ENPM 673 Robotics Perception Project 3b: Visual Odometry

Introduction:

In robotics and computer vision, the visual odometry is the process to determine the position and orientation of the camera center by analyzing the associated video frames. It is a crucial concept in Robotics Perception and has been widely used in robots, for example, on the Mars Exploration Rovers. This project is to determine the motion trajectory of camera center using given video frames.

Work Flow:

1. Demosic Image

The given images are Bayer pattern encoded images. A Bayer filter mosaic is the arrangement of color filters which each sensor in a single-sensor digital camera only record one of red, green, or blue color. Since human eye is more sensitive to green light the patterns locate more green sensors to mimic human eyes. It can be recovered to truecolor image. Here we convert Bayer patterns to GBRG patterns using function *demosic*. The figure 1 shows the different image before and after demosic.

Figure 1: The Image Before (Left) and After (Right) demosic

2. Undistort Image

The given frames are recorded using stereo camera. A stereo camera has two or more two lenses. It mimic binocular vision of human eye to make stereo views. However, it always cause the distortion of images. Thus, it is necessary to undistort the images. Here, we use the given function UndistortImage. The figure 2 shows the different image before and after undistort.

Figure 2: The Image Before (Left) and After (Right) Undistort

3. Feature Extraction and Feature Match

To find the visual odometry, the most important part is feature extraction and match in neighbor frames. At present, there are several algorithms to extract feature. Here we use Scale-invariant feature transform (SIFT) algorithm to extract feature points and match them by distance. Since the accuracy of matching points will significant affect the final results, we use the function *vl_sift* and *vl_ubcmatch* in *VL_FEAT* toolbox instead of the self-implemented functions. Thus, please do not forget setup the VL_FEAT toolbox before running this program. Figure 3 shows the feature extraction and match in two neighbor images.

Figure 3: The Matched Point in *k-1* Frame (Left) and *k* Frame (Right)

4. Fundamental Matrix and Essential Matrix

Once we get a set of corresponding points, we need to calculate the fundamental matrix first. Here, we use 8 points algorithm. In 8 points algorithm, there must be more 8 pair of matched points. Unlike a homography, where each corresponding point contributes two constraints (rows in the linear system of equations). To estimate the fundamental matrix, each point only contributes one constraint (row) *(see function F_Matrix).*

After we get the fundamental matrix, we can use formula $E = K^T F K$ to get the essential matrix. K is the intrinsic matrix of camera. We can use given function *ReadCameraModel* to obtain the elements for intrinsic matrix.

5. Decompose Essential Matrix

The essential matrix E can be decomposed to two skew-symmetric matrix S1, S2 and rotation matrix R1, R2. Since E = SR, there are four possible combination of S and R. Since the actual object must be in front of camera, we check which combination can keep the actual object in front of camera. First, we try to reconstruct one actual point object X using essential matrix E, the sets of corresponding points, x1 and x2 (see function *decompose*). We can get four possible coordinate of X. And then, format the coordinate X to [x y z 1]. Here, z is the depth of X. If object X is in front of camera, z must be positive. Thus, we choose R, S that let z is positive.

6. Plot Trajectory of Camera Center

Once we get essential matrix E and decomposed matrix R, S, it is easy to get the trajectory of camera center. Since the translation vector t is the right null space of essential matrix E (Et = 0), we use the MATLAB function *null()*. There is also another method that using SVD to find the singular vector with smallest singular value.

We assume that the coordinate of camera center in first image is (0, 0, 0). The relative coordinate of second frame C2 = R*t. Continue this increment of coordinate to plot the whole trajectory of camera center (Figure 4). X-Z plane represents the ground and Y represents the altitude.

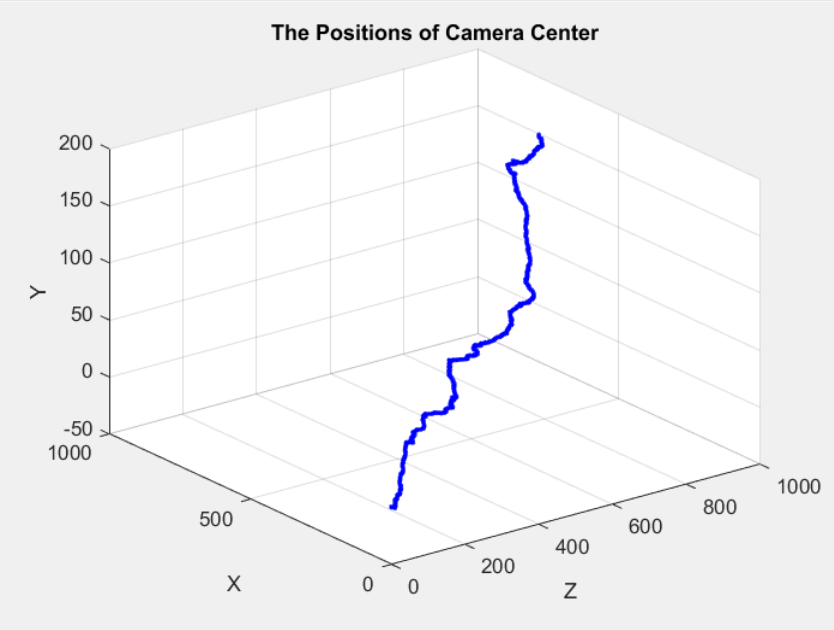

Figure 4: The Trajectory of Camera Center

7. Extra Credit: Trajectory comparison between myself-implemented functions and MATLAB built-in function.

MATLAB has build-in functions to find essential matrix and relative camera pose. The Figure 5 is the trajectory comparison between self-implement code and build in function in computer vision toolbox. Since the full computation will cause several hours, here we compare first 500 frames. It seems different. Since there is no SIFT algorithm for feature extraction in computer vision toolbox, we use harris feature extraction. Also, the camera parameter cannot be extract from image because there is too much distortion, we use the camera parameter same to self-implemented code. Right now, we are not sure that which is correct. We really hope that we can get the actual trajectory to check which is correct. If our trajectory has problems, we can use the actual trajectory to correct our code in the future.

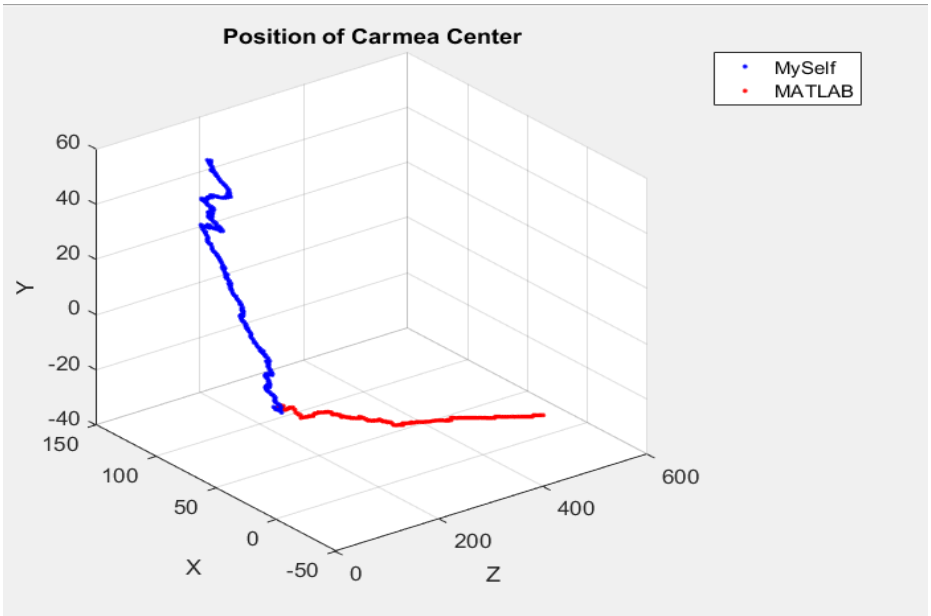

Figure 5: Comparison of Myself function and MATLAB build in Function

Appendix:

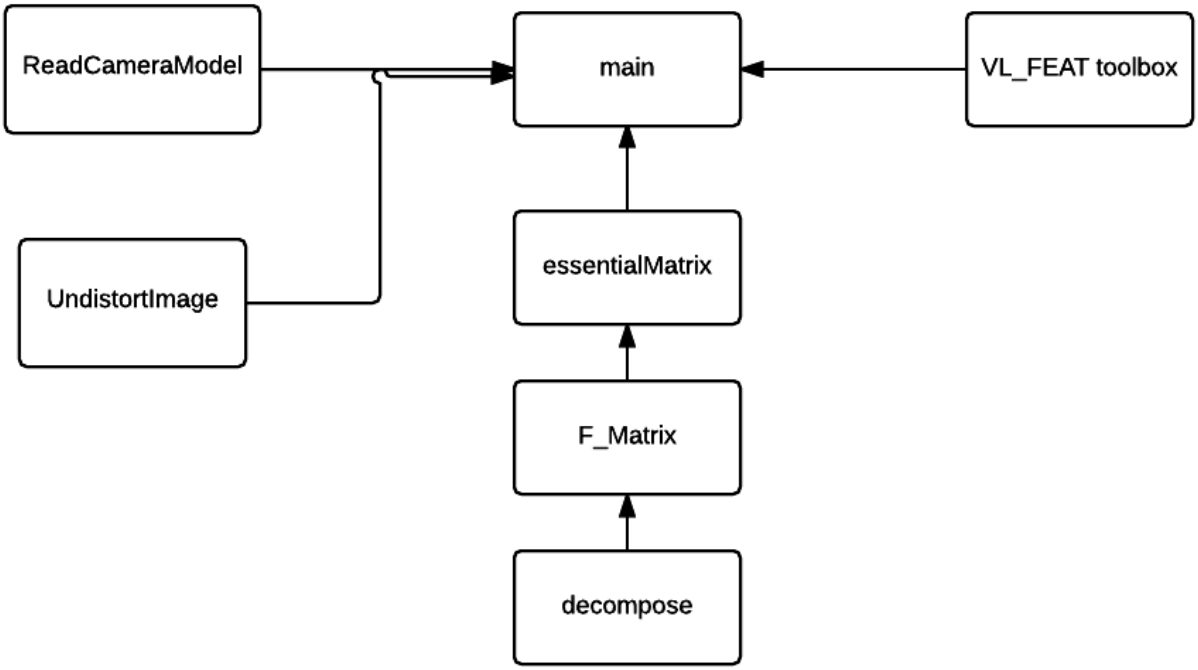

Figure 6: The Relationship between Functions