Implementation of Stereo Odometry Using Careful Feature Selection and Tracking

Mayank Mittal (14376), Ritwik Bera (14561)

Abstract

This report provides a brief overview of an algorithm for stereo odometry, which has been adapted from [1]. The algorithm, implemented on MATLAB, relies on careful selection and matching of features over a pair of stereo images, followed by egomotion estimation using the features chosen. To estimate the rotation incrementally Nister's Five Point Algorithm is used, while the translation is estimated by minimizing reprojection- based error function. Evaluating the code over KITTI dataset sequences yield reasonably accurate odometry estimates compared to the original algorithm in [1].

1 Introduction

With the recent advances in robotics, a common problem, often referred to as the localization problem, is to estimate a robot's pose. Odometry is one such method to do so. In odometry, the pose of the robot is estimated incrementally relative to the changes in its surrounding. Traditional systems used sensors like rotary encoders to find out position by measuring the angle rotated by the wheel's shaft. However, this method is futile for non-wheeled locomotion systems, such as legged robots, and aerial vehicles. Additionally, rotary encoders are erroneous as wheels tend to slip and drift which cannot be measured from the encoders. This error is compounded when the vehicle operates over non-smooth surfaces.

Hence, a better way to perform odometry is to rely on visual systems like cameras, and LiDARs. Visual odometry is considered a subpart of a bigger structure from motion (sfm) problem in computer vision. Although it can be used in both outdoor and indoor environments, it is considered accurate only in feature rich scenes as opposed to a texture-less environments.

A pictorial overview of the algorithm is shown in Figure 1. It has been adapted from the works mentioned in [1]. The remaining sections of the report explain the various steps in the flowchart followed by the results of the experiments done.

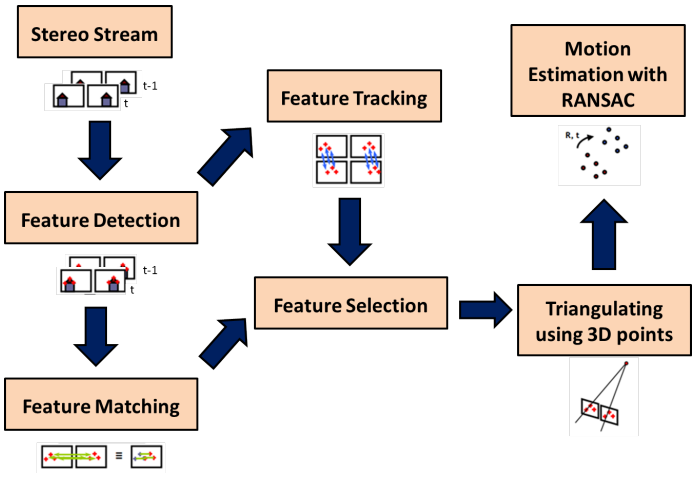


Figure 1: Overview of the Algorithm

2 Algorithm

2.1 Pre- Processing

The stereo pair receive stream of images at a frequency dependent upon their sampling rates. In this step, these images are undistorted and rectified using the stereo cameras parameters and two pairs of left and right camera images for consecutive time frames t and $t-1$ is given as the input further in the algorithm.

2.2 Feature Selection and Tracking

Feature Detection

Features in an image are often locations in the image which differ from the immediate surroundings. These may be corners, edges,or even blobs. Over the years, several methods to detect robust and invariant features have been developed such as ORB, SIFT, SURF, to name a few. We have used the minimum eigenvalue feature detector as described in [5] to find features across all the four images in our implementation.

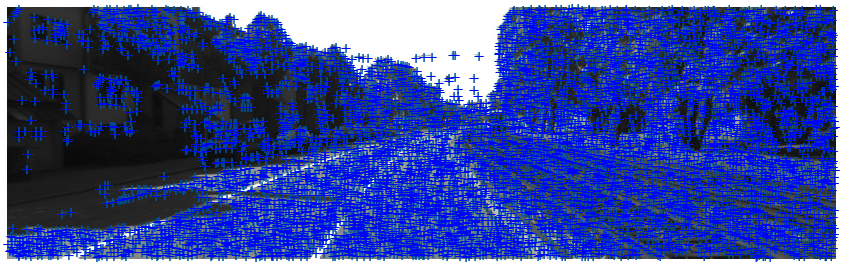


Figure 2: Minimum Eigenvalue Feature points detected in left camera at an instant t

Feature Matching

Feature matching helps in finding out correspondences between feature points in two images. A common way to do so is by calculating Sum-of-Absolute-Differences (SAD) over a window across each feature point in an image and taking the feature with minimum disparity one as a matched feature. However besides finding correspondences, we have performed circular matching, as given in [2], which tries to match features across all the four images, thereby rejecting outliers and selecting only invariant and strong features.

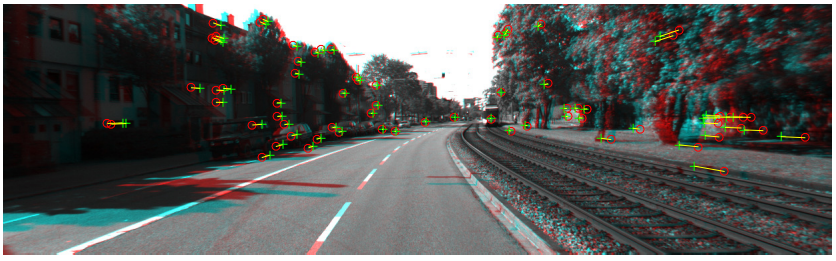


Figure 3: Matched features in left camera at instants t (features marked as green plus signs) and t-1 (features marked as red circles)

Feature Tracking

Along with feature strength, each feature point has been given an additional property called *age*, which is the number of frames a feature point has lasted since its first appearance. As suggested in [1], if a feature point becomes absent for more than three consecutive frames it is discarded.

In our implementation, we have applied Kanade–Lucas–Tomasi feature tracker [6] for tracking of feature points between frames.

Feature Selection

As reported in [1], using all matched features for ego-motion estimation simply increases the computation time of the algorithm, while carefully selecting features tend to increase the accuracy of the algorithm. We thus use bucketing over the image to select only a given number of features from each bucket on the basis of the following policy for two feature points x and y in a bucket:

$$select(x, y) = \begin{cases} stronger(x, y), & \text{if } age(x) = age(y) \\ older(x, y), & \text{if } age(x) \neq age(y) \end{cases}$$

This ensures a uniform distribution of far and near features across the image, thereby preventing biasing of the rotation and translation being estimated. A sample output of the procedure up to now is shown in Figure 4.

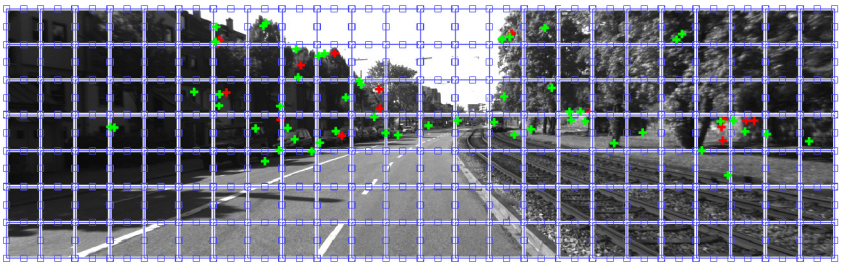


Figure 4: Feature points selected through bucketing: marked in green, while weak features in each bucket: marked in red

Triangulation

Triangulation of feature points observed in the left and right camera images is required while estimating the translation from the 3D point cloud generated from the $t-1$ frame. Given the cameras' optical model and the extrinsic parameters, we compute the disparity between the images and use it to estimate the coordinates of the feature points observed, on basis of following formula:

$$z = \frac{Tf}{x_l - x_r},$$
$$\frac{x}{z} = \frac{x_l}{f}$$

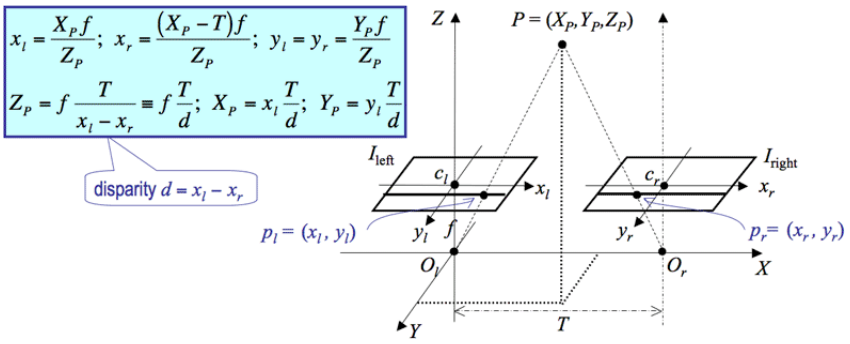


Figure 5: Triangulation of a point observed in the left and right camera images

2.3 Egomotion Estimation

The implementation so far has allowed generation of a 3D point cloud for the time frame $t-1$. It would be used to estimate translation, once we have estimated the rotation first. The reason for splitting the egomotion estimation into two different steps rather than a single optimization function with greater number of parameters is that it reduces the computational time.

So far we have been working in the reference frame of the cameras, so we assume the features in the world are moving around the camera. This might be the case (if there is any movement in the scene as seen by the camera, e.g. a person walks by), but for the purposes of visual odometry task, we assume that most of the features seen by the camera are fixed with respect to the world reference frame. This is a reasonable assumption, as long as the majority of the features in the frame belong to stationary objects. Also, we can accommodate for some movement in the world, as we will see later. Assuming the 3D points are stationary in the world reference frame, any apparent movement of the 3D point clouds between frames, as perceived in the camera reference frame, is actually due to the movement of the camera itself.

At this point we make the plausibility argument that by keeping track of the movement of a static point cloud from frame to frame, we can deduce the movement of the camera. We will develop this rigorously after developing some of the necessary pieces.

Rotation Estimation

Rotation is estimated with the five point method typically used in monocular cases. Therefore, only left camera is used for rotation estimation.

Nister's five point algorithm [4] is used commonly for estimation of extrinsic parameters in a stereo camera setup and exploits the epipolar constraint between points (set of five) in the image planes of the left and right cameras. For the purpose of this project rotation is estimated by plugging in the left camera's feature points at time t and t-1 and the corresponding rotation matrix obtained tells us the rotation of the camera between the two consecutive frames.

Nister's five point algorithm exploits the following constraints to estimate the extrinsic parameters with just five pairs of points (instead of 8 in the eight-point algorithm).

$$2EE^TE - trace(EE^T)E = 0$$
$$det(E) = 0$$

$$x_l^T E x_r = 0 \tag{1}$$

The advantage of using the five point algorithm is that it gives a closed form solution and is more accurate than the iterative methods. Also it is free from calibration errors because it is a monocular method and utilizes only the left camera.

Often, five point method is used in conjunction with RANSAC. A number of random five point subsets are taken from the total set of points, and essential matrix is calculated for each subset. Finally, the essential matrix that has the largest set of inliers among all the points is selected as the final solution.

Translation Estimation

3D feature points generated from the stereo data of the previous frame (t-1) are reprojected to the image plane in the current frame (t) after undergoing rotation and translation (which tells us about the motion of the camera between the two frames). The error between the reprojected 3D points and the 2D points in the image plane obtained from feature selection (in the current time frame) is computed.

An alternate way of computing error would have been two generate 3D point clouds for both time frames and evaluate the total euclidean error, but that method has been generally found to be more error prone.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \pi(X, R, t) = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

Translation is then calculated by iteratively minimizing the following error function.

$$\sum_{i=1}^{n} \|x_l - \pi(X_l, R, t)\|^2 + \|x_r - \pi(X_r, R, t)\|^2 \tag{3}$$

where $\pi(X, R, t)$ refers to the reprojected 3D points in the image plane, x_l and x_r are the coordinates of the feature points in the image planes for the current time step.

Translation estimation minimizes the reprojection error in both cameras and hence give a more accurate estimate than monocular odometry methods which rely on minimization of error in a single camera.

3 Experiments and Results

The algorithm as explained above was written on MATLAB while using a few functions from the Computer Vision and Optimization Toolboxes available with the student license version.

The code was evaluated on two KITTI Dataset [3] sequences: City 01 and Residential 07 with ground truth poses available for the latter one only. The former has been recorded by a vehicle moving on a straight road with a gentle curve in the start, as can be concluded from the trajectory estimate in Figure 6, while a comparison for the results obtained for the second dataset with respect to the ground truth is shown in Figure ??.

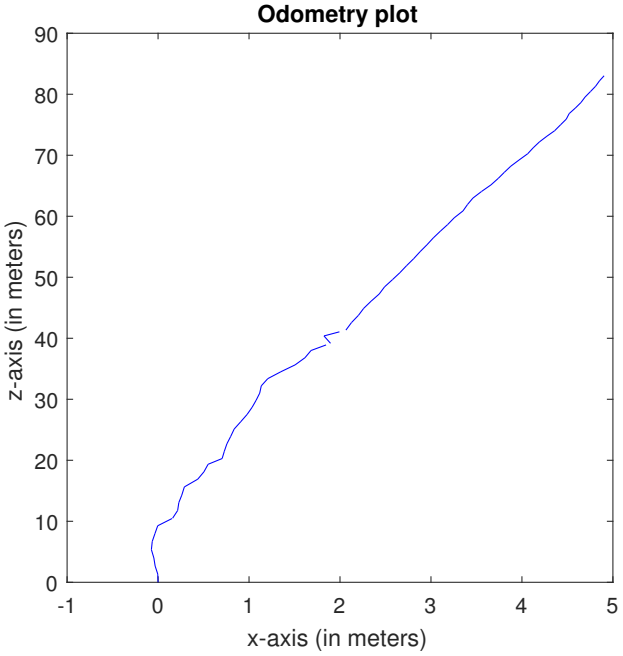
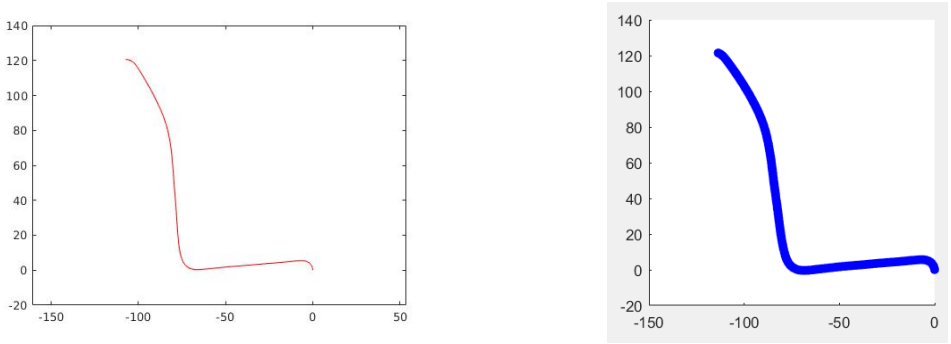


Figure 6: Estimated Trajectory of vehicle in KITTI City Dataset Sequence 01 (Ground truth unavailable)



(a) Ground Truth

(b) Estimated Odometry

Figure 7: Trajectory of Vehicle in KITTI Dataset: Residential Sequence 07

4 Conclusion

The results above show a successful implementation of stereo odomoetry based on careful feature selection and tracking as adapted from [1]. However, our current implementation on MATLAB takes 6-7 seconds for every time step. This is considerably high for real time implementation. Implementing this using OpenCV and C++ would probably help resolve this issue.

As mentioned in the paper [1], we can further improve the results by using dead reckoning sensors like IMU and apply Kalman filter to improve our rotation estimates. Due to the lack of time and availability of IMU dataset, this part has been left as one of the future works for our stereo odometry implementation.

References

[1] I. Cvišić and I. Petrović. Stereo odometry based on careful feature selection and tracking. In *Mobile Robots (ECMR), 2015 European Conference on*, pages 1–6. IEEE, 2015.

[2] A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 963–968. Ieee, 2011.

[3] B. Kitt, A. Geiger, and H. Lategahn. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 486–492. IEEE, 2010.

[4] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[5] J. Shi and C. Tomasi. Good features to track. pages 593–600, 1994.

[6] C. Tomasi and T. Kanade. Detection and tracking of point features, 1991.