Image Processing using Edge Detection and Segmentation in Python

Henri Setyo Pambudi

Overview

This Python code demonstrates various techniques in image processing, including edge detection and image segmentation. The code utilizes the OpenCV library to apply edge detection using the Roberts, Canny, and Sobel operators, and performs image segmentation using thresholding.

The core tasks performed are:

- Edge detection using the Roberts, Canny, and Sobel operators.
- Image segmentation using thresholding.
- Visualization of the results, including edge-detected images and segmented images.

Code Explanation

1. Importing Libraries

The following libraries are imported to perform image processing:

- **cv2**: For image manipulation, including reading and processing images.
- **numpy**: For numerical operations and matrix manipulations.
- **matplotlib**: For displaying images and plotting results.

Listing 1: Importing Required Libraries

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

2. Edge Detection

The code applies three types of edge detection techniques:

- **Roberts Operator**: A simple method for edge detection based on gradient approximation.
- **Canny Edge Detector**: A multi-step algorithm that finds edges by looking for areas of intensity contrast in the image.
- **Sobel Operator**: An edge detection technique that computes gradients in both x and y directions, combining them to get the overall edge strength.

Each of these methods is applied in the `edge_detection()` function.

Listing 2: Edge Detection Function

```
def edge_detection(image):

    edges_roberts = cv2.Canny(image, 50, 150)


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


    edges_sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize
       =5)
    edges_sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize
       =5)
    edges_sobel = np.sqrt(edges_sobel_x**2 + edges_sobel_y
       **2)

    return edges_roberts, edges_canny, edges_sobel
```

3. Image Segmentation

The code applies thresholding to convert the input image to a binary (black and white) image. This process is called segmentation, where the image is separated into regions based on pixel intensity.

The thresholding is performed in the `image_segmentation()` function, which converts the grayscale image into a binary image based on a threshold value of 128.

Listing 3: Image Segmentation Function

```
def image_segmentation(image):

    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


    _, binary_image = cv2.threshold(gray_image, 128, 255, cv2
        .THRESH_BINARY)

    return binary_image
```

4. Loading and Processing Multiple Images

The code loads three images from disk: `1.jpg`, `2.jpg`, and `3.jpg`. Each image is processed using the edge detection and segmentation functions.

If the image fails to load, an error message is printed. The edge detection results (for each method) and the segmented image are then visualized.

Listing 4: Loading and Processing Images

```
image1 = cv2.imread('1.jpg')
image2 = cv2.imread('2.jpg')
image3 = cv2.imread('3.jpg')


if image1 is None:
    print("Error: Unable to load image1")
if image2 is None:
    print("Error: Unable to load image2")
if image3 is None:
    print("Error: Unable to load image3")
```

```
edges1_roberts, edges1_canny, edges1_sobel = edge_detection(
    image1)
segmented_image1 = image_segmentation(image1)


edges2_roberts, edges2_canny, edges2_sobel = edge_detection(
    image2)
segmented_image2 = image_segmentation(image2)


edges3_roberts, edges3_canny, edges3_sobel = edge_detection(
    image3)
segmented_image3 = image_segmentation(image3)
```

5. Displaying the Results

The results are displayed using `matplotlib`, where:

- The first row shows the edge detection results using the Roberts, Canny, and Sobel operators.
- The second row shows the segmentation result.

The following Python code is used to plot the results:

Listing 5: Displaying Results

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


plt.subplot(3, 4, 1), plt.imshow(edges1_roberts, cmap='gray')
    , plt.title('Edges (Roberts) 1')
plt.subplot(3, 4, 2), plt.imshow(edges1_canny, cmap='gray'),
    plt.title('Edges (Canny) 1')
plt.subplot(3, 4, 3), plt.imshow(edges1_sobel, cmap='gray'),
    plt.title('Edges (Sobel) 1')
plt.subplot(3, 4, 4), plt.imshow(segmented_image1, cmap='gray
    '), plt.title('Segmentation 1')


plt.subplot(3, 4, 5), plt.imshow(edges2_roberts, cmap='gray')
    , plt.title('Edges (Roberts) 2')
```

```
plt.subplot(3, 4, 6), plt.imshow(edges2_canny, cmap='gray'),
    plt.title('Edges (Canny) 2')
plt.subplot(3, 4, 7), plt.imshow(edges2_sobel, cmap='gray'),
    plt.title('Edges (Sobel) 2')
plt.subplot(3, 4, 8), plt.imshow(segmented_image2, cmap='gray
    '), plt.title('Segmentation 2')

plt.subplot(3, 4, 9), plt.imshow(edges3_roberts, cmap='gray')
    , plt.title('Edges (Roberts) 3')
plt.subplot(3, 4, 10), plt.imshow(edges3_canny, cmap='gray'),
     plt.title('Edges (Canny) 3')
plt.subplot(3, 4, 11), plt.imshow(edges3_sobel, cmap='gray'),
     plt.title('Edges (Sobel) 3')
plt.subplot(3, 4, 12), plt.imshow(segmented_image3, cmap='
    gray'), plt.title('Segmentation 3')

plt.tight_layout()
plt.show()
```

Key Insights

- **Roberts Operator**: Provides a simple and fast method for detecting edges, but can be noisy for complex images.
- **Canny Edge Detection**: Offers more accurate results by reducing noise and detecting finer details, making it ideal for most edge detection tasks.
- **Sobel Operator**: Computes gradient magnitudes and is effective for detecting edges in both horizontal and vertical directions.
- **Segmentation**: Thresholding is a simple and effective method for segmenting an image, particularly when there is a clear distinction between objects and the background.