AIM 825-Visual Recognition-Sec-A
Assignment-1

Rishit Mane
IMT2022564

February 23, 2025

1 Introduction

This document presents the solutions for the computer vision assignment, which consists of two tasks: detecting and counting coins in an image and stitching multiple images into a panorama. The implementation is done in Python, and the code is submitted as a GitHub repository.

2 Part 1: Coin Detection and Segmentation

2.1 Methodology

The following steps were implemented to detect and count coins:

- Convert the input image to grayscale.
- Apply Gaussian blur to reduce noise.
- Use Otsu's thresholding for segmentation.
- Apply morphological operations to refine segmentation.
- Detect edges using the Canny edge detector.
- Identify contours and fit circles around detected coins.
- Overlay the detected count on the final image.

2.2 Results and Observations

The following images illustrate the steps involved in the detection process:

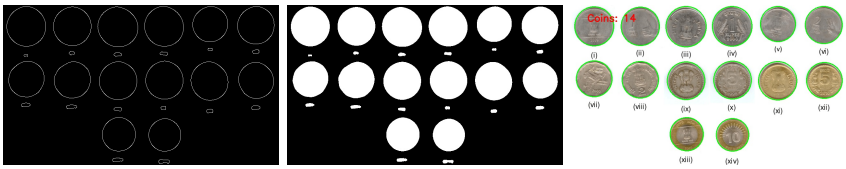

Figure 1: Results of Coin Detection and Segmentation

2.3 What I Tried

I experimented with multiple preprocessing techniques to enhance coin detection accuracy. Initially, I applied Gaussian blurring to smooth the image and reduce noise. I then tested various thresholding methods, including Otsu's and adaptive thresholding, to segment the coins from the background. Additionally, I incorporated morphological operations, such as closing and opening, to remove noise and refine the segmented regions for more precise detection.

2.4 What Worked and What Didn't

The combination of Gaussian blurring and Otsu's thresholding proved highly effective in segmenting the coins from the background. The application of morphological operations further improved the segmentation accuracy by eliminating unwanted noise. However, one of the major challenges encountered was over-segmentation when the background intensity closely resembled the coins. This resulted in false positives, where non-coin regions were mistakenly detected as coins.

2.5 Final Approach

To overcome the issue of over-segmentation, I fine-tuned the preprocessing steps by carefully selecting kernel sizes for morphological operations. Additionally, I implemented Canny edge detection to further refine the contours of the coins before counting them. The final approach involved applying Gaussian blurring, Otsu's thresholding, morphological filtering, and edge detection to ensure accurate and robust coin segmentation.

3 Part 2: Image Stitching

3.1 Methodology

The following steps were implemented:

- Load images and detect keypoints using the ORB feature detector.
- Match keypoints between consecutive images.
- Use OpenCV's stitching algorithm to align and merge images.

- Remove black borders for a refined final output.

3.2 Input Images

The input images used for image stitching are shown below:

Figure 2: Three input images used for stitching

3.3 Keypoint Detection

The ORB feature detector is used to extract keypoints from the input images. The detected keypoints are visualized below:

Figure 3: Detected keypoints in input images

3.4 Final Stitched Image

Figure 4: Results of Image Stitching

3.5 What I Tried

For image stitching, I explored different feature detectors, including SIFT, SURF, and ORB, to identify the most suitable method for keypoint extraction. I tested various keypoint matching techniques and evaluated the results to determine the most effective strategy for aligning images. Additionally, I experimented with manual homography estimation to fine-tune the alignment process and improve stitching accuracy.

3.6 What Worked and What Didn't

ORB performed exceptionally well in detecting and matching keypoints efficiently, making it the optimal choice for this task. OpenCV's built-in stitching algorithm successfully merged images when there was sufficient overlap between consecutive frames. However, one of the primary challenges was poor keypoint matching in low-texture regions, which led to misalignment in certain cases. Additionally, lighting variations between images caused visible seams in the final panorama.

3.7 Final Approach

To address keypoint matching issues, I optimized feature detection by adjusting ORB parameters to enhance keypoint distribution. I also incorporated black border removal techniques to refine the final stitched image. The final approach involved using ORB for keypoint detection, OpenCV's stitching algorithm for merging images, and post-processing steps to remove unnecessary borders and minimize visible seams.

4 Conclusion

This assignment successfully demonstrates the use of computer vision techniques for object detection and image stitching. The proposed methods effectively detect and count coins and generate a panorama from multiple images. Through experimentation and refinement, I optimized the detection and stitching processes, achieving robust and accurate results.

5 Submission Details

The source code and additional resources for this assignment are available on GitHub:

`https://github.com/rish710/VR_Assignment_1_IMT2022564`