Visual Odometry Pipeline

Pascal Buholzer, Fabio Dubois, Milan Schilling, Miro Voellmy

January 8, 2017

@Miro : add nice image of Poly-Up dataset with red/green kps overlayed

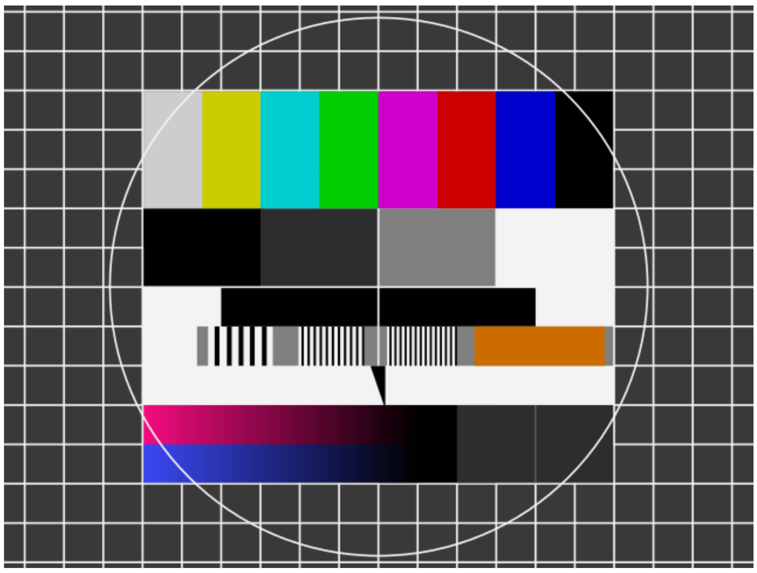


Figure 1: ...

Contents

1 Introduction

@Miro @Milan, read it trough please
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 to reduce the required computation effort.

2 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.

2.0.1 Coordinate Frames

In this mini project the coordinate frames were defined as shown in **??**. 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.

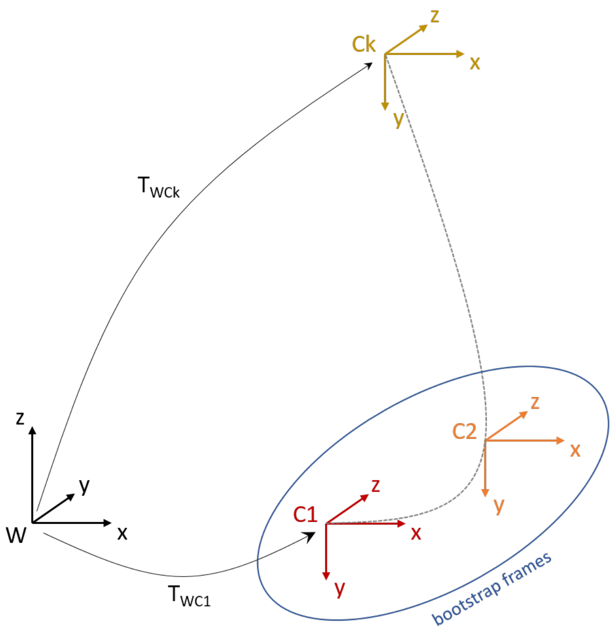


Figure 2: Coordinate Frames

2.0.2 Pipeline overview

As shown in **??** the pipeline consists mainly of three parts:
1. Bootstraping (see **??**)
2. Initialization (see (**??**)
3. Continuous operation (see **??**)

@Milan is this flowchart up to date? If yes all good. :-)

2.0.3 Conventions

- Index of previous frame: i
- Index of current frame: j
- Index of frame for newly added candidate 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.

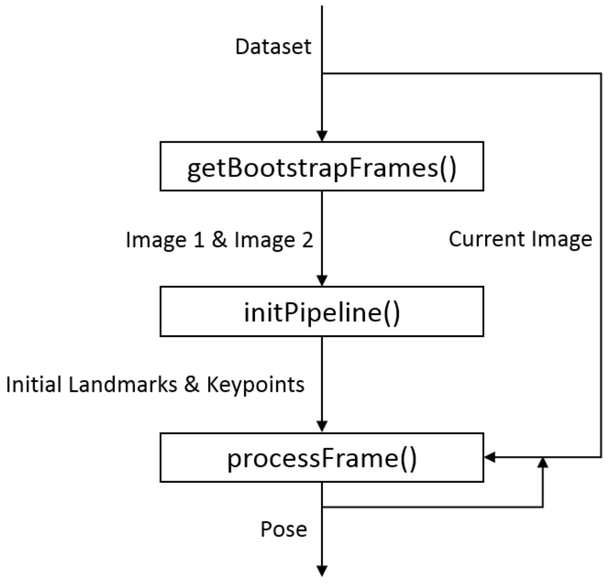


Figure 3: Overview flow chart

2.0.4 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:

- **Debug Mode:** 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.
- **Simple GUI:** 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 figure **??**. Only the most crucial entities, like number of landmarks, are visualized for intuitive understanding.

How to run the GUI 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

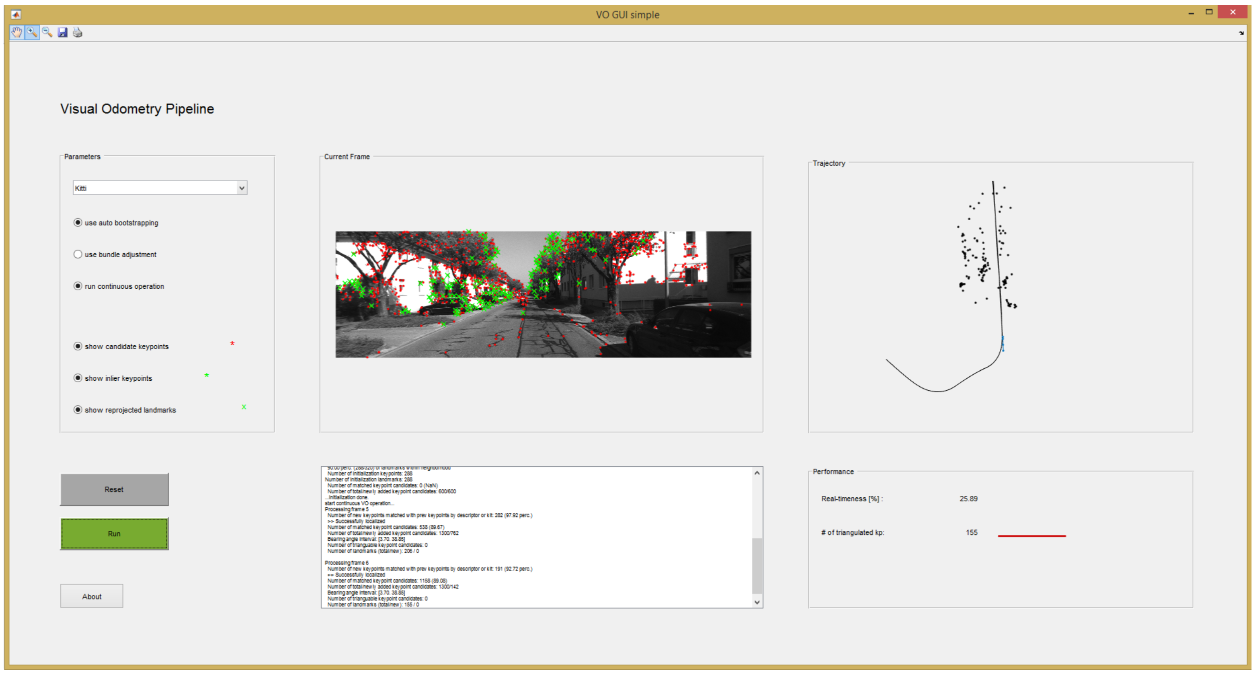
Advanced parameter tuning can be achieved by changing the default parameters in `loadParameters.m`.

2.0.5 Camera calibration

Using the Camera Calibration Toolbox for Matlab[1] from Jean-Yves Bouguet (Caltech) the camera of the an iphone 7 Plus was calibrated. A set of 17 calibration images where extracted from a calibration sequence filming a checker-board pattern from various angles.
The camera intrinsics parameters together with a fourth-order polynomial approximation of the radial-tangential distortion were calculated based on the assumption of the skew being zero. The exact values and error margins are to be found on the Github repository.

Given these calibration parameters the VO pipeline was also applied on self-generated datasets called *Poly-Up* and *Poly-Down*. Note, that no undistortion was performed, since testing showed it to be negligible.

[1] www.vision.caltech.edu/bouguetj/calib_doc/



Kitti

[✓ use auto bootstrapping]
use bundle adjustment
[✓ run continuous operation]
[✓ show candidate keypoints] *
[✓ show inlier keypoints] *
[✓ show reprojected landmarks] x

```
 90.00 perc. (288/320) of landmarks within neighborhood
  Number of initialization keypoints: 288
 Number of initialization landmarks: 288
  Number of matched keypoint candidates: 0 (NaN)
  Number of total/newly added keypoint candidates: 600/600
 ...initialization done.
 start continuous VO operation...
 Processing frame 5
  Number of new keypoints matched with prev keypoints by descriptor or klt: 282 (97.92 perc.)
  >> Successfully localized
  Number of matched keypoint candidates: 538 (89.67)
  Number of total/newly added keypoint candidates: 1300/762
  Bearing angle interval: [3.70. 38.85]
  Number of triangulable keypoint candidates: 0
  Number of landmarks (total/new): 206 / 0

 Processing frame 6
  Number of new keypoints matched with prev keypoints by descriptor or klt: 191 (92.72 perc.)
  >> Successfully localized
  Number of matched keypoint candidates: 1158 (89.08)
  Number of total/newly added keypoint candidates: 1300/142
  Bearing angle interval: [3.70. 38.85]
  Number of triangulable keypoint candidates: 0
  Number of landmarks (total/new): 155 / 0
```

Real-timeness [%] :	25.89
# of triangulated kp:	155

Figure 4: Graphical user interface
candidate keypoints (red •), inlier keypoints (green •), reprojected landmarks (green ×)

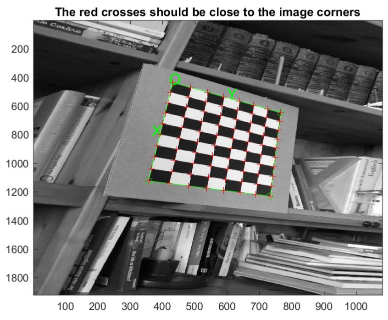


Figure 5: Calibration pattern coordinate system aligned after corner selection

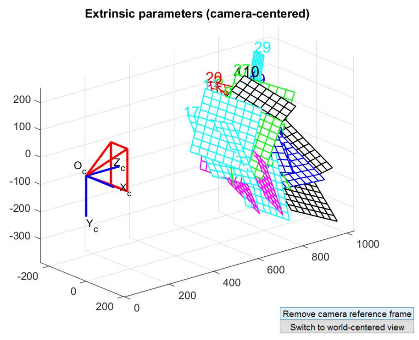


Figure 6: Relative to camera pattern poses

2.1 Initialization

The first step of the initialization is the bootstraping which outputs an image pair for further processing. This image pair varies depending on the dataset and is required since the baseline between consecutive images is often too small for accurate landmark triangulation and pose estimation.
Features are then matched across the two bootstrap images and the pose of the second camera is estimated using an 8-point-Ransac. Landmarks are generated using the pose and the matched features. These landmarks and the pose are then refined using bundle adjustment. To end the initialization the landmarks which are unrealistic are discarded.

2.1.1 getBootstrapFrames()

In order to pick reasonable image pairs for initialization an automatic procedure was implemented, as a additional degree of 'autonomy' compared to hard-coded index pairs.
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*:

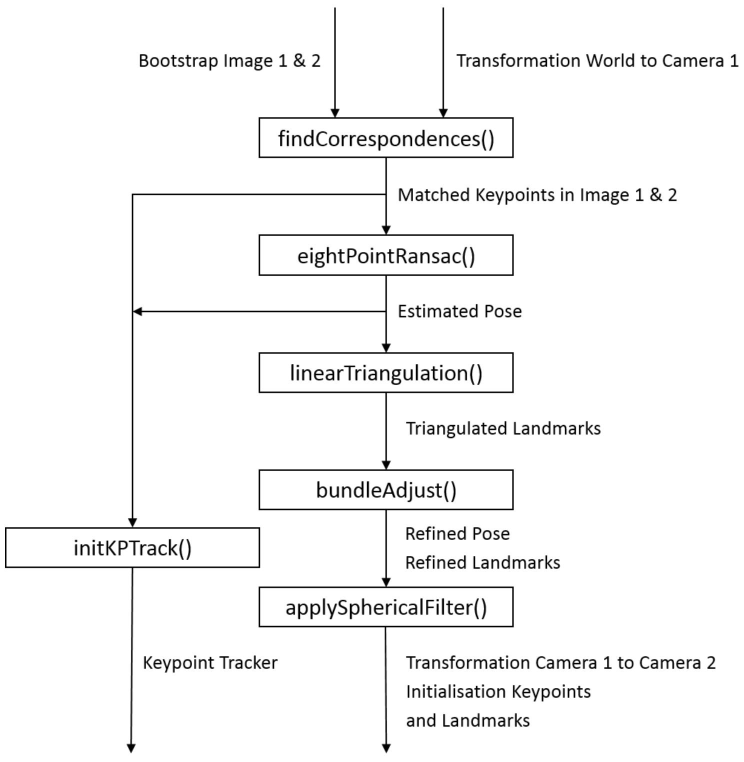


Figure 7: Initialization flow chart

correspondence search via sum of squared differences (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 Kanade-Lukas-Tomasi (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 **??** the second approach outperformed the other and is selected as auto-bootstrapping method.

2.1.2 findCorrespondences()

The same approaches discussed in **??** are used to find the final matching between the bootstrap frames.

2.1.3 eightPointRansac()

Normalized 8-point-Ransac is used to estimate the essential matrix and filter the outliers of the keypoint matching. The error function consists of the shortest distance from the query keypoint to the epipolar line. All keypoints which have an error smaller than a certain threshold are regarded as inliers. The final inlier-set is the one with the most inliers. The essential matrix obtained by the 8-point-Ransac is then decomposed with respect to the obtained translation such that we get the final transformation.

2.1.4 linearTriangulation()

3D-landmarks are generated using linear triangulation of the previously matched keypoints and transformation.

2.1.5 bundleAdjust()

The landmarks and transformation between the first two frames are optimized using bundle adjustment (non-linear least-squares minimization of the reprojection errors).

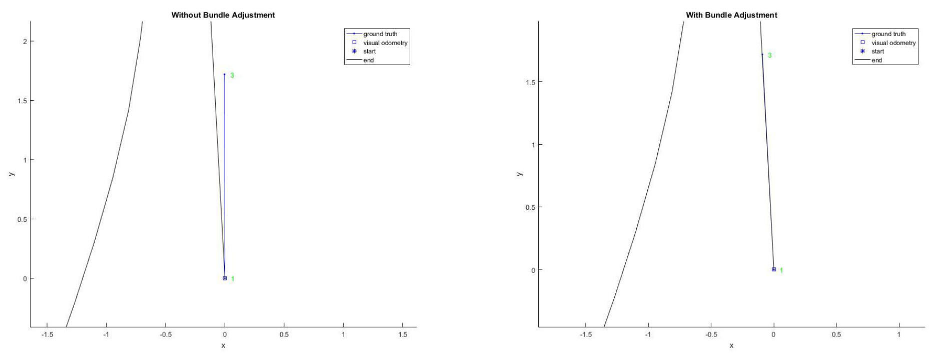


Figure 8: Initialization trajectory before (left) and after (right) bundle-adjustment

2.1.6 applySphericalFilter()

Discards all landmarks which are not within a half-sphere in front of the camera. This removes on one hand landmarks triangulated behind the camera (negative z-components) and also landmarks further away than the filter cutoff radius. With this filtering only landmarks in the neighborhood are kept, which are more reliably triangulated (smaller uncertainty cone). Note, that this absolute thresholding relies on an accurate scale estimation.

2.2 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 **??** are described below.

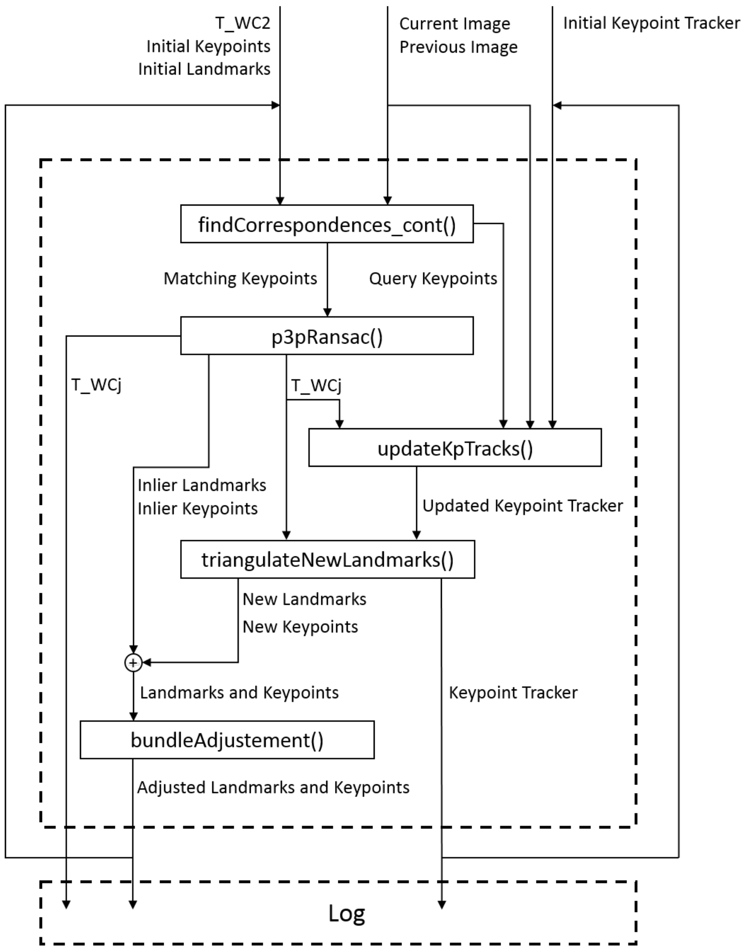


Figure 9: Continuous operation flow chart

2.2.1 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.

2.2.2 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.

2.2.3 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 $T_{WC_{first}}$.

2.2.4 triangulateNewLandmarks()

In order to localize correctly (see **??**), 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.
Adaptive bearing angle threshold: 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. See also parameter 'increase bearing angle threshold' in **??**.
Filters: Two filters are implemented to discard outliers within the newly generated landmarks (e.g. from image noise):

1. Spherical filter: Already described in **??**.
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.

2.2.5 Bundle Adjustment

Todo: @Miro Explain how it works - add to flowchart

2.2.6 Reinitialization

The quality of the transformation matrix estimated by the p3p-RANSAC depends on the number of inlier landmarks. If there are enough inlier landmarks, the pose found by the p3p-RANSAC is a good estimation. However, if there are too few inlier landmarks, the estimated pose can be considered as too inaccurate for a further use. The fact that too few landmarks were found is a sign that the last few poses weren't good enough. It is therefore a good idea to discard some of the previous poses. The reinitialization discards the last n poses and reinitializes the pipeline with the current image$j$ and the image $j-(n+1)$.

3 Results

3.1 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. ?? 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 ??.

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 ??.

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.

3.2 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.

3.2.1 Key parameters

?? shows the parameters that worked best for our implementation and led to the results presented. Tuning the datasets was always a tradeoff between speed, accuracy and robustness. The main parameter, that improved trajectory error a lot was the pixel tolerance of the RANSAC. It had to be set to a small value in order to not improve the trajectory and robustness.

Parameter / Dataset	**Kitti**	**Malaga**	**Parking**	**Poly-up**	**Poly-down**
min # of landmarks for reinit	65		80	50	50
min bearing angle for triangulation [deg]	1.5	@Milan	3.7	2	2
increase bearing angle threshold [# landmarks]	230		230	250	250
RANSAC tolerance [pixel]	2		2	3	3
# candidate keypoints in tracker	1500		1300	3500	3500

Table 3: Key parameters chosen for every dataset

3.2.2 Run time characteristics

?? summarizes the main run time characteristics of the different datasets with our pipeline.

The **parking dataset** performed fastest and had the best trajectory. Since we could not perform bundle adjustment, there is a slight drift of the estimated trajectory. However, the scene can also be nicely reconstructed in 3D from the point cloud generated (see **??**).

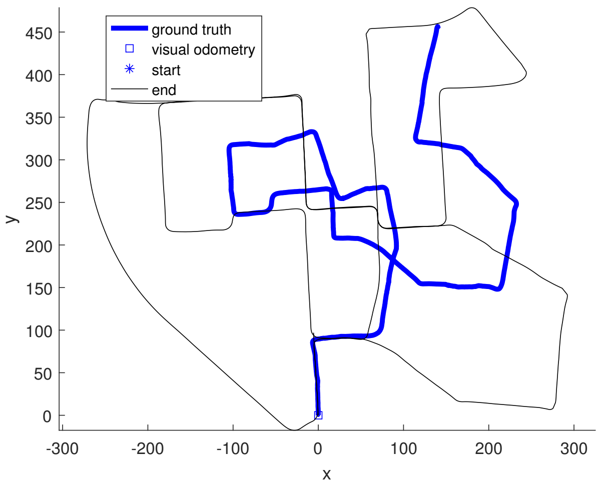
The **Kitti dataset** starts well but suffers a lot from scale drift also due to missing bundle adjustment and thus summing up errors. However, the first few curves are tackled relatively well.
@Milan und @ Fabio short summary here
(e.g. average nr inlier landmarks, and new landmarks, candidate kp)

Characteristics / Dataset	**Kitti**	**Malaga**	**Parking**	**Poly up**	**Poly down**
# reinits required	23		2	4	3
# average landmarks	∼350	@Milan	∼ 400	∼ 250	∼ 250
Frame rate [Hz]	1.4		2.6	1	1
Trajectory quality	+		++	+	+

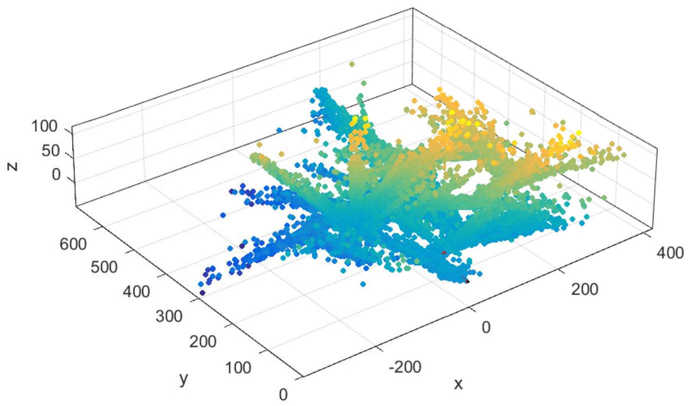
Table 4: Runtime characteristics

3.2.3 Comparison to ground truth

Kitti



(a) Trajectory vs. ground truth



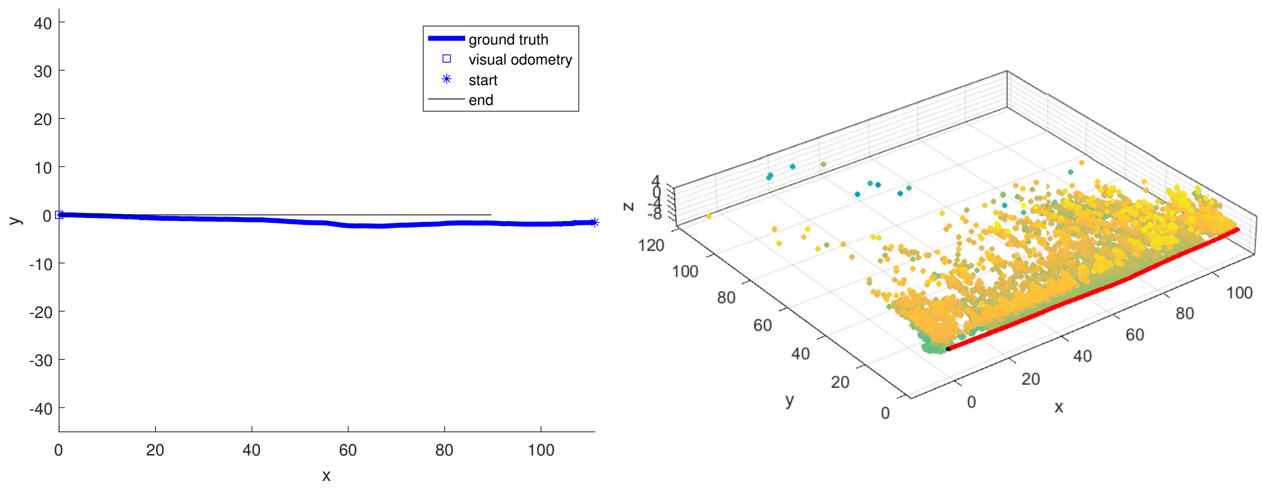
(b) 3D landmarks

Figure 10: Kitti Dataset Results

Malaga
Parking
Poly Up
Poly Down

3.2.4 Speed & real-timeness

The parameter with the biggest impact on the frame rate was the number of iterations for the P3P-RANSAC. The frame rate depends almost linearly on this parameter. Further the more landmarks tracked the slower the pipeline. However, we preferred to generate more landmarks over having a faster pipeline in order to increase robustness and accuracy.



(a) Trajectory vs. ground truth (b) 3D landmarks

Figure 11: Parking dataset results

A profiling run with the Matlab profiler tool showed that most of the computational resources is spent in the following three functions in descendant order (self-time): `p3p()`, `selectKeypoints()` and `harris()`.

4 Discussion

4.0.1 Correspondence search

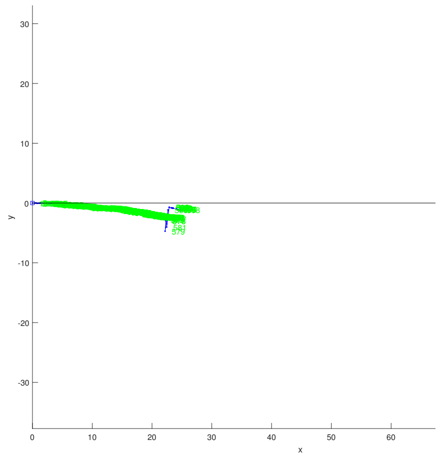
KLT vs Harris matching @ Milan

4.0.2 Bundle-adjustment

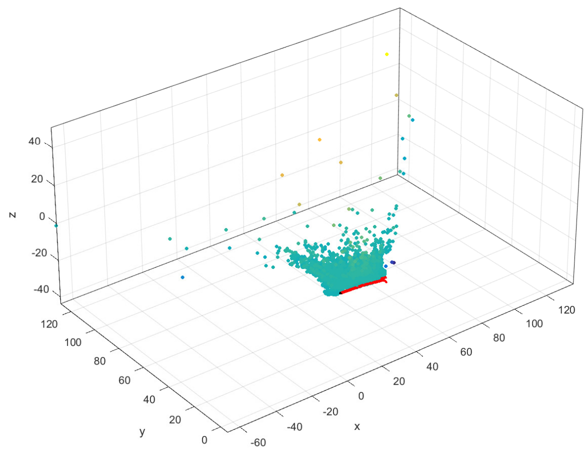
improvements reached @Miro

4.0.3 Pose estimation algorithm

We investigated pose estimation with DLT refinement after the P3P-Guess and without refinement (using the last P3P guess from RANSAC). It turned out, that very often the P3P-guess was a lot better and especially more robust then the DLT refinement. **??** shows the estimated trajectory with DLT refinement. It is clearly worse then the P3P estimate as shown in **??**. That's why we used the last P3P guess in our implementation.



(a) Trajectory vs. ground truth



(b) 3D landmarks

Figure 12: Parking dataset results with DLT refinement

4.0.4 Reinitialization

@Milan How often does it happen? Is the result better afterwards?

4.0.5 Feature work

A list of potential improvements @ Milan
Possible future work, improvements (loop closure, ...)

- bootstrapping with both information rotation through bearing angle diff and translation through baseline/depth r
- Include bundle adjustment
- Optimize runtime e.g. use a parallel for loop for P3P-RANSAC
- Reinitialize depending on $T_{C_iC_j}$ instead of number of landmarks remaining. An unusually big translation or rotation $T_{C_iC_j}$ is a sign that the pipeline start diverging.
- Loop closure and place recognition (e.g. with bag of words approach).

5 Conclusion

What have we learned, what worked? @Milan