Assignment-1

By

Amruth Gadepalli

IMT2022065

Part- A: Coin number counting using edge detection

This Python program detects and counts coins in an image using OpenCV. The process involves multiple steps to enhance the image, extract contours, and filter out unwanted regions. Below is a breakdown of the approach and the reasoning behind each step:

1. Image Loading and Preprocessing
    - The image is read using cv2.imread(). If the file is not found, the program exits with an error message.
    - It is resized to 800x800 pixels to ensure uniform processing across different images.
    - A copy of the original image is created (img_copy) to draw contours without modifying the processed image.
    - Gaussian Blur (7x7 kernel, sigma=100) is applied to reduce noise and smooth out unnecessary details, allowing better edge detection.
2. Grayscale Conversion and Thresholding
    - The image is converted to grayscale since contour detection is more effective in a single-channel format.
    - Binary thresholding (threshold = 120) is applied to create a black-and-white image, making the coins more distinguishable from the background.
3. Contour Detection and Sorting

- Contours are detected using cv2.findContours() with cv2.RETR_TREE, which retrieves all contours along with hierarchical relationships.
- The cv2.CHAIN_APPROX_NONE mode is used to store all contour points, preserving the shape details.
- The area of each contour is calculated using cv2.contourArea(), and the contours are stored in a dictionary.
- The contours are sorted in descending order of area, assuming the largest contour corresponds to the background.

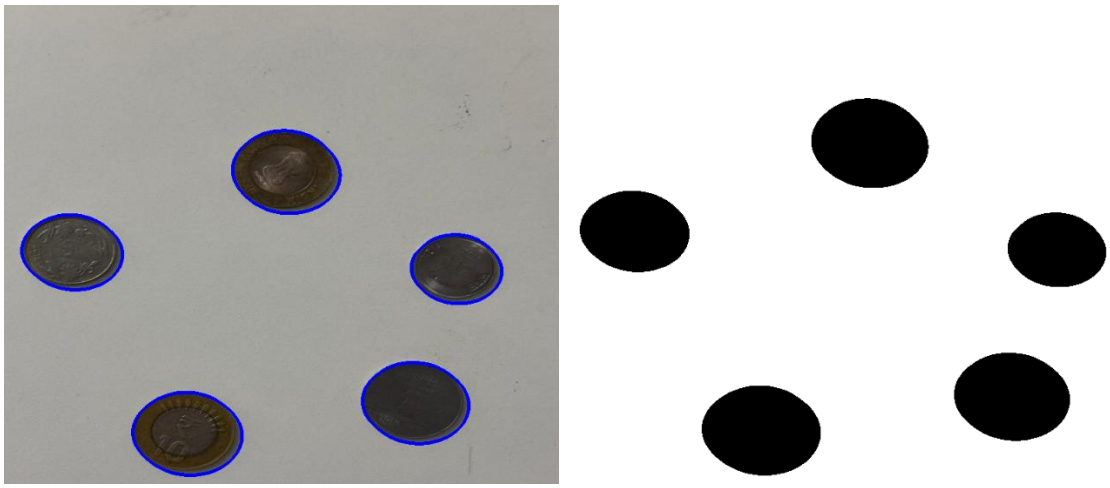
4. Filtering and Counting Coins
    - The program ignores the largest contour (likely the background).
    - The remaining contours are counted as coins and drawn on the original image with a blue outline (BGR: (255, 0, 0)) using cv2.drawContours().
5. Displaying Results
    - The number of detected coins (excluding the background) is printed.
    - The final processed image, with detected coins outlined, is displayed using OpenCV's imshow() function.

Overview:

- Blurring reduces noise and prevents false contour detection.
- Thresholding creates a binary image, making object separation easier.
- Contour sorting helps in distinguishing the background from objects of interest.
- Skipping the largest contour ensures the count reflects only actual coins or else the background gets counted as well.
- Drawing contours visually highlights the detected coins for verification.

This approach ensures accurate coin detection while minimizing false positives.

Results and Images:



There are 2 images here. The right one is what we get after thresholding. From this after applying contouring and using these contours for outlining we get the boundaries for the coin in the left image. The result displayed is 5.

Part- B: Taking 2 images and making a Panaroma from it by stitching.

This Python program performs feature matching and panorama stitching using OpenCV. The process involves detecting key points, finding correspondences between two images, and then attempting to merge them into a single panoramic image. Below is a breakdown of the approach and the reasoning behind each step:

1. Image Loading and Preprocessing

- Two images (left.jpeg and right.jpeg) are loaded using cv2.imread().
- They are converted to grayscale using cv2.cvtColor(), as grayscale images simplify feature detection and improve computational efficiency.

2. Feature Detection Using SIFT (Scale-Invariant Feature Transform)

- A SIFT detector (cv2.SIFT_create()) is initialized to detect key features in the images.

- Key points and descriptors are extracted from both images using sift.detectAndCompute():
    - Key points represent unique, identifiable points in an image (such as corners or edges).
    - Descriptors encode information about the surrounding region of each key point, allowing for feature matching.

3. Feature Matching Using Brute-Force Matcher (BFMatcher)

- A Brute-Force Matcher (BFMatcher) is created using cv2.BFMatcher(cv2.NORM_L2, crossCheck=True).
- It compares the descriptors from both images and finds the best matches.
- The matches are sorted in ascending order of distance, ensuring that the most similar key points are prioritized.

4. Visualizing Feature Matches

- The top 50 matches are drawn using cv2.drawMatches().
- The output image displays corresponding key points connected between both images.
- The result is displayed using matplotlib.pyplot to visualize how well the images align.

5. Panorama Stitching

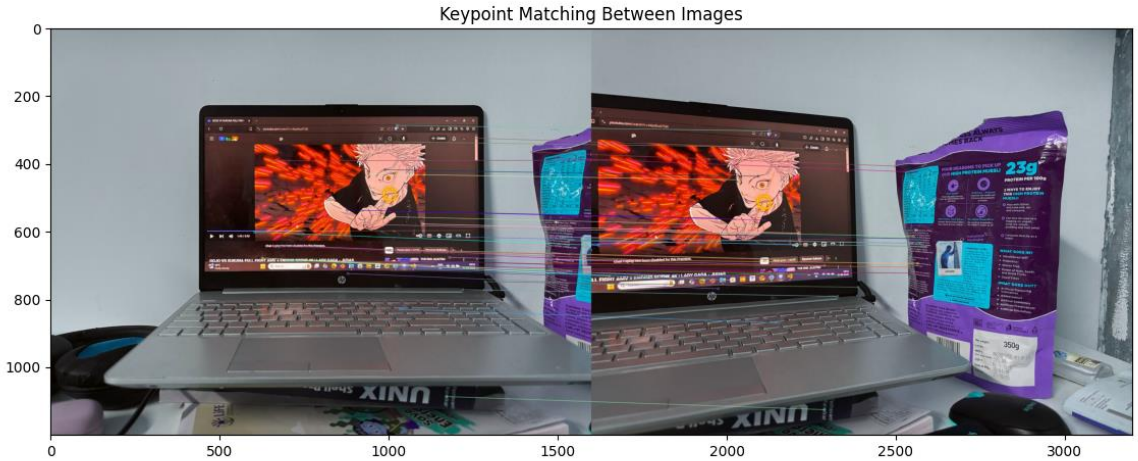
- OpenCV's Stitcher API (cv2.Stitcher.create()) is used to automatically merge the images into a panoramic view.
- The stitch() function attempts to align and blend the images based on detected features.
- If successful (status == cv2.STITCHER_OK), the final stitched image is displayed.
- If stitching fails, an error message is printed.

Overview:

- Grayscale conversion simplifies feature detection and reduces computation.
- SIFT is robust against scale, rotation, and illumination changes, ensuring accurate feature detection.
- BFMatcher efficiently finds correspondences between key points, improving alignment.
- Sorting matches ensures the best features are used for stitching.
- Panorama stitching combines images into a single seamless frame, useful for applications like landscape photography and 360-degree views.

This approach enables automatic image alignment and merging, producing a final stitched panorama if the input images have sufficient overlap.

Images:



Here, the first image shows how the 2 images look after stitching. In the second image, the two images left and right have been placed side by side. And the key points: the features that are common in both the images have been matched with a line.

Submission Details

All the details about the assignment have been uploaded in the repo

https://github.com/amruthg/VR_Assignment1_Amruth_Gadepalli_IMT2022065