VR ASSIGNMENT-1

Coin Detection and Segmentation

The following steps outline how to set up and run the program, along with methods used, results, and observations.

Setup and Requirements

Dependencies:

- Python 3.x
- Required libraries: OpenCV, NumPy, Matplotlib

Install dependencies using:

```
pip install opencv-python numpy matplotlib
```

Methods Used in the Code

CLAHE (Contrast Limited Adaptive Histogram Equalization): CLAHE was used to enhance the contrast of the grayscale image. It adjusted brightness locally by dividing the image into smaller tiles and equalizing their histograms while keeping noise amplification under control.

Bilateral Filtering: Bilateral filtering was applied to reduce noise without blurring edges. This method retained sharp edges by weighting nearby pixels based on both spatial proximity and intensity differences.

Canny Edge Detection: The Canny algorithm identified edges in the image. This process involved Gaussian smoothing, gradient computation, non-maximum suppression, and double-thresholding to isolate strong and weak edges effectively.

Morphological Operations (Closing): Morphological operations like "closing" were used to fix small gaps in edges by applying dilation followed by erosion with an elliptical kernel.

Contour Detection: Contours were extracted using `cv2.findContours` to isolate the shapes of coins. These were filtered based on their area and shape.

Circularity Check: To validate detected contours as coins, circularity was computed using the formula:

$$\text{Circularity} = \frac{4 \cdot \pi \cdot \text{Area}}{\text{Perimeter}^2}$$

Contours with circularity between 0.3 and 1.2 were considered valid.

Bounding Box for Segmentation: Bounding boxes were created around valid contours using `cv2.boundingRect`, allowing individual coins to be cropped and analyzed.

Visualization with Zoom Functionality: Matplotlib's scroll event handler enabled interactive zooming for inspecting detected coins and their edges in detail.

Workflow

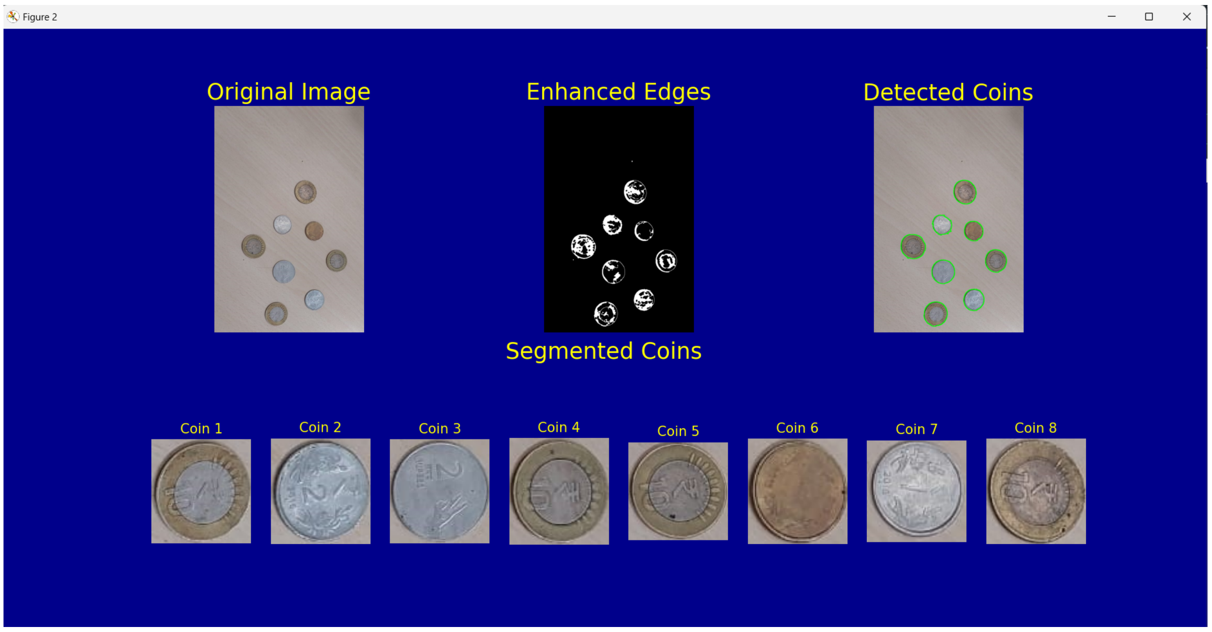
1. Preprocessed the image using CLAHE and bilateral filtering.
2. Detected edges using the Canny algorithm.
3. Refined the edges with morphological operations.
4. Extracted contours and filtered them by area and circularity.
5. Drew valid contours on the original image and segmented the coins.
6. Visualized the results, including the original image, edges, and segmented coins.

Results and Observations

The `images` folder in the `coin_detection` directory contains 9 images that users can choose from. For all these images, the code is able to correctly count and segment the coins. The following conditions were met for successful detection and segmentation:

- Uniform background.
- Proper lighting.
- All Indian coins.
- Images captured from a closer distance.
- Decent quality images.
- No other circular objects apart from coins.
- No reflections or light shining on the table.
- No overlapping of coins.

Users can also capture their own set of images, provided the above conditions are met and will get 100% accuracy.



figurecoin detection Output

Overall

The program combined advanced image processing techniques to efficiently detect and segment coins, focusing on preserving edges, reducing noise, and applying robust filtering criteria to ensure accuracy.

Panorama Creation

Dependency Installation

Dependencies were installed using the command:

```
pip install opencv-python opencv-contrib-python matplotlib numpy
```

Methods Used in the Code

ORB (Oriented FAST and Rotated BRIEF): ORB detected keypoints and extracted descriptors from images, analyzing intensity patterns and computing binary descriptors for matching points across images.

Feature Matching: Feature matching identified corresponding points in overlapping images, aligning features accurately for stitching.

Image Stitching: The `Stitcher` class was used to merge images into a seamless panorama by estimating homography transformations, warping overlapping regions, and blending them for smooth transitions.

Grayscale Conversion: Images were converted to grayscale using `cv2.cvtColor`, simplifying keypoint detection and descriptor computation.

Keypoint Visualization: Keypoints were visualized using `cv2.drawKeypoints`, ensuring meaningful features were correctly identified.

Image Resizing: The stitched panorama was resized using `cv2.resize` with `INTER_AREA` interpolation to maintain aspect ratio and optimize display quality.

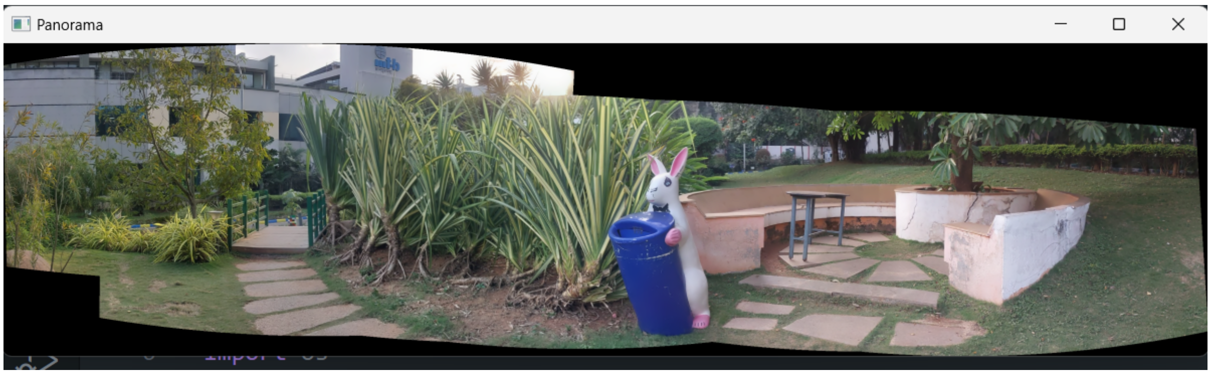
Error Handling: The program included error handling to manage scenarios where stitching failed, displaying appropriate error messages.

Results and Observations

The images were captured near the IIITB fish pond. A total of 7 images were taken by turning the camera slightly after each capture. The output panorama generated by the code is smooth, and no overlaps or visible artifacts are present in the final stitched image.

Summary

The code implemented a structured approach for panoramic image stitching, leveraging ORB for feature extraction, homography for alignment, and warping for blending. Each step ensured computational efficiency and robustness, resulting in high-quality outputs.



figurePanorama Output