ENPM673 - Project 5
Visual Odometry

Gnyana Teja Samudrala(115824737)
Dinesh Kadirimangalam(116353564)
Sai Chaitanya Balasankula(116137294)

May 6, 2019

1 Introduction

In this project we have been given the input of frames of a car moving around, by using two successive frames we generate the plot of the camera trajectory i.e path the car has taken. In this report, we will explain the pipeline followed to get the camera pose between successive frames. Also the Non-linear optimization is performed to get the right pose of the camera. The pipeline followed is as follows:

1. Data preparation. (*dataPrep.py*)
    - Get the RGB images
    - Undistort the images
2. Getting the key points between successive frames. ($featMatchCV.py$)
3. Outlier rejection using RANSAC based on Fundamental matrix. ($EstimateFundamentalMatrix.py$, $GetInliersRANSAC.py$)
4. Estimating the Essential matrix from Fundamental matrix. ($EssentialMatrixFromFundamentalMatrix.py$)
5. Extracting possible camera poses from Essential Matrix. ($ExtractCameraPose.py$)
6. Triangulating the 3-D points from two camera poses. ($LinearTriangulation$, $NonlinearTriangulation$)
7. Checking the cheirality condition to get the unique camera pose. ($DisambiguateCameraPose.py$)
8. Recomputing the camera pose from the optimized 3-D points using PnP RANSAC and Nonlinear optimization. ($LinearPnP$, $PnPRANSAC$, $NonlinearPnP$)

2 Data Preparation

The images given in GBRG alignment are converted into color image using the OpenCV function. The camera model is extracted and the calibration matrix K is constructed.

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

Using the python scripts provided the images are undistorted and saved in the order of the timestamp. The images sequence names according to the timestamp are saved in the text file in the undistort_images folder. This folder also has the undistorted color images.

3 Matching Key Points

The built in OpenCV SIFT feature detector is used to get the key points and the Flann based matcher is used to get the corresponding matching points.

4 Fundamental Matrix

Given a pair of successive images and corresponding matching points, we need a minimum of 8 points to estimate the fundamental matrix. The fundamental matrix F is estimated from the equation $x'^T F x = 0$ where x' and x are the corresponding image points.

4.1 The 8-point algorithm (EstimateFundamentalMatrix.py)

The image points are normalized by subtracting the mean (μ) and dividing with standard deviation (d). Then we get it into the form $Af = 0$, and solve it using the SVD.

$$T = \begin{bmatrix} \frac{1}{d} & 0 & -\frac{\mu_x}{d} \\ 0 & \frac{1}{d} & -\frac{\mu_y}{d} \\ 0 & 0 & 1 \end{bmatrix}$$

$$Af = 0$$

$$\implies \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_mx_m & x'_my_m & x'_m & y'_mx_m & y'_my_m & y'_m & x_m & y_m & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

The rank condition is enforced on the F.

$$U_f S_f V_f^T = F$$

The last value diagonal value of S_f is made 0 and the F is constructed. Then the matrix is denormalized by multiplying the T matrix.

$$F_{denorm} = T_2^T F_{norm} T_1$$

4.2 Inliers using RANSAC (GetInliersRANSAC.py)

The RANSAC algorithm is used to get the best inliers based on the points satisfying the fundamental matrix under a threshold value.

$$x'^T F x \approx 0$$

.

This is run for 1000 iterations and the threshold value chosen is 0.0006. The matching points before and after the RANSAC are shown in images 1 and 2.

Figure 1: The matchings between two frames before RANSAC.

5 Essential Matrix (EssentialMatrixFromFundamentalMatrix.py)

The Essential matrix is derived by using the formula $E = K^T F K$ where K is the camera calibration matrix. Also change the E to have the singular values (1,1,0).

6 Camera Pose from Essential Matrix (ExtractCameraPose.py)

All the 4 possible solutions for the camera poses are computed from the essential matrix. Also it is made sure that the determinant of the rotation matrix is 1, if not then the center and rotation matrix are negated. The values obtained from the equations provided are the translation vectors but not the camera center, in order to get camera center we use $C = -R^T t$.

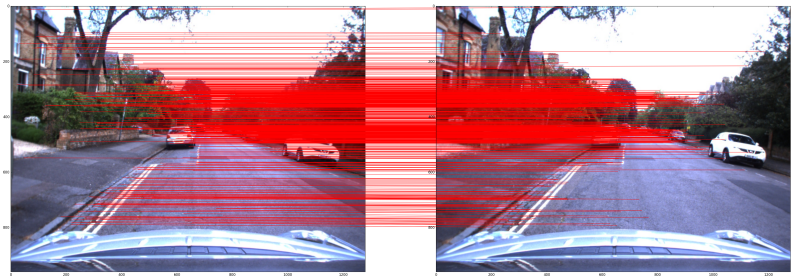

Figure 2: The matchings between two frames after RANSAC.

7 Triangulation

Given two camera poses and there corresponding points in two views, we can triangulate there corresponding 3-D points. In this, we assume one of the camera pose to be at the origin and no rotation i.e $C = [0, 0, 0]^T$ and $R = identity(3X3)$. The initial estimate is given by the Linear Triangulation and then using this as the initial value we optimize using a Non linear least square estimator.

7.1 Linear Triangulation (Linear Triangulation.py)

The matrix P is calculated which maps the world points to image points, where $P = KR[I_{3\times3} \quad -C]$. Here R is the rotation matrix, C is the center position of the camera and K is the camera intrinsic matrix.

$$x = PX$$

where x is the image point and X is the world point. This is rearranged to get into the form $Ax = 0$.

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

7.2 Non-linear Triangulation (Nonlinear Triangulation.py)

This step is performed after we get the unique pose by applying the cheirality condition. Now that we got the camera pose for the 2nd camera as well as the linearly triangulated points X from the previous step, we can apply a nonlinear optimization function to refine X such that we can minimize the reprojection error. We use the function *scipy.optimize.least_squares* to

minimize the following reprojection error:

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

8 Cheirality condition (DisambiguateCameraPose.py)

In this we will check for the solution with maximum number of inliers which has the positive depth. By this we find the unique camera pose.

9 Perspective-n-Points (PnP)

We have the world points in 3D(X) and their corresponding image points in 2D(x), using this correspondance we can calculate the rotation and translation of the camera in the world i.e 6 DOF pose of the camera. We first estimate the pose of the camera with linear least squares solution and make it robust by using RANSAC algorithm. This result from RANSAC is used as the initial estimate for the Non linear method. The steps are detailed below with plots for comparision and understanding the use of each method.

9.0.1 Linear PnP (LinearPnP.py)

The inputs to this are the image points(x), world points(X) and also the intrinsic parameters of the camera(K). At first the inverse of the intrinsic parameter matrix is calculated to normalise the image points. Then we get the equation 1 into the form $Ax = 0$ to solve it using linear least squares solution with SVD.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \tag{1}$$

where $P = R\begin{bmatrix} I_{3\times3} & -C \end{bmatrix}$

We manipulate this to arrive at,

$$\begin{bmatrix} 0_{1\times4} & -X^T & vX^T \\ X^T & 0_{1\times4} & -uX^T \\ -vX^T & uX^T & 0_{1\times4} \end{bmatrix} \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix} = 0$$

By doing SVD, the last column of V gives the solution to the equation. From this we extract the rotation matrix (R)and the 4th column will be the translation vector(T). The rotation matrix will not be orthogonal to make sure of which we decompose $R = USV^T$ and can get orthogonal R as $R = UV^T$. Also we check for the sign of the determinent and make it positive if not by negating both R and T. The centre is obtained by $C = -R^T T$.

9.0.2 PnP RANSAC (PnPRANSAC.py)

We use this to remove the outliers and get a better result of the initial linear estimate. This is run for a maximum of 1000 iterations and the threshold of the reprojection error for a point to be inlier is set to be 10. The formula used for calculating the reprojection error is

$$e = \left(u - \frac{P_1^T\tilde{X}}{P_3^T\tilde{X}}\right)^2 + \left(v - \frac{P_2^T\tilde{X}}{P_3^T\tilde{X}}\right)^2$$

9.0.3 Nonlinear PnP (NonlinearPnP.py)

We made use of the Nonlinear least squares solver from scipy to solve for optimal pose of the camera. The initial estimate for the solver is given from the values obtained by above method RANSAC PnP. In this to maintain the orthogonal condition of the rotation matrix we use quaternion as the input parameter for the solver. The equation to be minimized is

$$\min_{C,q} \sum_{i=1,J} \left(u^j - \frac{P_1^T \widetilde{X}_j}{P_3^T \widetilde{X}_j} \right)^2 + \left(v^j - \frac{P_2^{iT} \widetilde{X}_j}{P_3^{iT} X_j} \right)$$

10 Results

The plots of the camera trajectory obtained by using linear and Non-linear methods are presented below from fig. 3 to 6. The camera pose in the figs. 4 and 6 show the direction and the center of the camera, these are the poses plotted for every 10 frames for a neat visualization. Also the plot the trajectory obtained from using the OpenCV built in functions is shown in the fig. 7. The average drift between the Linear estimate and the built in function camera centers is 266.97 and that between Nonlinear estimate and the built in function is 269.17. The Nonlinear estimated result seems to be more accurate than the one from built in functions, this is evident from the ending part of the sequence.

References

[1] "Cmsc733- project 3 description." [Online]. Available: https://cmsc733.github.io/2019/proj/p3/

[2] T. Opsahl, "Lecture 7.2 triangulation." [Online]. Available: https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger/lecture_7_2-triangulation.pdf

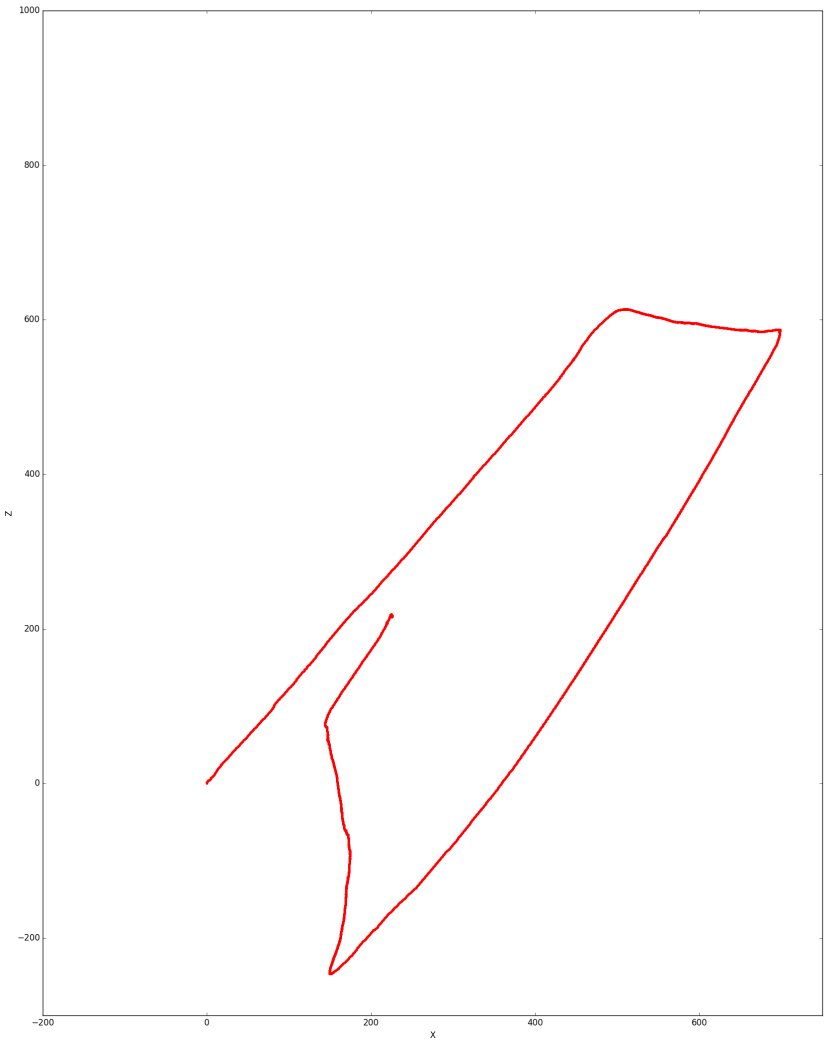

Figure 3: The camera center position obtained from Linear estimate.

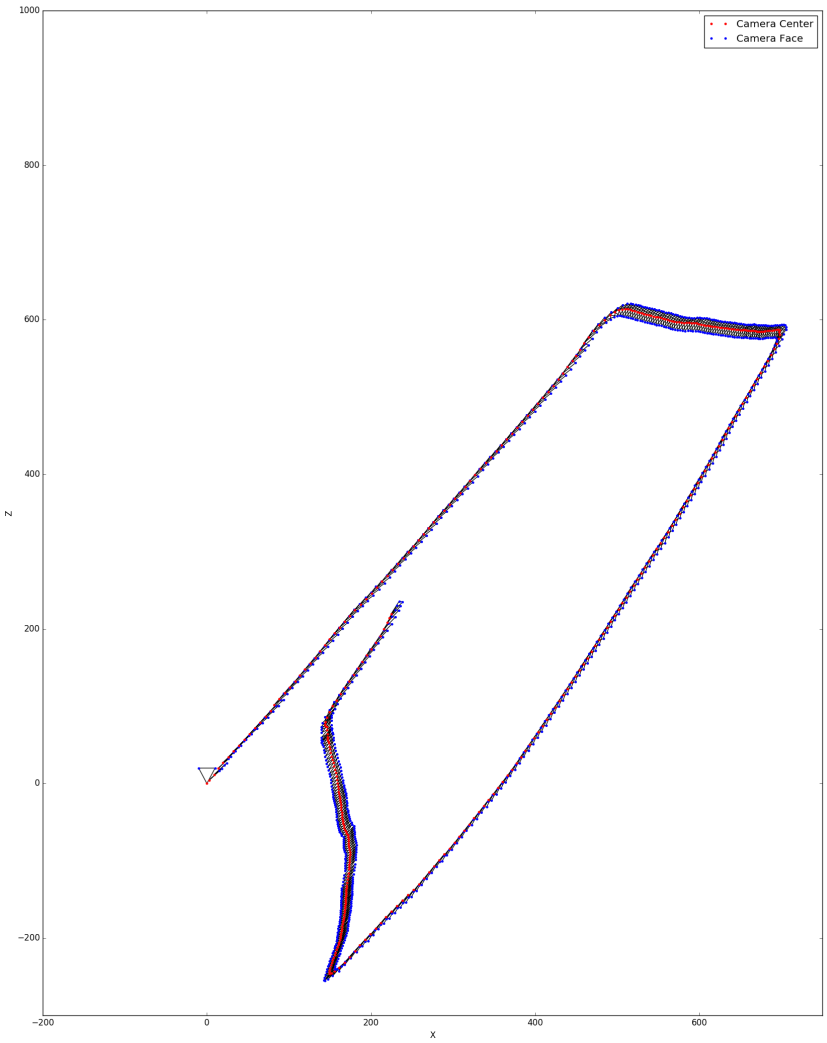

Figure 4: The camera pose obtained from Linear estimate.

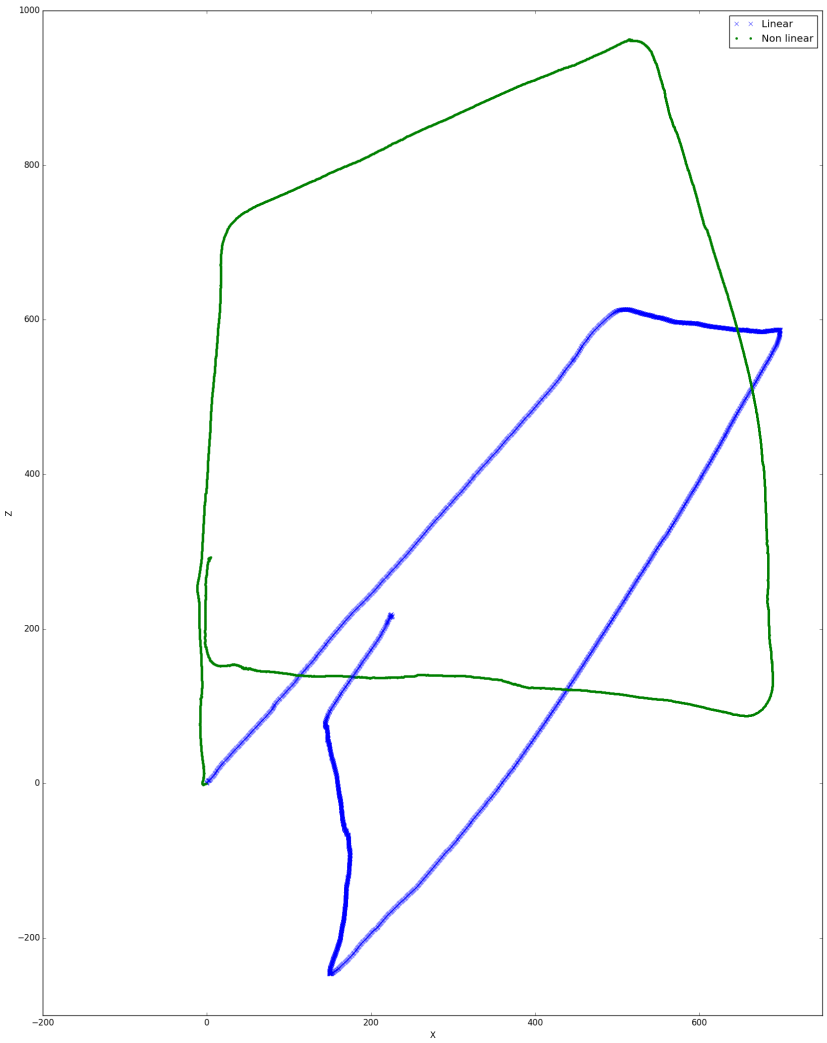

Figure 5: The camera pose obtained from Nonlinear estimate.

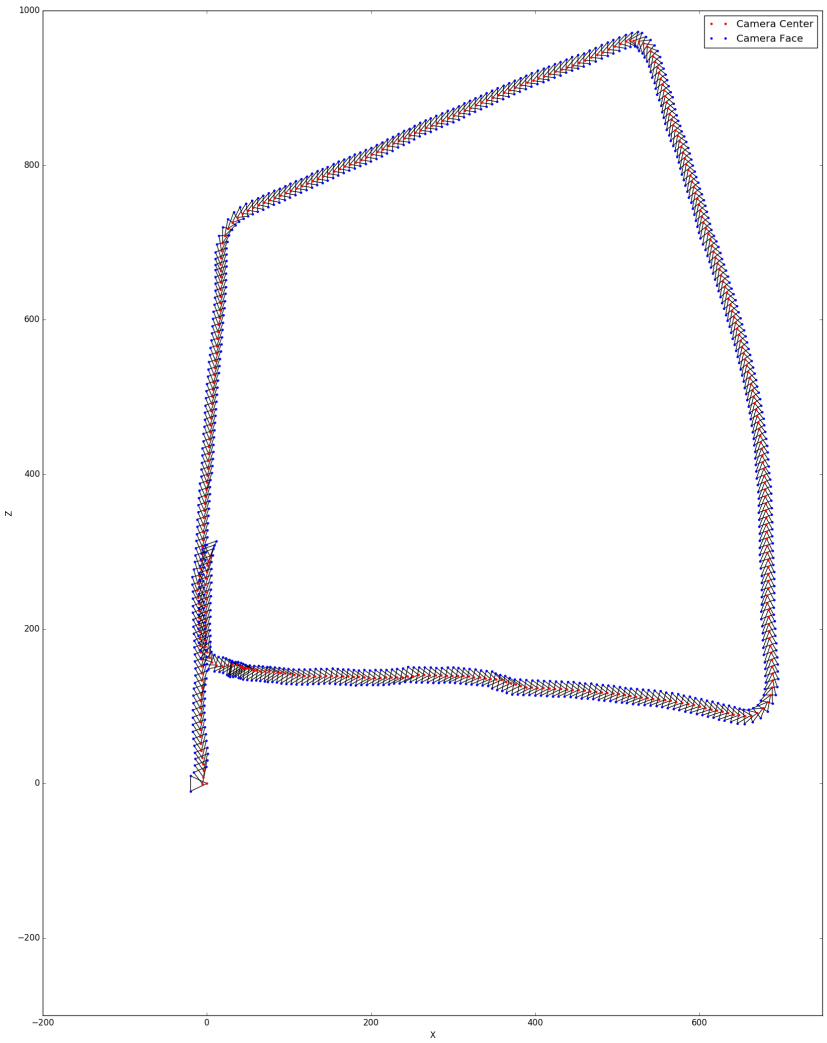

Figure 6: The camera pose obtained from Nonlinear estimate.

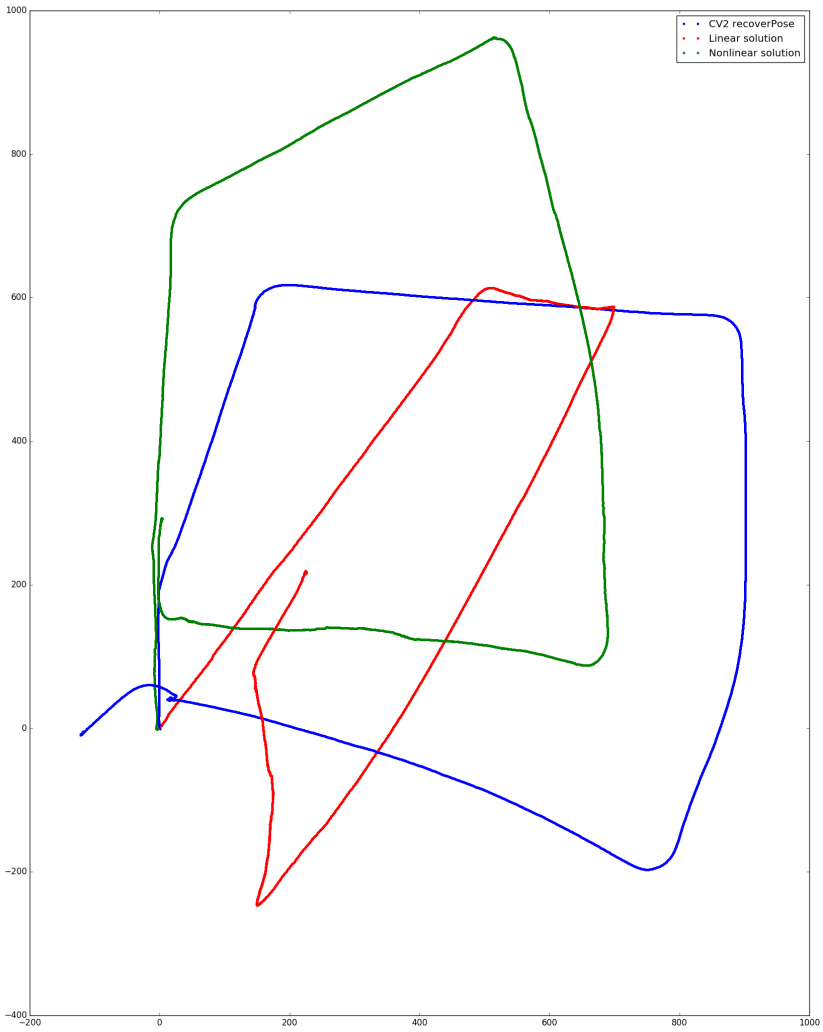

Figure 7: The comparison against inbuilt functions in OpenCV.