Visual Recognition Assignment Report

MT2024 Subha Chakraborty

March,2025

1 Q1: Coin Segmentation and Counting using Computer Vision Techniques

In this task, I applied various computer vision techniques to detect, segment, and count Indian coins from an image containing scattered coins. The methodology is divided into the following key steps:

1.1 Preprocessing the Image

The first step in the segmentation process was to load the image and convert it to grayscale. Then I applied a Gaussian blur to reduce noise in the image, which helps in improving the accuracy of subsequent edge detection.

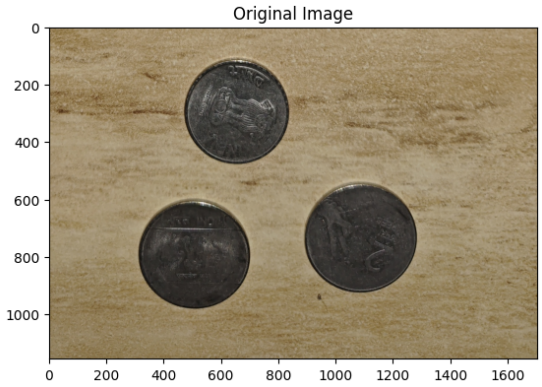

Figure 1: Original Image

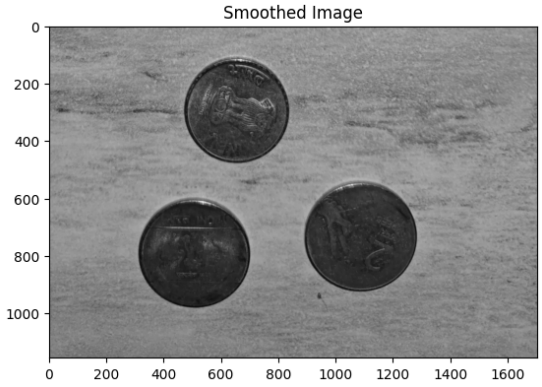

Figure 2: Preprocessed Image

1.2 Edge Detection

Edge detection is crucial for identifying the boundaries of objects, in this case, the coins. I used the Canny edge detector, which is effective in detecting edges by identifying areas of high intensity gradient. The result of this process was a binary image where edges are marked white, and other regions are black.

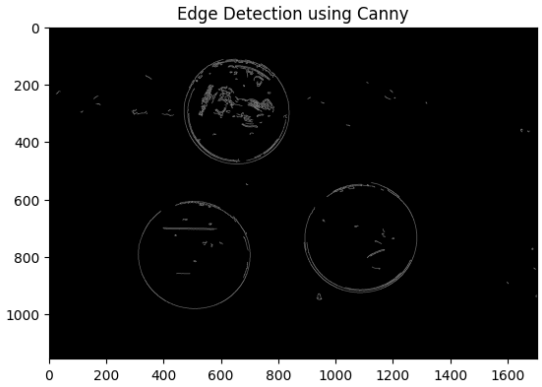

Figure 3: Edges of the Image

1.3 Region-Based Segmentation using Thresholding and Contour Area Mask

1.3.1 Thresholding

To further segment the image, I applied Otsu's thresholding method, which dynamically determines the optimal threshold value. This is particularly useful for images where lighting conditions vary or where different coin types have distinct

intensity ranges. Thresholding effectively separates the foreground (coins) from the background.

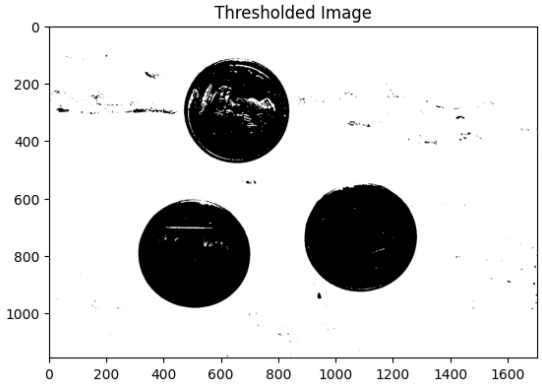

Figure 4: Thresholded Image

1.3.2 Contour Detection and Area Mask

Once the image was thresholded, I used the `cv2.findContours()` function to detect the boundaries of the individual coins. Contours represent the outlines of connected components in an image, which in this case correspond to the coins.

The contour area is calculated for each contour, and I applied a contour area mask to filter out irrelevant contours (such as noise or very small regions). A contour with an area above a threshold of 10,000 pixels is considered to correspond to a coin. This filtering step helps in removing background noise and focusing on significant regions representing the coins.

1.4 Results and Counting the Coins

The contours were sorted by their area, and the largest contour, which typically corresponds to the entire image, was ignored. By counting the number of contours with significant areas, I was able to determine the number of coins in the image. Finally, I drew contours around the coins and displayed the segmented image.

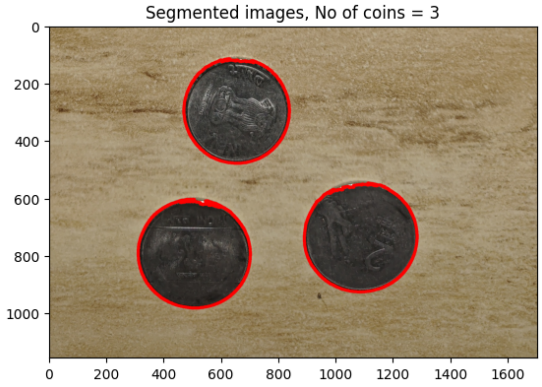

Figure 5: Segments and Count for 3 coins

1.5 Conclusion

This approach of dynamic thresholding and contour area masking provides a robust solution for segmenting and counting coins in images with varying lighting conditions and coin sizes. Below are few more examples:

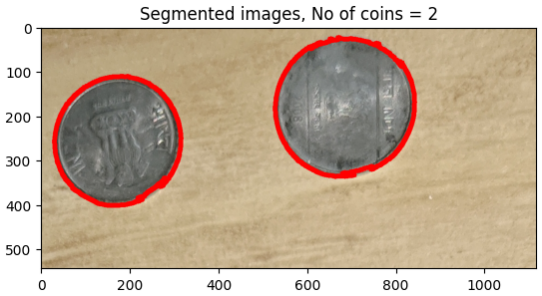

Figure 6: Segments and Count for 2 coins

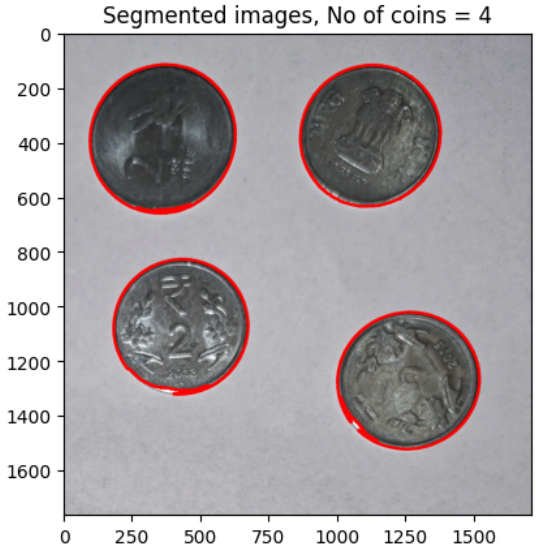

Figure 7: Segments and Count for 4 coins

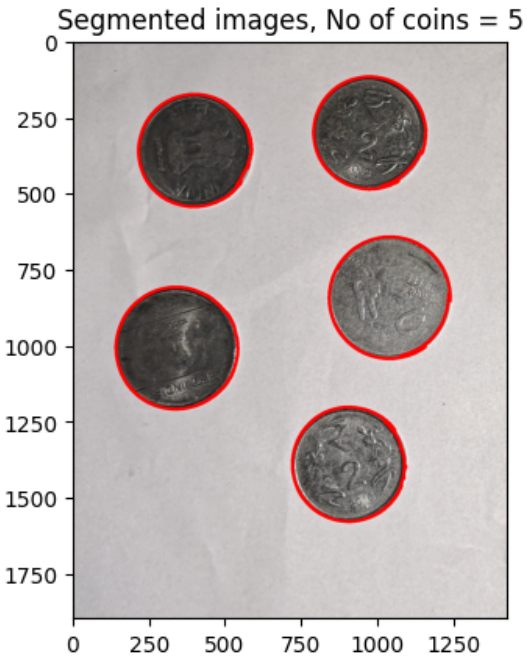

Figure 8: Segments and Count for 5 coins

2 Q2: Panorama Stitching from Multiple Overlapping Images

In this task, I created a panorama by stitching multiple overlapping images together.

2.1 Keypoint Detection with SIFT

The images were first loaded and keypoints were detected using the Scale-Invariant Feature Transform (SIFT) algorithm. This step is critical for aligning the images properly.

SIFT (Scale-Invariant Feature Transform) is an algorithm used to detect and describe local features in images. It works by detecting keypoints at different scales (i.e., locations with significant changes in intensity) and calculating descriptors that capture the distinctive characteristics around each keypoint. SIFT is particularly useful because it is invariant to scaling, rotation, and even partial affine transformations, which makes it ideal for image stitching tasks.

SIFT identifies keypoints that are stable across different viewpoints and scales. Each keypoint is associated with a descriptor, a vector that encodes the local image region around the keypoint. These descriptors are used to match keypoints between images, forming the basis for aligning and stitching images.

2.2 Feature Matching with BFMatcher

After detecting keypoints in each image, I used a brute-force matcher (`cv2.BFMatcher()`) to find the best matches between the descriptors of the keypoints. The BFMatcher works by comparing the descriptors of keypoints from two images and finding pairs that are closest in terms of Euclidean distance. In my code, I found it more reliable than KnnMatcher but it is most likely dependent on my smartphones ability to capture pictures for the assignment.

To ensure a good match, I applied Lowe's ratio test to filter out poor matches. This test checks that the ratio of distances between the closest and the second closest descriptor matches is below a threshold (typically 0.8). This helps eliminate ambiguous matches, ensuring that only reliable correspondences between keypoints are used.

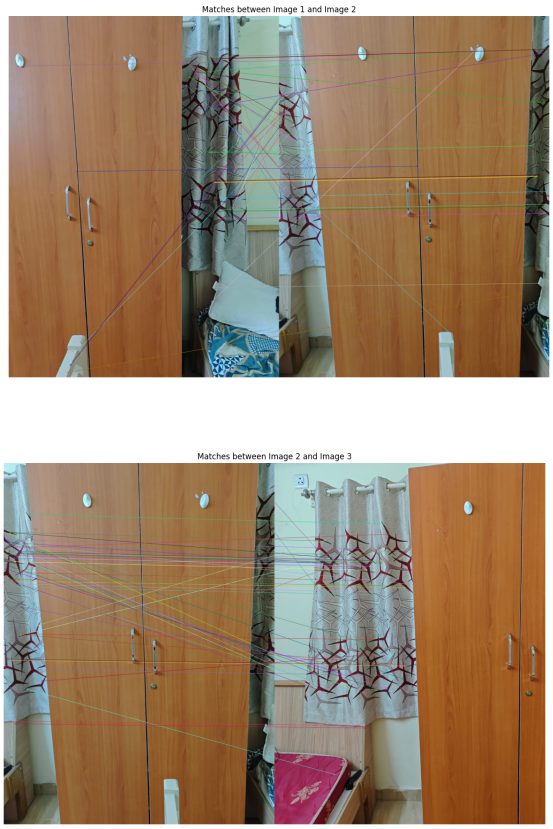

Figure 9: Matches between the Images

2.3 Homography Estimation with RANSAC

Once the keypoints from consecutive images were matched, I used RANSAC (Random Sample Consensus) to estimate a homography matrix that aligns one image to another. Homography is a transformation matrix that maps points from one image to another, accounting for changes in viewpoint, rotation, and scaling.

RANSAC is an iterative method used to estimate parameters (in this case, the homography matrix) by repeatedly selecting random subsets of the data and fitting the model to these subsets. Outliers (incorrect matches) are ignored during this process, which makes RANSAC robust to noise and false correspondences. The final homography matrix is computed by considering only inliers (reliable matches), which results in a more accurate transformation.

2.4 Image Warping and Stitching

Once the homography was calculated, I warped the second image to align it with the first image. This process was repeated for all subsequent images, progressively stitching them together. As each image was added, I displayed the intermediate results to visualize the progress of the panorama creation.

2.5 Cropping the Black Region

After stitching the images, there were black borders in the resulting panorama. To remove these regions, I applied a simple cropping technique. The image was converted to grayscale, and a threshold was applied to detect non-black regions. The bounding box around the non-black regions was used to crop the final panorama.

Figure 10: Final panorama

2.6 Conclusion

The panorama stitching process involves detecting keypoints using SIFT, matching the descriptors, and estimating the homography matrix using RANSAC. By applying image warping and carefully cropping the black regions, I was able to generate a seamless panorama.

2.7 A different approach using the cv::Stitcher Class

Instead of explicitly doing everything (as the assignment requires it), I decided to try using the Sticher class, to directly get the panorama. The panorama image generated was of better quality but also needed post-processing so clean

the dark background surrounding it. Nevertheless, it showed the limitiation of my ground up approach, but I couldn't outperform it.

Figure 11: Panorama using cv::Stitcher