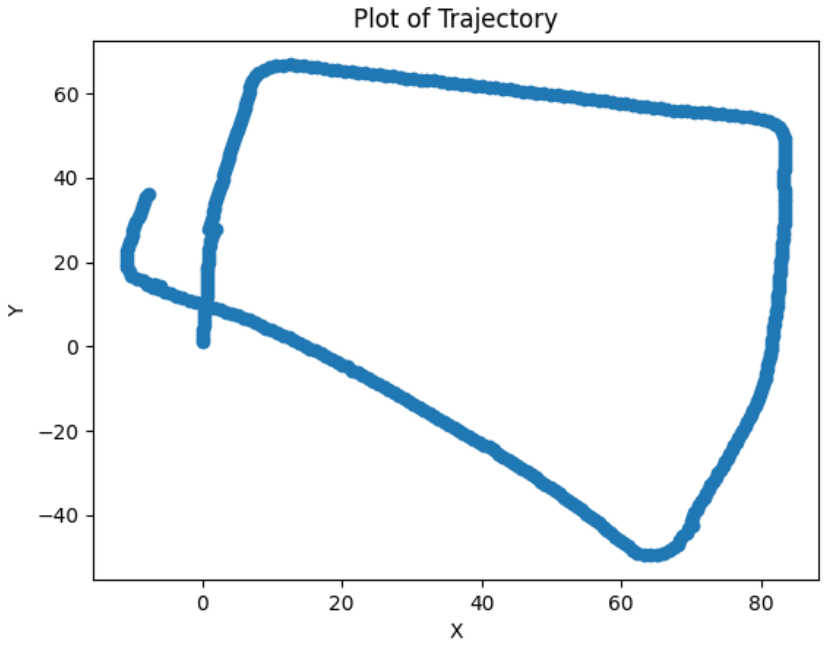
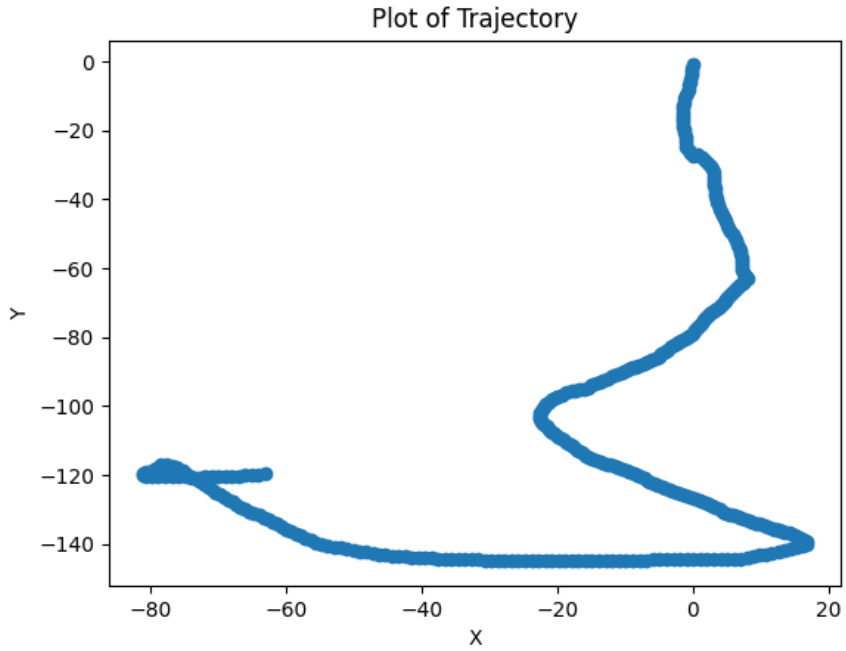
The following are the results I got for the final assignment. The first one used the cv2.recoverPose function, the second one used replacement recoverPose implemented by myself.





The following are detailed descriptions of my implementation of Visual Odometry:

Following the documentation provided, first we are using ReadCameraModel to extract the camera parameters. And compute the camera's intrinsic matrix K. Here, fx, fy, cx, cy are the camera's intrinsic parameters representing the focal length and principal point coordinates.

We then save all images' names to the array in order to read them after. Using a for loop to go over all the images. For each loop, we read two continuous images to find match points between them.

For the keypoint correspondences part, part of the code is from geeksforgeeks. We first create a SIFT object using cv2.xfeatures2d.SIFT_create() to detect keypoints and compute descriptors. Then, detect keypoints and compute descriptors for both img1 and img2 using the SIFT object's detectAndCompute method. We are using the FLANN matcher to find the nearest neighbor, one of the parameters for this matcher is FLANN_INDEX_KDTREE, which tells cv2.FlannBasedMatcher the algorithm we are using. After that, using knnMatch with parameters we got from sift.detectAndCompute to find matches. At last, we use ratio test to drop bad matches and save good matches to two arrays called pst1 and pst2, respectively. After checking this method separately, we can efficiently find corresponding keypoints. And we are good to do the next part.

We are then using the key points just identified to find the estimate fundamental matrix using cv2.findFundamentalMat(pts1, pts2, cv2.FM_RANSAC), where the parameter cv2.FM_RANSAC specifies the method to be used for computing the fundamental matrix. RANSAC (Random Sample Consensus) is a robust estimation method that can handle outliers in the matched keypoints. Using some numpy operations(K.T @ F @ K) to F and K we obtained previously to recover the essential matrix E.

For the next step, we use cv2.recoverPose(E, pts1, pts2, K) to reconstruct rotation and translation parameters from E. It first decomposes the essential matrix E into the SVD form: E = U * S * VT, where U, S, and VT are the matrices obtained from the SVD. Then do some operations to ensure that the essential matrix satisfies the mathematical requirements. Then, we can compute the rotation and translation and return it.

For reconstruction of the trajectory part, we just follow the documentation hint to do some matrix operations. We first do matrix transformation from first image to second image by horizontally stack R and t, then vertically stacks the previous result to [0,0,0,1] to compute a 4*4 homogeneous transformation matrix. Then we do the multiplication and extract the required points from that result. And we are done with this project!! Also, don't forget to plot the trajectory. My result at the beginning of this report is similar to figure 1 in the project documentation, which means our implementation did a pretty good job.