Challenge

April 21 U26267142

Design

In this challenge, the first task is to figure out how many coordinates are in the path. Then I found there are 801 images, and I created an empty path list first. It is necessary to add a zero coordinate to the list as every two images can generate one coordinate thus, we need an extra coordinate.

Second, a feature detector named the FAST algorithm for corner detection was used for each image before it goes through the loop. The most promising advantage of the FAST corner detector is its computational efficiency. When machine learning techniques are applied, superior performance in terms of computational time and resources can be realized. The FAST corner detector is very suitable for real-time video processing applications because of its high-speed performance. Then, I estimated the corners by using Lucas-Kanade optical flow. In two adjacent images, neighboring pixels have similar motions. It would output vectors of 2D points containing the calculated new positions of input features in the second frame. Both the functions above are used by importing the OpenCV library.

Next comes the step to calculate the essential matrix. At first, I completed a method with 8-points-algorithm manually. However, the scores on the leaderboard were lower than the baseline and I tried a lot but failed to boost the performance. Therefore, I decided to use the OpenCV library for the essential matrix and complete the pose recovery by myself.

The camera poses can be determined by the rotation and the camera center coordinates between two adjacent frames. The camera pose matrix can figure out the camera coordinates, which consists of three rotation vectors and three translation vectors. The camera pose matrix is already decided by the essential matrix as it gives the configuration that shows the camera positions and angles. Then I can calculate the singular value which is decomposed from the essential matrix. When we get the 4 possible camera poses, we triangulated 3D points to determine which is the real camera pose. As the projected point

must be in the camera's view, we need to compare linear least squares and check the depth of these points as well.

As we have already been provided with scale estimation for multiplying translation vectors, the final translation will be fixed by applying the rotation matrix. Meanwhile, these numbers are the coordinates of the camera in each image. The process iterates only until all frames have been processed. The path will finally be used to calculate mean square errors.

Improvement

As can be seen above, the Fast algorithm boosts not only the performance but also the processing speed. Compared to the BFmatcher, the Lucas-Kanade optical flow algorithm has a significant boost on the speed and the performance as well. The essential matrix from the OpenCV library is the most effective. Especially, its Ransac can efficiently reduce the outliners as long as we give it appropriate parameters. Another important improvement is the method called chirality condition in the pose recovery. Different camera poses can even result in the same projection point. Only the pose which can produce the most points is likely to figure out the correct rotation matrices and the translation vectors. Also, I added a method to estimate the camera pose by using the camera's parameters and the projection of 3D points on the image plane.

Result

The implementation by using the OpenCV library completely achieved the average MSE of 5.17 and scored 102 points on the leaderboard. By contrast, the version with manual pose recovery achieved the average MSE of 5.48 and scored 120.6 points on the leaderboard. It must be admitted that the latter is the result of many functions tuning parameters and not just the manual realization of pose recovery.