.
Implement dilation and erosion to understand their effects.
Use opening and closing to remove noise and small objects.
Apply the Hit-or-Miss transform to detect specific patterns.
Display the results of each morphological operation.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    """Load an image from a file path."""
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image not found or unable to load.")
    return image

def dilation(image, kernel_size=5):
    """Apply dilation to the image."""
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    return cv2.dilate(image, kernel, iterations=1)

def erosion(image, kernel_size=5):
    """Apply erosion to the image."""
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    return cv2.erode(image, kernel, iterations=1)

def opening(image, kernel_size=5):
```

```
    """Apply opening (erosion followed by dilation) to the image."""
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    return cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

def closing(image, kernel_size=5):
    """Apply closing (dilation followed by erosion) to the image."""
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    return cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)

def hit_or_miss(image):
    """Apply Hit-or-Miss transform to detect specific patterns."""
    # Define the structuring elements for hit-or-miss
    kernel1 = np.array([[1, 0, 0], [0, -1, 0], [0, 0, -1]], dtype=np.int32)
    kernel2 = np.array([[-1, 0, 0], [0, 1, 0], [0, 0, 1]], dtype=np.int32)

    hit_or_miss_image = cv2.morphologyEx(image, cv2.MORPH_HITMISS, kernel1)
    return hit_or_miss_image

def plot_comparison(original, processed_images, titles):
    """Plot the original and processed images for comparison."""
    plt.figure(figsize=(15, 10))

    plt.subplot(2, 3, 1)
    plt.imshow(original, cmap='gray')
    plt.title("Original Image")
    plt.axis('off')

    for i, (image, title) in enumerate(zip(processed_images, titles), start=2):
        plt.subplot(2, 3, i)
        plt.imshow(image, cmap='gray')
        plt.title(title)
        plt.axis('off')

    plt.show()

def main(image_path):
    original_image = load_image(image_path)

    # Step 1: Dilation
    dilation_image = dilation(original_image)
```

```
    # Step 2: Erosion
    erosion_image = erosion(original_image)

    # Step 3: Opening
    opening_image = opening(original_image)

    # Step 4: Closing
    closing_image = closing(original_image)

    # Step 5: Hit-or-Miss
    hit_or_miss_image = hit_or_miss(original_image)

    # Plot the comparison
    processed_images = [dilation_image, erosion_image, opening_image, closing_image,
hit_or_miss_image]
    titles = ["Dilation", "Erosion", "Opening", "Closing", "Hit-or-Miss"]
    plot_comparison(original_image, processed_images, titles)

if __name__ == "__main__":
    image_path = '1.jpg'  # Replace with your image path
    main(image_path)
```



Project 4: Color Image Processing

Description:
Develop a tool that processes color images, applying pseudocolor transformations and enhancing color features.

Techniques Used:

Pseudocolor image processing
Color enhancement techniques
Color space transformations

Steps:
Load a color image.
Convert the image to different color spaces (e.g., RGB to HSV).
Apply pseudocolor transformations to highlight specific features.
Enhance the color contrast using histogram equalization.
Display the original and processed images for comparison.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    """Load a color image from a file path."""
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError("Image not found or unable to load.")
    return image

def convert_color_spaces(image):
    """Convert the image to different color spaces (HSV, LAB)."""
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    return hsv_image, lab_image

def apply_pseudocolor(image):
    """Apply pseudocolor transformation to highlight specific features."""
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    pseudocolor = cv2.applyColorMap(grayscale, cv2.COLORMAP_JET)
    return pseudocolor
```

```
def enhance_color(image):
    """Enhance the color contrast using histogram equalization."""
    ycrcb_image = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
    channels = cv2.split(ycrcb_image)
    cv2.equalizeHist(channels[0], channels[0])
    merged = cv2.merge(channels)
    enhanced_image = cv2.cvtColor(merged, cv2.COLOR_YCrCb2BGR)
    return enhanced_image

def plot_comparison(original, processed_images, titles):
    """Plot the original and processed images for comparison."""
    plt.figure(figsize=(20, 10))

    plt.subplot(2, 3, 1)
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')

    for i, (image, title) in enumerate(zip(processed_images, titles), start=2):
        plt.subplot(2, 3, i)
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis('off')

    plt.show()

def main(image_path):
    original_image = load_image(image_path)

    # Step 1: Convert to different color spaces
    hsv_image, lab_image = convert_color_spaces(original_image)

    # Step 2: Apply pseudocolor transformation
    pseudocolor_image = apply_pseudocolor(original_image)

    # Step 3: Enhance color contrast
    enhanced_image = enhance_color(original_image)

    # Plot the comparison
    processed_images = [hsv_image, lab_image, pseudocolor_image, enhanced_image]
```

```
    titles = ["HSV Image", "LAB Image", "Pseudocolor Image", "Enhanced Color"]
    plot_comparison(original_image, processed_images, titles)

if __name__ == "__main__":
    image_path = '1.jpg'  # Replace with your image path
    main(image_path)
```

Project 5: Motion Detection and Segmentation

Description:
Create a motion detection system that segments moving objects in a video stream using techniques like frame differencing and background subtraction.

Techniques Used:

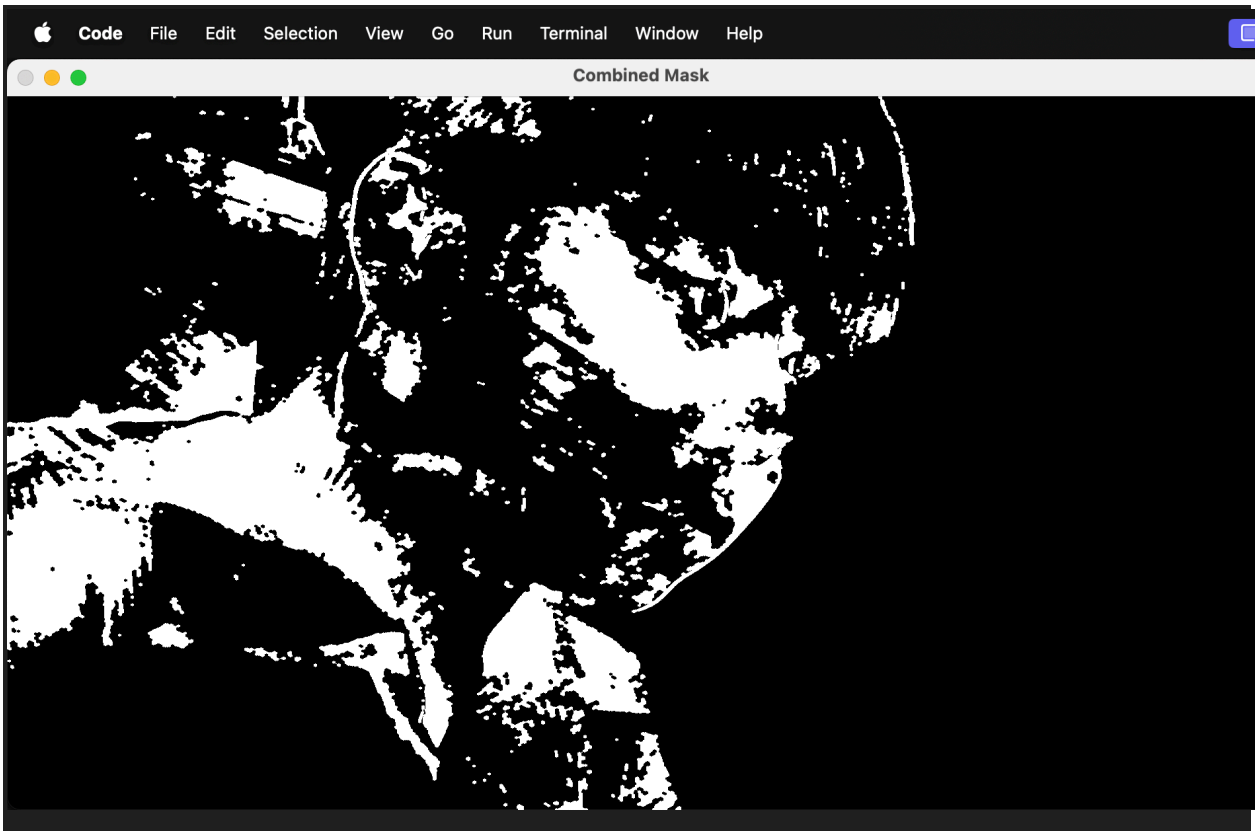
Frame differencing
Background subtraction
Motion-based segmentation

Steps:
Capture video from a camera or load a video file.
Implement frame differencing to detect motion.
Apply background subtraction to isolate moving objects.

Use morphological operations to clean up the segmented output.
Display the detected motion in real-time.

```
import cv2
import numpy as np

def frame_differencing(prev_frame, cur_frame):
    """Compute the absolute difference between two frames."""
    diff = cv2.absdiff(prev_frame, cur_frame)
    gray_diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray_diff, 30, 255, cv2.THRESH_BINARY)
    return thresh

def background_subtraction(bg_subtractor, frame):
    """Apply background subtraction to isolate moving objects."""
    fg_mask = bg_subtractor.apply(frame)
    _, fg_mask = cv2.threshold(fg_mask, 244, 255, cv2.THRESH_BINARY)
    return fg_mask

def morphological_operations(mask):
    """Use morphological operations to clean up the segmented output."""
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    cleaned_mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    cleaned_mask = cv2.morphologyEx(cleaned_mask, cv2.MORPH_CLOSE, kernel)
    return cleaned_mask

def main(video_path=None):
    if video_path:
        cap = cv2.VideoCapture(video_path)
    else:
        cap = cv2.VideoCapture(0)  # Use the camera

    if not cap.isOpened():
        print("Error: Could not open video.")
        return

    ret, prev_frame = cap.read()
    if not ret:
        print("Error: Could not read the first frame.")
        return
```

```
    bg_subtractor = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=50,
detectShadows=False)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Frame differencing
        diff_mask = frame_differencing(prev_frame, frame)

        # Background subtraction
        fg_mask = background_subtraction(bg_subtractor, frame)

        # Morphological operations
        cleaned_diff_mask = morphological_operations(diff_mask)
        cleaned_fg_mask = morphological_operations(fg_mask)

        # Combine masks for visualization
        combined_mask = cv2.bitwise_or(cleaned_diff_mask, cleaned_fg_mask)

        # Display results
        cv2.imshow('Original Frame', frame)
        cv2.imshow('Frame Differencing', cleaned_diff_mask)
        cv2.imshow('Background Subtraction', cleaned_fg_mask)
        cv2.imshow('Combined Mask', combined_mask)

        prev_frame = frame

        if cv2.waitKey(30) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    video_path = None  # Replace with your video path or None for webcam
    main(video_path)
```



Project 6: Feature Detection and Description

Description:
Build a project that detects and describes features in an image, such as edges, corners, and blobs, using different algorithms.

Techniques Used:

Edge detection
Corner detection (e.g., Harris corner detector)
Blob detection (e.g., Difference of Gaussians)

Steps:
Load an input image.
Apply edge detection algorithms like Canny to find edges.
Use the Harris corner detector to identify corners.
Implement blob detection to find blob-like structures.
Visualize the detected features on the image.

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    """Load an input image from a file path."""
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError("Image not found or unable to load.")
    return image

def edge_detection(image):
    """Apply Canny edge detection to find edges."""
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray_image, 100, 200)
    return edges

def corner_detection(image):
    """Use the Harris corner detector to identify corners."""
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_image = np.float32(gray_image)
    corners = cv2.cornerHarris(gray_image, 2, 3, 0.04)
    corners = cv2.dilate(corners, None)
    image[corners > 0.01 * corners.max()] = [0, 0, 255]
    return image

def blob_detection(image):
    """Implement blob detection to find blob-like structures."""
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    detector = cv2.SimpleBlobDetector_create()
    keypoints = detector.detect(gray_image)
    blob_image = cv2.drawKeypoints(image, keypoints, np.array([]), (0, 0, 255),
                                   cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return blob_image

def plot_comparison(original, processed_images, titles):
    """Plot the original and processed images for comparison."""
    plt.figure(figsize=(20, 10))

    plt.subplot(2, 2, 1)
```

```
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')

    for i, (image, title) in enumerate(zip(processed_images, titles), start=2):
        plt.subplot(2, 2, i)
        plt.imshow(image, cmap='gray' if title == "Edges" else None)
        plt.title(title)
        plt.axis('off')

    plt.show()

def main(image_path):
    original_image = load_image(image_path)

    # Step 1: Edge detection
    edges = edge_detection(original_image)

    # Step 2: Corner detection
    corners_image = corner_detection(original_image.copy())

    # Step 3: Blob detection
    blobs_image = blob_detection(original_image.copy())

    # Plot the comparison
    processed_images = [edges, corners_image, blobs_image]
    titles = ["Edges", "Corners", "Blobs"]
    plot_comparison(original_image, processed_images, titles)

if __name__ == "__main__":
    image_path = '1.jpg'  # Replace with your image path
    main(image_path)
```

Original Image

Edges

Corners

Blobs