ENPM673-Project 5

Mushty Sri Sai Kaushik, Alexandre Filie, Santhosh Kesani

May 5, 2020

1 Introduction

Visual odometry is the process of determining the position and orientation of a robot by analyzing the associated camera images. That is, the 2D representations allow us to extract 3D information about where the camera is and in which direction the robot moves.

For this project, we are provided with a set of camera frames obtained from a camera planted in a moving car and from this data we are required to find the position and orientation of the camera through a SfM pipeline and finally provide the plot of the trajectory of the camera.

2 Pipeline

2.1 Data preparation

The data(set of camera frames) provided to us are in Bayer format and they are to be converted into colour images. For this, as provided we used the function,

colour_images = cv2.cvtColor(img, cv2.COLOR_BayerGR2BGR)

Later, these images are undistorted for further processing using the provided script UndistortImage.py, through a function call in our script as,

undistorted_image = UndistortImage(original image, LUT)

Finally, the required camera's intrinsic parameters are extracted using the provided script ReadCameraModel.py, through a function call in our script as,

fx, fy, cx, cy, G camera image, LUT = ReadCameraModel('./ model')

From these obtained values, the intrinsic camera matrix/calibration matrix(K) is created, which is,

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

2.2 Determining Key Points

First step in this to pipeline is to determine the key features, for this we have used the cv2.orb_create() and cv2.BFmatcher() to generate the key features between the given two images. The first function generates all the key features in both the images and the second function matches the key features and gives out the best matching features between both the images.

2.3 Fundamental Matrix

The Fundamental matrix is a 3x3 matrix(of rank-2) which relates corresponding points in stereo images.
The point correspondences(Key points) which are obtained through the above mentioned algorithm are sent into a RANSAC algorithm to generate inliers. These obtained inliers are used to generate the required fundamental matrix through 8-point algorithm. The relation between the point correspondences and the fundamental matrix is as follows,

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x_mx'_m & x_my'_m & x_m & y_mx'_m & y_my'_m & y_m & x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_{1_1} \\ f_{1_2} \\ f_{1_3} \\ f_{2_1} \\ f_{2_1} \\ f_{2_3} \\ f_{3_1} \\ f_{3_2} \\ f_{3_3} \end{bmatrix} = 0$$

From this relation on Singular Value decomposition of the first matrix(A) and later selecting the last column of the V^T matrix and reshaping the obtained column into 3x3 matrix gives us the fundamental matrix.
But, due to the noise in the images the rank of the fundamental matrix generated

here is not 2. So, inorder to enforce the rank of the fundamental matrix we need to decompose it and rebuild to generated a rank 2 fundamental matrix, it is done as shown below,

$$\text{U, D, } V^T = \text{np.linalg.svd(Fundamental\_matrix)}$$
$$\text{D[-1]=0}$$
$$\text{F\_new} = \text{U * D * } V^T$$

F_new is the final required fundamental matrix of rank 2.

2.4 Essential Matrix

The Essential matrix is also similar to Fundamental matrix but with some additional properties, which also relates corresponding points in stereo images assuming that the cameras satisfy the pinhole camera model.
From the obtained Fundamental matrix in the above algorithm, we determine the essential matrix through the following equation,

$$\text{Essential\_matrix} = K^T \text{ x Fundamental\_matrix x K}$$

2.5 Determining Solution

This part of the pipeline involves two steps: First being the triangulation of the 3D point and next is to check cheirality function to determine the Rotation and Translation matrices of the camera.
Initially, we need to decompose the obtained Essential matrix using Singular Value Decomposition to obtain the four possible solutions of Rotation and Translation as follows,

$$\text{U * D * } V^T = \text{np.linalg.svd(Essential\_matrix)}$$

$$C_1 = \text{U(:,3) and } R_1 = \text{U * W * } V^T$$
$$C_2 = -\text{U(:,3) and } R_2 = \text{U * W * } V^T$$
$$C_3 = \text{U(:,3) and } R_3 = \text{U * } W^T * V^T$$
$$C_4 = -\text{U(:,3) and } R_4 = \text{U * } W^T * V^T$$

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

Now, after finding these four possible C and R, we need check for the cheirality, which is the checking of reconstructed points that are in front of the camera, and the condition is,

$$r_3*(\mathrm{X} - \mathrm{C}) > 0$$

Here, r_3 is the third column of the Rotation Matrix, C is the Translation Matrix and X is the 3D point generated through the Linear Triangulation.

2.6 Solution Output

After determining the best solution for the Rotation and Translation matrices, it is possible to translate and rotate the calculated camera pose based on the change between frames. The translation parameters reflect directly to the X, Y, and Z directions. The rotation matrix must be solved to provide a relative rotation to the respective matrices. The resulting plots represent the translation from several perspectives. The 3D plot provides an isometric view of the camera pose, with a Green marker at the camera position and a Red marker used to demonstrate the orientation.

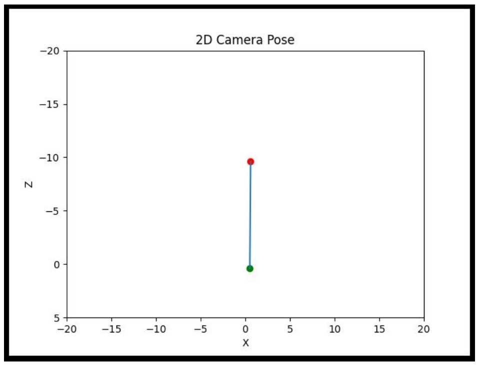

Figure 1: 2D Camera Pose Output

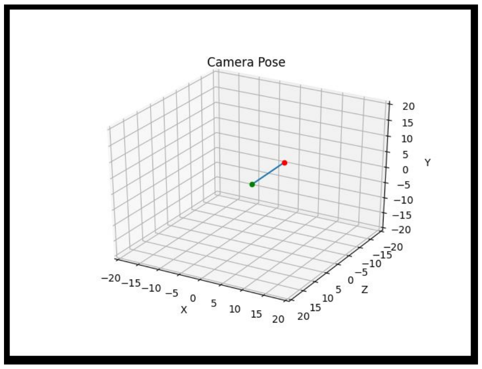

Figure 2: 3D Camera Pose Output

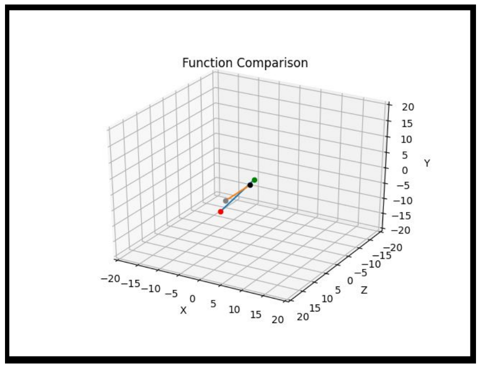

Figure 3: 3D Pose Comparison of built-in functions and calculated translation and rotation

3 Comparison to OpenCV Functions

An additional step in the solution output is the implementation of the built-in OpenCV functions that calculate the Essential Matrix and resulting Rotation/Translation matrices. This can be compared with our calculated output to demonstrate the drift between the calculated result from our implementation and the built-in features.

The comparison shows there can be a slight variation in orientation (angles). However, the translation drift demonstrates mostly little variation. This can be seen in an additional 3D data plot that places the calculation methods side-by-side. The Drift matrices can also be displayed in the command window as the script is running.

4 Non-Linear Depth Solution

The current implementation of the code does not utilize the non-linear depth solution step to solve for the 3D motion. The function for non-linear triangulation has been added to the code.