VR Assignment 1: Coin Detection and Panorama Creation

Author: Divyam Sareen

Roll Number: IMT2022010

---

1. Introduction

This project focuses on two key computer vision tasks:

1. **Coin Detection and Segmentation**: Identifying and isolating coins from images using edge detection and contour analysis.
2. **Panorama Creation**: Stitching multiple overlapping images into a single panoramic image using feature detection and homography.

The implementation is divided into two Python scripts:

- **Part1.py**: Handles coin detection and segmentation.
- **Part2.py**: Performs panorama stitching using images provided in a folder.

Additionally, various preprocessing and feature extraction techniques were tested, some of which were commented out due to unsatisfactory results. These are discussed in detail below.

---

2. Coin Detection and Segmentation (Part1.py)

2.1 Image Preprocessing

1. The input image is read and converted to **grayscale** for simplicity.
2. **Gaussian blur** is applied using a kernel size of (19,19) to smoothen the image and reduce noise, which enhances edge detection.

   Other blurring techniques were tested but commented out due to poor results:

- **Median Blur** (`cv2.medianBlur(gray, 5)`) was tried but resulted in incomplete edge detection.
- **Bilateral Filtering** (`cv2.bilateralFilter(gray, 9, 25, 5)`) preserved edges but created inconsistencies in thresholding.

3. **Histogram Equalization** (`cv2.equalizeHist(blurred)`) was applied to enhance contrast; however, it inadvertently highlighted background regions, leading to unwanted detections during thresholding.

2.2 Thresholding

1. **Binary Inverse Thresholding** (`cv2.threshold(blurred, 125, 255, cv2.THRESH_BINARY_INV)`) was applied to create a binary image where coins appear as white objects.
2. Connected Components Analysis (`cv2.connectedComponentsWithStats`) was initially tested to address small gaps in the detected edges. However, it was later commented out as the gaps were effectively resolved using thresholding.

2.3 Edge Detection and Contour Detection

1. **Canny Edge Detection** (`cv2.Canny(thresholded_img, 50, 150)`) was used to highlight the edges of coins.
2. Contours were extracted using `cv2.findContours()`, ensuring that only significant objects were considered.

   Morphological operations were attempted but later discarded:

   - **Dilation** (`cv2.dilate(edges, kernel, iterations=2)`) was tested to strengthen edges but led to over-segmentation.
   - **Morphological Closing** (`cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel, iterations=2)`) was applied to fill small gaps but was later removed as it did not improve results significantly.

2.4 Segmentation and Output Generation

1. The detected contours were drawn on the original image to visualize the identified coins.
2. Each segmented coin was extracted using **bounding boxes**, and masked images were saved in the output folder.

3. The final segmented images and intermediate results (edges, thresholded images, and blurred images) were saved in `Part1_output/`.
4. `{filename}_outlined.png` represents the input image with detected coins outlined, while `{filename}_segmented_coin_{i}.png` contains the extracted image of the `i` th segmented coin from the corresponding input image.

2.5 Results

Input Image

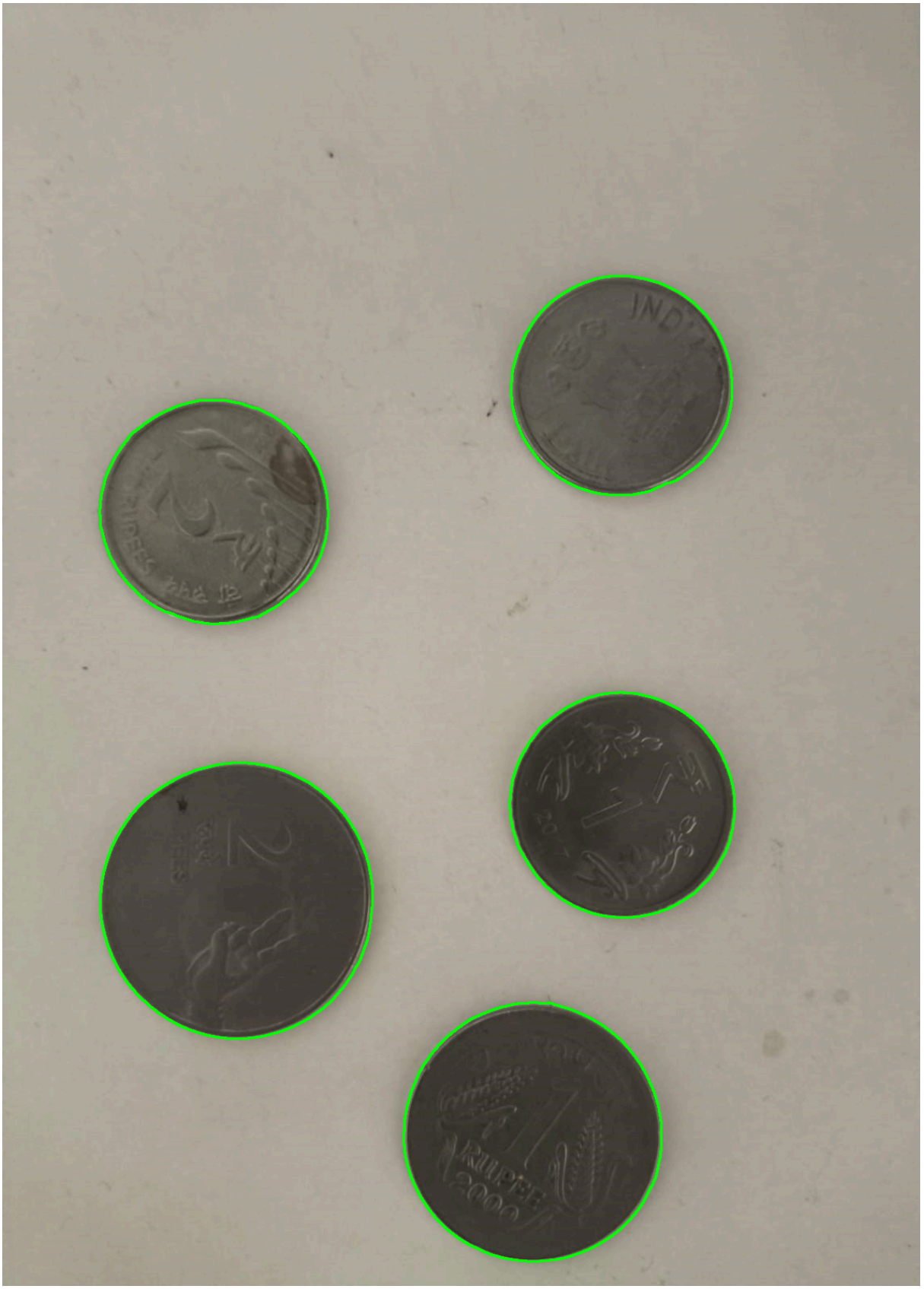


Image with coins outlined

Segmented coins

```
(myenv) PS C:\Users\T761\OneDrive\Desktop\VR_Assignment1_Divyam_IMT2022010> python3 .\Part1.py
Processing coins1.jpeg...
Number of coins in the image: 3
Processing coins2.jpeg...
Number of coins in the image: 5
Processing coins3.jpeg...
Number of coins in the image: 4
```

Number of coins in each image

3. Panorama Creation (Part2.py)

3.1 Input Handling

- The script automatically reads three images from the input folder: `left_{i}.jpeg`, `centre_{i}.jpeg`, and `right_{i}.jpeg` for different values of `i`.
- It processes each set of images and outputs the final stitched image in `Part2_output/`.

3.2 Feature Detection and Matching

1. The **Scale-Invariant Feature Transform (SIFT)** algorithm detects keypoints and descriptors for each image.
2. **Feature Matching** is performed using `cv2.BFMatcher()` with **Lowe's ratio test** to filter out poor matches.

3.3 Image Stitching and Homography Estimation

1. **Homography Matrix Calculation** (`cv2.findHomography()`) aligns the images.
2. **Image Warping** (`cv2.warpPerspective()`) transforms images to a common perspective.
3. The first stitching step combines `centre_{i}.jpeg` and `right_{i}.jpeg`.
4. The second stitching step merges `left_{i}.jpeg` with the first stitched result.

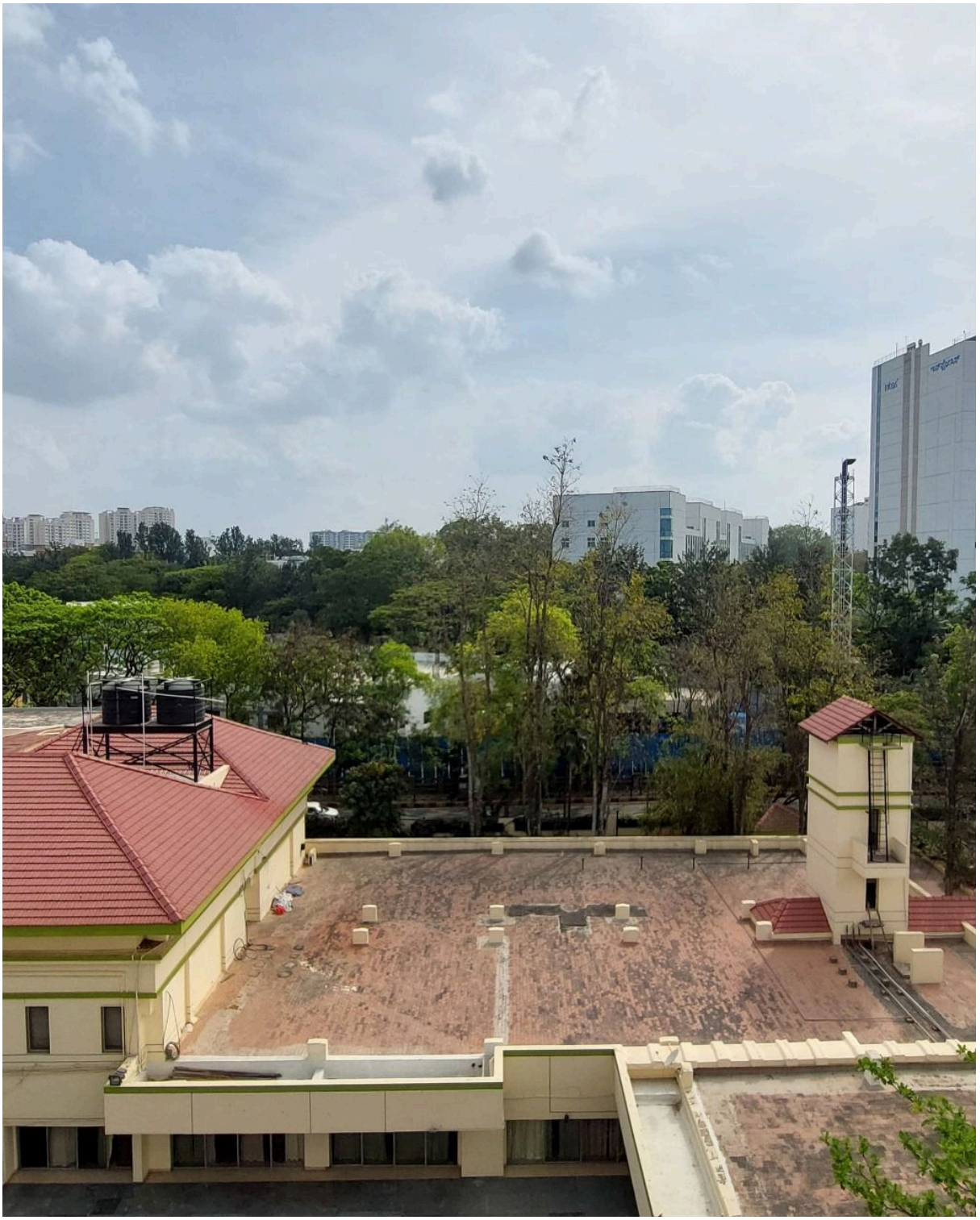
3.4 Post-Processing and Cropping

1. The stitched image often contains black borders. To remove them:
    - The image is converted to grayscale.
    - Thresholding (`cv2.threshold()`) is applied to create a binary mask.
    - **Contour Detection** (`cv2.findContours()`) is used to find the largest bounding box.
    - The final image is cropped to remove excess black regions.

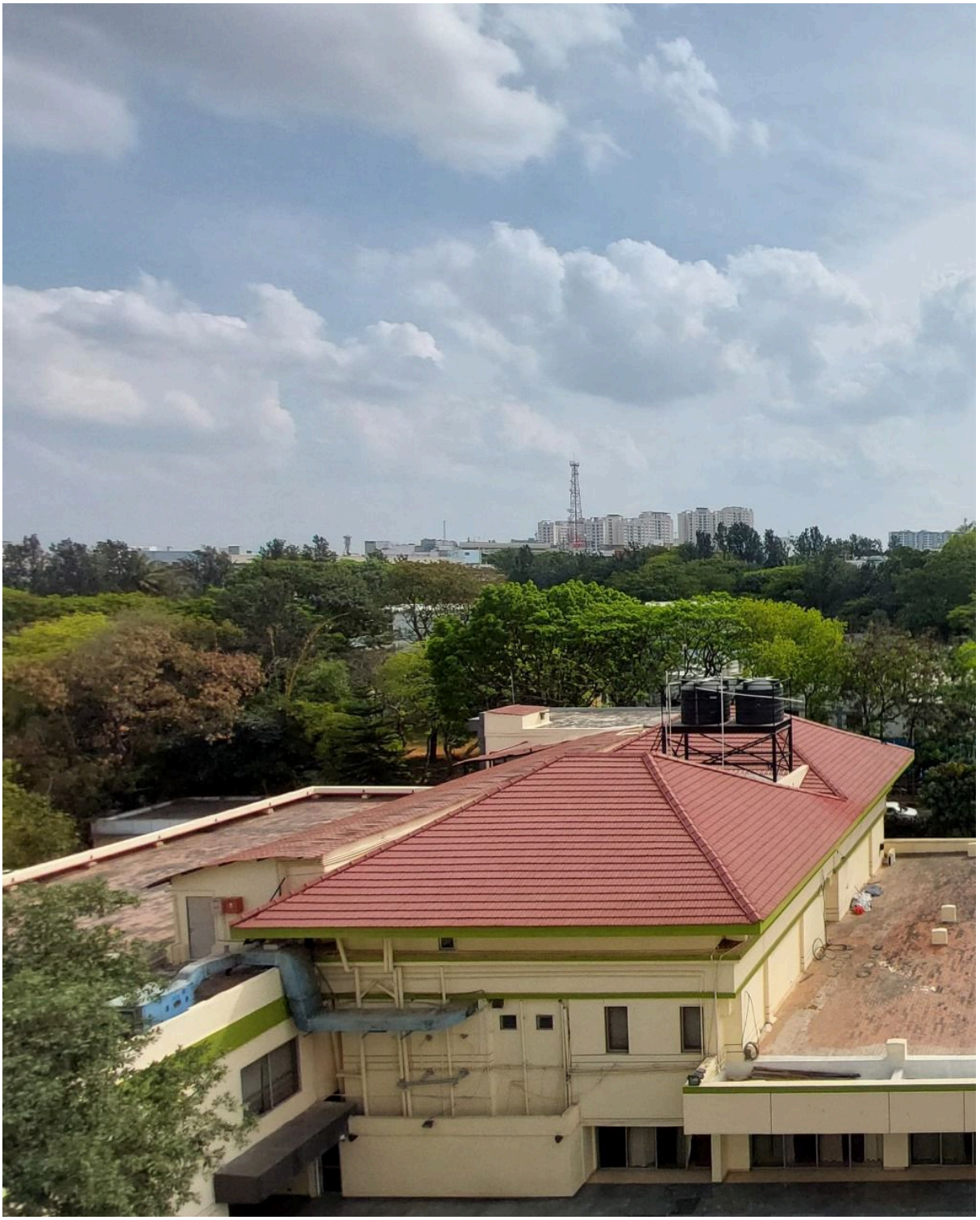
3.5 Output Generation

1. The Final stitched images can be found in `Part2_output/`.

3.6 Results



centre



left

right

final stitched image

4. Results and Observations

4.1 Coin Detection

- Successfully detected and segmented coins in various images.
- Filtering ensured that only circular objects were detected.
- Morphological operations were not effective in improving segmentation.

4.2 Panorama Creation

- **SIFT-based stitching** provided better results than previously attempted ORB-based methods.
- Feature matching and homography estimation ensured accurate alignment.
- Black border cropping improved final output aesthetics.
- **Issues:** Some minor distortions appeared in certain stitched images, possibly due to variations in lighting or perspective.

---

5. Conclusion

- The **Coin Detection System** successfully detects and extracts coins from images.

- The **Panorama Stitching Algorithm** correctly merges three images while removing unnecessary black regions.
- Multiple alternative approaches were tested, but only the best-performing methods were retained.

Future Improvements

- Enhance coin detection for overlapping or touching coins.
- Improve panorama blending to minimize seams.
- Experiment with deep learning-based segmentation and stitching techniques.

This report comprehensively documents the processes, alternative methods tested, and the final solutions implemented in **Part1.py** and **Part2.py**. The commented-out code represents trials that were unsuccessful or suboptimal in the final workflow.

Final code can be found at https://github.com/DeathlyMade/VR_Assignment1_Divyam_IMT2022010.