Akash Atharv, Rishabh Choudhary, Sanchit Gupta

Project-5 Report

Visual Odometry

Objective

To estimate the 3D motion of the camera and provide a plot of the trajectory of the camera as output given the frames of a driving sequence taken by a camera in a car.

Modules needed to run:

1. import cv2
2. import numpy
3. import matplotlib.pyplot as plt
4. from mpl_toolkits.mplot3d import Axes3D
5. import random
6. import ReadCameraModel
7. import UndistortImage
8. import os

In order to run this code import the dataset from the drive link and run the file odometry.py. Make sure that the files ReadCameraModel.py and UndistortImage.py are in the same folder/project from which the odometry.py is being executed.

The output camera center trajectory and dataset are included in the drive folder attached below.

The Google drive link is as given below :

Pipeline

Data preparation:

- The input images are provided in the BAYER format. The images were converted to Color format using cv2.cvtColor().
- Using the given scripts, camera parameters were read using ReadCameraModel()
- The images were undistorted using UndistortImage() script given as part of dataset.

Processing:

- **Feature extraction and matching:** For the undistorted images obtained in data preparation step, matching features were extracted for consecutive frames using SIFT feature descriptor. A Flann based matcher is used to find the best matches. After the SIFT descriptors are obtained, we find the coordinates of the key points that match in both the corresponding windows. We have created the function keypoints( ) to do this. We use the K-nearest-neighbors algorithm of the key-point matching technique to obtain the good key points.

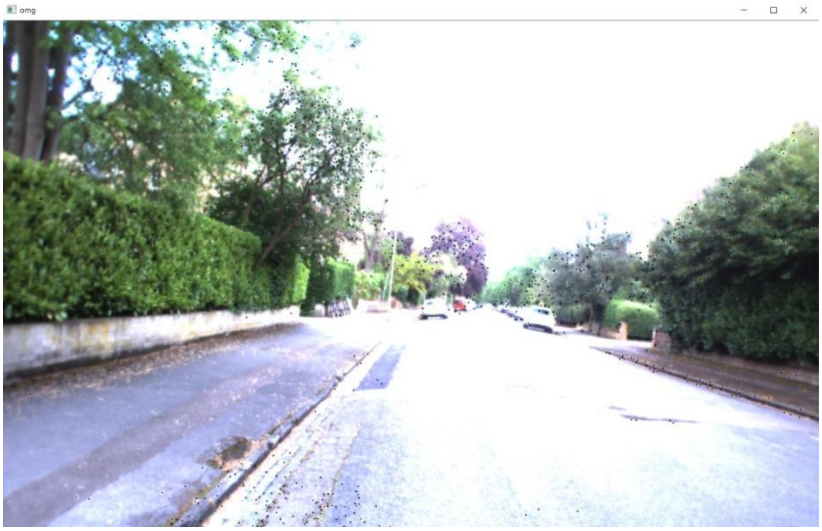


Fig all the key points drawn on a frame

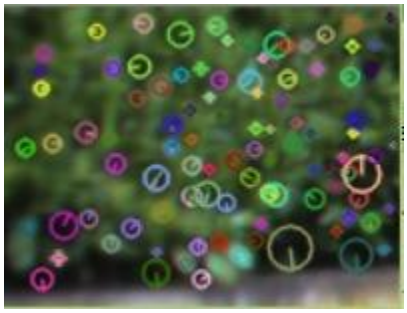


Fig: Key features in a window

- **Estimating Fundamental:** Using these m correspondences (m >=8), the Fundamental Matrix was obtained for the pair of images using the 8-point algorithm. The fundamental matrix was estimated over a set of random 8-point correspondences repetitively in a RANSAC algorithm until the desired number of outliers are rejected. The key points that are used here are normalized before they are sent in.

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

To solve the above equation, we use least square method and form the A matrix. This matrix is formed using the following equation:

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

After forming the A matrix, we solve the following equation of the form AX = 0

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx'_m & x_my'_m & x_m & y_mx'_m & y_my'_m & y_m & x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

After reshaping the result of the above, we get the fundamental matrix.

- **Normalization:** For normalizing the key points we calculate the mean of x and y and then subtract the same from all other values of x and y, using these values we construct 2 matrices T1 and T2 for 1st and the 2nd image respectively. Using these matrices, we multiply all our points which will be used to estimate F. this transformation centers the points at origin.

- **RANSAC:** The RANSAC has a cap on the maximum number of iterations as 500 or when it finds 50% inliers of the total points. Any key point is termed as inlier when the x'Fx = 0 is satisfied and the error is within 0.6 absolute value. The x' value is the point correspondence from the 2nd image and x is the same point correspondence from the first image. Using RANSAC we reject outliers and with good number of inliers form the F matrix which is then used to estimate E matrix.

- **Estimating Essential Matrix:** Essential matrix can be constructed from Fundamental Matrix by incorporating the intrinsic camera parameters to find the relative pose between the two images. The Essential matrix has been computed using the Fundamental matrix (f) and The Intrinsic camera matrix(K) using the given formula:

$$\boldsymbol{E = K^T.F.K}$$

Due to noise in K, the singular values of E are corrected by reconstructing it with [1,1,0] singular values as shown below:

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

where U and V are the SVD of matrix E.

- **Estimating Camera Pose from Essential Matrix:** The camera pose has six parameters, Rotation(Roll, pitch, Yaw) and Translation (X, Y, Z) of the camera center with respect to the world.
The camera pose can be written as

$$P = KR[I_{3x3} \quad -C]$$

From the Essential Matrix, four camera configurations have been obtained for (R,C) as shown below:

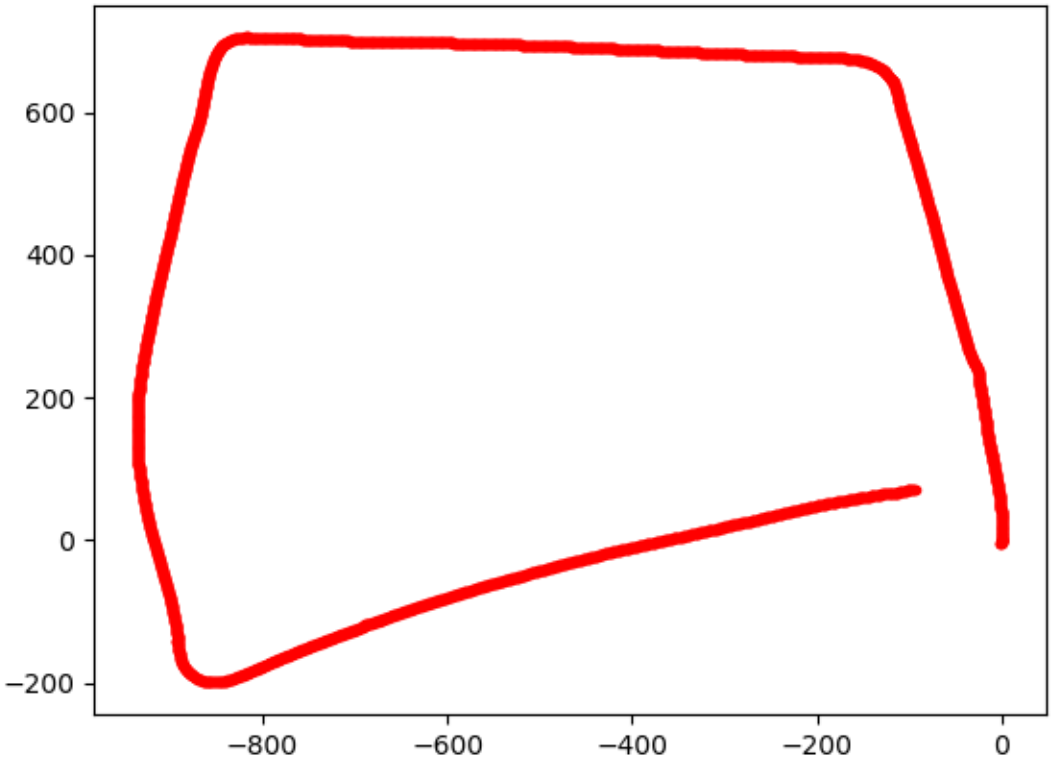
1. C1 =U(:3) and R1=UWV^T
2. C2 =−U(:3) and R2=UWV^T
3. C3 =U(:3) and R3=UW^TV^T
4. C4 =−U(:3) and R4=UW^TV^T

The determinant of the Rotation Matrices obtained should be equal to 1. If the det (R)= -1, the camera pose C and R needs to be negated in sign, i.e. C = -C, R = -R.

- **Triangulation using the camera poses:** Using each of the pairs of Camera poses obtained, we can triangulate a set of 3D points using linear least squares. This point can be triangulated using the condition that x = PX and x' = P'X, where x and x' are image points in two frames, P and P' are Camera poses from the two frames and X is the 3D point.
  For the 4 possible solutions, we will have 4 triangulated points. However, this is ambiguous. To find the correct unique camera pose, we need to remove this disambiguity. This can be done using chirality condition, which states that the reconstructed points must lie in front of the camera.
  The chirality condition specifies that a 3D Point X lies in front of the camera if $r_3(X - C) > 0$
  Where r3 is the 3rd row of rotation matrix and C is the camera center.

  Using this condition, we obtained the correct camera pose (R, C).
- **Track the Camera center:** Using the valid camera pose solution obtained in previous step, we track the camera position by plotting the value of C for each frame.

Results from custom functions:



The above graphs starts from (0,0) and plots the camera center as the frames move. The result is inverted(negative x), because we plotted camera center as (x_center, -z_center). The correct orientation can be obtained by plotting (-x_center, z_center).

The above code has been executed from the file **proj5_custom.py**

Comparison with OpenCV inbuilt functions:

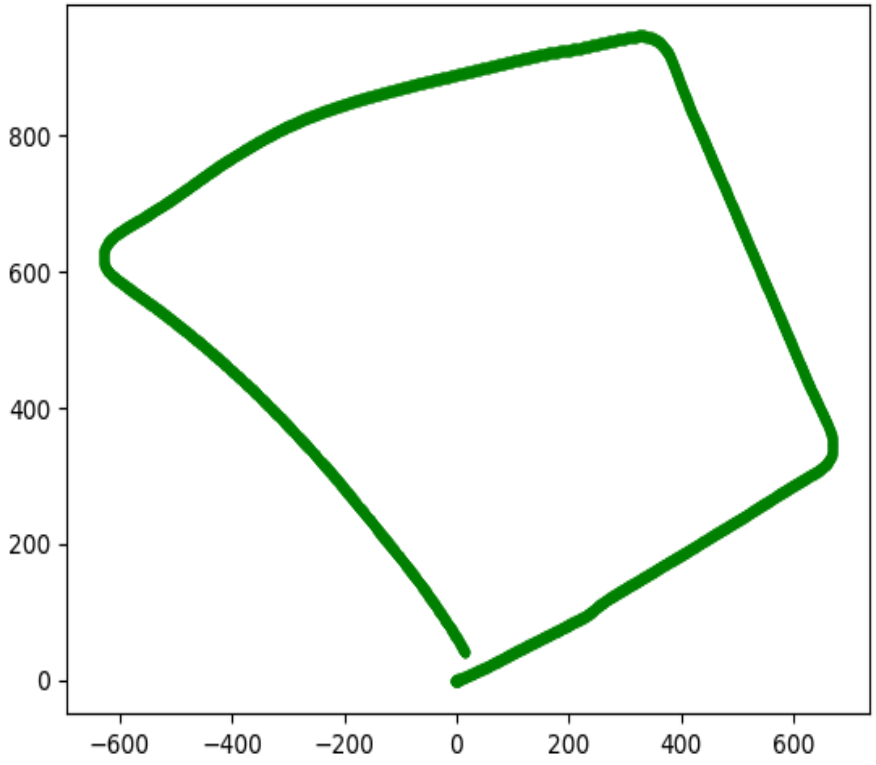
The above pipeline has been implemented with functions written from scratch except feature matching (for which we used opencv's SIFT feature descriptor). To compare the accuracy of the output, we have compared it against baselines results obtained from inbuilt 3D reconstruction functions using opencv. The following functions were used from OpenCV library to compare the results with our pipeline.

1. **cv2.findEssentialMat():** This matrix computes the best estimate of essential matrix given a set of matching points in two given frames.
2. **cv2.recoverPose():** This matrix computes the correct Rotation and Translation parameters of the camera using the essential matrix, paired points and Intrinsic camera matrix.

Using these two functions, we computed the Camera center for each frame relative to previous frame using $C = -R^T.t$, where R is the rotation matrix and t is the translation matrix obtained from cv2.recoverPose().

We then plotted the camera center for each frame to track the camera position as shown below

Results from built-in functions:



The above trajectory plots the camera center (starting from (0,0)) using inbuilt opencv functions. As we can see, the output of both the trajectories is quite similar.

The above code has been executed from the file **proj5_inbuilt.py**

Challenges:

1. **Slow Process:**
   Due to the number of iterations in RANSAC and multiple key points, the overall process is slow and takes a lot of time to plot the final result.

2. **Key Point selection, Windowing:**
   while selection of the key points coordinates from SIFT and then storing the coordinates, we faced a

lot of problems as every time each window would be treated as a separate image and the coordinates would come according to it and not according to the initial image. After plotting the key point correspondences, we realized this mistake and corrected the same.

3. **Linear PnP:**
   While researching on the internet we came across linear PnP as a faster and better alternative and we tried to implement the same from the 3rd image that we read. The PnP would take in the previously calculated 2d points, 3d points and K(calibration matrix) and output the new R and C. Using this we made our algorithm a lot faster but after couple of working for a couple of images, the result of R would drop to an identity matrix and C would become 0 and for rest of the images it would not come out of this. Due to this we had to drop using PnP.

References:

1. https://cmsc733.github.io/2019/proj/p3/#featmatch
2. https://www.youtube.com/watch?v=K-j704F6F7Q