VR Assignment 1

Aditya Priyadarshi

Part 1

Preprocessing

The input image undergoes preprocessing to enhance the quality and aid in further analysis:

- **Grayscale Conversion**: The input image is converted to grayscale to simplify processing.
- **Scaling**: The image is resized using a scaling factor computed as:

$$\text{scale\_factor} = \frac{700}{\max(\text{image dimensions})}$$

  This ensures a consistent size across different images, making processing more efficient.

- **Blurring**: A Gaussian blur with a 5×5 kernel is applied to reduce noise.
- **Thresholding**: Adaptive thresholding is applied to create a binary image that highlights coin edges.

Edge Detection

To detect coins, contours are extracted from the thresholded image. The steps are:

1. **Contour Detection**: Contours are extracted using OpenCV's `findContours()` function.
2. **Circularity Computation**: The circularity of each detected contour is calculated using:

$$C = \frac{4\pi \times \text{Area}}{\text{Perimeter}^2}$$

   A perfect circle has a circularity of 1. Since real-world coins are not mathematically perfect (and to take account for some noise), we set a threshold of $0.7 < C < 1.2$.

3. **Area Constraint**: A contour is classified as a coin only if its area exceeds a certain threshold. The area threshold was set as:

$$\text{Area} > 500 \times (\text{scale\_factor}^2)$$

The identified coin edges are visualized by drawing green contours around them.

Region-Based Segmentation

After identifying coins, region-based segmentation is applied:

- A mask is created where detected coin contours are filled.
- Bitwise '`and`' operations extract the coins from the original image using the mask.

Coin Extraction

Each detected coin is extracted individually:

1. The minimum enclosing circle for each contour is determined.
2. A mask is created by drawing a filled circle over the detected coin.
3. Bitwise operations isolate the coin from the background.
4. The extracted region is cropped and stored.

Example Results

Below are example images generated during processing:

Figure 1: Original Input Image

Part 2

Approach

The approach involves detecting key points using the Scale-Invariant Feature Transform (SIFT), matching them using the Brute Force Matcher, and computing homography to align images. The reasons for choosing this approach are:

- **SIFT Feature Detection:** SIFT is robust to scale, rotation, and lighting variations, making it ideal for panorama stitching.

Figure 2: Edges Detected on the Image

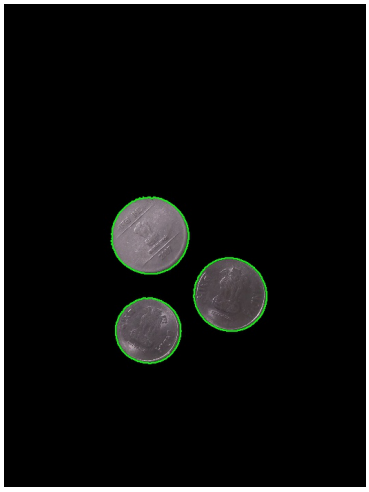

Figure 3: Segmented Coins Mask

- **Brute Force Matcher:** Ensures accurate keypoint matching using KNN and Lowe's ratio test.
- **Homography Transformation:** Allows proper alignment of images even with perspective changes.
- **Sequential Stitching:** Instead of blending all images simultaneously, images are combined in order to reduce computation and maintain alignment accuracy.

Functions Overview

Keypoint Detection and Feature Extraction `sift_detect_descriptor`

Detects key points and computes descriptors using SIFT.

- Converts keypoints to a float32 array.
- Extracts feature descriptors to use for matching.

Figure 4: Extracted Individual Coins

Matching Keypoints Between Images `interest_point_matcher`

Matches keypoints between two images using the Brute Force Matcher and Lowe's ratio test.

- Uses KNN matching to find the two best matches.
- Applies Lowe's ratio test to filter out poor matches.
- Computes the homography matrix using RANSAC.

Black Border Cropping `crop_black_region`

Removes extra black regions after warping by detecting the largest contour and cropping the bounding box.

Image Warping and Blending `stitch`

Stitches two images together by computing homography and warping one image onto another.

- Calls feature detection and keypoint matching functions.
- Computes homography and applies perspective transformation.
- Crops unnecessary black regions.
- Returns the stitched panorama.

Sequential Stitching for Multiple Images `main`

The implemented code stitches multiple images in sequence:

1. Reads images in a sorted order.
2. Starts with the first image and stitches the next image onto it iteratively.
3. Saves intermediate visualizations and the final panorama.

This approach ensures a smooth transition across images while maintaining computational efficiency.

Figure 5: Example input images

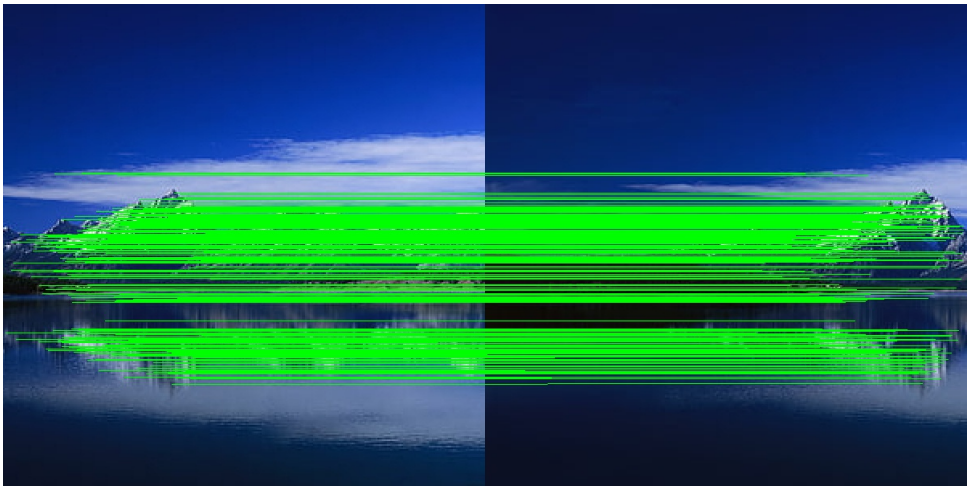

Figure 6: Keypoint Matches between Images (1-2 match)

Figure 7: Keypoint Matches between Images (Pano123 - 4 match)

Figure 8: Final Stitched Panorama