Visual Odometry Pipeline

Pascal Buholzer, Fabio Dubois, Milan Schilling, Miro Voellmy

January 7, 2017

Symbols

Contents

Introduction

During this mini project a monocular visual odometry pipeline was developed. This pipeline takes the consecutive gray-scale images of a single digital camera as input. The output of the pipeline is the position of the camera in relation to its initial position for each frame. The pipeline is programmed in such a way that the Markov assumption is valid. This means that the current computation step is only dependent on the previous step. This reduces the required computation effort.
keywords: (VO, sequential,monocular, markov assumption)

Implementation

This pipeline was developed in MATLAB. Since the group consisted of four students, a Git repository was used to be able to work on different files simultaneously, and to enable version control.

Coordinate Frames

In this mini project the coordinate frames were defined as shown in Fig. 1. The camera coordinates are in a way oriented, that the x-y plane lies parallel to the image plane, while the z-axis is pointing towards the scenery. The world frame however is oriented in such a way that the x-y plane is parallel to the ground and the z-axis is pointing upwards. The origin of the world frame is at the same location as the origin of the first bootstrap image.
Transformation between frames are described by homogenous transformation matrices. T_{AB} maps points from frame B to frame A.

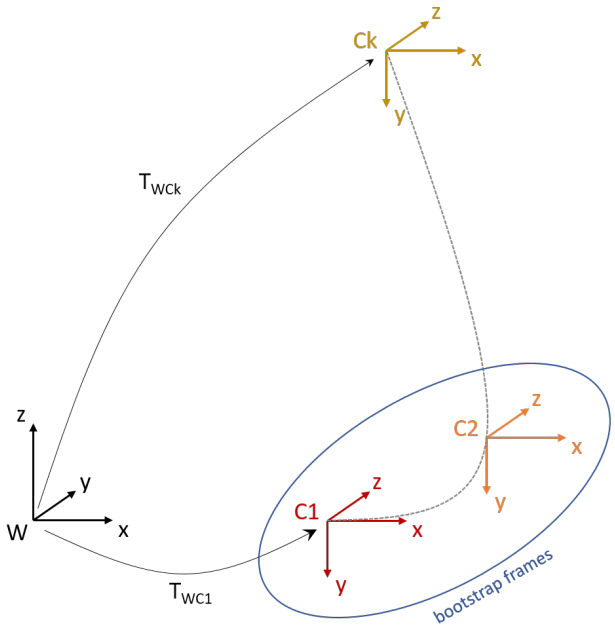

Figure 1: Coordinate Frames

Pipeline overview

As shown in Fig. 2 the pipeline consists mainly of three parts, a bootstrap, the initialisation and the continuous operation. In Section 2.1 and Section 2.2 the initialisation and the continuous operation are described in detail.

Conventions

- Index of previous frame: i
- Index of current frame: j
- Index of frame for newly added keypoint: $first$
- Pose difference between previous to current frame: $T_{C_iC_j}$
- $[u/v]$: Pixel coordinates
- Query keypoints: Keypoints newly generated in frame j
- Candidate keypoint: A keypoint without associated landmark
- Harris Matcher: Descriptor matching keypoint tracker (based on Harris features) developed during the lecture.

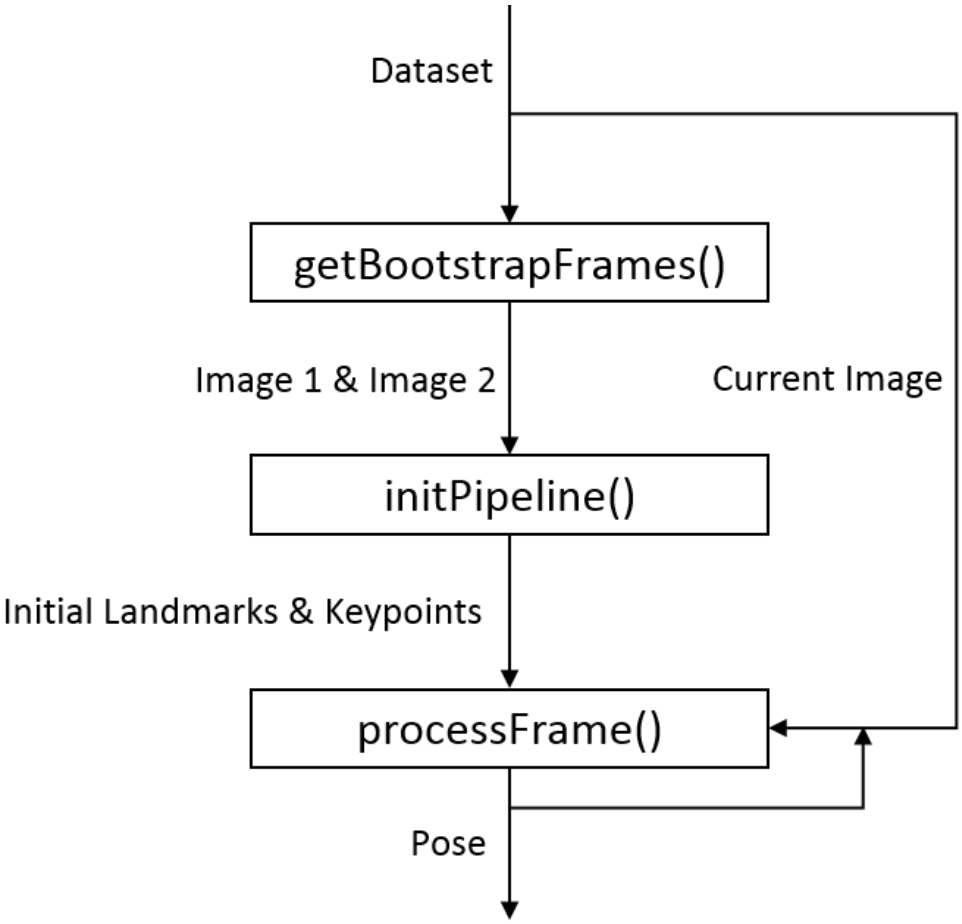

Figure 2: Overview flow chart

Options and parameters

The pipeline was designed in a modular way. Key algorithms are abstracted into self-contained functions, as described in the pipeline overview. Next to this 'functional programming' approach all the tuning variables (e.g. number of keypoints, bearing angle thresholds, etc.) were centrally aggregated in a parameter struct. For further insight please consult the function `loadParameters.m`.

In order to run the visual odometry two launch procedures where implemented: For development and debug mode the main.m with default parameters can be executed. Numerous individual plots are displayed with insightful information about matching, inlier rejection and triangulation.

Out of performance reasons a more compact and user-friendly display of the pipeline output was created with a GUI designed with the Matlab GUIDE application, see Fig. 3. Only the most crucial entities, like number of triangulated keypoints, are visualized for intuitive understanding and easy handling.
Please follow the steps below to run the visual odometry through the GUI environment:

1. adapt dataset paths in `loadParameters.m`
2. type into MATLAB command window: `gui_simple`
3. in *Parameters* panel select dataset to run on and toggle respective radio buttons
4. hit *Run* to trigger the visual odometry

For more advanced parameter tuning resort to changing the default parameters in `loadParameters.m`.

Initialization

Todo: @Miro

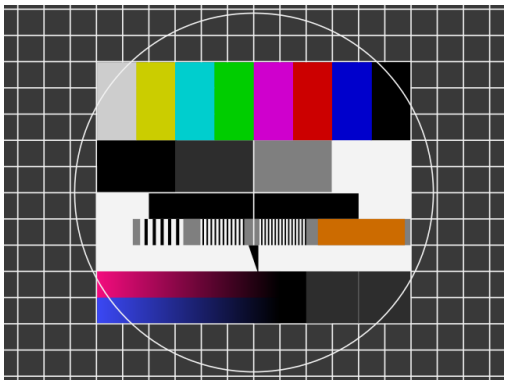

Figure 3: Graphical user interface
all features (red •), inlier features (green •), triangulated features (green ×)

Bootstraping

In order to pick reasonable image pairs for initialization an automatic procedure was implemented.
Since a good initialization relies heavily on the number of inlier keypoints a hard constrain on their minimum number of `min_num_inlier_kp` = 600 was set.

Two approaches were investigated:

- *SSD matching + Baseline/Depth*:
  correspondence search via SSD on Harris features & 8-point-Ransac for outlier rejection & baseline/depth-ratio (as proposed in the lecture)

$$\frac{C1\_baseline}{C1\_av\_depth} \geqslant min\_b2dratio = 0.1 \tag{1}$$

- *KLT + bearing angle*:
  correspondence search and inlier selection via KLT tracking on query Harris corners & minimum average bearing angle

$$av\_bearing\_angle\_deg \geqslant min\_av\_angle\_deg = 10deg \tag{2}$$

As elaborated in Section 3 the second approach outperformed the other and is selected as auto-bootstrapping method.

? findCorrespondences()

Todo: @Miro

eightPointRansac

Todo: @Miro

linearTriangulation

maybe only 1 sentences - “like in the exercise”

bundleAdjust

is here anything different then in the continuous bundle adjust?

applySphericalFilter

have a look at Section 2.2.4. Maybe put this up to here and make a cref there.

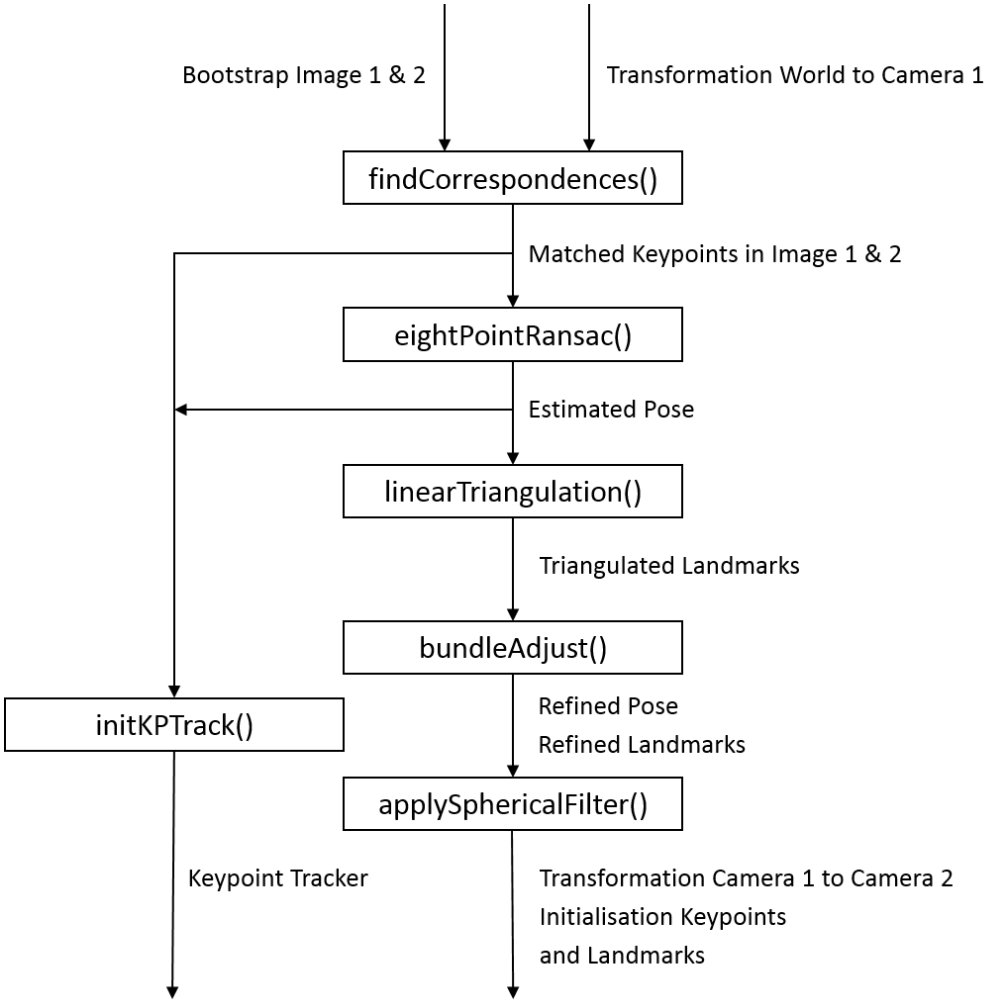

Figure 4: Init Flow chart

Continuous Operation

Continuous operation of the VO pipeline is implemented in the 'process frame' function. It tracks keypoints with corresponding landmarks over several frames while estimating the pose difference between successive frames. Further, a keypoint tracker finds new candidate keypoints which will become new landmarks if a candidate keypoint was tracked far enough and achieved 'good' trianguability. This ensures to never run out of landmarks and keypoints if the image changes over time. The routines of the continuous operation shown in Fig. 5 are described below.

Relative pose estimation

To estimate the pose difference $T_{C_iC_j}$ from frame i to j we use the p3p-RANSAC algorithm also used in exercise 5. If wished the RANSAC can also use DLT pose estimation. Using these RANSAC algorithm ensures to remove outliers from our landmarks. We don't use DLT refinement after the p3p RANSAC since the best guess from p3p often gave better results.

find_correspondences_cont()

Tracking keypoints with existing landmarks from frame i to frame j is achieved by the function $find\_correspondences\_cont$. The user can choose whether to use a KLT or Harris matcher. As a by-product, the generated query keypoints are saved to be used by the candidate keypoint tracker in a successive step so they don't have to be generated twice which saves computation time. The number of generated query keypoints is adjustable by a parameter. In case the KLT tracker is active (which does not return query keypoints by default) new query keypoints are generated using Harris features. The amount of newly generated keypoints is the difference of remaining and wished candidate keypoints.

updateKpTracks()

In every frame, candidate keypoints from previous frames are tracked to j-frame. Every candidate keypoint track consists of the following entries: $\left\{[u/v]_j, [u/v]_{first}, T_{WC_{first}}, nr\_trackings\right\}$. After successive tracking, $[u/v]_j$ as well as the number of successful successive trackings are updated. The user can choose whether to use a KLT or a Harris matcher. In case a candidate keypoint could not be tracked it's whole track gets removed from the tracker. If there are less candidate keypoints in the tracker then desired, newly generated keypoints (generated in find_correspondences_cont) are added to the tracker. Every newly added keypoint is stored together with the current pose $T_{WC_{first}}$.

triangulateNewLandmarks()

In order to localize correctly (see Section 2.2.1), a sufficient number of landmarks is required. If the number of landmarks in the current frame drops below a certain threshold, the 'triangulateNewLandmarks' function is called to generate new landmarks from the candidate keypoint tracks. To check, whether a candidate keypoint is ready for triangulation the bearing angle between it's first and it's current observation is calculated. This is done for every keypoint candidate. If the bearing angle of a candidate keypoint exceeds a certain threshold, it is removed from the keypoint tracker and a new landmark gets triangulated with the algorithm developed in exercise 4. The bearing angle threshold is adaptive to the number of remaining landmarks. The higher the number of remaining landmarks the higher the threshold. This ensures generation of new landmarks in case the filter starts to run out of landmarks but improves triangulation results once enough landmarks are in the pipeline. To filter outliers within the newly generated landmarks (e.g. from image noise) two checks are implemented:

1. Spherical filter: Discard all landmarks which are not within a half sphere in front of the camera. This removes landmarks triangulated behind the camera which are prone to bad triangulation due to the large distance (and small baseline therefore).
2. Reprojected error filter: Landmarks passed the spherical filter get reprojected into the current frame. If the reprojected point is too far apart from the candidate keypoint coordinates it is discarded.

Landmarks and their corresponding keypoints which passed these two filters are appended to the existing landmarks and keypoints vectors and will be used during the next localization iteration.

Bundle Adjustment

Todo: @Miro

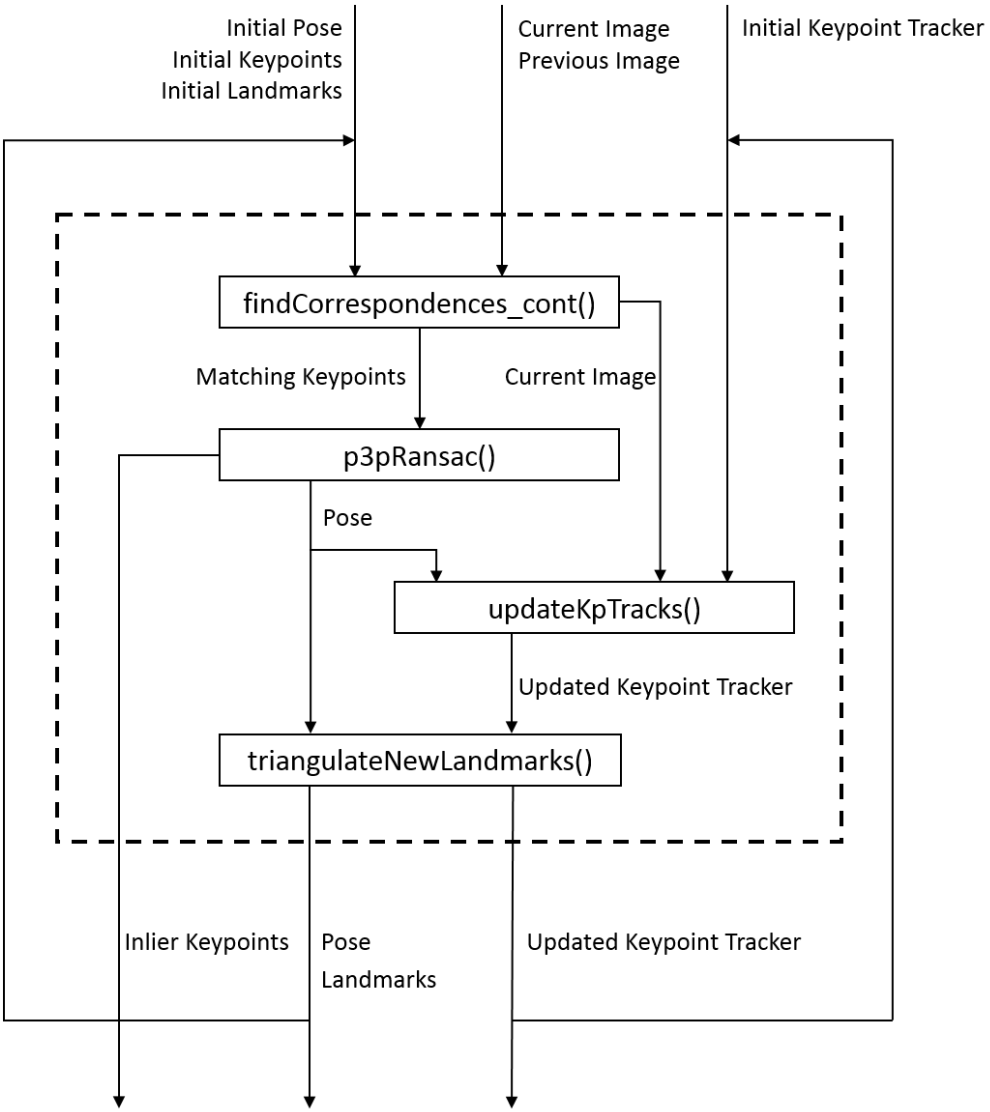

Figure 5: Cont Flow chart

Results

Bootstrapping methods

The comparison showed a significantly higher performance of the KLT tracker approach, this being orders of magnitude faster and yielding many more feature correspondences also over large distances. Table 1 depicts the index tuples retrieved through the second approach with `min_num_inlier_kp` = 600 being required.

approach		Kitti	Malaga	Parking
KLT + Bearing angle	first idx	1	1	1
	second idx	4	6	5
	# keypoints	600	600	600
	angle [deg]	1.37	4.59	4.07

Table 1: Bootstrapping pair indices for different datasets with minimum keypoint inliers

When relaxing the minimum number of inlier keypoints allowed to reach the desired baseline/depth ratio of 10 % for the first and a bearing angle of 10 degrees for the second bootstrapping approach results are shown in Table 2.

approach		Kitti	Malaga	Parking
SSD Harris desc. + baseline/depth ratio	first idx	1	1	1
	second idx	6	10	7
	# keypoints	34	39	54
	ratio [%]	10.1	10.9	11.5
KLT + bearing angle	first idx	-	1	1
	second idx	-	16	17
	# keypoints	-	99	74
	angle [deg]	-	10.4	10.55

Table 2: Bootstrapping pair indices for different datasets and no keypoint number constraint

We observed that the first approach yields far too few keypoints, due to the matchDescriptor() method. Furthermore, the pure lateral camera motion of dataset Parking suits well this baseline/depth bootstrapping approach, which confirms our intuition given the 'triangulability condition' described in Section 2.1.1.

Deploying the second approach on the Kitti dataset evidently fails due to the straight camera motion, inhibiting angles ($[0.5, 3.7]$ deg) to reach the desired 10 degrees. As soon as there is a distinctive rotational movement involved, e.g. in Malaga and Parking, a useful bootstrapping is performed.

Overall performance

(keywords: Real time ness, comparison to groundtruth, compare different datasets Impact of features) Here we describe it such as it runs best in our opinion.

- Chosen parameteters
- Runtime characteristics (e.g. average nr inlier landmarks, and new landmarks, candidate kp)
- plot of groundtruth
- 3d landmarks if possible
- speed & realtimeness, slowest functions - why?
- speed & realtimeness, slowest functions - why?

Discussion

Correspondence search

KLT vs Harris matching

Bundle Adjust

improvements reached

Pose estimation algorithm

DLT refinement vs p3p guess in continuous maybe not

Reinitialization

Feature work

What have we learned, what worked?
Possible future work, improvements (loop closure, ...)

- bootstrapping with both information rotation through bearing angle diff and translation through baseline/depth ratio
- ...

Conclusion