Project 3 – Visual Odometry

Submitted in partial fulfilment of the requirements for the course of

ENPM673 – Perception for Autonomous Systems

By

Adheesh Chatterjee

Andre Ferreira

Pablo Sanhueza

Course Faculty
Prof. Cornelia Fermuller

A. James Clark
School of Engineering

INTRODUCTION

The aim of this project is to implement the different steps to estimate the 3D motion of the camera and provide as output a plot of the trajectory of the camera. The concepts learned in class will allow us to write a script that solves the problem of visual odometry. As the car moves around the city, we track the change in position of the camera with respective to the initial point. This is accomplished with the different methods explained and provided in the pipeline of the project guidelines. Below you will see explanations of what we did to accomplish such an arduous task.

DATA PREPARATION

Since the input images were in Bayer format, we convert it into RGB scale using cv2.COLOR_BayerGR2BGR. We then extract the camera parameters from ReadCameraModel.py and undistort the current frame and the next frame using UndistortImage.py

Now that our data is ready to be processed, we start to implement our pipeline for Visual Odometry.

One very notable thing we did, was after the above steps, we saved the images out to a folder, so it would speed up our processing for the rest of the video. Although this step doesn't really affect the general pipeline, it does slow down the code considerably.

Sift Detection

Scale Invariant Feature Transform (Sift) is a feature detection used in computer vision to detect features in images. Sift detects key points in an image and stores them in a database. These keypoints are recognized in another image by individually comparing each feature from the new image with the ones stored in the database and finding possible candidates. Initially, we were applying Sift to the whole image and the plot of the trajectory of the car was working as expected. However, we noticed that there were a lot of corresponding points that were located in the front part the car whose location was not changing from frame to frame and did not affect the calculation of the fundamental matrix. It is important to refine the matches by rejecting outline correspondence.

We made sure these points were not affecting our result, where this was done by implementing RANSAC which filtered out the outliers. In order to better optimize our code, and after discussing the same with the TAs, we placed a mask in the front part of the car so that those points aren't really detected and hence making our code faster. We choose this approach since Sift and the functions provided for camera calibration and undistorted image were making the code slow.

Fundamental Matrix

The Fundamental Matrix is a 3x3 matrix which relates the corresponding points in stereo images , where x and x' are the cordinates of the matching points in each image and FX describes the epipolar line where the point x' must lie in the other image. All corresponding points must hold the following equation:

$$x'^{T} F X = 0$$

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

The Fundamental matrix must have rank two and it is necessary to get at least 7 correspoding points to calculate it. In this Project we used the 8 point algorithm to find it.

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

We can stack the above equation in a matrix A forming the equation:

$$Ax = 0$$

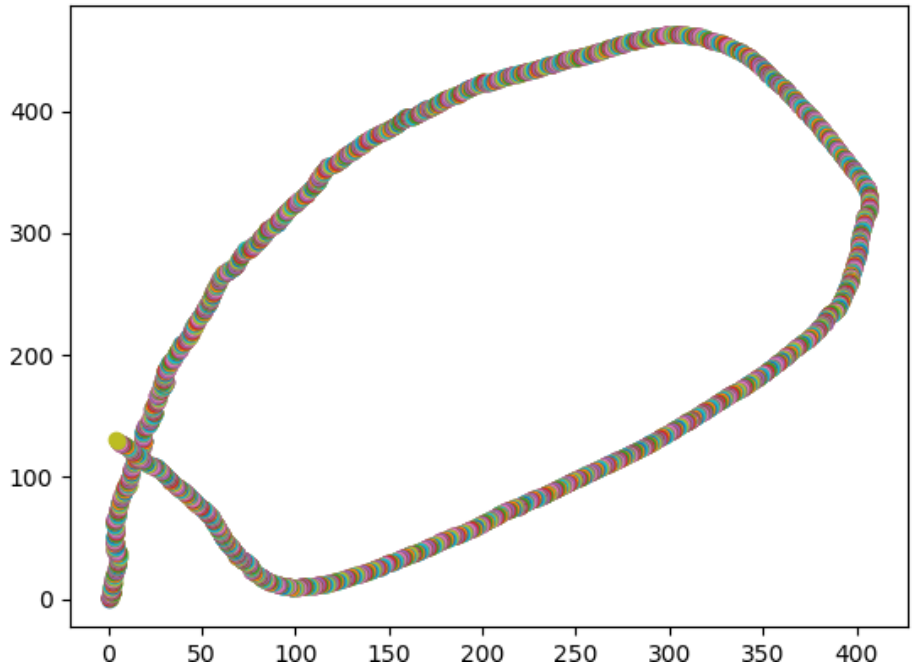
This way F can be obtained by solving the linear least squares using Singular Value Decomposition . Applying SVD to matrix A, it is decomposed to USV^T, where the last column of V is the solution. However, since the fundamental matrix must have rank 2 it is necessary to perform another SVD to the matrix and set the last singular value to 0 and transform back to get the final F matrix.

For our Project, we first obtained the fundamental matrix as explained above, however it was not giving us good results as compared to the inbuilt fundamental matrix function from opencv. In order to make it more accurate we searched for the source code in C++ of the inbuilt method to try and understand how they were perfoming the calculation. After analyzing the source code, we realized that they were normalizing all the points around the mean of the points and enclosing them at a distance of $\sqrt{2}$ from the new center location and then homogenizing it by dividing F by the last element of the matrix. After following a similar procedure in python we were able to get better results.

RANSAC

After using Sift to find corresponding points between two frames, we get a lot of matching points including outliers, which if used to perform the calculation for the fundamental matrix would not give us a good result. To fix this problem, we used Ransac the Ransac algorithm to find the best fundamental matrix.

We had some problems with Ransac too. Firstly, we were perfoming Ransac calculating the error with the traditional equation $x'^T FX = 0$ and checking which F would give us the best inliers but again it was not giving good results and when comparing with the inbuilt function the values were very distinct.



For this case we were not able to find the source code of how the inbuilt fuction works. To solve this issue, we looked for different methods to calculate the error, finally choosing the Sampson Error which gave us a better result.

The Sampson error is a better error measurement the sampson distance that is a first order approximation to the geometric error .The following equation computes the Sampson distance of a point correspondence to the corresponding epipolar line of the point correspondence in the other image.

$$\frac{(x^T FX)^2}{(FX)_1^2 + (FX)_2^2 + (F^T X')_1^2 + (F^T X')_2^2}$$

Where $(FX)_z^2$ is the square of the z-ith element of the vector FX.

So we get 8 random matched points using the 8-point algorithm and use those points to form the Fundamental matrix. Then we use the Sampson equation, with the F matrix and every single detected point, we set a threshold value and use it to detect the number of inliers. We

then iterate to detect another set of 8 points and follow the same procedure. Finally, we use the F matrix which gives us the max inliers.

Essential Matrix

The Essential matrix is also a 3x3 matrix, which relates corresponding points in stereo images if the cameras satisfy the pinhole camera model. Both matricies can be used to stablish constrains between corresponding points, however the essential matrix only can be used in relation to calibrated cameras, once the the intrisic parameters of the câmera must be know in order to achieve the normalization.
The essential matrix can be used to determine the relative position and orientation between the camera and the 3D position of the image matching points.

The calculation of the Essencial matrix can be given by the following equation:

$$E = K^T F K$$

Where :

- F is the fundamental Matrix
- K is the câmera Calibration Matrix

The singular values of the Essential Matrix are not goint to be necessarily (1,1,0), due to the noise in the câmera matrix .This can be fixed by decomposing E using SVD and reconstructing it with the singular values as (1,1,0).

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

ESTIMATE CAMERA POSE FROM ESSENTIAL MATRIX

The camera pose consists of 6 degrees-of-freedom (DOF) Rotation (Roll, Pitch, Yaw) and Translation (X, Y, Z) of the camera with respect to the world. Since the E matrix is identified, the four camera pose configurations: (C1,R1), (C2,R2), (C3,R3) and (C4,R4) where C is the camera centre and R is rotation matrix. We can calculate the 4 poses from the E matrix. Let

$$E = UDV^T \text{ and } W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1. $C_1 = U(:,3)$ and $R_1 = UWV^T$
2. $C_2 = -U(:,3)$ and $R_2 = UWV^T$
3. $C_3 = U(:,3)$ and $R_3 = UW^TV^T$
4. $C_4 = -U(:,3)$ and $R_4 = UW^TV^T$

Since we know that the determinant of a Rotation matrix must always be 1, we can check the 4 rotations obtained. We thus realize that the four poses are as follows –

1. Both cameras are looking at the same point
2. Both cameras are facing away from the same point
3. The first camera is looking at the point and the first is facing away
4. The second camera is looking at the point and the first is facing away

Here, we use the analogy of two cameras, to basically explain the the position of the same camera in 2 consecutive frames.
We thus realize that in case 3 and case 4, the camera was facing one way, then in the next frame it mirrored itself, thus the det(R)=-1 which isnt supposed to be possible, hence, we check that whenever the det(R) is -1, we multiply it with -1 to flip the sign.

TRIANGULATION CHECK FOR CHEIRALITY CONDITION

So now that we have 4 camera poses, we need to determine which one gives us the maximum number of points which will satisfy the cheirality condition. We hence take all 4 poses one by one and compare it to the initial pose which will have rotation as an identity matrix and camera centre as (0,0)
We triangulate by minimizing the algebraic error. This algorithm uses the two equations for perspective projection to solve for the 3D point that are optimal under least squares. Each perspective camera model gives rise to two equations on the three entries of (x,y,z). Combining these equations, we get an over determined homogeneous system of linear equations that we can solve with SVD.

$$\begin{bmatrix} v_i \boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ u_i \boldsymbol{p}^{3T} - \boldsymbol{p}^{1T} \\ v_i' \boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \\ u_i' \boldsymbol{p}'^{3T} - \boldsymbol{p}'^{1T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$

$$\begin{bmatrix} v_i p_{31} - p_{21} & v_i p_{32} - p_{22} & v_i p_{33} - p_{23} & v_i p_{34} - p_{24} \\ u_i p_{31} - p_{11} & u_i p_{32} - p_{12} & u_i p_{33} - p_{13} & u_i p_{34} - p_{14} \\ v_i' p_{31}' - p_{21}' & v_i' p_{32}' - p_{22}' & v_i' p_{33}' - p_{23}' & v_i' p_{34}' - p_{24}' \\ u_i' p_{31}' - p_{11}' & u_i' p_{32}' - p_{12}' & u_i' p_{33}' - p_{13}' & u_i' p_{34}' - p_{14}' \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$

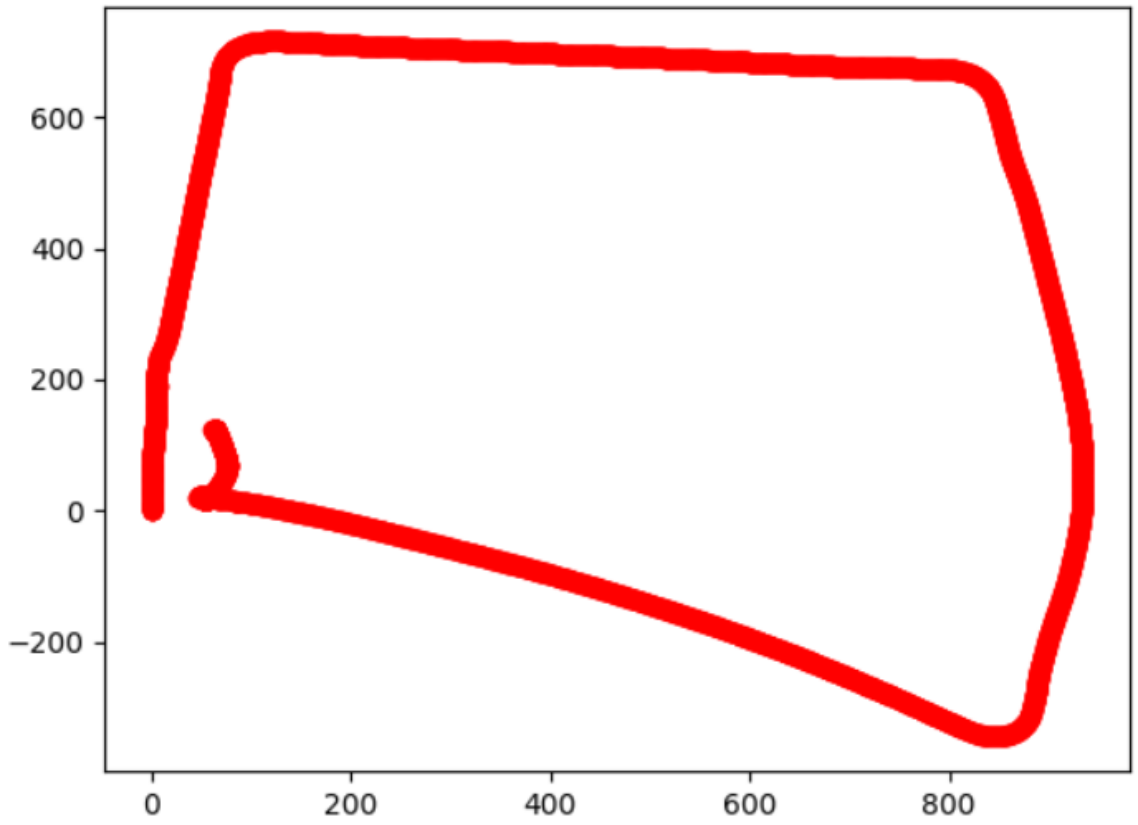
$$A\tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$

Where u and v are the (x,y) from one frame and, u' and v' are (x,y) from the next frame. P and p' are the perspective camera matrix.
Now that we have our 4 triangulated points, we can use the cheiraity condition to check and find the best (R,C) pair.

A 3D point X is in front of the camera if: $r3(X-C)>0$ where $r3$ is the third row of the rotation matrix (z-axis of the camera). The best camera configuration, (C, R, X) is the one that produces the maximum number of points satisfying the cheirality condition.
We thus return the best (R,C,X) for every frame.

FINAL PLOTTING

Now we form the homogenous matrix (H) given the R and C. We plot the original point, we then update the H matrix for every frame with the previous H matrix and plot the points obtained.
We are now able to estimate the 3D motion of the camera and provide as output a plot of the trajectory of the camera



SUMMARY

1. Prepare the data for processing
2. Use SIFT to find all the matching features between consecutive frames.
3. Use the 8 point algorithm to determine the F matrix
4. The points are normalized at the camera centre and the best F matrix is estimated using RANSAC. The rank 2 constraint is enforced
5. The E matrix is formed from the F matrix according the camera Calibration
6. The 4 poses are found from the E matrix
7. The points are triangulated and the best Rotation and Translation is found using cheirality conditions (depth positivity)
8. The final rotation and translation is used to estimate the position of the camera centre

CODE DETAILS

Since we had to process over 3000 frames, and running a ransac made it extremely slow, we ran the code overnight and logged our camera centre points to a file. We later created another file where we inputted the camera centre points just for plotting them faster.

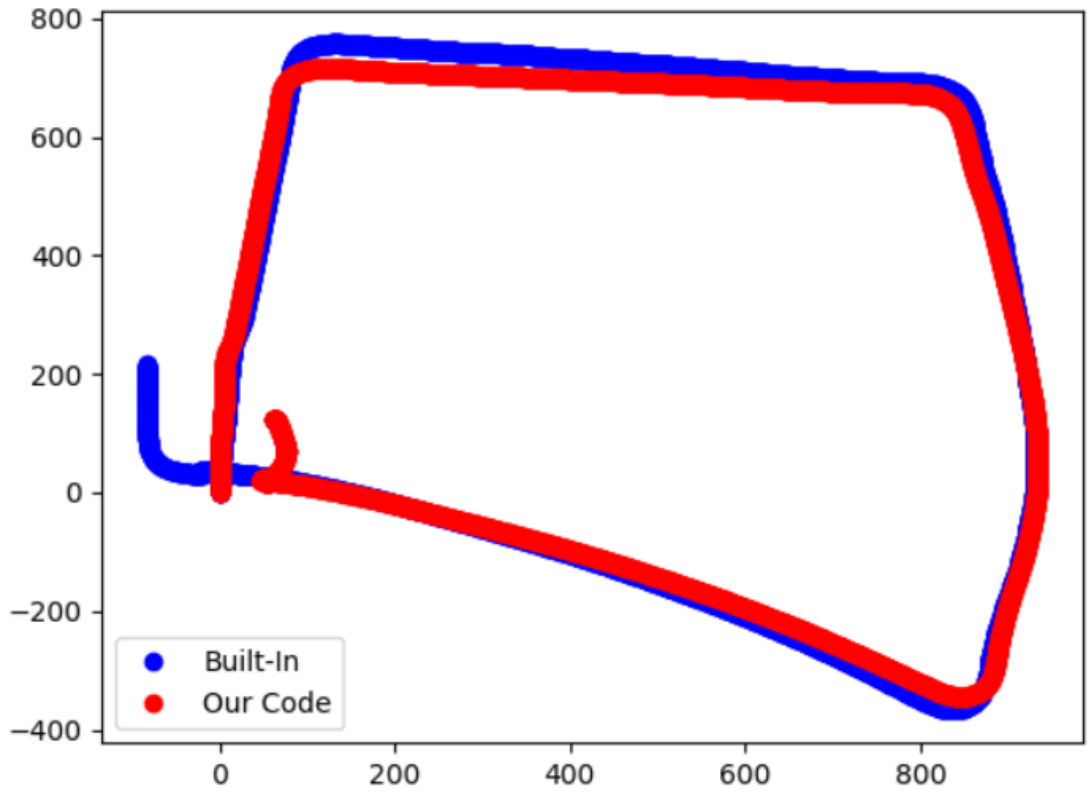
EXTRA CREDIT 1 – 20 Points

Unfortunately, we couldn't find the time to implement the non-linear triangulation and estimating the rotation and translation non-linear using the Nonlinear PnP.

EXTRA CREDIT 2 – 30 points

We use the built-in functions, cv2.findEssentialMat and cv2.recoverPose and plot the trajectory. We then compare the obtained output with our output and plot the two trajectories

We notice that the trajectory of the inbuilt function is very similar to the output trajectory of our own developed functions. You can notice a different in these 2 plots due to obvious accumulation of error for each frame of the video. One type of error that might be affecting our code compared to the inbuilt functions is the idea that the inbuilt funcions are optimizing and normalizing points in a different way. Since we analyzed the source code of the Fundamental Matrix in opencv, we tried to match ours to theirs, which reduced the error in the plots. Even though this was done, there are still some other types of errors along the way. One of these can be in the optimization and/or normalization of the Essential Matrix of the opencv function. It is very likely that the highest amount of error is coming from the different thresholding values that are applied to our RANSAC code.



REFERENCES

[1] https://www.cvg.ethz.ch/teaching/2010fall/compvis/assignments/Exercise4.pdf
[2] https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger/lecture_7_2-triangulation.pdf