1 Introduction

In this project, we have estimated the 3D motion of camera, and provided as output a plot of the trajectory of the camera. The inputs given in this project are image frames of a driving sequence taken by a camera attached in front of the car. By estimating the motion of camera. we are indirectly estimating the motion of the car using the concepts of Visual Odometry.

2 Pipeline followed

The following pipeline was followed to implement this project.

2.1 Data Preparation:

The input frames were in Bayer format from which we can recover the color images using the demosaic function with GBRG alignment. That is, convert the Bayer pattern encoded image to a color image using inbuilt Opencv function:

```
color_image = cv2.cvtColor(img, cv2.COLOR_BayerGR2BGR)
```

Then we extracted the camera parameters using *ReadCameraModel.py* file provided.

```
fx, fy, cx, cy, G_camera_image, LUT = ReadCameraModel('./model')
```

Then we undistorted the images frames using *UndistortImage.py* file provided. Since, there

```
undistorted_image = UndistortImage(original_image,LUT)
```

were many image frames, for easy access and optimize the code, we converted the undistorted image sequence into a single video.

2.2 Estimating Fundamental Matrix:

The fundamental matrix, denoted by F, is a 3x3 matrix of rank 2 that relates the corresponding set of points in two images from different views. Lets say a point X in 3D is represented as $(x_i, y_i, 1)$ in camera $C1$ and $(x_i^{'}, y_i^{'}, 1)$ in camera $C2$. Fundamental matrix related these points in two different cameras using the following relation.

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

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

The above equation can be modified to the matrix form $Ax = 0$ Hence, the fundamental

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

matrix can be found finding the SVD of matrix A, the decomposition $U * S * V^T$ would be obtained with U and V orthonormal matrices and a diagonal matrix S that contains the singular values. The last column of V is the true solution of the matrix equation $Ax = 0$ which lies in null space of matric A. The last column of V can be reshaped into fundamental matrx of size 3x3. This is done in the python file EstimateFundamentalMatrix.py.

2.3 Match Outlier Rejection via RANSAC:

The corresponding point in two images computed may be noisy. Thus, we may not get the correct estimate of the fundamental matrix. So, in order to get the correct estimate, we employ RANSAC algorithm to remove the outliers. After removing the outiers, the fundamental matrix F with maximum number of inliers is chosen. The RANSAC algorithm is implemented in GetInlierRANSANC.py pyrhin file.

2.4 Estimate Essential Matrix from Fundamental Matrix:

We can find the relative camera poses between two images using the fundamental matrix F. In order to compute relative poses between two images we need to first compute essential matrix(E). In order to compute, essential matrix, we need to intrinsic matrix(K). The relation is given by:

$$\mathbf{E = K^T F K}$$

By, due to noises present in matrix K, the third eigen value of matrix E is not zero. This can be corrected by reconstructing it with (1,1,0) singular values.

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

This is implemented in EssentialMatrixFromFundamentalMatrix.py python file.

2.5 Estimate Camera Pose from Essential Matrix

Camera pose refers to the position and angle at which the camera is places w.r.t to the world coordinates. Position of the camera refers to the center of the camera. The position and rotation of the camera can be determined by using essential matrix(E). First, we compute the svd of E.

$$\mathbf{E} = UDV^T$$

Then using the W matrix which is given by:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The camera center C and rotation of the camera R can be computes as follows:

1. $C_1 = U(:,3), R_1 = U * W * V^T$
2. $C_2 = -U(:,3), R_2 = U * W * V^T$
3. $C_3 = U(:,3), R_3 = U * W * V^T$
4. $C_4 = -U(:,3), R_4 = U * W * V^T$

If the $det(R) = -1$, the camera pose must be corrected i.e. $C = -C$ and $R = -R$. This is implemented in ExtractCameraPose.py python file.

2.6 Linear Triangulation:

Linear triangulation is a method to extract the 3D coordinates of a point in the image frame(X) from two different camera projections of the point$(x, x^{'})$ from 2 cameras. From the camera matrix of two cameras(C_1, C_2) which are $P, P^{'}$, we need to estimate the 3D coordinates of X using linear triangulation.
We know the relation between $(x, x^{'})$ and $P, P^{'}$ and X.

$$x = PX \tag{1}$$

$$x^{'} = P^{'}X \tag{2}$$

and we also know that

$$cross(x, PX) = 0 \tag{3}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{p}_1^\top \boldsymbol{X} \\ \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_3^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} \\ x\boldsymbol{p}_2^\top \boldsymbol{X} - y\boldsymbol{p}_1^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Third line is a linear combination of the first and second lines. So, we can remove it. So, we get: By concatenating points from two images, we get: This is a system of homogenous linear

$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p}_3'^\top - \boldsymbol{p}_2'^\top \\ \boldsymbol{p}_1'^\top - x'\boldsymbol{p}_3'^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

equation of the form $Ax = 0$. The solution is obtained using svd. The goal is to minimize the error caused due to noise. So, we find the total least square approach to minimize the error. So, the solution is the eigen vector with least eigen value of $A^T * A$. This is implemented in LinearTriangulation.py python file.

2.7 Triangulation Check for Cheirality Condition

After getting the 3D coordinates from linear triangulation, we need to check whether the obtained point is in front or behind the camera. We always want our point to be in front of the camera. This is checked by using Cheirality condition. Cheirality condition checks the sign of the depth in the camera coordinate w.r.t camera center. The condition is given by:

$$r_3(X - C) > 0 \tag{4}$$

where r_3 is the third row of the rotation matrix.The best camera configuration, (C,R,X) is the one that produces the maximum number of points satisfying the cheirality condition. This is implemented in DisambiguateCameraPose.py python file.

2.8 Plotting the motion of the camera:

The final part of this pipeline is plotting the camera center based on translation and rotation between successive frames.