Indirect Visual Odometry with Optical Flow

Erkam Uyanik

Vision-based Navigation
Project Report

Department of Informatics
Technical University of Munich

15.03.2020

Contents

1 Introduction **1**

2 Related Work **1**

2.1 Optical Flow 1

2.1.1 Lucas Kanade Method 2

2.1.2 Image Pyramids 3

2.2 FAST Keypoint Detector 4

3 Method **4**

3.1 Overview 4

3.2 Keypoint Selection 5

3.2.1 Grid Based Approach 5

3.2.2 Keypoint Detection 5

3.2.3 Distance between Selected Keypoints 6

3.2.4 Distance to Image Borders 6

3.3 Keypoint Tracking 6

3.4 Localizing Camera 8

3.5 Adding New Landmarks 8

3.6 Optimization 9

3.7 Keyframe Selection 9

4 Comparison to Descriptor Based Approach **9**

5 Results **10**

5.1 Evaluation 10

5.1.1 Absolute Trajectory Error 10

5.1.2 Runtime 11

5.2 Qualitative Results 11

5.3 Quantitative Results 11

6 Implementation **12**

7 Conclusion **12**

References **13**

1 Introduction

Odometry is a technique to estimate position and orientation. With visual odometry this is done based on sequences of images. In this project, our goal is to track the location of an agent through a trajectory given a sequence of images.

There are two approaches to visual odometry: i) directly computing odometry based on pixel intensities, ii) selecting a set of feature points and using only those feature points. We will use the latter approach which is named as *indirect* approach.

To match feature points there are two common approaches. One is computing distinctive descriptors for keypoints and matching keypoints from different images using their descriptors. This approach requires a good way to compute feature descriptors such as ORB descriptor [7] and it increases the required computational work. Another approach is using an *optical flow* method to estimate optical flow between images to track keypoints, and thus getting correspondences between keypoints in different images. Optical flow can be defined as the motion between two images based on the assumption that brightness values of voxels are preserved.

Prior to this project, we have also implemented an indirect visual odometry using feature descriptors. In this report, we will emphasize the differences between optical flow based approach and descriptor based approach.

2 Related Work

2.1 Optical Flow

Aim of optical flow is to estimate motion from images, most simply between two consecutive images. This is, in this form, is an *ill posed* problem as there can be infinitely many mappings from one image to the other. To solve ill posedness, further information or assumptions are needed. The main assumption for optical flow is that brightness values of voxels are preserved between images.

In a 2D case, we can formulate this *brightness constancy* assumption as follows:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

In addition to this, with the assumption that the movement between two images are small, we can use Taylor expansion to get:

$$\begin{aligned}
I(x + \Delta x, y + \Delta y, t + \Delta t) &\approx I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \\
\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t &= 0 \\
\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} &= 0 \\
\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} &= 0, \text{ where } V \text{ is velocity vector} \\
I_x V_x + I_y V_y + I_t &= 0 \\
\nabla I^T V + I_t &= 0
\end{aligned}$$

Here, with brightness constancy assumption, we derived an equation with two unknowns which is known as *optical flow constraint* and we cannot solve it without any other assumption. One cannot estimate motion in the direction of constant brightness, this equation is invariant to changes in motion field V normal to the image gradient. This is known as the *aperture problem*.

This second assumption we need is what differentiates between various optical flow methods.

2.1.1 Lucas Kanade Method

In this method [5], it is assumed that the optical flow is constant in local neighborhoods. With this assumption, optical flow constraint can be solved for all points in the neighborhood using the *least squares* approach.

For a local neighborhood with points $p_1, p_2, \ldots, p_n$, we can write:

$$A = Vb$$
$$v = (A^T A)^{-1} A^T b$$

where

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \quad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_n) \end{bmatrix}$$

As an improvement, a weight matrix can be used in above equation where a larger weight is given to points closer to the center of neighborhood.

Lucas Kanade method is suitable for use with indirect approaches as it can be used to track all feature points individually.

To track feature points for a sequence of images, one needs to apply Lucas Kanade method between consecutive images sequentially after detecting feature points in the first image. This way, a feature point can be tracked as long as possible, until it goes out of the frame or becomes occluded.

2.1.2 Image Pyramids

Optical flow constraint involves the assumption that the movement between frames are small. When this is not the case, optical flow methods may not give very reliable results. To prevent this, *image pyramids* are usually used.

The idea is to apply optical flow method on a coarse version of the image first and repeat it with more detailed versions iteratively. This way, given points can be tracked from coarse to detailed version of the image, providing that assumption holds at each iteration.

2.2 FAST Keypoint Detector

FAST (Features from Accelerated Segment Test) [6] is a corner detection method which is known for its computational efficiency.

For a candidate point, a circle of 16 pixels is taken into account.

The main condition is that for a threshold τ, if there are N contiguous pixels which are all brighter than $I_p + \tau$ or all darker than $I_p - \tau$, then the candidate point is a keypoint. Here N is selected as 12.

Additionally, there is a *high speed test* which is applied first to quickly filter out irrelevant points. It checks 4 equidistant pixels from the selected circle. If the candidate point is a corner, at least 3 of these 4 pixels should be brighter than $I_p + \tau$ or darker than $I_p - \tau$. Otherwise, the candidate is filtered out.

Although this is very fast, there are some drawbacks of this method. To overcome those drawbacks, a decision tree can be trained using a training set preferably from the target domain.

To prevent selecting keypoints very close to one another, non maximal suppression is applied. For this, a score is computed for each pixel. It is the sum of absolute intensity differences between the candidate point and each pixel from the surrounding circle. When compared, candidate with the lower score is discarded.

3 Method

3.1 Overview

The main flow of our visual odometry framework is as follows:

For each frame:

1. track points from previous frame
2. detect new feature points if tracked points are not enough

3. localize camera
4. if current frame is a keyframe
   (a) track points between stereo images
   (b) add new landmarks
   (c) remove old keyframes
   (d) optimize

What differentiate this optical flow based approach from feature descriptor based approaches are how we select feature points and tracking of feature points. Since we track the points, we already know matching information, so we skip matching step normally done in descriptor based approach.

For completeness, we will also explain other steps but emphasis will be on above mentioned steps.

3.2 Keypoint Selection

3.2.1 Grid Based Approach

During feature point tracking, due to the movement of the camera, some feature points may leave the frame or get occluded. If we heavily depend on a small segment of the image, e.g. an object with distinctive patterns, when we select feature points, at some point we can lose track of many feature points which would affect our method badly. To prevent this, we apply a grid based approach while selecting keypoints. The aim is to select keypoints from all parts of the image, a more homogeneous selection compared to directly choosing best keypoints.

We first divide an image into a grid of cells and evaluate each cell separately. We select as many keypoints from a cell as we determine if there are enough keypoint candidates.

3.2.2 Keypoint Detection

We use FAST detector as it is a fast method which helps us to decrease our computational load. Also, we are not too vulnerable due to using FAST

instead of a more robust method since we have other steps where poorly selected or poorly tracked keypoints are filtered such as epipolar constraint check or steps where RANSAC is used.

OpenCV implementation cv::FAST is used in the project. The threshold parameter is handled as a hyperparameter to tune.

3.2.3 Distance between Selected Keypoints

Although FAST detector inherently has a non maximal suppression step, we try to keep some distance between selected keypoints because keypoints that are very close to each other may become an issue if tracked points accumulates some spatial noise after a number of iterations. This could cause a mismatch between keypoints while tracking. Thus, when selecting new keypoints, we skip candidates that are too close to previously tracked or previously selected keypoints. This threshold is controlled as a hyperparameter.

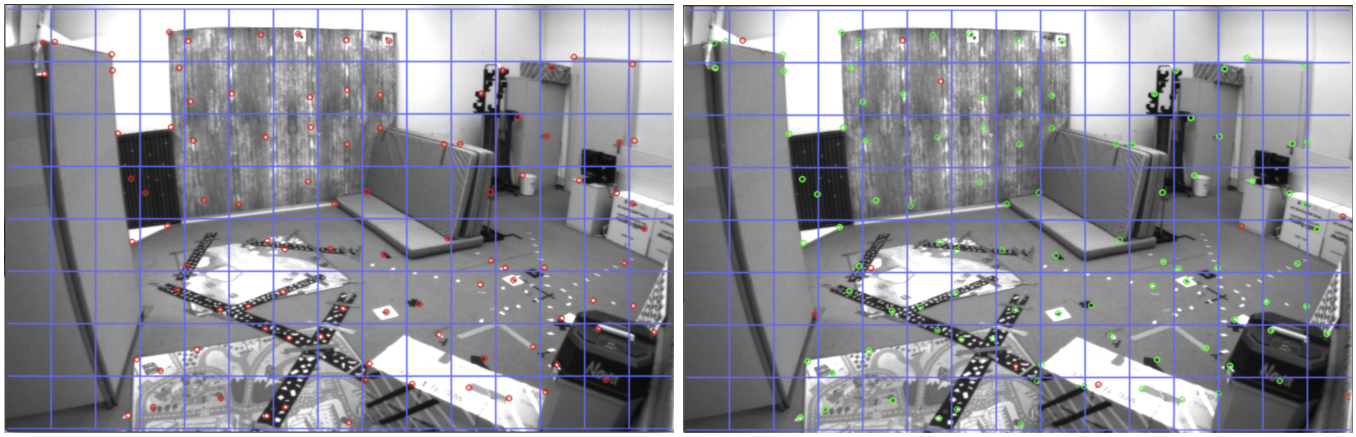
3.2.4 Distance to Image Borders

We mentioned earlier that when tracking keypoints we can lose track when keypoints leave the image frame. Since it is more probable for keypoints closer to image borders (a small movement of camera can cause those points to leave the image frame), we check the distance to image borders when selecting new keypoints, controlled by a hyperparameter.

Overall, we have a keypoint selection step with various hyperparameters. We aim to select more dominant keypoints while also keeping total number of keypoints large enough since next steps depend on enough number of tracked keypoints.

3.3 Keypoint Tracking

For tracking keypoints, we use Lucas Kanade method. It is suitable for us as we want to track a number of keypoints.



(a) Without any tracked points (b) With tracked points

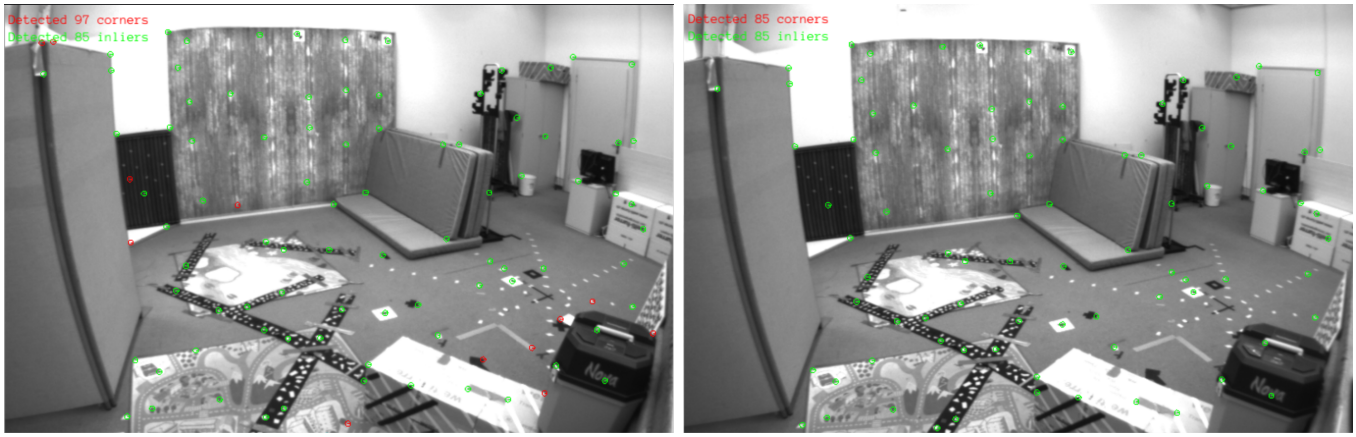
Figure 1: Keypoint selection procedure with visible grids. Number of keypoints per cell is 1. Newly added keypoints are denoted with red and tracked points denoted with green.

We apply Lucas Kanade method with image pyramids to track points. For this, cv::buildOpticalFlowPyramid is used to generate image pyramids and cv::calcOpticalFlowPyrLK to track points using Lucas Kanade method.

To filter out poorly tracked points, we track back points and compare them with the original points. If tracked back points are not very close to the original point, we discard them. This allows us to keep the quality of tracking high so that following steps can rely on the tracked points.

For keypoint tracking step, we have parameters for window size for optical flow algorithm (the size of the neighborhood), for number of levels for image pyramids, for termination criteria of iterative search for optical flow, and for the required maximum distance between original points and tracked back points.

The important part of the keypoint tracking is that successfully tracked keypoints should be precise. For the given example 2, we are concerned with 85 keypoints that are successfully tracked, and all of them appears to be correctly tracked.



(a) Left image of a stereo frame (b) Right image of a stereo frame

Figure 2: An example of keyframe tracking between stereo images. Out of 97 keypoints, 85 of them are tracked from left image to right image.

3.4 Localizing Camera

Localizing camera step has the same logic as before from descriptor based approach. First, we find keypoints in current frame that belong to a landmark and unproject the keypoints. We use unprojected 3D positions and 3D positions of landmarks to solve central absolute pose problem with RANSAC [2]. OpenGV [3] library is used for this. We use Kneip's P3P algorithm [4].

With this step, we find camera pose for each frame using the fact that we already know correspondence between existing landmarks and tracked points from previous frames.

3.5 Adding New Landmarks

For a keyframe, before adding new frames, we initially track keypoints from left image to right image 2. We use *epipolar constraint* to filter poor matches.

For stereo matches, we add them as observations if they already belong to an existing landmark. Otherwise we triangulate them to create new landmarks. We use OpenGV to compute triangulations.

3.6 Optimization

For each added keyframe, we optimize the results using bundle adjustment. We optimize camera intrinsics and extrinsics, and landmarks. We optimize *reprojection error* for the landmarks based on observations in keyframes. For bundle adjustment, Ceres Solver [1] is used.

Optimization process is done in another thread while visual odometry keeps running.

3.7 Keyframe Selection

We select a new keyframe if number of keypoints that belong to a landmark is less than a threshold and if there have been enough frames since the last keyframe (we don't want to select keyframes temporally too close to each other). We also wait for any ongoing optimization to finish before selecting a keyframe.

4 Comparison to Descriptor Based Approach

The main difference from descriptor based approach is the implicit matching mechanism of tracking. Instead of detecting keypoints for all images, computing descriptors for all of these keypoints and comparing them, we avoid both detecting keypoints for all images and computing descriptors entirely.

Since we either track existing keypoints or detect new keypoints, using unique identifiers for keypoints is enough to keep track of correspondences of keypoints between images and correspondences between keypoints and landmarks. In our implementation, for instance, we use same identifier number for keypoints and corresponding landmarks since the relation is obvious.

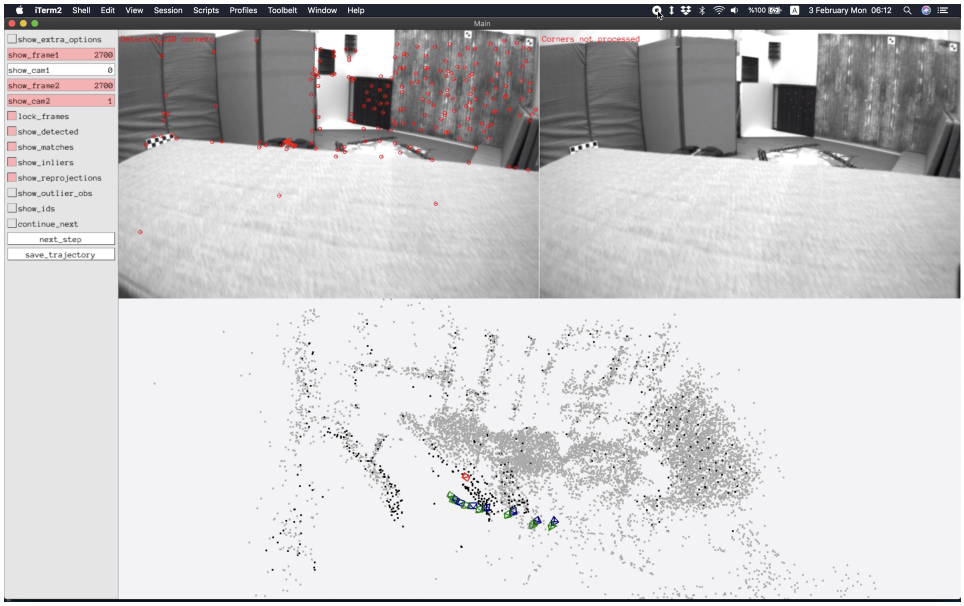


Figure 3: End result of application GUI after 2700 frames

5 Results

In GUI, we display camera positions for keyframes and all landmark positions in a 3D setup. Although we do not create a map of the environment, this helps to see whether our odometry works as expected. In the figure 3, you can see the end result of the program after 2700 frames.

5.1 Evaluation

For evaluation we have 2 criteria.

5.1.1 Absolute Trajectory Error

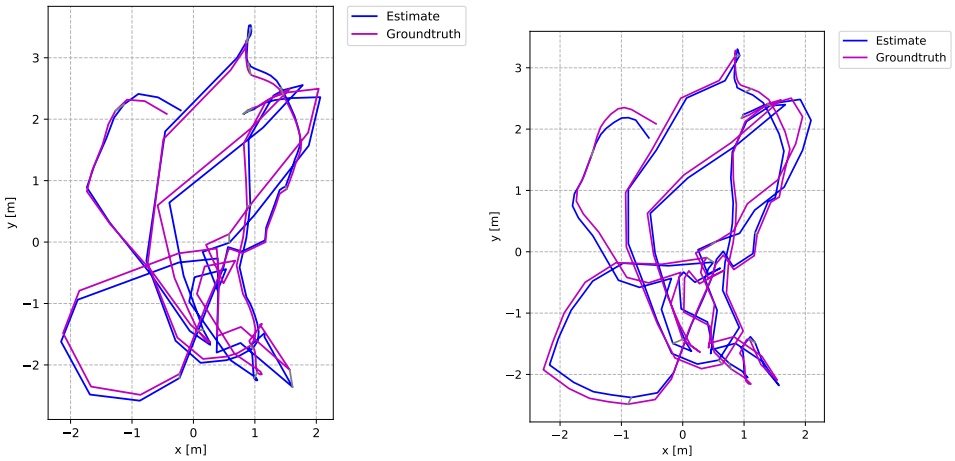
Absolute trajectory error (ATE) is the common metric to compare predicted trajectory with the groundtruth. After aligning trajectories, differences between each pair of camera poses are computed. Root mean square error is used to evaluate the prediction.

For ATE, we use tools from [8] for evaluation and [9] for visualization.

5.1.2 Runtime

Although our priority is having a good performance in terms of accuracy, runtime is also an important metric since we want odometry to be able to run in real time.

5.2 Qualitative Results



(a) Optical flow (0.144867) (b) Descriptor (0.127981)

Figure 4: Two qualitative results from optical flow approach and descriptor approach. Figures show aligned trajectories. Numbers are RMS ATE (m) results

From qualitative results 4, we see that both approaches works and that we can predict the route using optical flow without use of descriptors.

5.3 Quantitative Results

When we compare quantitative results 1, we see that both methods perform similarly according to ATE, although VO descriptor performs slightly

better. On the other hand, VO optical flow runs %10 faster than VO descriptor.

Method	RMS ATE (m)	RMS ATE stddev	Runtime (s)	Runtime stddev
VO optical flow	0.143596	0.043500	47.44	0.41
VO descriptor	0.122272	0.031707	52.76	0.97

Table 1: Results of visual odometry with optical flow and with descriptors. Each method is run 5 times.

6 Implementation

You can find the source code at gitlab repo under project branch.

To run the code from root directory, use:

time (./build/odometry −−show−gui=0 && python2 rgbd_benchmark_tools /evaluate_ate.py stamped_groundtruth.txt stamped_traj_estimate.txt)

This will run the code without GUI, print the ATE for predicted trajectory, and print runtime.

Similarly, you can run descriptor based visual odometry using:

time (./build/odometry −−show−gui=0 −−use−optical−flow=0 && python2 rgbd_benchmark_tools/evaluate_ate.py stamped_groundtruth.txt stamped_traj_estimate .txt)

7 Conclusion

We have implemented an indirect visual odometry based on optical flow for tracking keypoints. It works without computing any descriptors and since we track keypoints it computes considerably fewer keypoints. Overall it works and predicts a reasonable trajectory. It performs similarly to descriptor based indirect visual odometry implementation, slightly worse accuracy but with faster runtime.

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[2] Martin A Fischler 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):381–395, 1981.

[3] Laurent Kneip and Paul Furgale. Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2014.

[4] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR 2011*, pages 2969–2976. IEEE, 2011.

[5] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[6] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[7] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[8] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[9] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.