ENPM673

Project – 5: Visual Odometry.

Submitted by –

| Chayan Kumar Patodi |

I Prasanna Marudhu Balasubramanian |

| Saket Seshadri Gudimetla |

INTRODUCTION

In this project, we were supposed to plot the trajectory of the camera, by estimating the 3-D motion of the camera. We were given frames of the driving sequence and the required scripts to prepare the dataset.

DATA PREPARATION

1. We read all the images in grayscale using glob.glob from the path "stereo/centre" and cv2.imread(image,0).
2. The images were in Bayer format, so we recovered the images using demosiac function with GBRG (cv2.COLOR_BayerGR2BGR), to convert the Bayer pattern encoded image to a color image.
3. Once the images were converted into a color format, we used the python file (ReadCameraModel) provided with the project description to read the camera parameters (fx, fy, cx, cy, G_camera_image, LUT). We calculate the **intrinsic camera matrix (K)** using fx, fy, cx, cy. We will use this K later in the code.
4. We used UndistortImage function provided with the project description to undistort the color images.
5. We also used a gaussian filter on the images, to detect more features using the feature detector.

6. We used a loop to store all the processed images in a folder named 'Data', so that we can use this dataset later in the code. This was done to save the computational time of the code.

We also tried using Histogram equalization for better feature detection, but it wasn't giving the desired output and thus we only used Gaussian Filter.

This code for above mentioned steps can be found in the file named **'Set.py'** attached with this project report. We have commented the lines to save the images.

P.S. We are uploading the dataset extracted on the google drive and the link would be shared in the comment section of the project submission on ELMS.

BASIC PIPELINE

1. Feature Detection:

   As suggested in the project description, the very step of the pipeline was to find point correspondences between successive frames.
   We considered a few key-point algorithms for feature detection, such as SIFT, SURF, ORB, BRIEF, FAST, goodfeatures2track.We finally used ORB detector, because it was faster and more accurate, compared to others.

   To calculate using the **ORB** algorithm, we initiate the ORB detector. We then compute the key-points and match the descriptors b/w 2 images. We then extract the [x,y] coordinates of both the images from the key-points using query index and train index.

2. Fundamental Matrix:

   The fundamental matrix is a 3 x 3 (rank 2) matrix that maps corresponding set of points in two images from different angles.
   We wrote a function for estimation of Fundamental Matrix using eight-point algorithm. For selecting the best Fundamental Matrix for our usage, we have used RANSAC, to get a fundamental matrix with minimum error. It calculates the error using epi-polar constraints.
   We also tried using all the feature points for calculating the fundamental matrix , but the result was inaccurate to a very great extent and thus we decided to go with RANSAC and eight point algorithm.

   We first **normalized** the image coordinates i.e. we centered the image data at origin , and scaled the data according to the eight point algorithm.
   The second step after this was , to use **RANSAC** to estimate Fundamental Matrix.
   Then we established a rank 2 constraint on F , by making the last singular value of F to be zero.
   After estimating the Fundamental matrix , it is denormalized before we proceed. It is done using the Transformation Matrix for both the images and the normalized Fundamental Matrix and by using the Formula: F = P2' * F * P1.

This is done using the function created , named **“, normalise , RANSAC_fundamental”** and **“get_fundamental_matrix”.**

3. Essential Matrix:

The Essential Matrix is another 3x3 matrix, that relates the corresponding points, keeping in mind the camera calibration matrix. The essential matrix can be extracted from F and K by the formula: E = K’FK.
Though the singular values of E are not (1,1,0) , due to the noise present in K and thus we reconstruct the E by making the S matrix as a diagonal matrix with (1,1,0).

4. Estimating Camera Pose from Essential Matrix:

The essential matrix can be decomposed into a Translation and Rotation. As suggested in the project description, we get four camera pose configurations The four configuration were calculated as follows:

1. C1 = U(:,3) , R1 = UWV’.
2. C2 = -U(:,3) , R2 = UWV’.
3. C3 = U(:,3) , R3 = UW’V’.
4. C4 = -U(:,3) , R4 = UW’V’.

Then we created a function to check det(R) , to correct the sign of R and C , as suggested on the reference page provided with project description.
This is done using the function **“compute_P_from_essential”**.

5. Cheirality Equation:

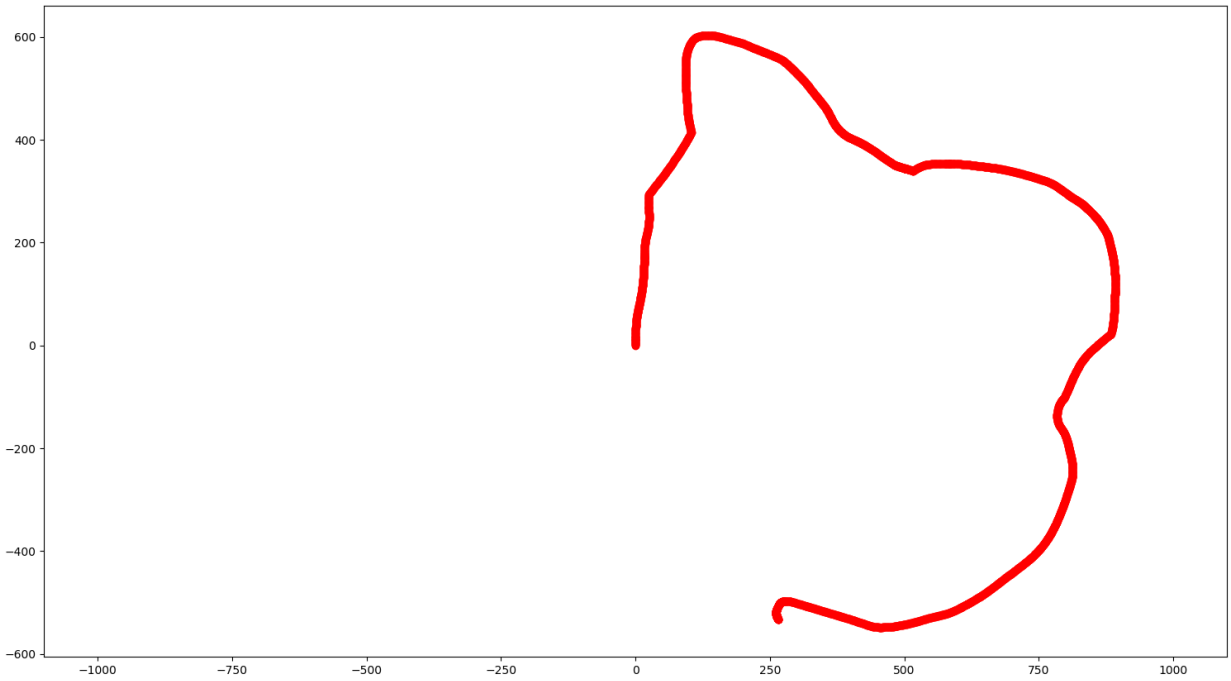
The correct translation ‘T’ and Rotation ‘R’ values are obtained from the depth positivity by estimating for each of the four solutions, the depth of all points linearly using the Cheirality equations. From the four camera pose configuration, (C[i],R[i]) , and correspondences, x1 and x2, the 3D points are triangulated as mentioned above using linear least squares.But in order to find the correct unique camera pose, we need to remove the disambiguity and it is accomplished by checking the cheirality condition. The reconstructed points are obtained such that it lies in front of the cameras. For checking the cheirality condition, the 3D points are triangulated as using linear least squares to check the sign of the depth Z in the camera coordinate system with respect to the camera center. The 3D point ‘X’ is in front of the camera if and only if the following condition is satisfied: r3*[X−C]>0 , where ‘r3’ is the third row of the rotation matrix (z-axis of the camera). The cheirality condition is checked in the algorithm using the condition mentioned above.Due to the presence of noise, this cheirality condition will not be

satisfied by all the triangulated points. Hence the steps are included in the algorithm to reduce the noise which reduces the error . The (C,R,X) value that produces the maximum number of points satisfying the cheirality condition is observed to be the best camera configuration. The R and T values are chosen such that it gives the largest amount of the positive depth value after triangulation and cheirality condition check. The unique camera pose is obtained by checking this cheirality condition.

This is done using the functions named "**LinearTriangle and Cheirality**".

6. Camera Center:

To plot the camera center we plot the x and z axis of the Translation matrix. We also keep updating the rotation and translation matrix with a certain formula.



Output 1

The above mentioned functions , outputs and steps followed an be found in the file named **"Final_Code.py".**

Extra-Credit

From the obtained camera poses (Rotational and Translational matrix values) and the linearly triangulated points we obtain the location of the 3D points that minimizes the reprojection error and it can be computed from the 'NonLinearTriangulation' function in the algorithm.

Though, the reprojection error is geometrically meaningful error and can be computed by measuring the error between measurement and projected 3D point.

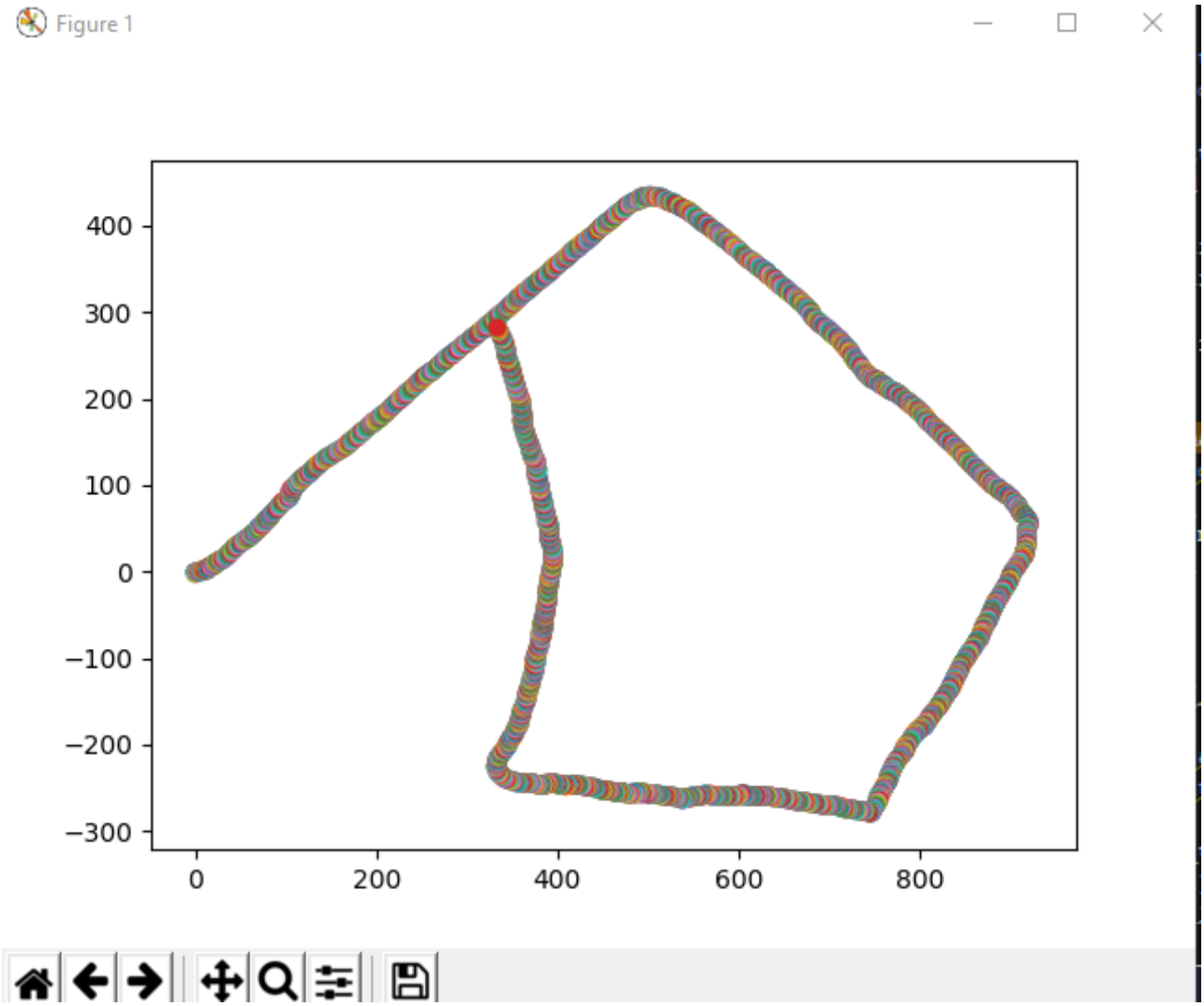
The initial value of the solution, 'Xset', is estimated via the linear triangulation to minimize the cost function. This minimization is solved again through the 'NonLinearTriangulation' function in the algorithm. The 'Xset' value obtained from this 'NonLinearTriangulation' can be used to calculate the Rotation and Translation matrix values to obtain the camera pose.

This method of minimization for the camera pose is highly nonlinear because of the divisions and quaternion parameterization. The initial guess of the solution, (C,R), estimated via the linear PnP is needed to minimize the cost function. This minimization is solved using a nonlinear optimization function.

Comparison

For the comparison part, we have used the inbuilt functions of OpenCV as suggested by the projected description. We have calculated the points using ORB detector and we are passing those points in **cv2.findEssentialMat** to calculate the essential matrix.
Once we obtain the essential matrix, we pass that to **cv2.recover pose** along with points and camera calibration matrix. Once we obtain the R and t , we plot it using the similar approach as mentioned above.
Even with inbuilt function the trajectory is not coming as expected.

The above-mentioned part can be found in the file named **"Inbuilt.py".**



We have calculated the drift using the distance formula and we are printing the final drift at the very end of the code.