Assignment 2, Report
COL 780, Semester 2, 2022-2023
Shashank Krishna Vempati (2022AIY7509)

Note: Most of the lines in code have been commented in single-line notes to make it easy to understand the logic and functionality.

Introduction :

The code is a Python script that implements the creation of a panorama from multiple images. The script reads images from a specified directory, extracts keypoints and descriptors, finds matches between images, and computes the affine transformation between each pair of images using RANSAC. Finally, the images are stitched together into a single panorama.

Libraries :

The following libraries are used in the script:

- cv2 (OpenCV): is an open-source computer vision library that includes several hundreds of computer vision algorithms.
- numpy: is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Directory Structure :

```
2022AIY7509
├── COL780_Assignment2_Report.pdf
├── Output
│   ├── Panorama_dataset1.jpg
│   ├── Panorama_dataset2.jpg
│   ├── Panorama_dataset3.jpg
│   ├── Panorama_dataset4.jpg
│   └── Panorama_dataset5.jpg
├── assignment2.py
└── requirements.txt
|____New_Dataset
```

Requirements :

```
Python version : 3.8.15
Numpy version : 1.23.2
OpenCV version: 4.5.5
```

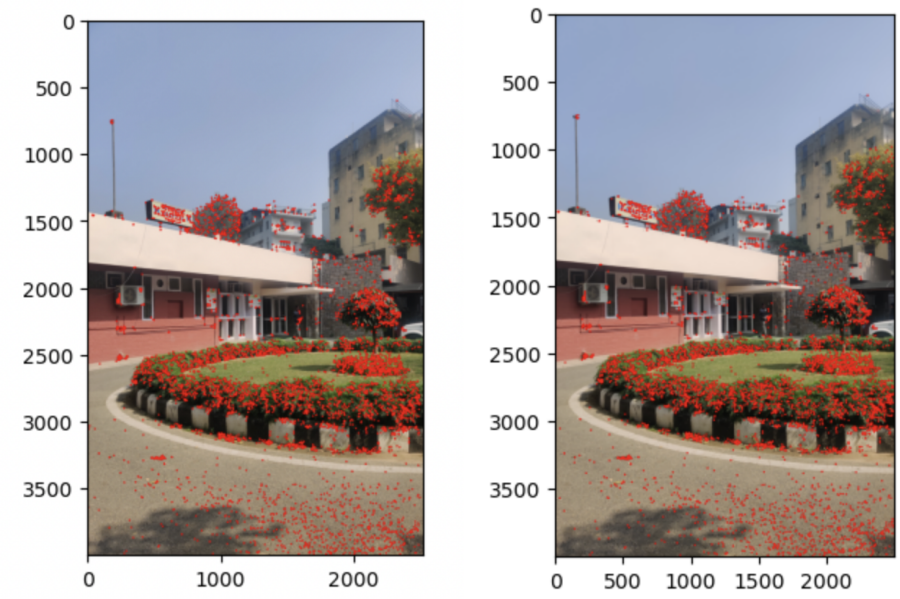
Running the code :

python assignment2.py path_to_directory_of_images output_image_name

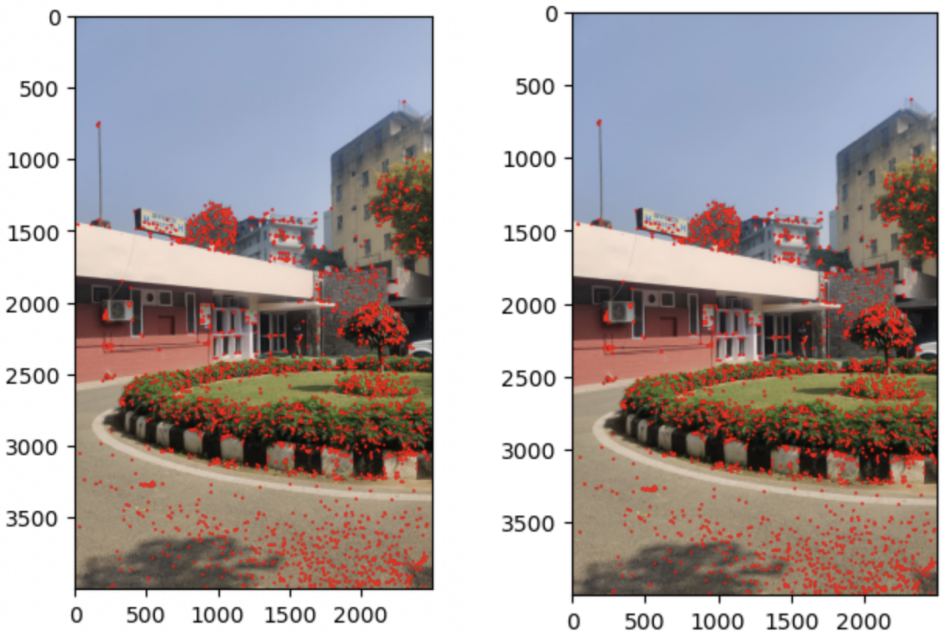
Example: python assignment2.py New_Dataset/1/ Panorama_dataset1

Steps :

1. The script first reads all the image files with the extensions ".jpg" or ".png" in the specified directory using OpenCV's cv2.imread() function. It then converts each image to grayscale using “cv2.IMREAD_GRAYSCALE” and stores them in separate lists.
2. The Hessian corner detection algorithm is used to extract the key points from the images. The get_corners() function implements this algorithm. The get_corners() function first applies a Gaussian filter to the image to smooth it. It then computes the image derivatives using Sobel filters and applies another Gaussian filter to the products of derivatives.
3. The first step in creating a panorama is to identify the corners in each image. To do this, we use specific hyper parameters that are tailored to the particular dataset we are working with. We then take a patch of pixels around each corner and attempt to match the corners by comparing these patches between the images.

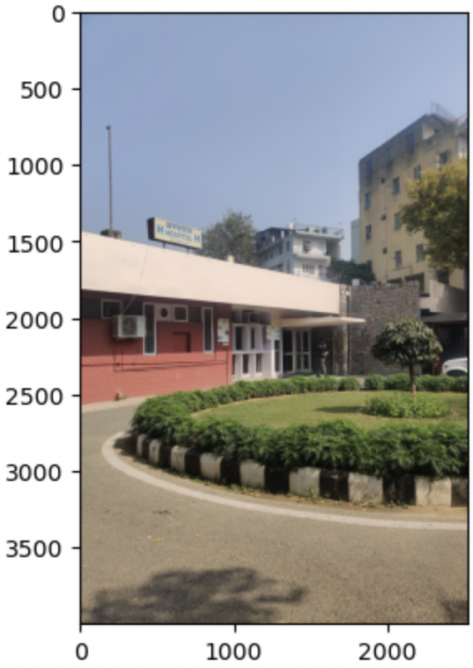


4. After matching the corners, we set a threshold to filter out matches that are too close together. This helps to ensure that only the most accurate matches are used in the final transformation. `Fig shows matching corners :`



5. Once we have a set of matching points, we use them to compute an affine matrix that describes the transformation required to align the two images. To do this, we use RANSAC, a robust estimation method, to eliminate outliers and obtain the best possible transformation matrix.

6. With the transformation matrix in hand, we apply it to one of the images and then stitch the two images together to create a seamless panorama. We repeat this process for all the images in the sequence, ultimately resulting in a complete panorama composed of all the images. `Panorama (Merging above two images):`



Thresholds used :

1. Corner response threshold :
   Threshold to eliminate the corners which are computed using corner response value
2. Search space hyperparameters :
   As the dataset is more oriented towards slight change in x axis, search space based hyperparameters are set in such a way that the corners are searched along the same axis with slight deviations along y axis and huge deviations along x axis. These parameters can be tuned or modified according to different scenarios. Affine properties include scaling, rotation and translation.

`Note: Based on these hyperparameters and other changes we can deal with different datasets. But the code is optimized to suit the needs of the dataset given.`

Pitfalls and Improvements :

There are few pitfalls to the above implementation that can affect the accuracy and robustness of the panoramic image generation:

1. Matching algorithm: The matching algorithm used to match feature points across images may not work well in all scenarios. In cases where there is significant distortion or occlusion, the matching algorithm may fail to find the correct correspondences, leading to poor quality panoramas.
2. Threshold setting: Setting the threshold for filtering out close corners may not work well in all cases. If the threshold is too high, then some true matches may be filtered out, whereas if the threshold is too low, then some false matches may be retained, leading to poor quality panoramas.
3. RANSAC parameters: The parameters used for RANSAC may not be optimal for all cases. If the number of iterations or the threshold for inliers is too low, then some false matches may be retained, whereas if they are too high, then some true matches may be filtered out, leading to poor quality panoramas.
4. Image distortion: In cases where there is significant distortion or perspective change, the affine transformation may not be sufficient to correct for these distortions, leading to poor quality panoramas.

To improve the above implementation, we can consider the following:

1. Use a more robust feature matching algorithm that is able to handle occlusion and distortion better, such as SIFT or SURF.
2. Use adaptive thresholding techniques to set the threshold for filtering out close corners, such as Otsu's thresholding.
3. Tune the RANSAC parameters based on the characteristics of the dataset being used.
4. Use more advanced techniques such as homography or perspective transforms to handle distortion and perspective changes.
5. Consider using multiple image stitching techniques to obtain the best possible panorama.

Reasonable results can be obtained by experimenting with different parameter settings and using appropriate techniques to handle various challenges such as occlusion, distortion, and perspective changes. Careful preprocessing of the images, such as correcting for lens distortion, can also help to improve the quality of the results.

Conclusion :

In conclusion, image stitching is a challenging task that involves several steps, including feature extraction, feature matching, and image transformation. In this process, there are many potential pitfalls that need to be addressed, such as incorrect feature matching, image distortion, and color balancing issues.

To overcome these challenges, various techniques have been developed, such as using RANSAC to eliminate outliers during feature matching, applying non-linear warping functions to account for image distortion, and using image blending to balance color and luminosity. Additionally, deep learning-based methods have shown great promise in recent years for image stitching, particularly for handling complex scenes and large-scale image datasets. Deep learning techniques will help us solve such issues with better accuracy.

Overall, image stitching is an important technique in computer vision that has numerous practical applications, from creating panoramas to producing high-resolution satellite images. With continued advancements in technology and research, we can expect to see even more sophisticated techniques and tools for image stitching in the future.