VR_Assignment1_HarshShah_MT2024136

Coin Detection and Image Stitching to form Panorama

1. Coin Detection

- **Edge Detection**
  The image of coins was converted to grayscale and then smoothened using a *Gaussian filter* and the *Canny edge detection* was used to detect the edges in the image. The detected edges were overlayed on the original image.

- **Thresholding**\

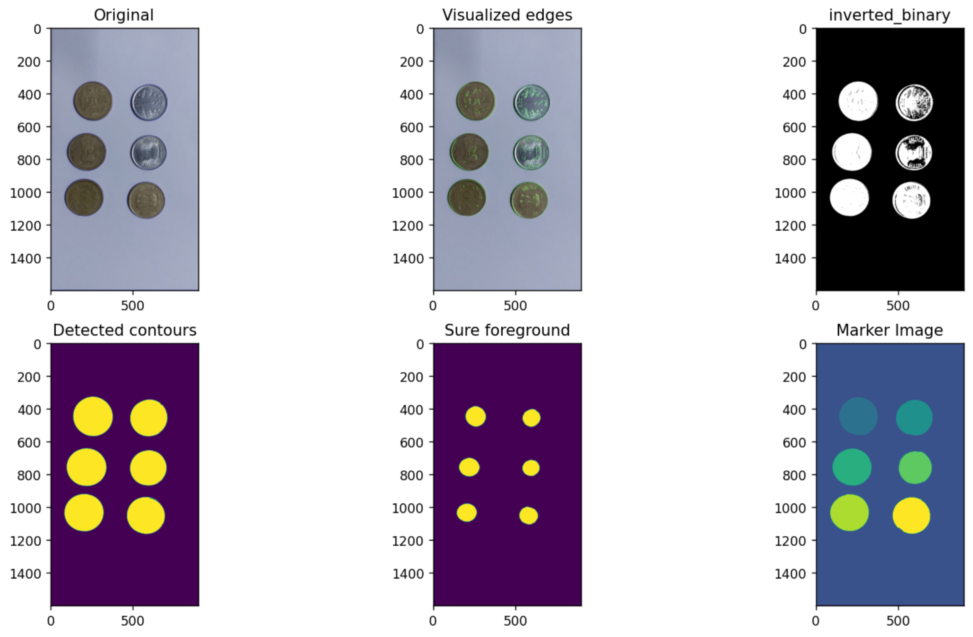
- Inverse Binary thresholding* was used to make the image binary with black in the background. Due to the designs on the coin there were some black patches inside the coin(white area) and they were not vanishing after morphological transformations, so *Contour detection* was used to eliminate them completely.

- **Contours detection and drawing**\

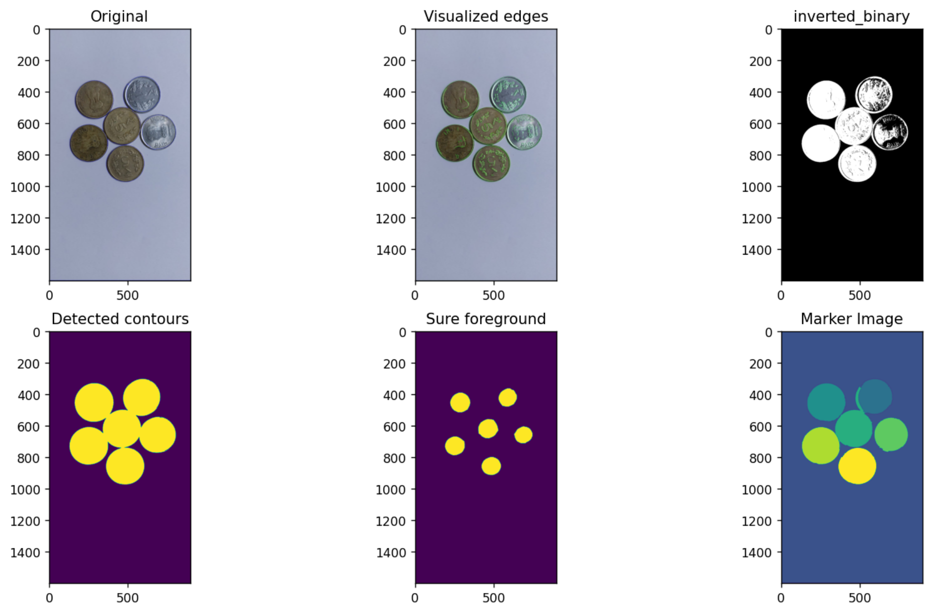
- Contour detection* was used to surround the coins and fill(cv2.FILLED) the gaps inside coins. Again, the noisy contours were rejected by a simple if condition checking if the area is greater than 500, giving us the *filtered_contours*. The final count of coins was equal to the number of filtered_contours. But this much processing is not enough for touching coins because we get only 1 contour in this so we will use watershed algorithm.

- **Watershed Algorithm**
  This algorithm was used to select the sure foreground(the area which surely represents the coins) lastly cv2.connectedComponents() was used to count the number of connected white pixels. Each connected component represents a coin. So the number of coins is equal to the number of components. The cv2.watershed() algorithm gives unique markers to each components.

- **Normal Image of Coins:**



- **Image of touching Coins:**



\

2. Image Stitching with Feature Matching and Feather Blending

Image Preprocessing

- Input images are read using OpenCV (`cv2.imread`).
- Images are resized using a **custom resize function** to adjust dimensions while maintaining aspect ratio.

Feature Detection and Matching

- **SIFT (Scale-Invariant Feature Transform)** detects keypoints and computes descriptors.
- **FLANN-based matcher** finds correspondences between overlapping images.
- **Ratio test (Lowe’s test)** filters good matches.

Homography Computation

- Using the good matches, corresponding points are used to estimate a **homography matrix** with **RANSAC**.
- This matrix transforms one image into the coordinate space of another.

Image Warping

- **Perspective transformation** warps images onto a common coordinate plane.
- Image boundaries are determined using **warped corner detection**.
- A **translation matrix** ensures all images align within a non-negative coordinate space.

Canvas Initialization

- A large blank canvas is created to accommodate the stitched images.
- The **reference image** is placed at its correct position.

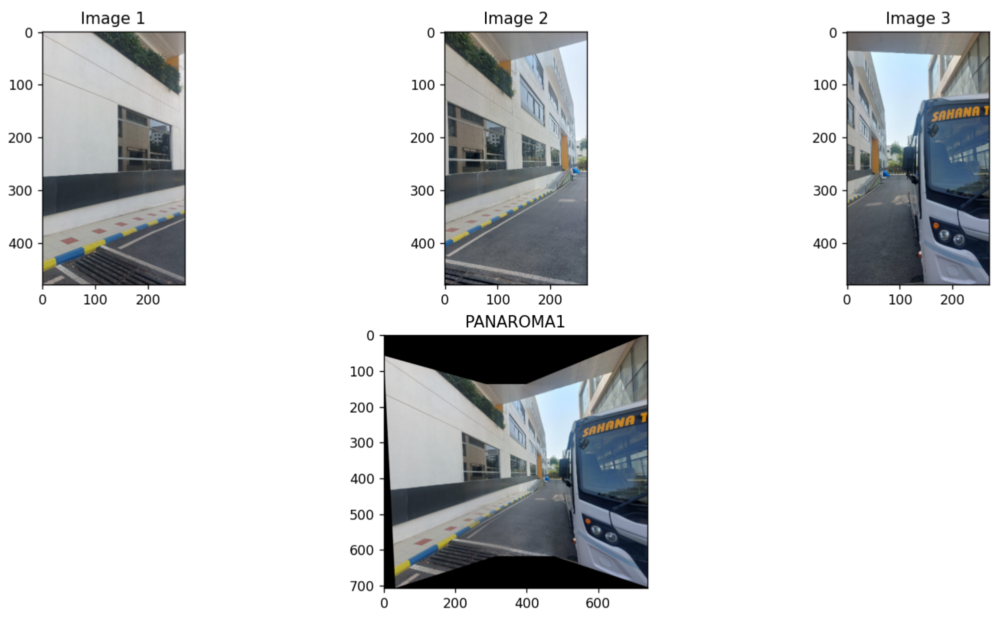
Feather Blending for Seamless Merging

- A **distance transform-based feather blending** method is applied:
  - Compute **distance maps** for overlapping regions.
  - Generate blending weights for smooth transitions.
  - Use weighted averaging to blend overlapping areas.

Visualization of Results

- **Matplotlib** is used to visualize:
  - The individual input images.
  - The final **stitched panorama**, spanning the entire width of the second row.

Output The final stitched image is displayed with a smooth transition between the images, ensuring minimal visible seams.



Dependencies Ensure you have the following installed:

- Python 3.x
- OpenCV (`cv2`)
- NumPy (`numpy`)
- Matplotlib (`matplotlib`)

Running the Code Run the script using:

```
python image_stitching.py
```

This structured approach ensures an **accurate and seamless panorama** by combining feature matching, homography-based warping, and feather blending.