ENPM673: Perception for Autonomous Robots

Project 5

Umang Rastogi UID: 116773232

Sayani Roy UID: 116818766

Prateek Bhargava UID: 116947992

Date: 4 May 2020

1. Introduction

Visual Odometry is used to determine the position and orientation of a robot by analyzing the camera images associated to the robot. It is an integral part of perception. Moreover, its concepts are analogous to that of SLAM.
In this project, we were provided an image dataset from a camera mounted on a car. We have implemented the necessary algorithms involved behind visual odometry to plot the trajectory of the camera.

2. Pipeline

2.1 Data Preparation

Before implementing the various algorithms involved in visual odometry, the dataset has to be processed and converted into the correct format for feature matching.

- First, we extract the various camera parameters such as focal lengths, principal points, and the look-up table using the function in given *ReadCameraModel.py*.
- The camera parameters, focal lengths and principal points, are used the calculate the camera matrix, K.
- We now read the dataset in gray-scale and convert the images from the Bayer to BGR format.
- After converting the images into BGR format, they are undistorted using the function provided in *UndistortImage.py* and the look-up table retrieved while extracting camera parameters.
- We save the undistorted images during the first run to reduce processing delays during the implementation of the basic pipeline.
- Please note that the functions provided in *ReadCameraModel.py* and *UndistortImage.py* have been added in *Code/utils/data-prep.py* without any modifications.

2.2 Basic Pipeline

The basic pipeline can be further divided into 5 major sections, namely, feature matching, evaluation of fundamental matrix using RANSAC, computation of the essential matrix, estimation of the camera pose, and plotting of the camera pose.

- We iterate through the recently saved undistorted image dataset.
- Each iteration involves reading 2 image frames, the current and the next frame.
- The images are read in gray-scale mode and feature extraction is employed on each of them using SIFT.
- We get around 3000 features for each image frame which are matched using KNN match employed via a FLANN based matcher.
- The locations of these matched features are used to evaluate the fundamental matrix.

- We now employ RANSAC with 70 iterations to get an optimum fundamental matrix with the most number of inliers.
- We calculate a fundamental matrix for each of the iterations using the equation:

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx_m' & x_my_m' & x_m & y_mx_m' & y_my_m' & y_m & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

- We have set an epsilon value of 0.01. We evaluate the fundamental matrix for each of the random selected feature based on the inequality:

$$x_1{}^T F x_2 < \epsilon$$

- The fundamental matrix with the most inliers during the 70 iterations is our final fundamental matrix for the current set of image frames.
- Now, we calculate the essential matrix, E, using the equation: $E = K^T F K$.
- We eliminate noise from the essential matrix as the fundamental matrix is just an estimation using RANSAC. We take the SVD of the essential matrix obtained and regenerate the essential matrix by changing its singular values to (1,1,0).

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

- We estimate the camera pose using the essential matrix and inliers found while employing RANSAC to find the fundamental matrix.
- We then compute the 4 possible combinations of the rotation and translation matrices. Note that the pose of the camera is a combination of rotation and translation matrix.
- After that, we used linear triangulation on these rotation and translation matrices to estimate the shift in pose of the camera with respect to the previous frame.
- We perform cheirality check on the rotation and translation matrices using the condition:

$R_3(X - C) > 0$, where R_3 is the 3^{rd} row of the rotation matrix

- We generate a homogeneous matrix from the final rotation and translation matrix retrieved from the function *disambiguate-camera-pose*.
- This homogeneous matrix is post-multiplied with the homogeneous matrix of the previous frame to get the final homogeneous matrix.

- The first 3 rows of the last column of the homogeneous matrix gives us the final estimated translation.
- We plot the trajectory of the camera using x and z coordinates of the translation.

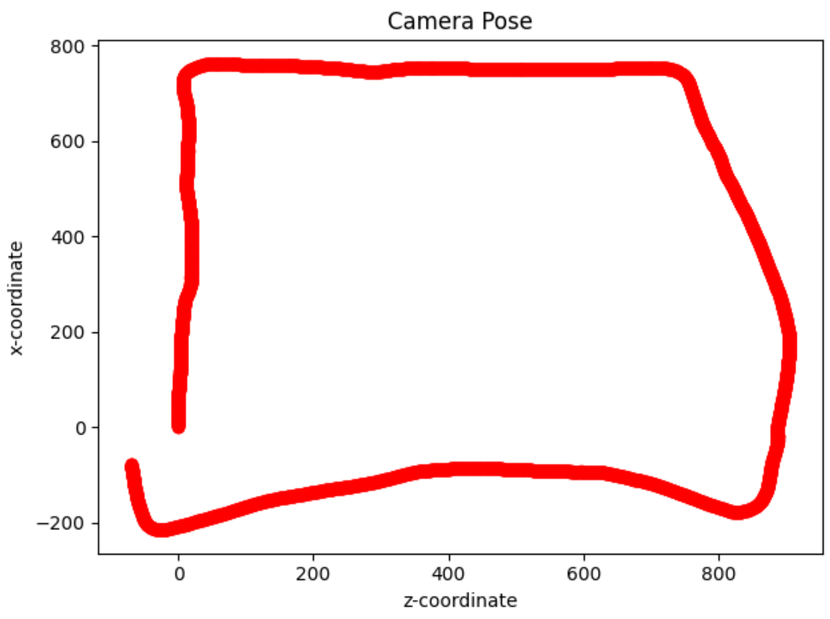

Figure 1: Trajectory of the camera using custom-built functions

2.3 Implementation using OpenCV Function

We compare our results against rotation/translation parameters using *cv2.findEssentialMat* which obtains the points needed to calculate the accumulated drift from the current fundamental matrix and the next fundamental matrix. The function *cv2.recoverPose* finds the final camera position using the image frames. The accumulated drift is precisely calculated by comparing the user defined function and the opencv function. The optimal value of the fundamental matrix using RANSAC and then a robust algorithm such as 8 point algorithm is used to get the essential matrix and the fundamental matrix accurately from a set of corresponding image points.
The figure below depicts the accumulated drift per frame in the pipelines as compared to built in values.

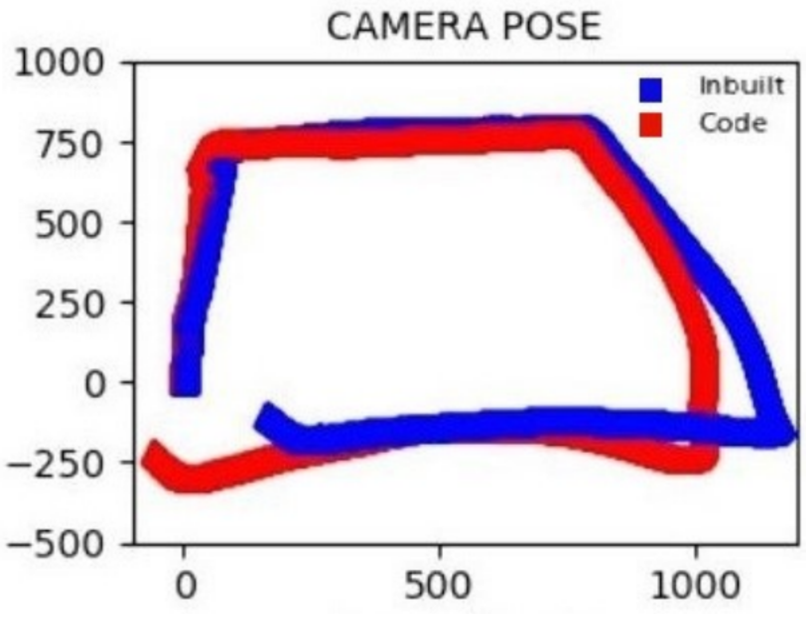

Figure 2: Comparison with OpenCV in-built functions

Please note that our final drift between the 2 trajectories came out to be 58176.447.

2.4 Non-Linear Triangulation

After obtaining the camera poses, we triangulate the 3-D points. This is done using the cheirality condition which is performed to find the correct unique camera pose. The steps performed to find the cheirality condition and Non-Linear triangulation are as follows:

- We check the depth of the 3-D points and triangulate them using the least squares function imported from scipy package.
- The sign of the depth of Z in the camera co-ordinate system with respect to the camera center frame depicts the cheirality condition. The pose (R,C) produces the maximum number of points which also satisfies the cheirality condition.
- We then reduce the error in reprojection of 3-D points using non-linear triangulation by using the camera poses and 3D triangulated points which were calculated linearly, rather than just performing linear triangulation which only reduces the algebraic error.
- The formula given below minimizes the geometrical error:

$$\min_{x} \sum_{j=1,2} \left(u^j - \frac{P_1^{jT} \widetilde{X}}{P_3^{jT} X} \right)^2 + \left(v^j - \frac{P_2^{jT} \widetilde{X}}{P_3^{jT} X} \right)^2$$

  Here, j is the index of each camera, X˜ is the homogeneous representation of X. P is each row of camera projection matrix, P. We estimate initial guess of the solution, X_0 via linear triangulation to minimize the cost function.

- We then refine the locations of the 3D points that minimizes reprojection error using the two camera poses and linearly triangulated points.

3. Problems encountered

The problems faced are as follows:

- The time taken to get the undistorted images is large due to the high number of images present in the dataset. Therefore, we saved the processed images obtained from the initial run of the code and then retrieve them every time from the saved directory whenever the code is rerun again.
- Many blogs and tutorials suggest the use of opencv functions SIFT and FLANN based matcher for feature extraction and matching specially for visual odometry. We observed that the latest opencv package does not include these and thus we had to downgrade our opencv package to version 3.4 for our project.
- The fist few frame are very bright to detect any significant features, especially the ones after the 5^{th}frame till the 17^{th}frame. Therefore, we have started employing the pipeline starting from the 18^{th}.
- It was hard to find documentation on linear triangulation and its implementation online, even though the tutorial provided gives an understanding on the working of the same.

4. Links

GitHub repository link : Motion-Estimator-3D

5. References

- CMSC-733 Project-3 Guildelines
- OpenCV Epipolar Geometry Tutorial
- Evaluation of Fundamental Matrix