VR Graded Assignment - 1

Part 1: Use computer vision techniques to detect, segment, and count coins from an image containing scattered Indian coins.

Part 2: Create a stitched panorama from multiple overlapping images.

Pranav Laddhad

IMT2022074

Part 1

Introduction

In this part of the assignment, we had to follow these steps:

a. Detect all coins in the image

- Use edge detection techniques to detect all coins in the image.
- Visualize the detected coins by outlining them in the image.

b. Segmentation of Each Coin

- Apply region-based segmentation techniques to isolate individual coins from the image.
- Provide segmented outputs for each detected coin.

c. Count the Total Number of Coins

- Write a function to count the total number of coins detected in the image.
- Display the final count as an output.

Processed images after each step

This section displays the images and the corresponding changes made at each step of the process.

Example 1:

Figure 1: Original image containing coins

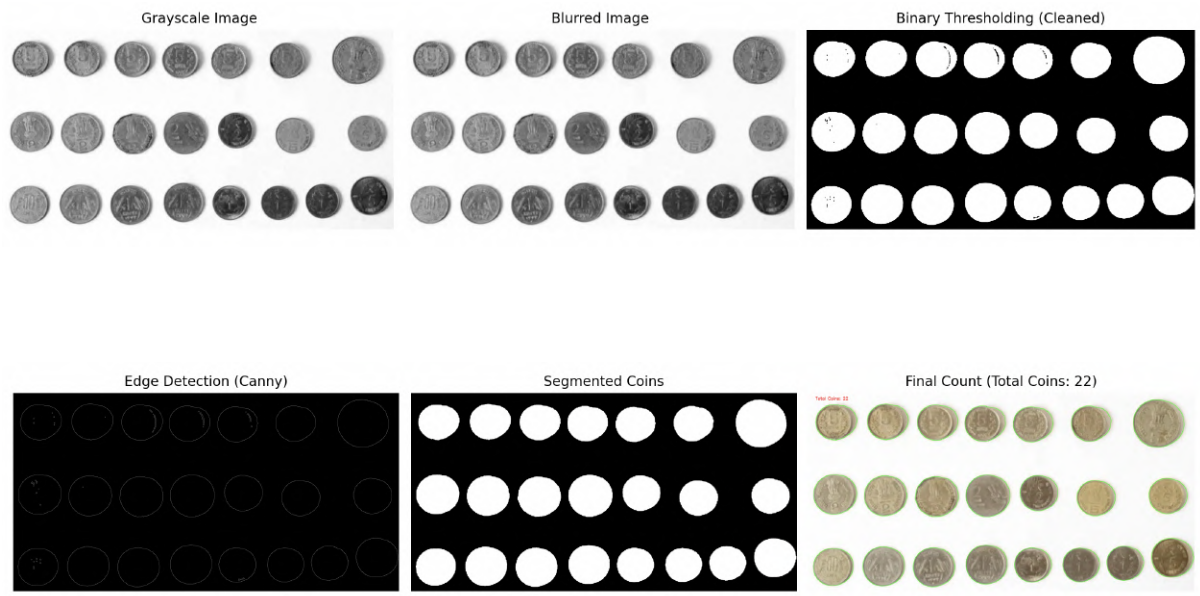

Figure 2: Images showing the preprocessing and the final coin detection

As it can be seen here, the correct number of coins (i.e. 22) is being detected.

Example 2:

Figure 3: Original image containing coins)

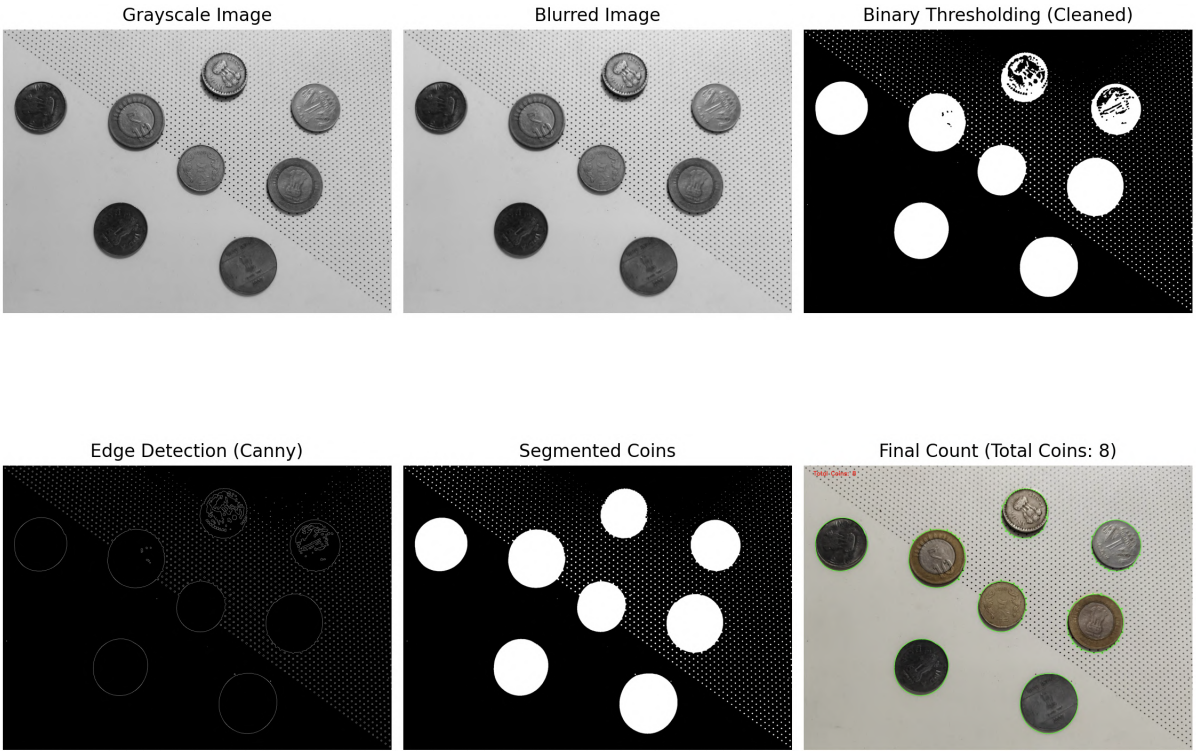

Figure 4: Images showing the preprocessing and the final coin detection

As it can be seen here, the correct number of coins (i.e. 8) is being detected.

Brief explanation of each step

1. Loading the original image

The input image containing coins for which is loaded using OpenCV's cv2.imread().

2. Converting to grayscale

Since color is not required for coin detection, the image is converted to grayscale using cv2.cvtColor().This step simplifies the processing by reducing the color channels from three (RGB) to one (grayscale).

3. Blurring the image to reduce noise

A Gaussian Blur is applied using cv2.GaussianBlur() to remove minor variations and smooth out the image.This helps in better edge detection by reducing unwanted noise. We have used 5×5 kernel size and standard deviation of 0. The idea for this was to remove minor noise while preserving coin boundaries for better edge detection.

4. Edge detection using canny edge detector

The Canny edge detection technique is applied to detect the edges of the coins which helps in distinguishing the coin boundaries from the background. For this the thresholds

(120, 250) were used to define the sensitivity of edge detection. Edges with intensity gradients below 250 were ignored. The idea was to come up with threshold for ensuring sharp coin boundaries while filtering out weaker edges caused by noise.

5. Binary thresholding

Binary thresholding is applied using cv2.threshold() with Otsu's binarization. This converts the image into a black-and-white format, where coins appear as white objects on a black background. So, with the range (0, 255), coins appear as white objects on a black background.

6. Segmentation to isolate individual coins

Contours are detected in the thresholded image using `cv2.findContours()`. `cv2.RETR_EXTERNAL` extracts only external contours, ignoring nested contours inside coins. `cv2.CHAIN_APPROX_SIMPLE` compresses contour points for efficiency. These contours are drawn on a blank image to create a segmented version.

7. Counting the number of coins

The contours found in segmentation are filtered based on their area (cv2.contourArea()). Small unwanted contours (e.g., noise) are ignored. Contours with an area less than 500 are ignored, ensuring only significant objects (coins) are counted. This threshold was decided after trial and error. The remaining contours are drawn on the original image, and the total count is displayed.

Part 2

Introduction

In this part of the assignment, the goal is to create a stitched panorama from multiple overlapping images.

We had to follow these steps:

a. Extract Key Points

- Detect key points in overlapping images.

b. Image Stitching

- Use the extracted key points to align and stitch the images into a single panorama.
- Provide the final panorama image as output.

Input images

Figure 5: Input image (Left)

Figure 6: Input image (Middle)

Figure 7: Input image (Right)

Input images with key points

Figure 8: Input image (Left)

Figure 9: Input image (Middle)

Figure 10: Input image (Right)

Figure 11: Final stitched image (Panorama)

Brief explanation of each step

1. Loading the Input Images

The program takes three overlapping images.

2. Converting to grayscale

Since feature detection works best on grayscale images, each image is converted to grayscale

3. Detecting Keypoints using SIFT

Scale-Invariant Feature Transform (SIFT) is used to detect and extract keypoints. The detected keypoints are then drawn as small red dots.

4. Image Stitching using OpenCV's Stitcher

We have used OpenCV's in-built Stitcher class that automates the entire stitching process.