Assignment 1 Report

Sai Venkata Sohith Gutta (IMT2022042)

Keywords

OpenCV, Image Processing, Segmentation, Edge Detection, Thresholding, Morphological Operations, Image Stitching, Feature Matching, Homography, Distance Transform, Blending.

GitHub Repository

The complete implementation can be found at: GitHub Repository and LMS submission

Introduction

This report details the implementation of a coin segmentation algorithm and an image stitching technique using OpenCV. The primary goal of the first part is to identify and extract individual coins from an image using image processing techniques. The second part focuses on blending multiple images to create a seamless panorama using feature matching and homography transformation.

Methodology

The implemented code follows a structured approach to process and segment the coins in an image and stitch multiple images into a panorama. The key steps involved are:

Part 1: Coin Segmentation

Image Preprocessing

- The input image ('Coins.jpg') is read using OpenCV.
- The image is rescaled to 25% of its original size using the `rescaleImg` function to optimize processing efficiency.
- The rescaled image is converted to grayscale to simplify further processing.

Figure 1: Original image used for counting coins

Noise Reduction

- A Gaussian Blur with a 3x3 kernel is applied to reduce noise and smooth the image.

Edge Detection and Thresholding

- The Canny Edge Detection algorithm is applied with threshold values of 100 and 150 to highlight edges.
- Otsu's Thresholding is used to create a binary image, enhancing object separation.

Figure 2: Edges of coins detected using Canny Edge Detector

Figure 3: Coins image after thresholding

Morphological Processing

- A morphological closing operation with a 5x5 kernel is applied for six iterations to fill holes and enhance object contours.

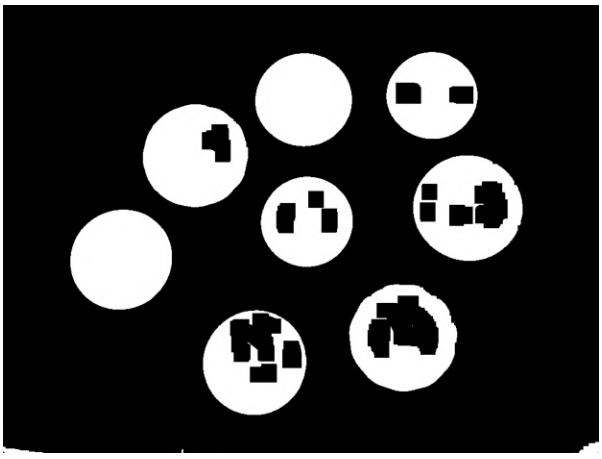

Figure 4: Thresholded image after using morphological close

Contour Detection and Filtering

- Contours are detected using `cv.findContours` with `cv.RETR_EXTERNAL` to retrieve only external contours.
- A filtering step ensures only contours with an area greater than 15,000 pixels are considered valid coins.

Figure 5: Contours after filtering about the required areas

Coin Extraction and Masking

- A mask is created for each detected coin, and `cv.bitwise_and` is used to extract individual coins.
- Each segmented coin is saved to the `seg_coins` directory as a PNG image.
- The total number of coins detected is printed to the console.

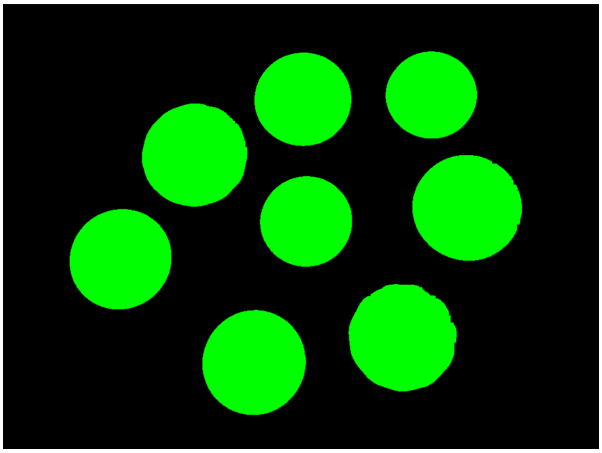

Figure 6: Full Mask which will be further used to extract mask for each coin using `'bitwise_and'` and then extracting each coin

Final Output Images

(a) Extracted Coin 1 (b) Extracted Coin 2 (c) Extracted Coin 3

(d) Extracted Coin 4 (e) Extracted Coin 5 (f) Extracted Coin 6

(g) Extracted Coin 7 (h) Extracted Coin 8

Figure 7: Final extracted coins

Part 2: Image Stitching

Image Preprocessing

- Two images ('img1.jpg' and 'img2.jpg') are read from the 'images' folder.
- The images are rescaled to 15% of their original size using the `rescale` function.
- Each image is converted to grayscale using the `makeGray` function.

Figure 8: Image 1 for panaroma

Figure 9: Image 2 for panaroma

Keypoint Detection and Feature Matching

- The SIFT (Scale-Invariant Feature Transform) algorithm is used to detect key points and extract descriptors.

- A Brute Force Matcher with L2 norm is used to match the key points between the two images.
- The matches are sorted based on their distance, and the top 75 matches are visualized.

Figure 10: Matching the key points of images

Homography and Image Warping

- If at least four good matches are found, a homography matrix is computed using RANSAC.
- The second image is warped onto the first image using the homography transformation.

Distance Transform and Image Blending

- A distance transform is computed for both images to determine the blending weights.
- The warped second image and the first image are blended using these distance transform values to create a seamless transition.

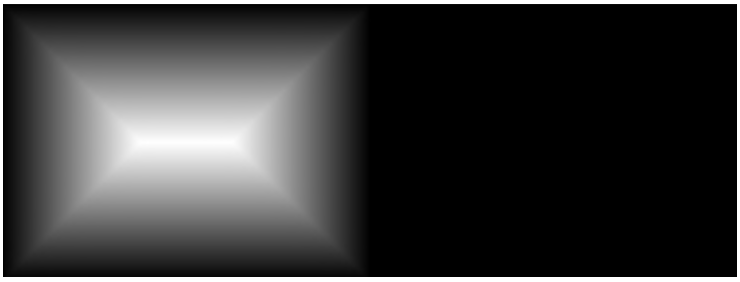

Figure 11: Distance Transform for image 1

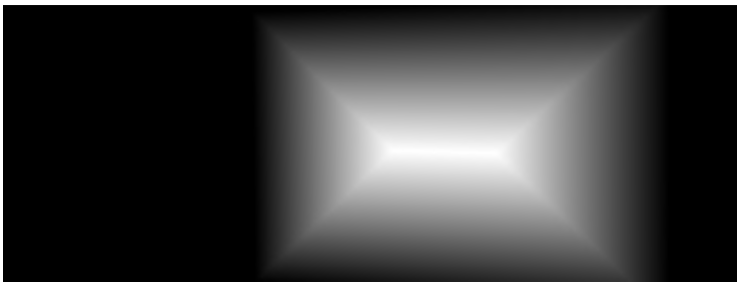

Figure 12: Distance Transform for image 2

Output Generation

- The final stitched image is displayed and saved as 'Panorama.jpg'.

Figure 13: Panaroma made using both the images

Implementation Details

The implementation for Part 2 includes the following functions:

- `makeGray(img)`: Converts an image to grayscale.
- `rescale(frame, scale)`: Resizes an image while maintaining aspect ratio.
- `distanceTransform(img)`: Computes the distance transform for blending.
- `blendImages(img1, img2, dist1, dist2, p_w, p_h)`: Blends two images using their distance transforms.
- `detectKeyPoints(img1, img2)`: Detects key points and matches them using SIFT and Brute Force Matcher.
- `stitchImages(kp1, kp2, matches, img1, img2, gray_img1, gray_img2)`: Uses homography to align and stitch images into a panorama.

- `main()`: Reads input images, performs keypoint detection, stitching, and displays the final result.

Results and Observations

- The coin segmentation script successfully extracts individual coins from the image.
- The morphological operations improve segmentation by closing gaps within coin edges.
- The image stitching algorithm successfully combines two overlapping images into a single panorama.
- Feature matching using SIFT provides robust alignment for homography transformation.
- The distance transform-based blending reduces visible seams in the stitched image.

Conclusion

The implemented scripts effectively perform coin segmentation and image stitching using OpenCV. The combination of image preprocessing, feature detection, and transformation techniques provides a structured approach for both tasks. Further enhancements in adaptability and accuracy can be achieved through parameter tuning and more sophisticated algorithms.