ENPM673 - Perception for Autonomous Robots
Project 5 - Visual Odometry

Abstract
The process of Visual Odometry involves determining the position and orientation of a robot by analyzing its camera images and can be used in navigation to estimate the change in position over time. In this approach, we have a camera rigidly attached to a moving car from which we construct a trajectory.

1. Data Preparation

- The given images are black and white in Bayer format. To reconstruct a full color image and render these images in viewable format, we apply the demosaic function. The alignment of the images are in GBRG.
- Secondly, we extract the camera parameters fx, fy, cx, cy, G camera image. LUT using `ReadCameraModel.py`
- Next step is to undistort the image using the undistortion lookup table LUT obtained from above to get a better image which we can use for feature detection and to compute visual odometry. This is done by applying gaussian filter to remove the noise from the color images.
- The front part of the car always remains stationary with respect to the camera and hence we ignore this part of the region where there should be no change in the correspondence points between the images when the car moves forward.
- We convert the RGB image to grayscale and to improve the contrast we tried to equalize the image and also increasing contrast of the image for better feature detection.

2. Visual Odometry Pipeline

We use Visual Odometry method to determine the trajectory of the moving car and implement the following pipeline to estimate the translation and rotation between successive frames in the input sequence.

2.1 Feature Matching

- Firstly, to find the point correspondences between successive frames, the keypoint algorithm we have started with SIFT, Scale-Invariant Feature Transform which is a feature detection algorithm we use to detect and describe local features in images. SIFT keypoints of objects are first extracted from the input image and an object is recognized in a new image by individually comparing each feature from the input image and finding candidate matching based on the Euclidean distance of its feature vectors.
- Since SIFT feature descriptor is invariant to orientation and illumination changes, it is able to robustly identify objects even among cluster and partial occlusion.
- However, we found ORB (Oriented FAST and Rotated BRIEF) which turned out to be a better alternative to SIFT or SURF. ORB basically uses FAST to find keypoints and Harris corner measure to to find top N points. Since FAST doesn't compute orientation, it

used the intensity weighted centroid of the patch to determine the direction of the vector from its corner point to centroid and hence the orientation.

- Finally, we used Lucas-Kanade method with pyramids to calculate optical flow iteratively to sparsely track features.
- The matching features were sorted by the error and selected best 1000 points to compute RANSAC in the next step comparatively faster.

2.2 Fundamental Matrix F via RANSAC

- In this step, we estimate fundamental matrix F, relates the corresponding set of points in two different view images from the previous step. However, to estimate the fundamental matrix from all the corresponding points it is necessary to remove unwanted points and normalize the points to make our results robust. The homogeneous linear system with F and and 9 unknowns can be given as below.

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

- Where (x_1,y_1), (x_1', y_1') are normalized corresponding points and the solution to this problem is the last column of V in [U, S, V] = svd(A). In F matrix estimation, each point only contributes one constraints since the epipolar constraint is a scalar equation and hence we require at least 8 point to solve the above equation and hence is called Eight-point algorithm. We linearly estimate the fundamental matrix F such that $x_2^T Fx_1 = 0$ and solving for the linear least squares Ax=0.
- In this process, the centroid of all the points are found and then by appropriate translation, centroid is translated to the origin. Nextly, the mean distance of all the points from origin is found and by applying proper scaling factor to the points the mean distance of all the points from origin is converted to $2^{1/2}$.
- Using feature descriptors to find the point correspondences as in the previous step renders the data to be noisy and contains several outliers. We use RANSAC algorithm to remove these outliers to get better estimate of the fundamental matrix.

- For normalizing the scaling factor was decided using the mean distance computed using square root of variance.
- After normalization, all the normalized corresponding 8 inlier points satisfying the fundamental matrix are found by randomly selecting 8 points from set of corresponding points to check how much they satisfy the fundamental matrix and the best is chosen. After estimating the fundamental matrix it is denormalized.

2.3 Estimate Essential Matrix E from F

- With camera parameters K and fundamental matrix F found previously, the essential matrix E is determined as E = K^T F K where E will be

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

2.4 Decomposing E into Translation T and Rotation R

- The value of rotation R and translation T are found by singular value decomposition (SVD) of the essential matrix E.

2.5 Estimating Camera Pose from E

- The above step gives four possible camera pose configurations given by (C_1,R_1), (C_2,R_2), (C_3,R_3), (C_4,R_4) and the pose configurations are given by

$$\begin{aligned} C_1 &= U(:,3) \text{ and } R_1 = UWV^T \\ C_2 &= -U(:,3) \text{ and } R_2 = UWV^T \\ C_3 &= U(:,3) \text{ and } R_3 = UW^TV^T \\ C_4 &= -U(:,3) \text{ and } R_4 = UW^TV^T \end{aligned}$$

- If the det(R) has to be 1. If det(R) = -1 we correct both C and R as C = -C and R = -R.
- We get the following translation and rotation parameters such that,
  R1 = UWV^T , R2 = UW^TV^T, T1 = U(:,2) and T2 = -T1

2.5 Plotting Camera Center

- After finding the rotation R and translation T of the camera, the camera center is obtained and plotted and the trajectory can be found without knowing the exact scale.

3. Additional Steps in Pipeline

To obtain better estimates, we solved non-linearly for the depth and 3D motion using the algebraic error.

3.1 Observations

- Gaussian blur filtering in the data preparation pipeline drastically improved our results for the trajectory.
- In Triangulation, Chirality condition was quite unreliable so we customized to this condition below

  np.sum([R t] [x y z]^T) > 0
- To compute fundamental matrix instead of using inbuilt RANSAC we resorted to customized RANSAC for better results but the computing was considerably slowed down.

3.2 Non-linear Triangulation

- For two camera poses and linearly triangulated points X, the locations of the 3D points are refined to minimize reprojection error which is computed by measuring error between measurement and projected 3D point and can be given as below.

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

- $\widetilde{X}$ is the homogeneous representation of X and P^T_i is each row of camera projection matrix P.
- The minimization is highly nonlinear due to the divisions.

3.3 Non-linear PnP

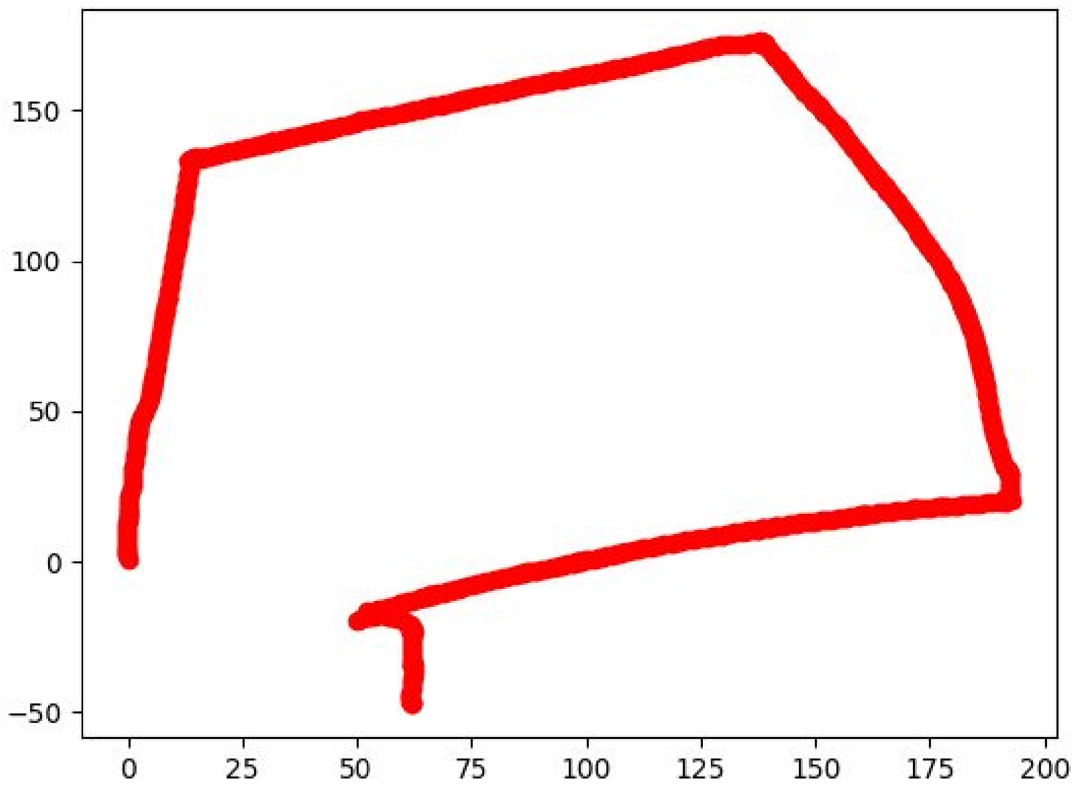
- The linear PnP minimizes algebraic error. In non-linear perspective-n-points, the camera pose is refined to minimize the reprojection error between the measurement and projected 3D point by enforcing orthogonality of the rotation matrix R=R(q) and hence represented as below.

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

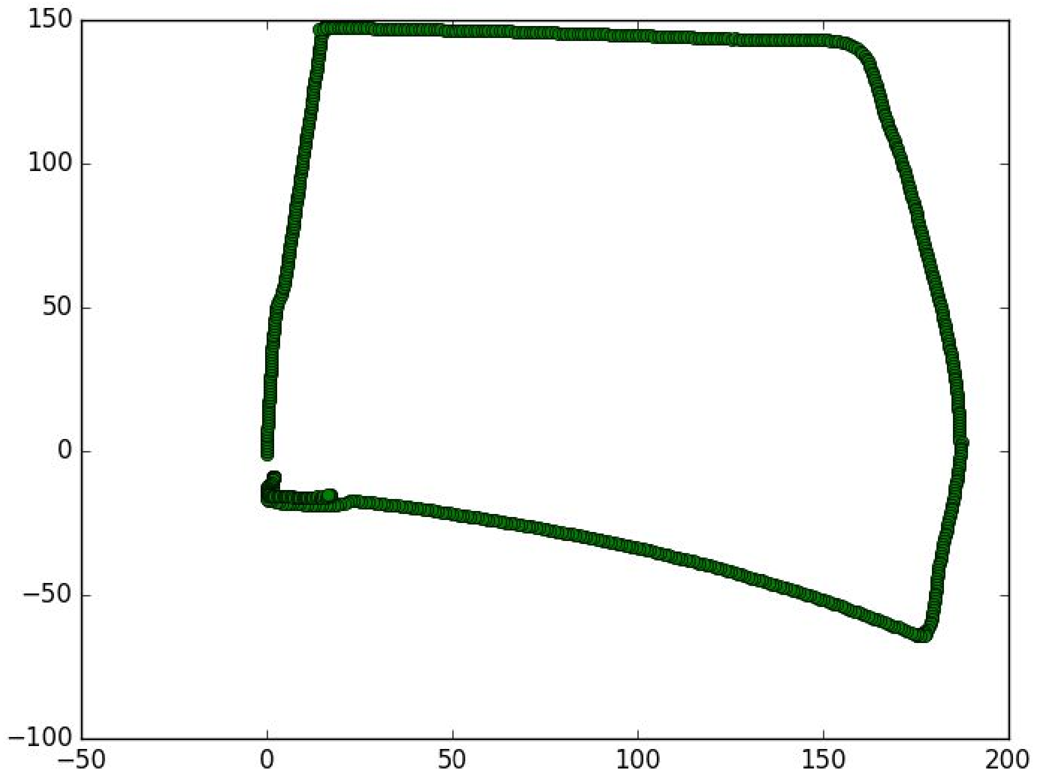
- We compute P such that P = KR [I_{3x3} - C] , $\widetilde{X}$ is the homogeneous representation of X and q is the 4-dimensional quaternion.
- The minimization is highly non-linear owing to the divisions and quaternion parameterization.

4. Comparison

In addition to the above pipeline, we have also found the trajectory of the camera by using OpenCV's inbuilt functions. On a close comparison the trajectories found by both the methods are found to be one and the same with a slight overshoot in the proposed theoretical approach.



Trajectory from the proposed Pipeline



Trajectory from Inbuilt functions Pipeline

How to run code

1. Unzip the folder which has the code, input sequences and the datasets.
2. Copy the Stereo images to its folder in Oxford_Dataset/Stereo/
3. Run data_preparation.py inside the Oxford_Dataset folder
4. Each of the following code parts needs to be separately run.
5. To compute fundamental matrix instead of using inbuilt RANSAC we resorted to customized RANSAC for better results but the computing was considerably slowed down. We have used both the code for comparison.
6. For Visual Odometry proposed pipeline run

   ```
   python3 VO.py
   ```

8. For Comparison with inbuilt OpenCV functions run

   ```
   python3 verification.py
   ```

Output Video:
https://drive.google.com/open?id=12Wfx0IMpCe19iwCPdsoLIl3x8J590kgm

Project Team

1. Nantha Kumar Sunder (UID: 116104777)
2. Nithish Kumar (UID: 116316958)
3. Rama Prashanth (UID: 116428941)

References

1. Computer Vision, A Modern Approach, Forsyth and Ponce , http://cmuems.com/excap/readings/forsyth-ponce-computer-vision-a-modern-approach.pdf
2. https://cmsc733.github.io/2019/proj/p3/
3. https://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html
4. Fundamental Matrix, https://www.youtube.com/watch?v=K-j704F6F7Q
5. Dataset used is by courtesy of Oxford's Robotics Institute