Rohan
Abhinav
Kamakshi

Project 5

ENPM 673
July 31, 2019

Visual Odometry

Visual odometry is the process of determining the position and orientation of a robot by analyzing the associated camera images. In this we reconstruct a 3D scene and simultaneously obtain the camera poses of a monocular camera w.r.t. the given scene. This procedure is known as Structure from Motion (SfM). As the name suggests, you are creating the entire rigid structure from a set of images with different view points (or equivalently a camera in motion).

1.Keypoint Matching

The epipolar geometry is the intrinsic projective geometry between two views. It only depends on the cameras internal parameters (K matrix) and the relative pose i.e. it is independent of the scene structure. Knowing the epipolar geomrtry helps in finding the fundamental (F) matrix. The F matrix is only an algebraic representation of epipolar geometry and can both geometrically and arithematically, i.e $x_i'^T F x_i = 0$, also known as epipolar constraint or correspondance condition.
In F matrix estimation, each point only contributes one constraints as the epipolar constraint is a scalar equation. Hence we used the eight point algorithm- as we require at least 8 points to solve the above homogenous system.

SIFT Scale-Invariant Feature Transform
A corner may not be a corner if the image is scaled. In such case the Scale Invariant Feature Transform (SIFT) algorithm is used which extracts distinctive Image Features from Scale-Invariant Keypoints, i.e extracting keypoints and computing its descriptors.
We are finding the point correspondences using SIFT, the data bounded is noisy and contains several outliers. Thus, to remove these outliers, we used RANSAC algorithm to obtain a better estimate of the fundamental matrix. So, out of all possibilities, the F matrix with maximum number of inliers is chosen.

Given the fact that finding SIFT features is a relatively slower process, we also tried **ORB features** to make our pipeline faster. The only problem was that ORB features gives less number of points to run RANSAC which leads to wrong pose estimation in the Disambiguate pose step. Given below are outputs of the feature matching using SIFT(1) and ORB(2):

2. Computing Essential Matrix

To calculate Essential Matrix we first compute the Fundamental matrix using the 8-point algorithm. The matrix obtained using this algorithm is not always of rank 2 due to noise present. Thus, rank 2 constraint is enforced by making the smallest singular value 0. The output obtained using our algorithm and one form Cv2's builtin function are given below for

Figure 1: Feature matching between two consecutive frames using SIFT features

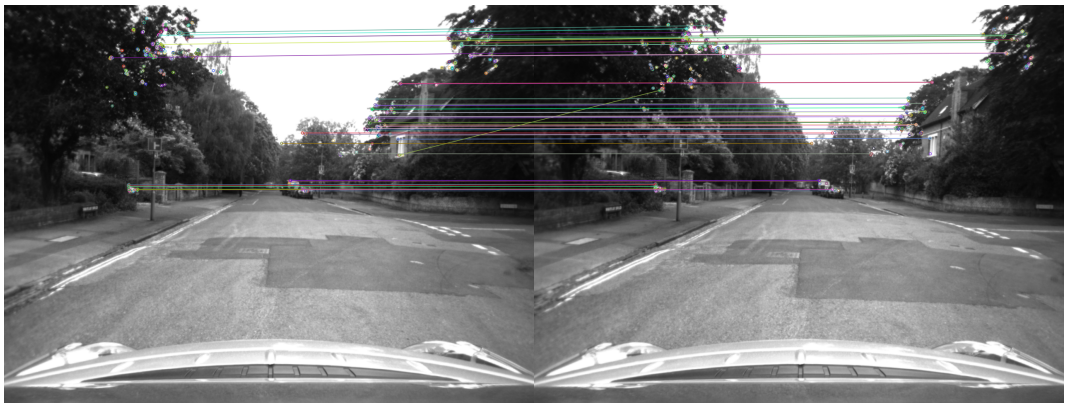

Figure 2: Feature matching between two consecutive frames using ORB features

comparison:

$$F_{cv2} = \begin{bmatrix} 8.67408149e-07, & -2.38251493e-03, & 1.23752372e+00 \\ 2.37765708e-03, & 7.28554718e-06, & -1.20816650e+00 \\ -1.23731840e+00, & 1.20428856e+00, & 1.00000000e+00 \end{bmatrix} \tag{1}$$

$$F_{implemented} = \begin{bmatrix} 2.87769034e-06, & -1.55260470e-03, & 8.63412717e-01 \\ 1.54369152e-03, & 1.61282008e-05, & -1.71750824e+00 \\ -8.66583877e-01, & 1.71655206e+00, & 1.00000000e+00 \end{bmatrix} \tag{2}$$

Essential matrix is another 3x3 matrix, but with some additional properties that relates the corresponding points assuming that the cameras obeys the pinhole model. The E matrix was extracted from F and K where K is the camera calibration matrix using the formula $E = K^T F K$.

Note: The F is defined in the original image space (i.e. pixel coordinates) whereas E is in the normalized image coordinates. Normalized image coordinates have the origin at the optical center of the image. Also, relative camera poses between two views can be computed using

E matrix. Moreover, F has 7 degrees of freedom while E has 5 as it takes camera parameters in account.

Estimate Camera Pose

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. We have identified the E matrix in the above step and can now get the four camera pose configurations: (C1,R1),(C2,R2),(C3,R3) and (C4,R4) where C is the camera center and R is the rotation matrix. The camera pose can be written as: $P = KR[I_{33}|\text{-}C]$.

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

The four configurations can be written as:

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

Triangulation Check for Cheirality Condition

So far we have found 4 possible camera poses. After this we triangulate the 3D points, given two camera poses.Given two camera poses, (C1,R1) and (C2,R2), and correspondences, x1 $\Longleftrightarrow$ x2, triangulate 3D points using linear least squares.Though, in order to find the correct unique camera pose, we need to remove the disambiguity. This can be accomplish by checking the cheirality condition i.e. the reconstructed points must be in front of the cameras.

To check the cheirality condition, we triangulate the 3D points (given two camera poses) using linear least squares to check the sign of the depth Z in the camera coordinate system w.r.t. camera center.

A 3D point X is in front of the camera iff: $\mathbf{r3(X-C)>0}$ where r3 is the third row of the rotation matrix. Not all triangulated points satisfy this condition due of the presence of correspondence noise. The best camera configuration, (C,R,X) is the one that produces the maximum number of points satisfying the cheirality condition.

Perspective-n-Points

We have: a set of n 3D points in the world, their 2D projections in the image and the intrinsic parameter.

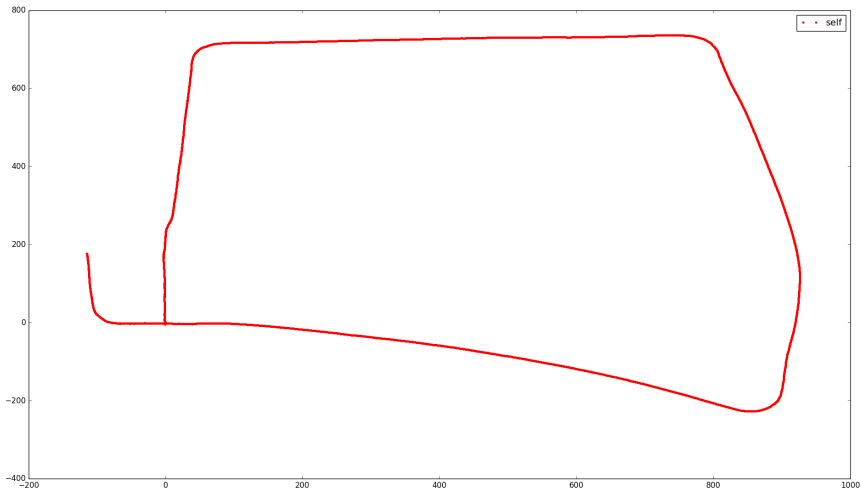

Figure 3: Visual Odometry using Implemented pipeline

To find the camera pose using linear least squares we use Persepective-n-Points (PnP). 2D points can be normalized by the intrinsic parameter to isolate the extrinsic camera parameters, (C,R), i.e. $K^1x = R[I|-C]X$. A linear least squares system that relates the 3D and 2D points is solved for (t,R) where $t = R^TC$. PnP is prone to error as there are outliers in the given set of point correspondences. To overcome this error, we used RANSAC to make our camera pose more robust to outliers.

The output of the our pipeline for the complete course(dataset provided) is shown below in the figure(7):

Additional Steps in the Pipeline

To better estimate, we solve non-linearly for the depth and 3D motion. So far we were solving the linear triangulation to minimize the algebraic error. A better estimate is to solve for the reprojection error, which is geometrically meaningful error and can be computed by measuring error between measurement and projected 3D point as shown in equation below. This minimization is highly nonlinear due to the divisions.

$$min_x \sum_{j=1,2} (u_j \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X})^2 + (v_j \frac{P_2^{jT}\tilde{X}}{P_3^{jT}X})^2$$

Here, j is the index of each camera, $\tilde{X}$ is the homogeneous representation of X. P_i^T is each row of camera projection matrix, P. The initial guess of the solution, X_0 and is estimated via the linear triangulation to minimize the cost function we did above.
After this step we have N6 3D-2D correspondences, X$\Longleftrightarrow$ x, and linearly estimated camera pose, (C,R). Finally we can refine the camera pose that minimizes reprojection error using Nonlinear PnP. The output after solving nonlinear PnP and nonlinear-Triangulation is shown below:
The figures below show the comparison of the output trajectories of the linear and nonlinear pipelines with the builtin.

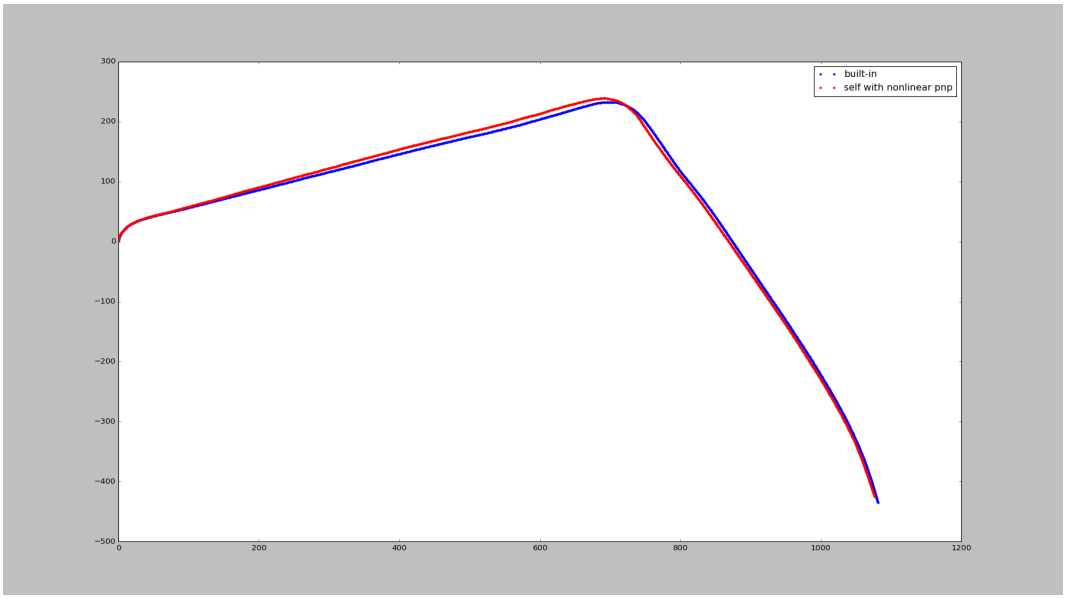

Figure 4: Outputs for nonlinear optimization vs builtin functions

The drift accumulated during the course is also shown in the figure(5):

Comparison

This section compares our result against the rotation/translation parameters recovered using cv2.findEssentialMat and cv2.recoverPose from opencv. Plot for both the trajectories is shown.
Also for the right turn in previous section, the comparison of the linear and nonlinear optimizations vs builtin functions is shown in the figure below:

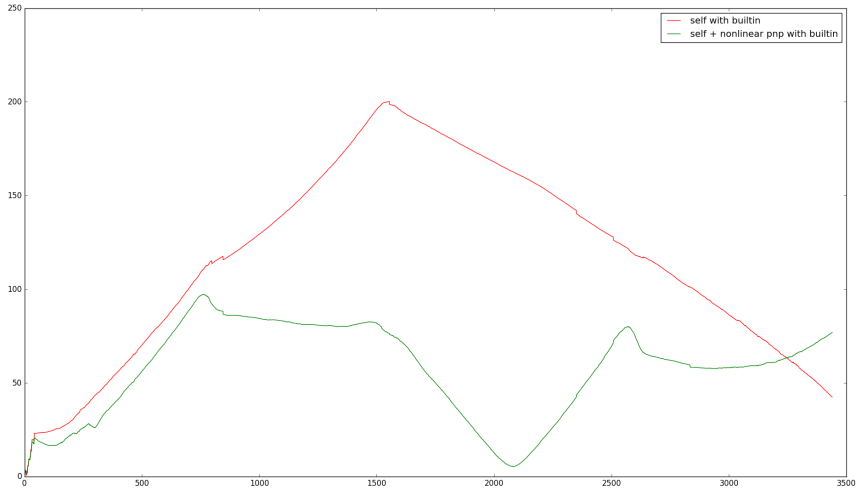

Figure 5: Drift per frame in linear and nonlinear pipelines as compared to builtin

Problems Faced

Mostly the problems were faced in figuring out the correct interpretation of C and -RC with their respective coordinate frames. The coding was fairly straightforward. This project helped in understanding reprojection error few things about nonlinear optimizatioin using python's scipy library.

One problem identified was that when vectors are used under different conditions using **np.matmul()** gives out different results. For eg., in one case a 3x1 vector was used to create a matrix of the form $\begin{bmatrix} 1,0,0,-C[1] \\ 0,1,0,-C[2] \\ 0,0,1,-C[3] \end{bmatrix}$ and an "array-like" vector of size (3,) was used to create the same matrix and when multiplied by "R" using np.matmul() gave out different results because numpy was treating the first case as an object. Don't know why exactly this occured but reshaping the first one helped.

This pipeline is highly dependent on the initial frame, as slight deviation in the rotation matrices of the initial frame can cause a lot of drift even if the relative poses calculated for the following frames are precise.

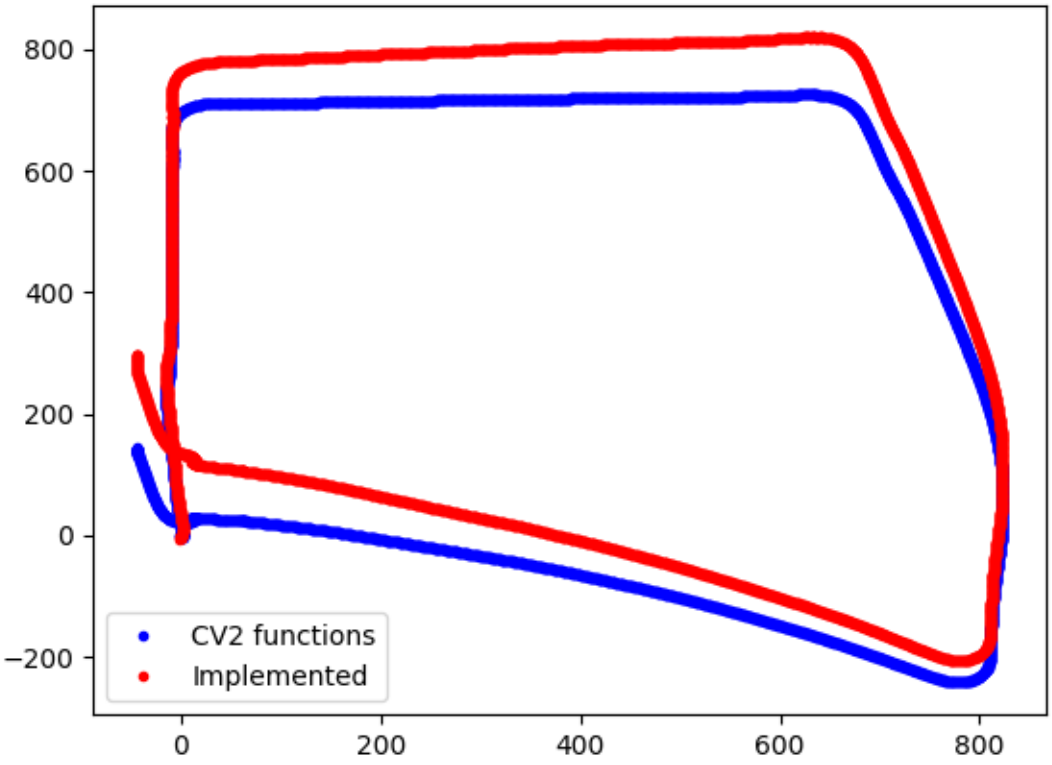

Figure 6: Visual Odometry using built_CV2 pipeline vs Implemented pipeline with Linear Optimization

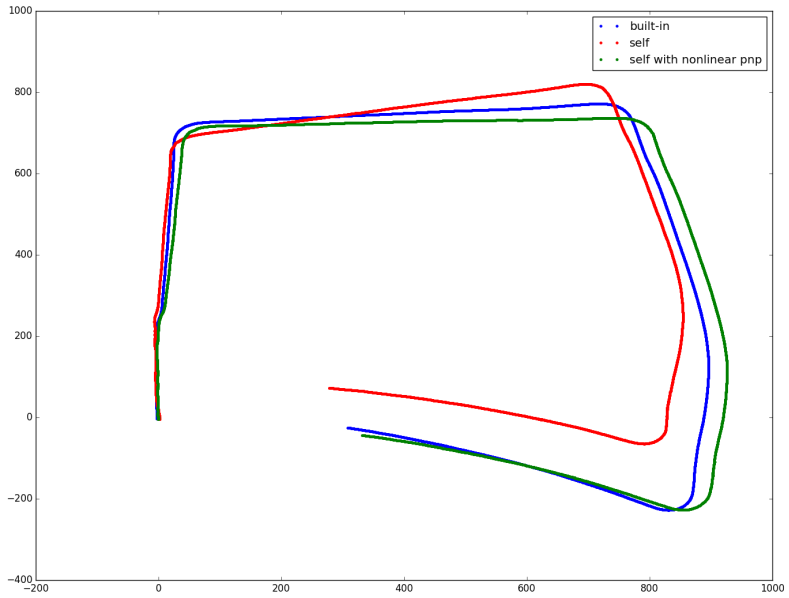

Figure 7: Visual Odometry using built_CV2(blue) pipeline vs Implemented pipeline with Linear(green) and nonlinear Optimization(red)

NOTE

To run the code follow the README file. The videos are available on our drive. Follow this link