IIIT-Bangalore

Visual Recognition

AIM - 825

Professors:
Prof. Viswanath G
Prof. Sushree Behera

Submitted by:
Madhav Girdhar (IMT2022009)

Task-1 : Detect, Segment, and count coins from an image

Introduction

Coin detection and segmentation using computer vision techniques is a crucial application in object recognition. This report presents a systematic approach to detecting, segmenting, and counting Indian coins from an image using OpenCV. The approach involves edge detection, morphological processing, contour filtering, and segmentation to achieve accurate coin detection.

Image Pre-processing

1. Image Loading and Resizing: The input image is read and resized to 800×800 pixels to maintain consistency during processing.
2. Grayscale Conversion: The image is converted to grayscale to simplify processing and enhance edge detection.
3. Noise Reduction: A Gaussian blur with a 5×5 kernel and a standard deviation of 1.5 is applied to reduce noise and smoothen the image.

Coin Detection and Segmentation

1. Edge Detection: The Canny edge detector is applied with threshold values of 50 and 150 to detect edges in the pre-processed image.
2. Morphological Closing for Noise Removal: A 5×5 kernel with two iterations of morphological closing is applied to fill small gaps and remove unwanted noise.
3. Filtering Contours by Area: Contours are extracted, and only those with an area greater than 500 are considered valid to filter out unwanted noise and non-coin objects.
4. Segmentation Using a Mask: A binary mask is created based on valid contours, isolating coins from the background. The mask is used to extract segmented coins.
5. Saving Segmented Coins: Each detected coin is cropped and saved individually in the output directory.
6. Result Display: The total number of detected coins is displayed on the segmented image, and the final results are stored in the output directory.

Result

The final image of detected coins and the segemented image are shown below, respectively:

Figure 1: Image with detected Coins.

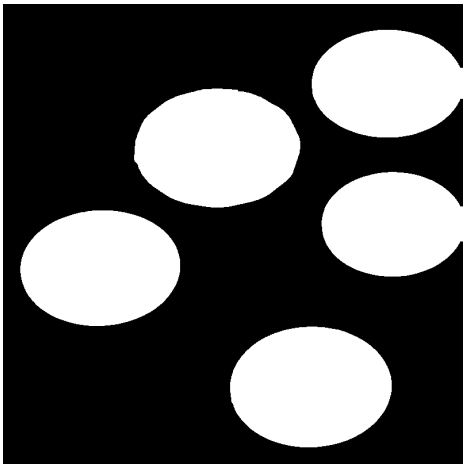

Figure 2: Segmented Image with Coins.

(a) Segmented Coin 1

(b) Segmented Coin 2

(c) Segmented Coin 3

(d) Segmented Coin 4

(e) Segmented Coin 5

Figure 3: Segmented Images of Coins

Observations and Challenges

1. The proposed method performed effectively in detecting and segmenting coins.

2. Challenges encountered:

    - Simple thresholding / Binary thresholding resulted in false detections due to reflections and uneven lighting in an image.
    - Direct contour detection without prior smoothing led to fragmented edges.
    - Without morphological closing, small gaps in the edges caused broken contours, leading to incorrect segmentation.
    - The kernel size and standard deviation must be carefully chosen to achieve optimal results. A larger kernel excessively smooths the image, causing a loss of fine details, while a smaller kernel fails to remove noise effectively, leading to fragmented contours and inaccurate detections.

Conclusion

This project successfully applied computer vision techniques to detect, segment, and count Indian coins from an image. The use of Canny edge detection, morphological operations, and contour filtering ensured accurate coin extraction. Future improvements, such as deep learning-based segmentation, could further enhance accuracy and robustness.

Task-2 : Stitched panorama from multiple overlapping images.

Introduction

Image stitching is a technique used in computer vision to combine multiple overlapping images into a single seamless panorama. This process is widely used in applications such as panoramic photography, satellite imaging, and medical imaging. The key steps involved in image stitching include detecting key points, extracting features, matching key points across images, and blending the images to create a smooth output.

Extract Keypoints and Stitching

The image stitching process involves the following steps:

- Keypoint Detection: Extract key points using the SIFT algorithm to detect distinctive features in each image.
- Feature Matching: Use the BF (Brute-Force) matcher to match key points between consecutive images based on their descriptors.
- Keypoint Visualization: Draw key points and matched features between image pairs to analyze the feature detection and matching process.
- Image Stitching: Use OpenCV's built-in Stitcher to align and blend the images into a panorama.
- Keypoint Detection on Stitched Image: After stitching, detect key points again on the final output image and visualize them.

Results

The final stitched panorama image is obtained after aligning and blending the input images. Below is the output of the image stitching process.

(a) Key points in first image

(b) Key points in second image

(c) Key points in third image

Figure 4: Key points detected in Input Images

(a) Key points matching in first and second image

(b) Key points matching in second and third image

Figure 5: Key points Matching in overlapping Images

Figure 6: Final panorama image with all image stitched

Observations and challenges

1. The proposed method performed effectively in detecting keypoints and stiching to produce desire result.
2. Challenges encountered :
    - Images should have sufficient overlap; otherwise, feature matching may fail, leading to stitching errors.
    - Good feature detection and matching are crucial for seamless blending; poor keypoint matching can result in misalignment.
    - Feature Matching Issues: ORB-based stitching failed due to an insufficient number of matched keypoints, especially in low-texture regions.

Conclusion

In this report, we successfully implemented an image stitching technique using keypoint detection, feature matching, and homography transformation. The results demonstrate an effective method for creating seamless panoramas from multiple overlapping images. Future improvements could include using deep learning-based image alignment techniques for more robust stitching.

Note

1. Code for both Task: Click here
2. All the images shown above are also present separately in the output directory of both task.
3. All input images used for both tasks were either sourced from Google or captured by me.