International Institute of Information Technology
Bangalore

Visual Recognition
AIM 825

Graded Assignment 1 Report

Submitted by:

Abhinav Deshpande (IMT2022580)

March 23, 2025

1 Coin Detection, Segmentation, and Counting

1.1 Introduction

The goal of this part is to detect, segment, and count Indian coins in an image using computer vision techniques. The process consists of three stages: detecting coins using edge detection, segmenting the detected coins, and counting them accurately. A key challenge in this task is handling overlapping coins, which requires advanced segmentation techniques.

1.2 Methodology

1.2.1 Edge Detection and Initial Observations

To detect the coins, edge detection techniques were applied to extract the outer boundaries. The Canny edge detector was initially used, but the raw edge-detected image contained significant noise.

Figure 1: The input image of coins

Figure 2: Initial Canny Edge Detection Output

The above image shows the raw output of Canny edge detection. While some coin edges are detected, there is considerable noise and broken edges, making further processing difficult. The next step was to refine the edge detection process.

1.2.2 Enhancing Edge Detection with Preprocessing

To improve edge detection, Gaussian blurring was applied before Canny edge detection. This helped smooth the image, reducing high-frequency noise while preserving the coin edges.

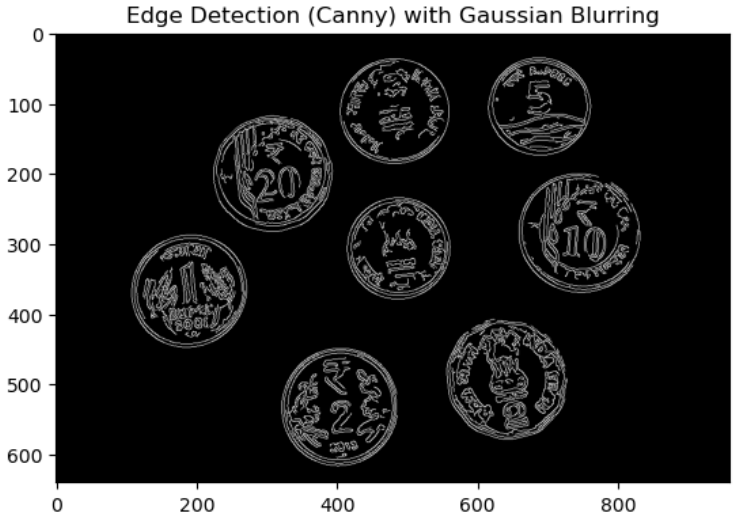


Figure 3: Canny Edge Detection After Gaussian Blurring

By applying Gaussian blurring before edge detection, the noise was significantly reduced, and the contours of the coins became more defined. However, some gaps were still present in the detected edges, requiring further refinement.

1.2.3 Improving Edge Continuity with Morphological Operations

Morphological closing was applied to fill gaps in the detected edges and improve edge continuity.

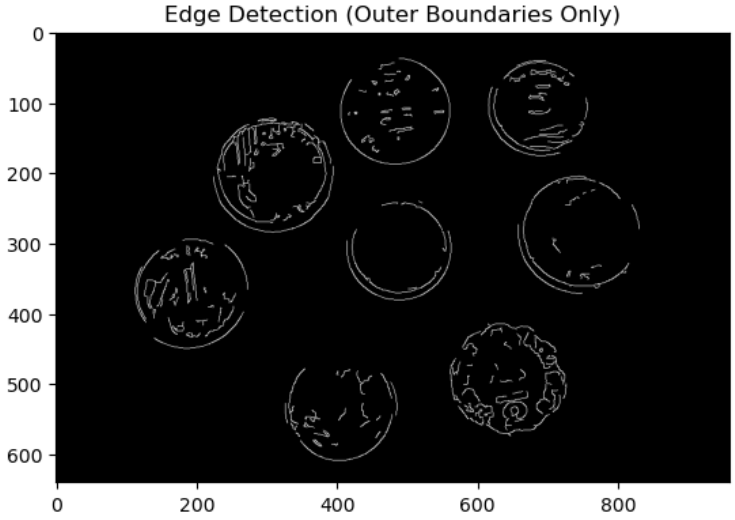


Figure 4: Canny Edge Detection After Morphological Closing

The results show that morphological closing helped in connecting broken edges, making the contours more complete. This was crucial for accurate segmentation in the next stage.

1.2.4 Coin Segmentation

Once the edges were detected, segmentation was performed using contour extraction. The contours of the coins were detected using OpenCV's `findContours` function, which identifies continuous boundaries of objects in a binary image. This approach works well for separating non-overlapping coins, where each coin can be individually identified based on its contour.

The contours were filtered based on their area and circularity to exclude noise and irrelevant objects. A circularity metric was computed using the formula:

$$\text{Circularity} = \frac{4\pi \times \text{Area}}{\text{Perimeter}^2} \tag{1}$$

Contours with circularity values close to 1 were more likely to be actual coins, while irregular shapes were discarded. This helped refine the segmentation and remove unwanted artifacts.

Figure 5: Segmented outputs on original image for each detected coin.

Figure 6: Bounding boxes drawn on original image around each detected coin

However, when the coins were overlapping, this method struggled to differentiate individual coins. The contours of touching coins often merged into a single contour, leading to under-segmentation. To handle this issue, more advanced techniques such as the Watershed algorithm were explored.

For non-overlapping coins, contour detection was effective. However, for overlapping coins, standard contour-based methods failed to distinguish between individual coins.

1.2.5 Enhanced Contour Detection Using the Watershed Algorithm

Traditional contour detection methods, such as OpenCV's `findContours`, work well for clearly separated objects but often struggle with touching or overlapping objects. The **Watershed algorithm** provides a more advanced approach to contour detection by incorporating **region-based segmentation**, making it more robust than simple edge-based methods.

Figure 7: Input image of Overlapping coins

The Watershed algorithm treats an image as a **topographic surface**, where pixel intensities represent elevation. The segmentation process follows these steps:

1. **Preprocessing:** The image is converted to grayscale, and thresholding is applied to create a binary mask. Morphological operations, such as opening, are performed to reduce noise and improve object boundaries.

Figure 8: Binary Image after Thresholding

2. **Identifying Foreground and Background:** The **distance transform** helps estimate object centers (sure foreground), while dilation is used to identify the sure background. The unknown region is computed by subtracting these two.

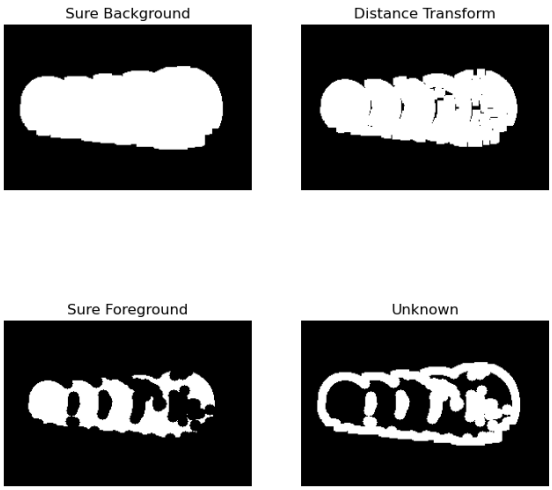


Figure 9: Distance Transform for Foreground Estimation

3. **Applying the Watershed Algorithm:** Markers are assigned to the foreground and background, and the Watershed algorithm floods the image from these markers. The boundaries where regions merge become **watershed lines**, effectively separating the objects.

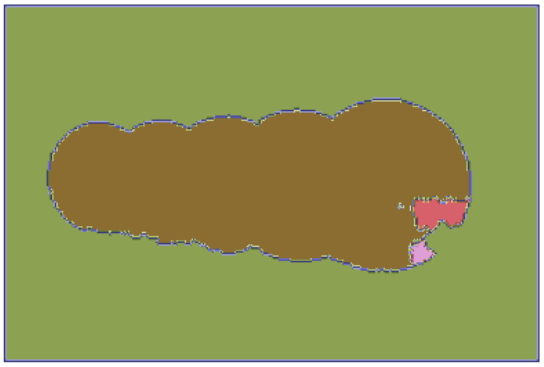


Figure 10: Final Segmentation using Watershed Algorithm

Figure 11: Outlining the coins on the original image

Unlike basic contour detection, which only identifies object edges, the **Watershed algorithm** provides a **structured and reliable contour detection method**. It incorporates **morphological transformations, distance-based region growing, and marker-based segmentation**, ensuring better object separation and more accurate contour extraction.

The results demonstrate that the Watershed algorithm is superior to traditional contour detection methods. While conventional methods may merge touching objects into a single contour, the Watershed algorithm successfully segments them into distinct regions, improving both detection accuracy and object counting.

Figure 12: Final Contours Extracted after Watershed Segmentation

1.2.6 Counting the Coins

After segmentation, the total number of coins was counted using a combination of contour-based counting and the Hough Circle Transform. This dual approach helped ensure robust detection, reducing errors caused by overlapping coins or partial visibility.

Hough Circle Transform for Coin Detection The Hough Circle Transform is a feature extraction technique that detects circular objects in an image using gradient information. The following parameters were used:

- **dp = 1.2** (Inverse ratio of the accumulator resolution to the image resolution)
- **minDist = 150** (Minimum distance between detected circle centers)
- **param1 = 50** (Higher threshold for edge detection)
- **param2 = 150** (Threshold for center detection)
- **minRadius = 20, maxRadius = 100** (Range of possible circle radii)

Contour-Based Coin Counting Alongside the Hough Circle Transform, contours were extracted using OpenCV's `findContours` function. The number of detected contours was counted and compared against the Hough Circle Transform results. Both methods provided an **identical count**, confirming the reliability of the approach.

Figure 13: Count of coins in **overlapping** coins image using watershed algorithm

Figure 14: Coin counting using hough transform in **non-overlapping** coin image

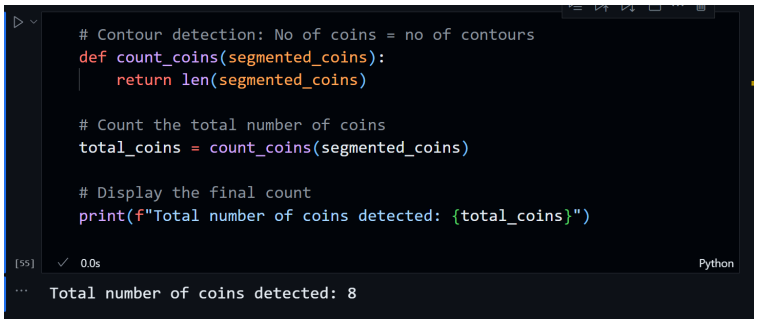


Figure 15: Coin counting using contour detection in **non-overlapping** coin image

The combination of these two methods strengthened the accuracy of the coin detection process. While the Hough Circle Transform was effective in identifying well-formed circular structures, contour-based counting ensured robustness against variations in shape, lighting, and minor occlusions. The consistency between the two techniques validated the correctness of the coin count.

2 Image Stitching for Panorama Creation

2.1 Introduction

This section describes the process of creating a stitched panorama from multiple overlapping images using feature-based image alignment techniques. The goal is to extract key points, match them, compute homographies, and blend the images seamlessly into a single panoramic view.

2.2 Methodology

2.2.1 Key Point Detection

Feature detection is the first step in image stitching. We use the Scale-Invariant Feature Transform (SIFT) algorithm to detect key points from the input images. SIFT provides robustness against scale, rotation, and illumination changes, making it ideal for panorama generation.

(a) SIFT keypoints detected in the left image.

(b) SIFT keypoints detected in the right image.

Figure 16: Key points detected in two input images using SIFT.

2.2.2 Feature Matching

Once key points are extracted, feature matching is performed using a Flann-based matcher. A ratio test is applied to retain only strong matches, ensuring reliable homography estimation.

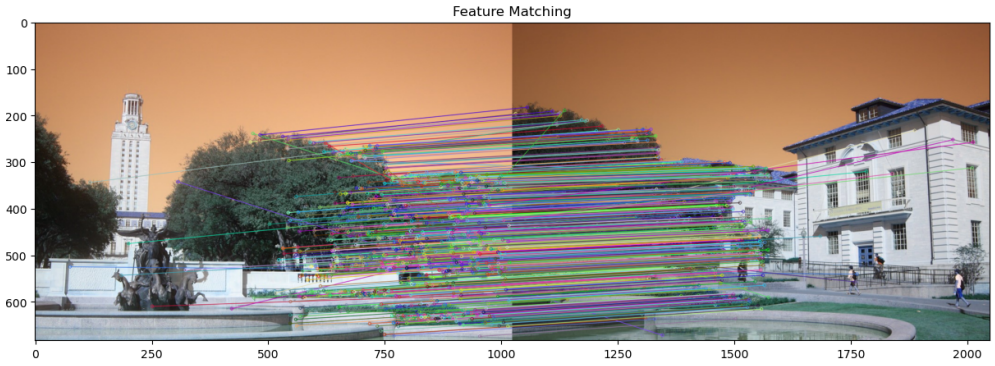
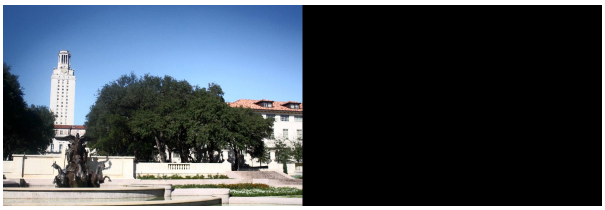


Figure 17: Feature matching between two overlapping images.

2.2.3 Homography Estimation

With the matched key points, we compute a homography transformation using RANSAC (Random Sample Consensus). The homography matrix is then used to warp images into a common coordinate space for alignment.



(a) Warped Image 1

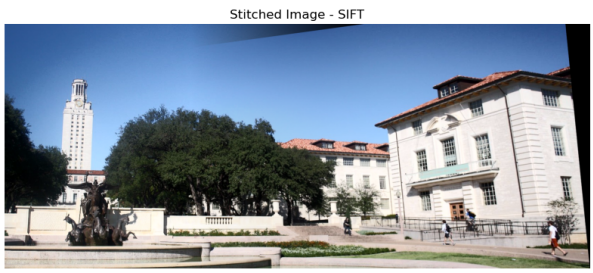
(b) Warped Image 2

Figure 18: Warped images using estimated homography.

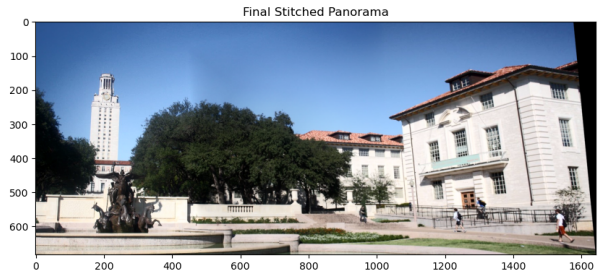
2.2.4 Blending and Panorama Generation

To create a seamless panorama, a smooth transition between overlapping image regions is necessary. We employ:

- **Feathering Blending (Gradient Mask Blending):** A weighted blending technique where a smoothing mask is applied to the overlapping region. The transition between images is gradual, reducing visible seams. This method is computationally efficient, making it well-suited for high-resolution images.
- **Weighted Blending (Distance Transform):** A more advanced technique that assigns pixel-wise weights based on distance transforms. While effective at minimizing artifacts, it is computationally expensive, especially for large images.



(a) Feathering Blending



(b) Weighted Blending (Distance Transform)

Figure 19: Comparison of Feathering and Weighted Blending techniques.

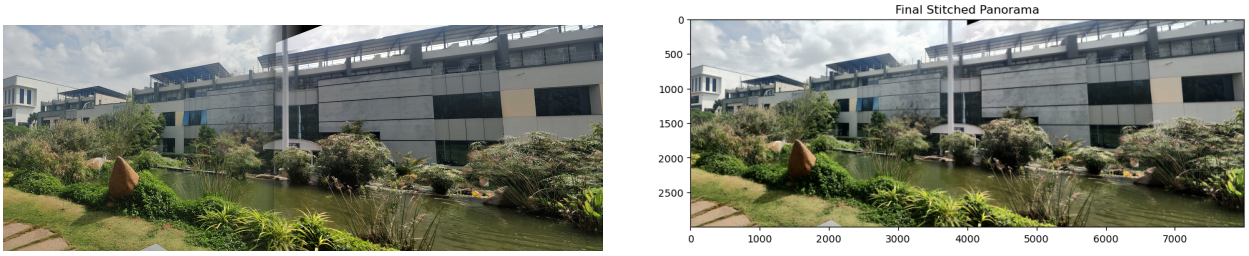
2.2.5 High-Resolution Panorama on images captured in our college

To further evaluate the robustness of our approach, we tested the homography estimation and blending pipeline on high-resolution images captured at our institute using a smartphone camera. These images were not sourced from the internet, ensuring real-world applicability and practical constraints such as lens distortion, varying lighting conditions, and perspective shifts.

(a) Left Image (High-Resolution)

(b) Right Image (High-Resolution)

Figure 20: High-resolution images captured at our institute before stitching.



(a) Feathering Blending (b) Weighted Blending (Distance Transform)

Figure 21: Comparison of Feathering and Weighted Blending techniques.

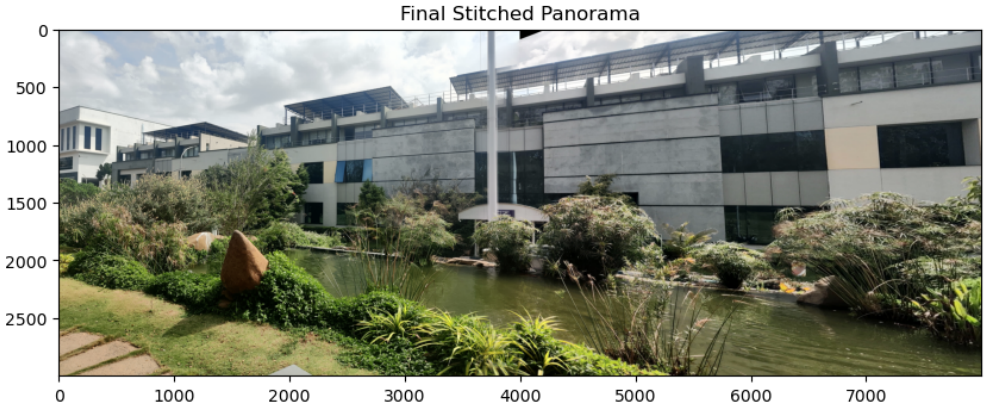


Figure 22: Final Stitched Panorama from High-Resolution Institute Images.

2.3 Extension to Multiple Images

After successfully stitching two images, the same approach was extended to multiple images. The process remains the same, but instead of a pairwise homography estimation, the images are aligned iteratively, with each new image being warped onto the existing stitched result. This extension allows for the creation of a larger and more complex panorama.

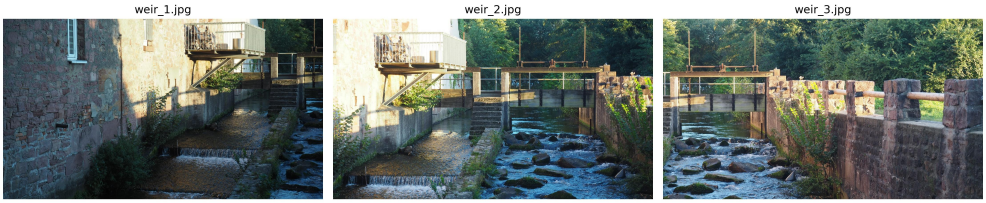


Figure 23: Set of three input images used for panorama creation (arranged in a 3×1 grid).

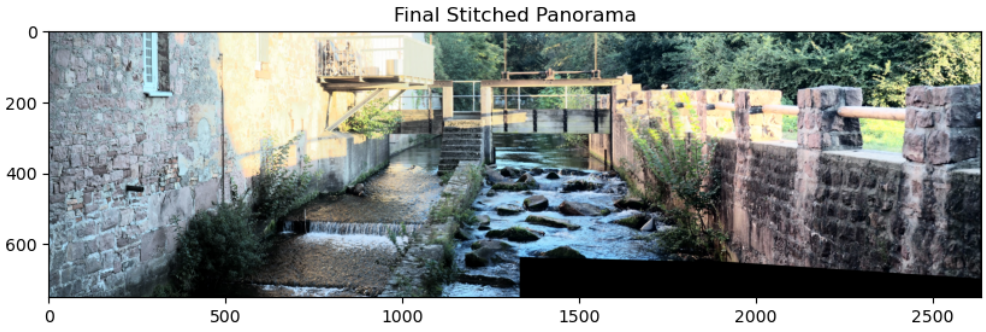


Figure 24: Final stitched panorama from three images.

2.4 Discussion

The SIFT-based approach demonstrated effectiveness in aligning images for panorama creation. RANSAC improved homography estimation by filtering incorrect matches, and weighted blending resulted in a smoother transition compared to linear blending. However, minor misalignments were observed due to parallax errors in input images. Future improvements could involve exposure compensation and multi-band blending.

2.5 Conclusion

This section presented an image stitching approach using feature detection, homography estimation, and blending techniques. The proposed method was first applied to two images, achieving a successful panorama. The approach was then extended to multiple overlapping images, demonstrating its scalability for larger panoramic image generation.

The complete implementation, including code, datasets, and additional experiments, is available on GitHub: `click here`