ENPM673 - Perception for Autonomous Robots Project 5

Sandeep Kota Sai Pavan
116911603
University of Maryland
College Park, MD - 20740
Email: skotasai@umd.edu

Satyarth Praveen
116752749
University of Maryland
College Park, MD - 20740
Email: satyarth@umd.edu

Revati Naik
116723015
University of Maryland
College Park, MD - 20740
Email: revatin@umd.edu

I. Introduction

This project is about finding the Odometry of the moving vehicle using the output of the camera. The techique is called Visual Odometry. It is essential for estimating the trajectory of any moving robot/vehicle. Here, we have a single calibrated moving camera and the relative pose between the camera positions is unknown. This project depicts the use of various steps to determine the camera pose w.r.t the given scene and plot the trajectory of the camera.

II. Data Preparation

The data set provided consists of image frames in Bayer format which are converted into color images and used further for processing. With the provided ReadCameraModel.py, we can extract the camera parameters f_x, f_y, c_x, c_y. The Camera Matrix which is given by Equation 1 is computed with the values obtained from the above given file.

$$CameraMatrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The Camera Matrix for the given data set is

$$K = \begin{bmatrix} 964.828979 & 0 & 643.788025 \\ 0 & 964.828979 & 484.40799 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Next, the image frames are undistored as a part of the data preparation pipeline. The undistored image frames are further used in the main pipeline.

III. Basic Pipeline

The translation and rotation between two successive frames and the camera pose is calculated using the the given steps.

1) Feature Matching to find point of correspondence between two successive frames.
2) Calculation of Fundamental Matrix F using Eight-Point Algorithm with RANSAC for outlier rejection.
3) Calculation of Essential Matrix E using F and Camera Matrix.
4) Estimating Camera Pose from the E which leaves us with 4 possible camera poses.

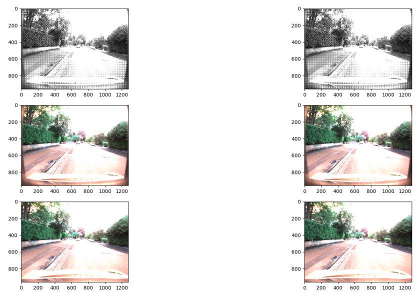


Fig. 1: (a) Read Input Image (b) Color Image (c) Undistorted Image for current and next image frame

5) Find the correct Camera Pose, Triangulate the 3D points to check for depth positivity and then choose the correct pose by checking for Cheirality Condition .
6) Plot the camera center pose and the trajectory of the camera.

A. Feature Matching

To find the point of correspondence between the two consecutive frames, we extract the features which match in these frames. This step is implemented using the SIFT feature extraction algorithm. We obtain the points x and x' where x lies in the first frame and x' is its point of correspondence in the next frame. Refer to Figure 2 where the first 150 matches have been shown.

B. Fundamental Matrix

The fundamental matrix F is a 3x3 matrix which is defines the relation between the two set of points from the image frames. It is calculated using the points x and x'. F is calculated such that $x'^T F x = 0$. It is defined by Equation (3).

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{3}$$

If i is the image frame number, (x_i, y_i) are the coordinates in image frame 1 and (x'_i, y'_i) are the coordinates of the point

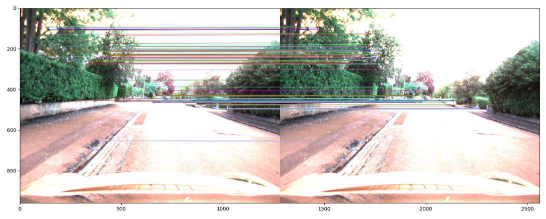


Fig. 2: Feature Matching using SIFT

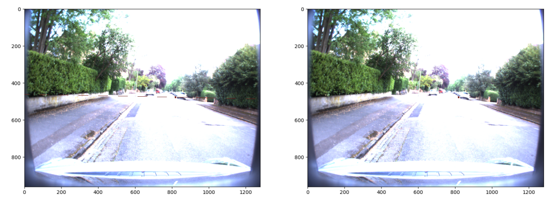


Fig. 3: Epipolar Line Plot

correspondence in the next frame. And since the Fundamental Matrix has 8 degrees of freedom, we require 8 points to solve this equation (Eight-Point Algorithm).

The above equation can be simplified as

$$x_i x_i' f_{11} + x_i y_i' f_{21} + x_i f_{31} + y_i x_i' f_{12} + \\ y_i y_i' f_{22} + y_i f_{32} + x_i' f_{13} + y_i' f_{23} + f_{33} = 0 \tag{4}$$

Normalizing the Image Coordinates: We normalize the image coordinates to make sure there is no unnecessary offset or scaling factor. Thus, we shift the coordinates at the image centre.

Equation (4) calculated for every point and its corresponding point is then stacked to create the a matrix A and we obtain the equation $Ax = 0$ which is solved using SVD. On decomposition using SVD, the last column of V is the solution for F. Due to the noise in the point correspondences, we obtain F with a rank 3 which violets the Eight-Point Algorithm, hence we enforce the rank to 2 by assigning the last singular value of the calculated F matrix to 0. We recalculate F after this step and then perform de-normalization. The code in $EstimateFundamentalMatrix.py$ takes the feature points as the input and calculates the fundamental matrix F. This F value is further used to calculate the Essential Matrix.

Outlier Rejection with RANSAC: The features obtained from feature matching step contains several outlier points which might affect the fundamental matrix. To remove these outliers, we use the RANSAC algorithm to obtain a better estimate for the F matrix. We implement a function in $GetInLierRANSAC.py$ which takes all the feature points and returns back the inliers by using the error from the Fundamental Matrix calculation. This function outputs the list of inlier feature points. This is then used to calculate the corrected F matrix.

Plotting of Epipolar Lines: To verify the correct calculation of the Fundamental Matrix F, we plot the epipolar lines using F and the feature points. The epipolar line for the image frame 1 is drawn on the image frame 2 which passes through the point of correspondence x' in image frame 2 for point x in image frame 1. This verifies that the calculated F is correct. Refer to Figure 3 which shows the epipolar line for the image frame 1 in image frame 2.

C. *Essential Matrix*

For calculating the Essential Matrix, we need to have the Camera Matrix (K) which is already computed in Section II. Refer to Equation (2). Essential Matrix is a 3x3 matrix given by,

$$E = K^T F K \tag{5}$$

We solve for E using SVD and decompose it into U, S, V.

$$U, S, V^T = svd(E) \tag{6}$$

Due to noise factor, we assign the singular value matrix of E to be $(1, 1, 0)$.

$$S = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \tag{7}$$

We then recalculate E for the updated singular value matrix. The function in $EsstimateEssentialMatrix.py$ takes in the Camera Matrix K and the Fundamental Matrix F and calculates E.

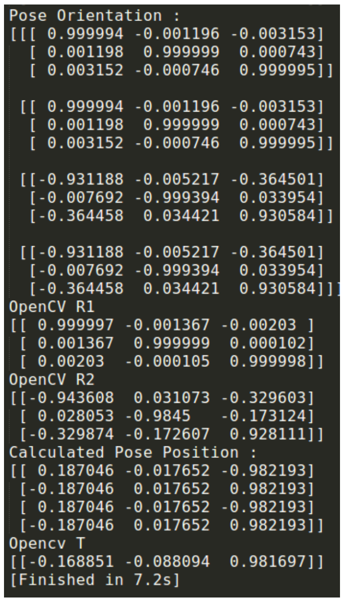
D. *Camera Pose*

The camera matrix is composed of the camera position and the camera orientation. E matrix is used to identify the camera pose. The function in $ExtractCameraPose.py$ takes in E and provides with four camera pose configurations $(C_1, R_1), (C_2, R_2), (C_3, R_3)$ and (C_4, R_4). While decomposing the matrix E using SVD (refer Equation (6)), we obtain U and V which are used for the calculation of the position and orientation of the camera. The four configurations are computed using the below given equations.

1) $C_1 = U(:, 3)$ and $R_1 = UWV^T$
2) $C_2 = -U(:, 3)$ and $R_1 = UWV^T$
3) $C_3 = U(:, 3)$ and $R_1 = UW^TV^T$
4) $C_4 = -U(:, 3)$ and $R_1 = UW^TV^T$

where W is given by $\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Camera pose correction is also required when the $det(R) = -1$. In this case, we correct the camera pose by changing the signs of C and R i.e. $C = -C$ and $R = -R$.



```
Pose Orientation :
[[[ 0.999994 -0.001196 -0.003153]
  [ 0.001198  0.999999  0.000743]
  [ 0.003152 -0.000746  0.999995]]

 [[ 0.999994 -0.001196 -0.003153]
  [ 0.001198  0.999999  0.000743]
  [ 0.003152 -0.000746  0.999995]]

 [[-0.931188 -0.005217 -0.364501]
  [-0.007692 -0.999394  0.033954]
  [-0.364458  0.034421  0.930584]]

 [[-0.931188 -0.005217 -0.364501]
  [-0.007692 -0.999394  0.033954]
  [-0.364458  0.034421  0.930584]]]
OpenCV R1
[[ 0.999997 -0.001367 -0.00203 ]
 [ 0.001367  0.999999  0.000102]
 [ 0.00203  -0.000105  0.999998]]
OpenCV R2
[[-0.943608  0.031073 -0.329603]
 [ 0.028053 -0.9845   -0.173124]
 [-0.329874 -0.172607  0.928111]]
Calculated Pose Position :
[[ 0.187046 -0.017652 -0.982193]
 [-0.187046  0.017652  0.982193]
 [ 0.187046 -0.017652 -0.982193]
 [-0.187046  0.017652  0.982193]]
Opencv T
[[-0.168851 -0.088094  0.981697]]
[Finished in 7.2s]
```

Fig. 4: Pose values from our algorithm and OpenCV function

E. Triangulation Check for Cheirality Condition

The task in this step is to find the correct solution among the possible configurations mentioned above. So pairs of these configurations are used to compute the depth for the two camera frames. The configuration that results is the largest positive depth is chosen to be the solution for the Essential matrix decomposition into Rotation matrix and Translation vector.

The cheirality condition is the check whether the depth of the reconstructed 3D point lies infront of the camera. The condition for check is given by $r_3(X - C) > 0$. Here, r_3 is the third row of the rotation matrix R. The function in $DisambiguateCameraPose.py$ takes in the camera matrix and the four camera pose configurations and returns back the correct Camera pose (position and orientation). Given the noise in point correspondence between the two frames, not all points are expected to satisfy this condition but the idea is to use the configuration that leads to maximum number of points satisfying the criteria.

This Camera pose is also cross verified with the output of the OpenCV function $cv2.recoverPose()$. Refer to Figure 4.

F. Plotting the Trajectory

The rotation and translation matrix are multiplied to plot the camera pose to obtain the trajectory of the camera.

IV. Results

Even though we followed the exact steps mentioned in the PRG website, we could not get the correct pose of the camera after the cheirality condition. It can be seen that the correct pose of the camera is present in one of the 4 poses that are generated from the $ExtractCameraPose.py$, by printing the generated outputs. It can be visually verified by the numbers. Also there were other problems faced while executing the openCV's functions. The in-built functions of openCV also seemed to be giving us wrong pose values for the cameras. The plot of the output generated by our implementation for the first 500 images (i.e straight line) is shown in the figure 5.

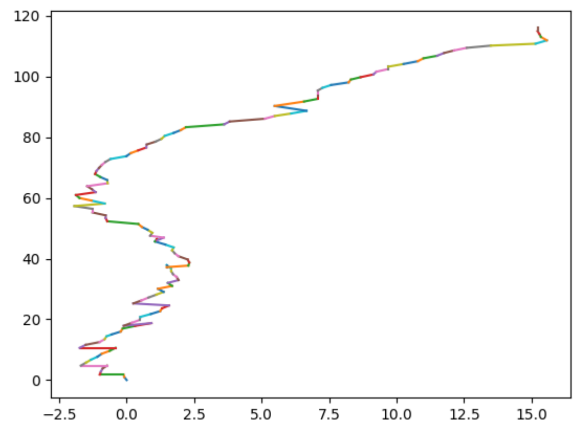


Fig. 5: Result from our implementation

References

[1] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[2] Anisha Gartia. Camera calibration and fundamental matrix estimation with ransac.