Stereo Visual Odometry

1st Benjamin Bahto
dept. Automatic control & Electronics
Faculty of Electrical Engineering
Sarajevo, Bosnia & Herzegovina
bbahto1@etf.unsa.ba

2nd Edin Omerović
dept. Automatic control & Electronics
Faculty of Electrical Engineering
Sarajevo, Bosnia & Herzegovina
eomerovic3@etf.unsa.ba

Abstract

This paper describes a visual odometry algorithm for estimating frame-to-frame camera motion from successive stereo image pairs. The paper includes a detailed description of the algorithm and experimental evaluation.

Index Terms

computer vision, visual odometry, motion tracking, ego-localization

I. Introduction

An important application of computer vision is autonomous navigation of vehicles and robots. Effective use of video sensors for obstacle detection and navigation has been a goal in ground vehicle robotics for many years.

Localization is a crucial capability for autonomous ground vehicles. In the past, localization was typically performed using a combination of wheel odometry (using encoders) and inertial sensor measurement (accelerometers and/or gyroscopes). However, this approach has two limitations; inertial sensors are prone to drift because they rely on integrating measured value to calculate movement, and wheel odometry is unreliable in rough terrain (wheels tend to slip and sink).

Visual odometry, which estimates vehicle motion from a sequence of camera images, offers a complement to the aforementioned approach. It is insensitive to soil mechanics, produces a full 6DOF motion estimate, and has lower drift rates than all but the most expensive inertial measurement units (IMUs) [1]. Visual odometry is a prerequisite for applications like obstacle detection, simultaneous localization and mapping (SLAM), and other tasks. Visual-SLAM (VSLAM) is a much more evolved variant of visual odometry, which obtains a global estimate of robot position and orientation while mapping the environment at the same time, using only visual inputs from a camera.

Still, with significant improvements both in sensors and computing hardware, VSLAM is one of the most challenging open problems for developing autonomous robots and vehicles. However, VSLAM will not be discussed in our paper. That is, our interest lies in incremental, real-time, low-latency methods for estimating camera motion.

Some of the challenges encountered by visual odometry algorithms are:

1) Varying lighting conditions
2) Insufficient scene overlap between consecutive frames
3) Lack of texture to accurately estimate motion

Our implementation of stereo visual odometry algorithm is a variation of [1] one by Andrew Howard. We have used KITTI visual odometry dataset for experimentation and validation of models. The detailed explanation of the algorithm and results generated by it are presented in the following chapters.

II. Algorithm

This chapter describes a real-time stereo visual odometry algorithm that is particularly well-suited for ground vehicle applications. The motive for adopting a stereo, rather than monocular, approach derives from the observation that many ground vehicles are already equipped with stereo cameras for obstacle detection and terrain sensing and that the stereo range data produced by these systems can be exploited for visual odometry.

For our stereo visual odometry algorithm, we assume that the robot is equipped with a stereo camera pair, and that the images have been processed using some kind of preprocessing. Real-time stereo algorithms typically go through the following steps:

- Resizing: change the actual number of pixels that represent a picture.
- De-colorization: convert color images to grayscale to reduce computation and memory complexity.
- Rectification: warp the original left/right images such that epipolar lines are aligned with the rows of the image. These rectified images correspond to the images one would obtain from a virtual pair of perfectly aligned perspective camera models.
- Pre-filtering: smooth the rectified images with an edge-preserving filter to remove high-frequency components.

After the preprocessing steps, a sequence of stereo images is forwarded to the algorithm. Let J_a and J_b denote the preprocessed images for frames a and b respectively. The basics of the algorithm are as follows, for a given pair of frames:

1) Detect features in each frame
2) Match features between frames
3) Find the largest set of self-consistent matches (inliers)
4) Find the frame-to-frame motion that minimizes the reprojection error for features in the inlier set.

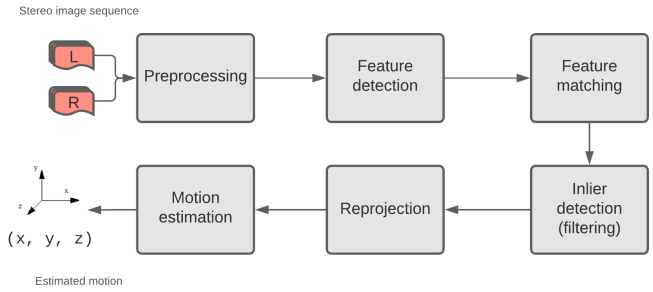

Fig. 1. Stereo visual odometry pipeline

A. Feature detection: $(J_a; J_b) \rightarrow (F_a; F_b)$

Let $f_a = (j, w, s)$ denote a feature with image location j, world location w and descriptor s; let F_a denote the set of all such features.

In this case, for the process of feature detection, we used FAST(Features from Accelerated Segment Test) corner detector [4], where the features are detected on the left picture at time T. Reason for using this corner detector rather than SIFT or SURF is because FAST is computationally less expensive. There are two conditions that a set of pixels needs to satisfy in order for a FAST to detect a corner. These conditions can be written in mathematical form ($\mathbf{I}_p$ is the denoted pixel p, $\mathbf{I}_x$ are a set of N adjacent pixels in the form of a circle and, **t** is the threshold):

1) A set of N adjacent pixels S, $\forall$ x $\in$ S, the intensity of $\mathrm{I}_x > \mathrm{I}_p$ + t
2) A set of N adjacent pixels S, $\forall$ x $\in$ S, the intensity of $\mathrm{I}_x < \mathrm{I}_p$ - t

So when either of the two conditions is met, candidate p can be classified as a corner (i.e., feature).

The feature matching stage inevitably produces some incorrect correspondences, which, if left intact, will unfavorably bias the frame-to-frame motion estimate. A common solution to this problem is to use a robust estimator that can tolerate some number of false matches (e.g., RANSAC [2]). In our algorithm, however, we adopt an approach described by Hirschmuller et al. [3], and exploit stereo range data at the inlier detection stage. The core intuition is that the 3D locations of features must obey a rigidity constraint, and that this constraint can be used to identify sets of features that are mutually consistent (i.e., a clique) prior to computing the frame-to-frame motion estimate.

After we determine all features on both images (F_a and F_b), we divert our attention to matching features from frame a to features from frame b

B. Feature matching: $(F_a, F_a, S) \rightarrow M$

Matching the features from different frames is done by computing the score matrix S for all pair-wise combinations of features in F_a F_b using the sum-of-absolute differences (SAD) between the feature descriptors. Low values in the scoring matrix indicate that two features are very similar; high values indicate that they are very different.

Using the score matrix S, match features F_a in frame a with features F_b in frame b. The process of matching features can be described in following steps: for each feature $f_a \in F_a$ find the feature $\bar{f}_b$ with the minimum SAD score; for each $f_b \in F_b$ find the feature $\bar{f}_a$ with the minimum SAD score; If $\bar{f}_a = f_a$ and $\bar{f}_b = f_b$ then the features f_a and f_b are declared a match.

Let (f_a, f_b) denote a match between features in frames a and b, and let M be the set of all matches.

C. Inliner detection: $M \rightarrow Q$

We do this by computing a consistency matrix W for all pairwise combinations of matches in M, using a simple rigidity constraint on the world coordinates. A pair of matches is consistent if the distance between two features in the frame a (measured in world coordinates) is identical to the distance between the corresponding feature in frame b. Any pair of matches for which this is not the case must contain at least one incorrect match or must contain features from some independent mover. At the simplest level, a pair of matches (f_a, f_b) and (f'_a, f'_b) are consistent iff they satisfy the inequality:

$$|w_a - w'_a| - |w_b - w'_b| < \sigma \tag{1}$$

where w denotes the world coordinates of a frame and σ is a fixed threshold.

Using the matrix W, find the largest set of mutually consistent matches. This is equivalent to finding the maximum clique on a graph with adjacency matrix W. We use the following sub-optimal algorithm for this:

1) Initialize the clique to contain the match with the largest number of consistent matches (i.e., choose the node with the maximum degree).
2) Find the set of matches compatible with all the matches already in the clique.
3) Add the match with the largest number of consistent matches.

Steps 2 and 3 are repeated until the set of compatible matches is empty.

We will use Q to denote the set of matches (f_a, f_b) in the inlier set discovered by this algorithm.

D. Motion estimation: $Q \rightarrow \Delta_{ab}, \epsilon$

We estimate frame-to-frame camera motion by minimizing the image reprojection error for all matches in the clique Q. We seek the homogeneous transform Δ_{ab} that minimizes the reprojection error:

$$\epsilon = \sum_{(f_a, f_b) \in Q} (j_a - P\Delta w_b)^2 + (j_b - P\Delta^{-1} w_a)^2 \tag{2}$$

where j and w are the homogeneous image and world coordinates, respectively, and P is the camera projection matrix. The solution is found using the standard Levenberg-Marquardt

least-squares algorithm, with one small modification: after finding the initial solution Δ_{ab} we discard any matches whose reprojection error exceeds a predetermined threshold and re-run the optimization. This second pass discards any outliers that survive the inlier detection step.

After this, we validate the solution against the following three criteria:

- The number of points in the clique Q. At a minimum, we require three points to generate a unique motion estimate. However, in practice, we typically demand at least ten points to ensure that the clique captures the camera egomotion (as opposed to some other motion in the scene).
- The co-linearity (or otherwise) of feature in the image. This is done by computing the eigenvalues of the feature distribution and computing the maximal ratio; values close to 1 indicate a good spread of features.
- The reprojection error ϵ, which must be below some threshold.

So finally, the algorithm outputs the frame-to-frame motion estimate Δ_{ab} and a status flag indicating if the solution is valid.

III. Implementation

For the implementation of the algorithm described above, we have used Python and OpenCV [6]. KITTI visual odometry [5] dataset is used for evaluation. In the KITTI dataset, the ground truth poses are given with respect to the zeroth frame of the camera.

A. Feature detection

As it is previously mentioned, the features are detected using FAST algorithm. In the python version that is used in this case, and that is python 3.x the FAST algorithm is already implemented in the OpenCV package.

To improve the results of feature detection we used a method called *feature bucketing*. The image is divided into non-overlapping rectangles and FAST algorithm is applied to every one of the rectangles. Every rectangle has a limit on feature points, and only features with high response values are selected from each bucket. There are two benefits of bucketing:

- Input features are well distributed throughout the image which results in higher accuracy in motion estimation.
- Due to fewer features, the computation complexity of the algorithm is reduced which is a requirement in low-latency applications. A disparity map for time T is also generated using the left and right image pair.

B. Feature matching and feature filtering

Feature matching is done by searching for features in the same image in the next frame. For this KLT(Kanade-Lucas-Tomasi) tracker is used. Features from the image in the previous frame (previous frame is frame at time T) are tracked in the next frame at, so at the time T+1 using 15x15 search windows and 3 image pyramid level search. Because the KLT tracker outputs the corresponding coordinates, it is necessary

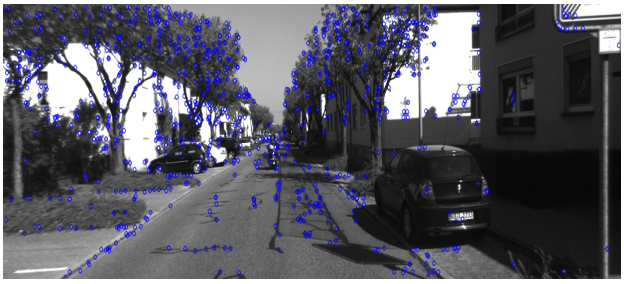

Fig. 2. Detected features in the sample image frame

to measure accuracy and error measure for each input feature. The metric that is used in this case for error measurement is SAD(Sum of absolute differences). Feature points that are tracked at corresponding frames T and T+1 and which have a higher measure of error or lower accuracy are dropped out from the further computation.

Fig. 3. Detected features at time T

Fig. 4. KLT tracked features at time T+1

C. Inlier detection

Instead of using the rejection of the outliers, this implementation uses the inliner detection algorithm which exploits the rigidity of scene points to find a subset of consistent 3D points at both time steps. Idea of this approach lies in the relative distance of two feature points. The absolute positions of two feature points will not be the same at two different time points, the relative distance between them will remain the same. If any feature that is analized does not satisfy this condition there is an error in 3D triangulation in at least one of two features, or we have triangulated is moving which cannot be used in the next step.

D. Motion estimation and reprojection error

In case of the analyzing the pictures in stereo system to get an adequate motion estimation it is necessary to get reprojection error estimation. The reprojection error is used to quantify how closely an estimate of 3D point recreates the points true projection in 2D. For this we used Levenberg-Marquardt algorithm for non-linear least squares also known as DLS(Damped Least Squares) algorithm. The only downside of using this method is that the LMA can find local minimum which in some cases is not necesseraly the global minimum, but the LMA is more robust than GNA(Gauss-Newton algorithm). This algorithm can be viewed like Gauss-Newton using a trust-region approach [7]. Frame-to-frame camera motion is estimated by minimizing the reprojection error using the algorithm mentioned earlier. Image re-projection in this particular case means that for a pair of corresponding matching points **Ja** and **Jb** at time T and T+1 there exists a corresponding world coordinates **Wa** and **Wb**. The world coordinates are reprojected back into the image using a transform to estimate 2D points for complementary time step (distance between the true and projected 2D point is minimized using LMA).

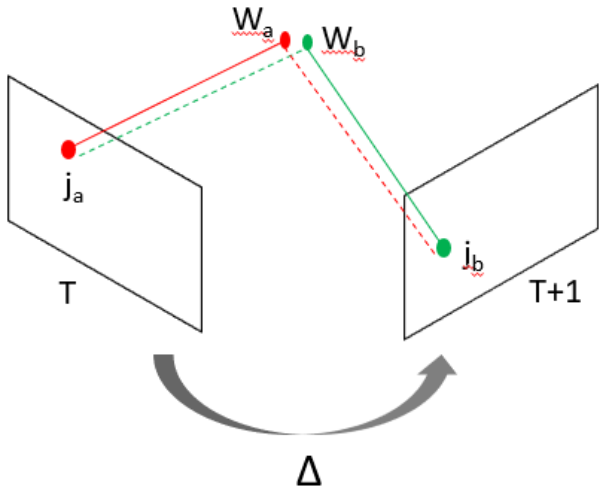

Fig. 5. Image reprojection

IV. Results

We have applied the implementation of a previously described algorithm to the stereo image sequence. Before images can be processed by the algorithm, it's necessary to compensate for lens distortion. To simplify the task of disparity map computation stereo rectification should be done in a way so that epipolar lines become parallel to horizontal. KITTI dataset has the input images that are already corrected for lens distortion and stereo rectified.

As it is possible to see in Figure [6] that, the results obtained match the ground truth trajectory initially, but small errors accumulate resulting in incorrect poses if the algorithm is run for a longer travel time. It is to be noted that although the absolute position is wrong for later frames, the relative motion (translation and rotation) is still tracked. To help correct this drift in measurement, it's possible to incorporate measurements from external sensors using some kind of estimators (e.g. Kalman filter) or use SLAM characteristics like loop closure .

There are several tunable parameters in the algorithm which can be tuned to adjust the accuracy of output, some of the parameters are: block size for disparity computation and KLT tracker, various error thresholds such as for KLT tracker, feature re-projection, clique rigidity constraint. More work is required to develop an adaptive framework that adjusts their parameters based on feedback and other sensor data.

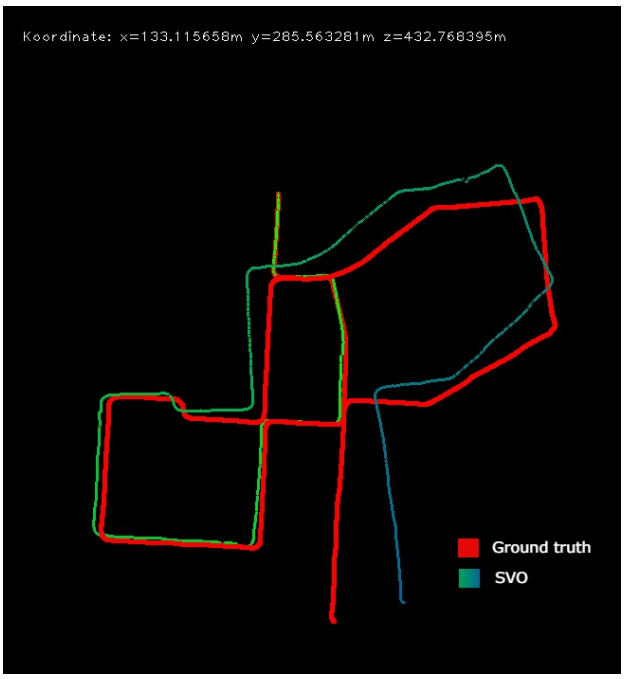

Fig. 6. First sequence

Fig. 7. Second sequence

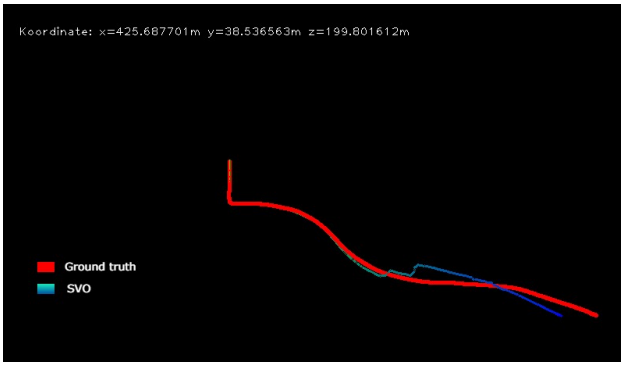

Fig. 8. Third sequence

REFERENCES

[1] Howard, Andrew. "Real-time stereo visual odometry for autonomous ground vehicles." 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2008.

[2] Fischler, Martin A., and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." Communications of the ACM 24.6 (1981): 381-395

[3] Hirschmuller, Heiko, Peter R. Innocent, and Jonathan M. Garibaldi. "Fast, unconstrained camera motion estimation from stereo without tracking and robust statistics." 7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.. Vol. 2. IEEE, 2002.

[4] Rosten, Edward, and Tom Drummond. "Machine learning for high-speed corner detection." European conference on computer vision. Springer, Berlin, Heidelberg, 2006.

[5] KITTI - visual odometry dataset

[6] OpenCV - library.

[7] Wikipedia contributors. "Levenberg–Marquardt algorithm." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 28 May. 2021. Web. 7 Sep. 2021.