Mini-project: A visual odometry pipeline!

Contents

1 Preliminaries

1.1 Goal of the project

The goal of this mini-project is to implement a simple, monocular, visual odometry (VO) pipeline with the most essential features: initialization of 3D landmarks, keypoint tracking between two frames, pose estimation using established 2D $\leftrightarrow$ 3D correspondences, and triangulation of new landmarks. During the exercises, you have seen some building blocks for this, now it is time to put everything together!

1.2 Datasets

You are provided with three datasets to test your pipeline:

- The parking dataset (monocular)
- The KITTI[1] dataset (stereo)

[1] http://www.cvlibs.net/datasets/kitti/eval_odometry.php

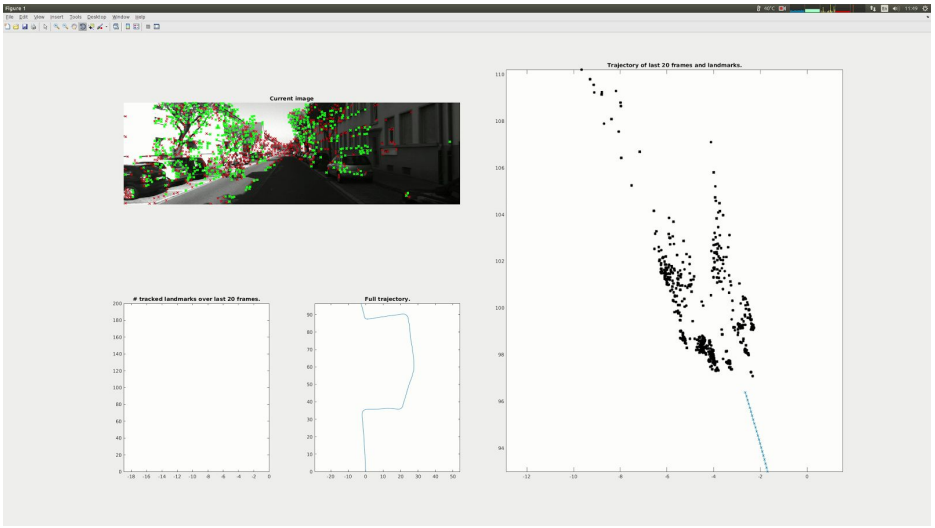

Figure 1: Screenshot from our reference implementation. A YouTube playlist with example runs can be found at http://tinyurl.com/ref-vo-vids .

- The Malaga[2] dataset (stereo)

These can be downloaded from the teaching website.

1.3 Grading

Depending on the outcome of the mini-project, your grade will be increased by 0, 0.25 or 0.5: 0 if the VO pipeline does not really work, 0.25 if it works and 0.5 if it works well and you add some interesting additional feature.

You will be graded based on three things:

1.3.1 VO pose estimation quality

We will not do a quantitative evaluation of the quality of your VO pipeline, but a qualitative one. It is important to us that your approach and problem-solving methodology is right, rather than that you happen to stumble upon the right parameter tune for the provided datasets. Besides submitting Matlab code that is easy to run, you have to record a screencast of your running pipeline. Because it is very hard to avoid scale drift in the KITTI and Malaga datasets, it is not important that you estimate the trajectory globally consistently, but it needs to be locally consistent for a reasonable span of frames. See "Full trajectory" versus "Trajectory of last 20 frames" in the videos (http://tinyurl.com/ref-vo-vids). Achieving a good global trajectory estimate can be considered an interesting additional feature and can result in a grade increase of 0.5. The VO pipeline must be fully monocular. As a consequence, you will of course not have to estimate the trajectory at the correct absolute scale.

1.3.2 Report quality

The project report should summarize the overall work that you did. Write here how your impemen-tation deviates or expands upon the recommendations written below, a succinct recount of how you set and achieved milestones, and what interesting problems you encountered and how you solved them. Also, quickly summarize your screencasts, commenting on the VO quality, highlighting in-teresting things and explaining why maybe things are not the way you expected them to be. The project report is also the place to describe the interesting additional feature and how it impacts the quality of your VO pipeline. Note that having implemented an additional feature which degrades

[2] http://www.mrpt.org/MalagaUrbanDataset

the quality of your VO pipeline will be accepted and valued as long as i) the implemented feature is properly motivated and described, and ii) an analysis showing the effect of your additional feature is provided in the report. If the additional feature degrades the quality, provide both a screencast with additional feature enabled and additional feature disabled.

1.3.3 Interesting additional feature

The following are examples of interesting additional features that could be accepted for a 0.5 grade increase. You can propose any other feature as long as you can justify its merit, though in that case we recommend that you verify with us whether a feature is acceptable first.

- Record your own dataset, and make your VO work with it. You can for example use your smartphone's camera calibrated using the calibration toolbox [3].
- Propose and implement improvements to combat scale drift (without using stereo frames).
- Detailed, quantitative analyses that compare several approaches that can be applied to the same component of the VO.

1.4 Allowed matlab functions and external libraries

You can use any Matlab function or library that achieves what has been done during any of the exercise sessions. Use of any code that you did not write yourselves must be declared in the report. We will not provide help for any function or library that is not part of the exercise solutions or this project statement.

1.5 Submission

Your final submission is to be sent per e-mail to all TAs by the date specified on the website. Please make sure that you only send one submission per team. Links to any data that are not in the e-mail attachment (incl. YouTube videos) need to be provided in the e-mail body. The submission needs to comprise:

1. At least one screencast per dataset, showing the output of your VO on it. Use a similar layout to Figure 1. Please use top-down 2D plots with an aspect ratio of 1 (`set axis equal`). Feel free to provide more screencasts if you want to show something with them. To simplify data transmission, we recommend uploading the screencasts to YouTube as listed or unlisted video, but you can also send us the screencasts in any other way. YouTube videos that are not unlisted will be added to a playlist that will be published on the website of our research group! For Ubuntu, we recommend the screen capture program "kazam". Please specify the last names of the members of your group in the video file name or YouTube video title.
2. When submitting code, always use relative paths instead of absolute paths. Write a Readme file specifying how to run it and the specifications of the machine on which you captured the screencasts (maximum number of used threads, CPU frequency, RAM).
   (a) Matlab code: Also mention the Matlab version you are using.
   (b) Python code: Use Anaconda environments with Python 3.x. Submit your exported conda environment (conda env export) together with your code.
   (c) C/C++ code: Submit your code together with a functioning Makefile or CmakeLists file. Ensure that we can compile your software in Linux (Ubuntu 18.04). There must be a detailed description of how to compile and run your code.
3. The report, see Section 1.3.2. An ideal report length is 5 pages excluding images or references, if any. Be concise! The absolute maximum allowed is 10 pages excluding images or references.

Please do **not** provide raw images of custom datasets, if you submit any. Instead, describe them in the report and only provide videos / screencasts.

[3] http://www.vision.caltech.edu/bouguetj/calib_doc/

2 Overview of the proposed pipeline

We first give a global overview of the proposed pipeline and the different components involved. We will then go into more details separately for each component.

2.1 Notation

We denote the set of all N frames in a dataset by: $\{I^i \equiv I(t^i)\}, i \in [1, N]$. We denote the pose of the camera at time t^i by T^i_{WC}.

2.2 Overview

The proposed pipeline is composed of two main components:

- An initialization module that extracts an initial set of 2D $\leftrightarrow$ 3D correspondences from the first frames of the sequence and **bootstraps** the initial camera poses and landmarks.
- A continuous VO module that processes each frame I^i, estimates the current pose of the camera T^i_{WC} using the existing set of landmarks, and regularly triangulates new landmarks.

These two modules can be developed independently from one another. To test the continuous VO module before the initialization module is completed, you can use the initial 2D $\leftrightarrow$ 3D correspondences provided in the exercise 7 (KITTI only).

3 Initialization

As you learned in lecture 6, one can use two-view geometry to estimate the relative pose between two (sufficiently distant) frames, and triangulate a point cloud of landmarks.

You can proceed as follows:

1. Manually select two frames I^{i_0} and I^{i_1} at the beginning of the dataset.
2. Establish keypoint correspondences between these two frames using either patch matching (exercise 3) or KLT (exercise 8, consider using the intermediate frames as well).
3. Estimate the relative pose between the frames and triangulate a point cloud of 3D landmarks (exercise 6). Since the keypoint correspondences from the previous step will inevitably contain some outliers, you will need to use RANSAC (some hints are given below) to filter them out.
4. Initialize the continuous VO pipeline with the inlier keypoints and their associated landmarks.

Implementation hints:

- Make sure that the baseline between the two initialization frames is large (i.e. it is better not to use two adjacent frames). Don't pick too distant frames either, otherwise it becomes more difficult to establish keypoint correspondence. For the KITTI dataset, we obtained good results using frame 1 and frame 3.
- Although we have not strictly implemented the robust eight-point algorithm with RANSAC during the exercises, you are allowed to use the Matlab function `estimateFundamentalMatrix` which implements RANSAC.

4 Continuous operation

The continuous VO pipeline is the core component of the proposed VO implementation. Its responsibilities are three-fold:

1. Associate keypoints in the current frame to previously triangulated landmarks.
2. Based on this, estimate the current camera pose.

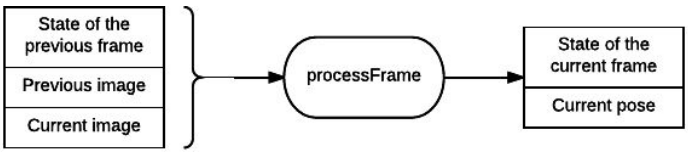

Figure 2: Recommended data flow / function design for continuous operation. This allows continuous operation without the need to keep a global data structure with the full history of previous observations

3. Regularly triangulate new landmarks using keypoints not associated to previously triangulated landmarks.

We recommend to implement this in a Markov way using the data flow / function design shown in Figure 2. Formally, we define as S^i the state of the current frame, whose contents are specified further below. Then, we can define a function for processing incoming frames, updating S^i and returning the pose T^i_{WC} as follows:

$$[S^i, T^i_{WC}] = \text{processFrame}(I^i, I^{i-1}, S^{i-1}) \tag{1}$$

The key idea in this design is that the function inputs solely depend on the output of the previous function call (and the new frame to process), i.e. it has the Markov property. That means we don't need to build a data structure to maintain the history of the past frames, all that is needed is contained in the state S^i.

4.1 Associating keypoints to existing landmarks

This can be achieved like in exercise 7. Remember that there, the function `ransacLocalization` took as input a query image, here I^i, a database image, here I^{i-1}, keypoints in the database image, 3D landmarks and the projection matrix. To fit this with the Markovian design, we add to S^i the keypoints in the i-th frame and the 3D landmarks associated to them. We denote the keypoints with $P^i = \{p^i_k\}, k \in [1, K]$, K being the keypoint count. The 3D landmarks are denoted with $X^i = \{x(p) \forall p \in P^i\}$, meaning that $x(p^i_k)$ is the 3D landmark associated to p^i_k.

We could now propagate S^i according to exercise 7: Obtain P^i from Harris corners and set each landmark $x(p^i_a) = x(p^{i-1}_b)$ if the keypoint p^i_a is matched to p^{i-1}_b, and if this matching is a RANSAC inlier. However, we strongly recommend to use KLT tracking (see exercise 8): instead of independently extracting P^i and then matching them, keypoints can be tracked from I^{i-1} to I^i. This tends to have far fewer outlier associations than extracting keypoints from scratch and then matching them. Furthermore, KLT can track the position of keypoints to sub-pixel accuracy, which leads to more accurate pose estimations. However, KLT is still not perfect, and you might have dynamic objects in the environment that would corrupt your pose estimate, so you still need the RANSAC step.

Note that with that, not all $p^i_k \in P^i$ might end up with an associated landmark. Those that do not can be discarded: they will no longer be useful for I^{i+1}, I^{i+2}... As a consequence, P^i will shrink over time, unless we actively expand it, which is discussed in Section 4.3. For KLT we recommend to use the Matlab class `vision.PointTracker` .

4.2 Estimating the current pose

Whether you obtain keypoint-to-landmark associations from patch matching or from KLT tracking, you should jointly estimate pose and inliers using RANSAC, like in exercise 7. This will give you T^i_{WC} as an automatic by-product. In exercise 7, we had suggested to refine the P3P guess with a DLT solution for all inliers, once the maximum set of inliers has been determined. However, while developing the reference VO, we have found that the DLT solution is very often worse than the best P3P guess.

To sum up, so far $S^i = (P^i, X^i)$. We recommend to use for S^i a Matlab struct and a $2 \times K$ and $3 \times K$ matrix for P^i and X^i, respectively.

4.3 Triangulating new landmarks

So far, the pipeline can use the landmarks X^1 from the initialization phase to localize subsequent frames. However, once the camera has moved far enough, these landmarks might not be visible any more. It is thus necessary to continuously create new landmarks. We propose an approach which maintains the Markov property of our design and provides new landmarks asynchronously, as soon as they can be triangulated reliably.

The idea is to initialize, for each new frame I, a set of **candidate keypoints**, and try to track them through the next frames. Thus, at every point in time, we maintain a set of candidate keypoints $C^i = \{c_m^i\}, m \in [1, M]$ which have been tracked from previous frames. Let us define $\Psi_i^{i+1}(c^i)$ as the expression for the keypoint that is obtained from tracking c^i from frame I^i to I^{i+1}. Then, we assume that this operation can be inverted such that

$$\Psi_{i+1}^{i}(\Psi_i^{i+1}(c^i)) = c^i \tag{2}$$

and define the concatenation of tracking a keypoint across several frames as

$$\Psi_i^{i+n}(c^i) = \Psi_{i+n-1}^{i+n}(\Psi_{i+n-2}^{i+n-1}(...\Psi_i^{i+1}(c^i))),$$

which can also be inverted in the sense of (2). For every candidate keypoint $c^i \in C^i$, we call the sequence $\Gamma(c_m^i) = \{\Psi_i^{i-L_m}(c_m^i), \Psi_i^{i-L_m+1}(c_m^i), ..., c_m^i\}$ of tracked keypoints from frame I^{i-L_m} to frame I^i a **keypoint track** of length L_m. As soon as a given keypoint track Γ_m meets some conditions (more details on that below), we can reliably triangulate a new landmark from the keypoint observations $\{\Psi_i^{i-L_m}(c_m^i), \Psi_i^{i-L_m+1}(c_m^i), ..., c_m^i\}$, and the corresponding camera poses $\{T_{WC}^{i-L_m}, T_{WC}^{i-L_m+1}, ..., T_{WC}^i\}$. To simplify, we assume that a good enough triangulation for a given track can be achieved using only the most recent observation c_m^i, the first ever observation of the keypoint $f(c_m^i) := \Psi_i^{i-L_m}(c_m^i)$, and the corresponding poses T_{WC}^i and $\tau(c_m^i) := T_{WC}^{i-L_m}$. Hence, all we need to remember of the track for a given keypoint c_m^i is $f(c_m^i)$ and $\tau(c_m^i)$.

We can add the following data to the state S^i to reflect this:

- The set of candidate keypoints C^i.
- A set containing the **first** observations of the track of each keypoint $F^i := \{f(c) \forall c \in C^i\}$.
- The camera poses at the first observation of the keypoint $\mathcal{T}^i := \{\tau(c) \forall c \in C^i\}$.

With this, the state ends up being:

$$S^i = (P^i, X^i, C^i, F^i, \mathcal{T}^i)$$

$C^i, F^i, \mathcal{T}^i$ can respectively be represented as matrices of shape $2 \times M$, $2 \times M$, $12 \times M$ (or $16 \times M$ for the latter), where the transformation matrices $\tau \in \mathcal{T}^i$ are reshaped to vectors.

To propagate the new components of this state, we can track $C^i = \{\Psi_{i-1}^i(c) \forall c \in C^{i-1}\}$ in the same way that $\{p_k^i\}$ are tracked in Section 4.1(we had not used this formalism there yet). Trivially, F^i is defined by $f(c) = f(\Psi_i^{i-1}(c)) \forall c \in C^i$, and $\mathcal{T}^i$ analogously. If a candidate keypoint $c \in C^{i-1}$ fails to be tracked, it and $(f(c), \tau(c))$ get discarded. As a consequence, $C^i, F^i, \mathcal{T}^i$ also shrink over time. To mitigate this, you will need to continuously add newly detected keypoints c' to C^i and set the corresponding $(f(c'), \tau(c')) = (c', T_{WC}^i)$. Consider making sure that these newly added keypoints are not redundant with existing keypoints C^i and P^i.

Finally, to obtain new 3D landmarks, you can attempt to triangulate a landmark x from each $(c, f(c), \tau(c)) \forall c \in C^i$ similarly as the triangulation in exercise 5. However, you should require a minimum baseline to make sure that the triangulations you get have good quality. We recommend finding a threshold for the angle $\alpha(c)$ between the bearing vectors corresponding to the keypoint observations and camera poses, see Figure 3. If the angle exceeds that threshold, you can remove $(c, f(c), \tau(c))$ from $C^i, F^i, \mathcal{T}^i$ and append $(c, x(c))$ to P^i, X^i.

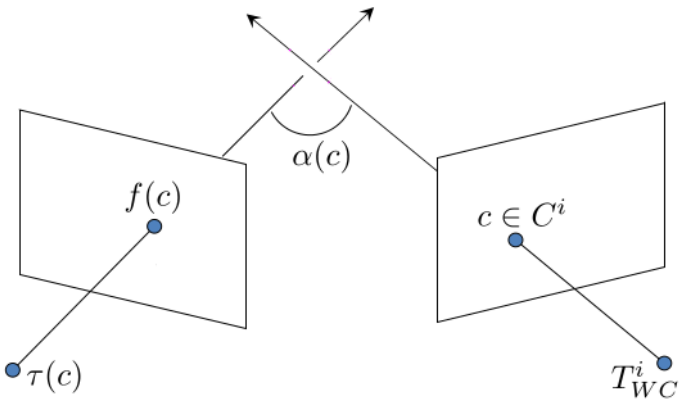

Figure 3: Illustration of the angle $\alpha(c)$ which we recommend to define the threshold on for deciding whether or not to add a triangulated landmark and its keypoint to X^i, P^i.

5 General hints

- In general, proceed step by step and verify your intermediate results visually. For example, make sure that matching, localization and landmark propagation work properly before triangulating new landmarks.
- You can save a lot of tedious coding and avoid bugs if you learn to master Matlab indexing[4]. We recommend that you carefully read and try out **all** of that tutorial except the advanced examples using linear indexing. Don't forget the logical indexing section.

[4] https://ch.mathworks.com/company/newsletters/articles/matrix-indexing-in-matlab.html