DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

# Photometric Bundle Adjustment for Globally Consistent Mapping

Simon Klenk

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

# Photometric Bundle Adjustment for Globally Consistent Mapping

# Photometrischer Bündelausgleich für global konsistente Kartenerstellung

| | |
|---|---|
| Author: | Simon Klenk |
| Supervisor: | Prof. Dr. Daniel Cremers |
| Advisor: | M.Sc. Nikolaus Demmel |
| Submission Date: | 12.06.2020 |

I confirm that this master's thesis in Robotics, Cognition, Intelligence is my own work and I have documented all sources and material used.

Munich, 12.06.2020 Simon Klenk

# Acknowledgments

I would like to thank my advisor Nikolaus Demmel for the technical discussions and support during the thesis. I would also like to thank Prof. Dr. Daniel Cremers for the possibility of writing the thesis at the chair of Artificial Intelligence & Computer Vision. Thanks to all my family, friends and teachers who have supported me during my educational path. Thanks to my parents, my brother and my grandmother who have always encouraged me and gave me valuable advice.

# Abstract

In this work, we revise and extend an existing formulation of Photometric Bundle Adjustment (PBA). Bundle Adjustment is a computer vision technique to improve an initial reconstruction acquired from a collection of images. The reconstruction includes an estimated three dimensional map as well as an estimated camera trajectory. PBA improves the reconstruction based on optimization criteria which contain pixel intensities of the images. PBA can be used in various applications such as robot navigation or autonomous driving. There are numerous design choices made in the PBA pipeline, such as detecting occlusions in the initial map, choosing an image interpolation method, selecting a suitable representation of three-dimensional points, efficiently transforming pixel locations between frames, reducing the effect of outlier pixels which degrade the optimization, etc. In this thesis, we examine a number of design choices in the current PBA pipeline and implement alternatives as well as extensions. We provide an extensive evaluation on the Euroc MAV, Kitti odometry and a small simulated custom data set. We focus on improving the global consistency of the reconstruction by measuring how much the estimated camera trajectory deviates from the ground truth trajectory. This thesis contributes to a better understanding of the current implementation of PBA and how the proposed modifications can help to enhance the system.

# Contents

# 1 Introduction

## 1.1 Motivation

We are currently in the midst of a robotic revolution: Whereas robotic systems have previously been used almost exclusively for monotonous and well-defined tasks in production plants, a new generation of flexible robots is being developed today. These new robots can perform a wide variety of tasks by freely moving in the world and flexibly interacting with it. For example, a cleaning robot can notice rearrangement of furniture to adapt its cleaning strategy.

To notice such changes of the world, a robot must be able to localize itself in a map of the environment. Only then can the robot reasonably plan its actions in our ever-changing world. In most robotic applications, a three dimensional (3D) map is required because navigation is usually performed in 3D space. In many cases, a map is not available, considering that the robot might operate in unexplored or simply unmapped terrain. Even if large 3D maps of main city districts or buildings are available in the future, it can hardly be ensured that these maps are always up to date.

Therefore, for independent and general localization capabilities, it is very beneficial for a robot to build and update an accurate map of the environment by using its own sensing devices. The approach for jointly building a map and at the same time localizing itself is called Simultaneous Localization and Mapping (SLAM). Implementation details for SLAM heavily depend on the available sensor modalities, computing resources and general system requirements. Since cameras are versatile and cheap sensors that are part of most state-of-the-art robotic systems, a lot of research has been conducted in visual SLAM (VSLAM) which solely relies on captured images [14].

There are many unsolved issues in current VSLAM algorithms which prevent them from being integrated seamlessly into robotic systems. A fundamental issue is the accumulation of drift, which causes the position and map estimation to degrade for an increasing map size [17]. Another major problem is instability of the system due to spurious or noisy observations, which can result in a temporary or complete system failure and degrades the estimation locally. Ultimately, the goal is to obtain an accurate and globally-consistent map, which is not affected by accumulated drift or noise. Global consistency means that all map segments are consistently aligned with each other. This is the key difference to visual odometry (VO), where the generated map is only used for

navigation purposes [42]. In VO, different map segments do not need to align strictly as long as navigation performance does not degrade.

In this thesis, we investigate a post-processing approach which increases global consistency of a given initial map. This approach is in principle independent of the underlying implementation for the initial map estimation.

## 1.2 Problem Statement and Contributions

VSLAM can be divided into indirect and direct approaches. Indirect approaches, such as the influential paper ORB SLAM [40], first abstract an image to features. Usually a set of re-identifiable keypoints for each image is extracted and matched across images [17]. The resulting correspondence sets are used to recover camera motion and 3D scene structure using classical techniques from 3D geometry. A common approach called Bundle Adjustment (BA) refines these initial estimates further. Geometric BA minimizes the difference between detected 2D keypoint location and reprojected 3D points (landmarks) into the images. The minimization variables in BA are the structure and motion parameters of the scene.

Direct approaches on the other hand use raw pixel data. The main idea is the photoconsistency assumption [27]: If two identical cameras observe a 3D point from different vantage points, the response (pixel value) in the cameras at the respective pixel position is the same. In contrast to indirect methods which require re-identifiable keypoints, direct methods can sample from all pixels in the image. Similar to indirect methods where BA minimizes a geometric error, a common approach in the direct setting is called Photometric Bundle Adjustment (PBA) minimizing pixel intensity differences at pixel locations viewing the same 3D landmark.

In this thesis, we investigate how the direct image error (Equation 1.1) introduced by Engel et. al. in the VO system Direct Sparse Odometry (DSO) [21] can be adopted in a global PBA module to increase global consistency of the map. To keep DSO real-time capable, Engel et. al. perform the PBA formulation only over a sliding window of active keyframes $F$. Keyframes are a subset of all frames to reduce the number of frames used in the system. Once a keyframe becomes inactive in DSO, i.e. once it leaves the sliding window, it is marginalized from the optimization problem. In contrast to DSO, we use the direct error in a post-processing step to improve the global consistency of an initial map estimate. In our formulation, we include most frames and points in the global optimization. We do not strictly require real-time capability. We propose and investigate different modifications to the error function and analyze the assumptions that are made in the formulation.

The *research questions* that are examined ask:

- Which modifications of the photometric error function affect the global consistency of PBA? What are the significant effects on runtime?
- Which other modification such as occlusion detection algorithms or modified optimization routines affect global consistency of PBA?

The *contributions* of this thesis include:

- To our knowledge, we perform the first systematic modification study of the error function in [21] in the context of global PBA. This formulation has been widely used in systems building on the idea of DSO [23], [59], [63], [5]. We implement and analyze possible useful modifications and extensions to the direct image error. We analyze using occlusion detection algorithms before performing PBA. Additionally, we analyze and experiment with details in the optimization procedure.
- We evaluate the proposed modifications quantitatively using common error metrics on the widely used datasets Euroc MAV [13], Kitti odometry [24] and an existing simulated image dataset obtained by CARLA Simulation software [19].
- We extend an existing manual custom solver for the PBA formulation.

$$E_{photo} = \sum_{i \in F} \sum_{\mathbf{p} \in P_i} \sum_{j \in obs(\mathbf{p})} \sum_{\mathbf{p} \in N_p} w_p \| \underbrace{(I_j[\mathbf{p}'] - b_j) - \frac{t_j e^{a_j}}{t_i e^{a_i}} (I_i[\mathbf{p}] - b_i)}_{residual} \|_\tau \tag{1.1}$$

Equation 1.1 states the investigated error function used in [21]. It contains the pixel value difference between a pixel $\mathbf{p}$ in the host frame $i$ and the corresponding pixel $\mathbf{p}'$ in the target frame $j$. The pixel $\mathbf{p}'$ is determined by the rotation $\mathbf{R}_{ji}$ and translation $\mathbf{t}_{ji}$ from frame $i$ to frame $j$, the landmarks inverse distance $id_p$ and the camera's intrinsic parameters $c$. The relationship is captured by the warp function in Equation 1.2:

$$\mathbf{p}' = \Pi_c(\mathbf{R}_{ji}\Pi_c^{-1}(\mathbf{p}, id_p) + \mathbf{t}_{ji}) \tag{1.2}$$

The *projection* from 3D world to 2D image coordinates considering the intrinsic camera parameters $c$ is denoted by $\Pi_c$. The *unprojection* from a 2D image point to a 3D point with inverse distance $id_p$ is given by $\Pi_c^{-1}(\mathbf{p}, id_p)$. A detailed overview of common camera models can be found in [56]. Since photoconsistency does not hold in most real world scenarios, [21] formulates a relaxed assumption called irradiance constancy: If two cameras observe a 3D point from different vantage points, the received irradiances

(in Watts per square meter) [36, p. 40] in the cameras at the respective pixel positions are the same. To obtain approximate irradiance from pixel values, the exposure times $t_i, t_j$ and the non-linear camera response $\gamma()$ which maps from received energy to pixel values must be taken into account. It is also common to model spatially varying sensitivity of the camera sensor. One popular choice is a *vignetting* [7] model which assumes less sensitivity towards the sensor's outer boundary. In case of unknown exposure times $t_i, t_j$, the affine brightness parameters ($a_i$, $b_i$, $a_j$, $b_j$) can be optimized. If exposure time is known, the brightness parameters are set constant. The Huber norm $||\cdot||_\tau$ is employed to reduce the influence of outlier residuals to the optimization problem, see Section 3.4.3. In a direct setting, outliers include occlusions, dynamic scene changes or illumination change. The observed objects need to be lambertian, i.e. they radiate the same brightness independent of the observer's viewing angle, as opposed to for example shiny or transparent surfaces [7]. To obtain the total photometric error $E_{photo}$, we sum over: (I) all frames $F$, (II) all points $P_i$ hosted in frame $i$, (III) all target frames $obs(p)$ where the point $p$ projects to, and (IV) all points of the residual pattern $N_p$. In DSO $F$ consist of all active frames which form the sliding window. In this thesis, $F$ contains all keyframes in the initial map estimate. Figure 1.1 visualizes how the total photometric error $E_{photo}$ is obtained. DSO uses the displayed residual pattern consisting of multiple hosted points which belong to the same landmark, i.e. they share the inverse distance parameter $id$. This increases robustness by providing additional information to the optimization. One of the ideas we investigate in this thesis is the residual pattern size and shape on the PBA output.

## 1.3 Scope

*Programming:*
To modify the cost as proposed above, we adapt existing code for a ceres-based [4] and for a manual solver implementation. Ceres is a C++ open-source library for solving non-linear least-squares optimization problems. The manual solver offers more flexibility in modeling and logging and shows superior runtime.

*Initial Map:*
In this thesis, we use the refined output of Direct Sparse Odometry with Loop Closure (LDSO [23]). LDSO is an extension of DSO by a loop closure module which increases global consistency. Loop closures arise when the camera returns to a previously visited area. LDSO employs Pose Graph Optimization (PGO), explained in chapter 2.1. The map after PGO is the input to our PBA module. We denote this initialization as *init_pgo* in the results table in chapter 4

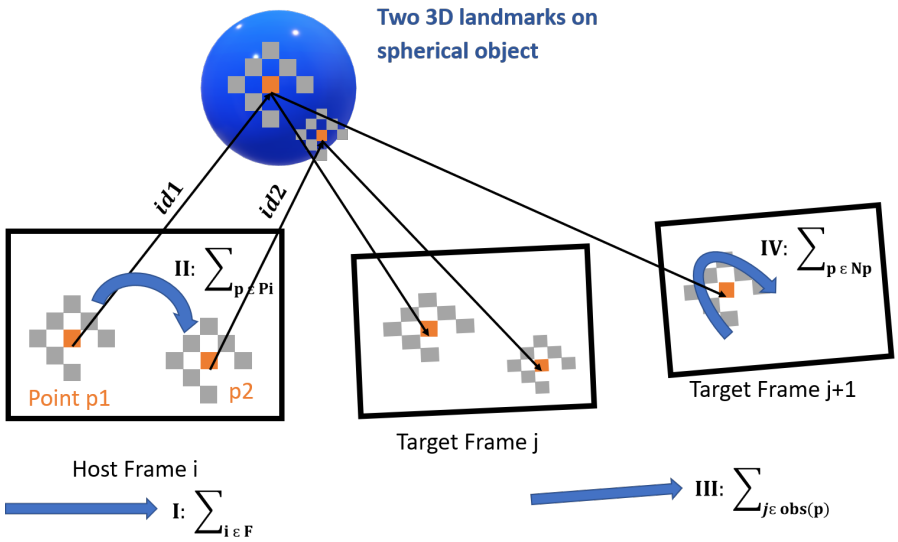



Figure 1.1: **Photometric Bundle Adjustment Visualized.** Orange points show the central pixels. Blue Arrows indicate the summations. Grey pattern is the residual pattern $N_p$. Point p1 is hosted in frame i and observed in frame j and j+1. Point p2 is hosted in frame i and observed by frame j. We estimate the transformation $T_i$ from world to camera frame $i$ and one inverse distance $id$ parameter for each landmark. Note that with our warp function, the residual pattern can be rotated, translated and distorted in the target frame.

*Map and Trajectory Representation:*
We represent the camera trajectory as a discrete collection of $N$ transformation matrices $\{\mathbf{T}_t\}_{t=1}^{N}$ from the world frame to any camera frame $i$. The first camera frame is considered our world reference frame. We represent the 3D map by an unordered collection of sparse landmarks. The landmarks are parameterized relative to their host frame. We do not model surfaces in the 3D space, and we do not model semantics of the reconstructed objects. We restrict our estimation to rigid bodies, i.e. non-deformable objects.

*Global Consistency Measure:*
To evaluate global consistency, the absolute trajectory error (ATE) is used. At first, our estimated trajectory is aligned with the given ground truth trajectory in a common reference frame by associating poses for each discrete timestamp $t$. ATE measures the Euclidean distance $\|\mathbf{x_t} - \tilde{\mathbf{x}}_\mathbf{t}\|$ in the common reference frame between the estimated camera position $\tilde{\mathbf{x}}_t$ and the ground truth position $\mathbf{x}_t$ at time $t$. In this thesis we focus on the root mean squared error (RMSE) over the whole trajectory, which measures deviations in a phyiscally meaningful unit such as meters:

$$ATE_{rmse} = \sqrt{\frac{1}{N}\sum_{t=0}^{N}\|\mathbf{e}_t\|_2^2} = \sqrt{\frac{1}{N}\sum_{t=0}^{N}\|\mathbf{x}_t - \tilde{\mathbf{x}}_t\|_2^2} \tag{1.3}$$

In contrast to the common arithmetic mean $ATE_{mean}$, taking the RMSE gives bigger influence to single outlier distances between the two trajectories. This is due to squaring the individual distances. It allows for a better comparison of *consistency*: Imagine trajectories A and B have the same $ATE_{mean}$ compared to the ground truth trajectory G. A is consistently close to G with mostly small errors whereas B deviates a lot for some time stamp but exactly matches G at other time stamps. We prefer trajectory A with a lower $ATE_{rmse}$.

*Data and Performance Comparison:*
Instead of using stereo information, we only we only operate on monocoluar images. In most cases, we evaluate our methods on all sequences of Euroc MAV, Kitti odometry and our simulated data called Carla. We make use of a *test data* set which allows for small sanity checks of our implementation. This test data consists of the pose-graph optimized (see Section 2.1) sequences EurocV101, Carla circle and Carla 12cam. For most experiments we show the *averaged* $ATE_{rmse}$ over multiple sequences relative to our PGO initialization. Note that we employ the geometric mean here, as it is the most

suitable mean to compare relative performance measures [22].

$$ATE_{rmse,geo} = (\prod_{i=1}^{N} \frac{ATE_{rmse,\ dataset\ i}}{ATE_{PGO\ rmse,\ dataset\ i}})^{1/N} \tag{1.4}$$

Additionally, for some experiments we show detailed comparison in terms of convergence behaviour during optimization, i.e. plotting for example intermediate ATE and cost values as well as solver parameters. All ATE measures are displayed in meters, except otherwise stated.

## 1.4 Outline

This thesis is organized as follows: In chapter 2 we present state-of-the-art methods for achieving global consistency. Chapter 3 describes the mathematical notation and ideas used throughout this thesis. Chapter 4 shows experiments and results. The conclusion is given in chapter 5.

# 2 Related Work

The fundamental idea of photometric minimization in image analysis was popularized by Horn and Schunk [25] as well as Lucas and Kanade [34] in 1981. Similar to their ideas we assume photoconsistency. We cannot model actual irradiance constancy since no full photometric calibration is available in our datasets, i.e. the exposure times, vignetting and camera response function $\gamma()$ are missing. The topic of *structure from motion* is closely related to VSLAM/VO, where the focus in structure from motion lies in offline applications having access to all acquired data [53, p. 345-377]. The focus of VSLAM lies in incremental and real-time applications [14]. In our work, we focus on scenes consisting of rigid, i.e. non-deformable objects. The topic of non-rigid large-scale VSLAM is largely unexplored [14].

## 2.1 Pose Graph Optimization

SLAM problems can be described as graphs: Each node represents a state variable and a connection relates two states by a relative measurement [31]. Kümmerle et al. propose the popular framework *g2o* for solving graph-based optimization problems in [31]. In landmark-based SLAM the states consist of camera poses and landmark positions. A connection from one camera pose to another camera pose is a relative pose measurements from the VO front-end. Similarly, a connection of a camera pose to a landmark is given by the VO position estimation of the landmarks. The landmark positions are usually initialized by triangulation of detected 2D correspondences [14]. To increase global consistency of the map, many SLAM systems [40], [23], [27] use the idea of *pose graph optimization* (PGO): A global graph of all (keyframe) camera poses connected by relative pose measurements is jointly optimized. PGO is part of the SLAM *back-end*. While the SLAM *front-end* is responsible for building a sufficiently accurate initial map, the SLAM *back-end* deals with optimizing and managing the map for long-term usage [14]. The main advantage of PGO is that the number of optimization variables grows only with the number of selected keyframes. A main disadvantage is its lack of robustness, i.e. its property to converge to nonsensical solutions if a few poses are incorrectly associated [52]. Furthermore, omitting the landmarks in the PGO formulation neglects their influence on the solution. In this thesis we use a PGO module to obtain an initial guess as input to our global PBA layer.

## 2.2 Global Bundle Adjustment

Global geometric BA has been studied widely and is used in many SLAM applications for structure refinement [54]. The geometric error is a 2D Euclidean distance between detected keypoints and reprojected landmarks. Therefore, geometric BA is limited by the precision of the keypoint detection algorithm as well as the intrinsic calibration for computing the projections.
The published literature surrounding PBA is rather limited. In this thesis we use and modify the cost function introduced in DSO [21]. DSO and its extension to more sophisticated camera models such as omnidirectional DSO [37], perform PBA in a sliding-window fashion only. LDSO is an extension of DSO, which performs global map optimization in a direct setting [23]. In contrast to our work, it performs pose-only BA. It uses DSO as direct odometry front-end, changing the point selection strategy to additionally select corner features. The combination of corner features and originally sampled points from DSO are used to notice loop closures. The estimated trajectory is optimized by employing PGO. LDSO estimates similarity transformations between poses, which take into account a rigid body motion as well as a scale factor $\alpha \neq 0$ [36, p. 269]. This scale factor models the scale drift which is present in monocular geometry reconstructions [23]. In this thesis, our global PBA implementation is initialized using LDSO. In our work, we only model point clouds consisting of sparse landmarks. Note that similar cost functions to the one presented in DSO can also be employed for dense 3D geometry reconstruction and texture estimation using a PBA approach, see [18].

Similar to LDSO, Direct Sparse Mapping (DSM) [63] by Zubizarreta et al. improves the global map acquired by a direct odometry frontend. In contrast to LDSO and DSO, DSM proposes a new criterion to select keyframes that are included in the PBA problem: Their new criteria is a combination of temporal proximity (as in DSO) and covisibility of shared map points (as in LDSO but without computing corner features explicitly). Zubizarreta et al. perform full PBA over the resulting set of keyframes, solving for optimal increments to absolute poses and inverse depth parameters. DSM provides a quantitative evaluation of map quality, whereas most VO/VSLAM systems only report trajectory errors. Inspired by [27], Zubizarreta et al. model the residual histogram of each keyframe by a t-distribution, to reduce the influence of outlier residuals on the optimization result. In Section 3.4.2 in this thesis, we propose a corrected robust weight formula based on a per-frame modelling of residual distributions. We show the results of using this corrected formula in Section 4.6.

In our work, we do not keep track of image features, but we revert to the intensity-based formulation of PBA, see Section 1.2. We do not perform PBA on a small subset of frames such as in DSO, omnidirectional DSO or DSM, but we include most of the keyframes in the optimization problem. Alismail et al. provide a PBA formulation in

[5]: In their work, the residual pattern can only be offset by a 2D translation vector in the target image plane. We assume that this simple warp function can improve the initialization solely because *small* increments to the initial structure and motion parameters are estimated, in a sliding-window fashion. Their PBA problem is initialized using ORB-SLAM [41], which performs geometric BA based on features. Since image projections generally result in subpixel positions in the target image, Alismail et al. use bilinear interpolation for image values in their work. The Huber loss function is employed to reduce the influence of outlier on the global optimization. In contrast to our work and DSM, they do not model residual distributions on a per-frame basis to chose the robust loss weights. Landmark positions are parametrized using absolute 3D Euclidean coordinates, and camera poses are parametrized using absolute poses. This parametrization allows the appearance of observed landmarks to change in the host frame during optimization. This could introduce biased solutions where both patches drift to the same constant brightness areas such that the photometric error is minimal [5]. Therefore, Alismail et al. fix the patch appearance in the host. The quantities of interest in the optimization problem are the optimal updates to the structure and motion parameters given the initialization. Their PBA is implemented in a sliding window fashion, which requires a strategy of avoiding re-initializing new scene points at similar image locations as before. Occlusions are detected using zero normalized cross correlation (ZNCC), which is a photometric measure for similarity of the residual pattern between host and target frame. We also perform photometric occlusion detection based on ZNCC in Section 4.13.

Note that we deviate from Alismail et al. as follows in this thesis : Our method allows a general six DOF rigid transformation of the residual pattern in 3D. This allows for more general patch configurations in the target image. For example, only our formulation can model a host-target pair of cameras which is given by a rotation around the coinciding principal axis of the cameras. We use efficient approximations of our more expressive warp function in Section 4.5 to make it computationally feasible. Furthermore, we propose interpolation schemes which give superior results over simple bilinear interpolation, see Section 4.3. Our method is initialized using LDSO. We do not implement a sliding window approach but we add all poses and landmarks to a global PBA problem. For outlier detection, we further propose a geometric occlusion detection algorithm, see Section 4.12. In addition to the Huber loss with a fixed tuning parameter as in the work of Alismail et al., we investigate data-dependent tuning parameter estimation techniques. Furthermore, we employ the Student's t-distribution with data-adaptive tuning parameters such as in [27], [63] and propose a corrected mathematical formulation suitable for our PBA setup in Section 3.4.2.

## 2.3 Robust Estimation Techniques in SLAM

Sünderhauf et al. [52] argue that *false positive* loop closures often cause a failure in the least squares formulation of PGO. This can lead the map converging towards heavily distorted and unusable results. A false positive loop closure wrongly connects two camera poses, hence incorrectly enforcing a similar 3D position for both cameras. This unavoidable error originates from wrong data associations in the VO front-end's place recognition module that detects which poses share common landmarks [52]. Simply using standard robust loss functions such as the Huber loss (see Section 3.4.3) is not sufficient to deal with false positive constraints [51] in PGO. Therefore, in [51], Sünderhauf et al. introduce the idea of *switchable constraints* jointly optimizing over the poses and the topology of the pose graph. An extension of this work by Agarwal et al. [3], which drastically improves the convergence speed, is called *dynamic covariance scaling*. Both these methods share the same idea: Each pose constraint is augmented with one *switch variable*, a multiplicative factor between 0 and 1. The pose connection in the graph can be turned off by setting the factor to 0, it can be fully kept by using a factor of 1 or it can be in between those states only influencing the overall optimization up to its multiplicative factor. Similarly, in [33], a latent weight variable is assigned to each loop closure constraint, but the graph is optimized using an Expectation Maximization (EM) algorithm. The EM-algorithm is an iterative algorithm which is generally applied to find the maximum likelihood of probability models containing latent (un-observable) variables [8, p. 439]. A main disadvantage of the mentioned methods is that the search space increases by adding additional variables. This demands more computational power and the optimization might get stuck in local minima more often [2]. Additionally for [51] and [3], regularization techniques which introduce new tuning parameters are required.

Another algorithm called *Realizing, Reversing, Recovering* [51], [32] by Latif et al. tries to identify false positive loop closures by checking the consistency of loop closure edges in accordance with the final trajectory estimate. The set of all loop closure edges which stem from the place recognition module is divided into disjoint subsets called *clusters*. Each cluster links topologically similar parts of the robot trajectory. A *plain graph* which contains all poses and VO constraints is extended by only one loop closure cluster. The resulting error has to fall below a threshold or else the whole cluster is rejected. This consistency check is also performed for single constraints inside the cluster, comparing the error introduced by single constraints to the error of the whole cluster. Furthermore, the error of extending the plain graph by individual clusters is compared among different clusters to find mutually consistent clusters. The output of the algorithm is a set of loop closure constraints which can produce useful PGO results.

As explained in chapter 3.4.1, maximum likelihood estimation of the map state

leads to a least-square cost function under a Gaussian likelihood assumption. In [44], Olson and Agarwal propose to use a multi-modal *max-mixture* likelihood. The multi-modality allows to keep track of probabilities for loop closure associations during the optimization. The optimization process is computationally feasible and the robustness is increased. However, a max-mixture model is a non-smooth approximation of a standard Gaussian mixture model [2]. Therefore, only an approximate solution is found, and no convergence guarantees can be given.

In this thesis, we follow yet another approach called *M-estimation*, which is very popular for BA problems. The **M** stands for maximum likelihood estimation, but for an unknown, non-Gaussian model [10]. Least-square costs which arise from Gaussian noise assumptions are extremely sensitive to outliers [8, p. 104]. Probabilistically, this can be derived from the fact that Gaussians are so called *light-tailed* [10] distributions meaning that the probability for an event which is far away from the expected value is very low. The M-estimation approach replaces the least-squares criterion by a *robust loss function*. Optimizing the robust loss function is less severely influenced by existing outliers in the data. Agamennoni et al. [2] show that the noise model often follows *elliptical distributions* for certain *robust loss functions*. Note that M-estimation can be applied to any kind of least-squares estimation process.

# 3 Mathematical Foundations for Bundle Adjustment

This chapter describes the mathematical concepts and notation used throughout the thesis. Lowercase light symbols ($f$) represent scalars or functions. Lowercase bold letters ($\mathbf{t}$) represent vectors. Uppercase bold symbols ($\mathbf{R}$) represent matrices.

## 3.1 Skew Symmetric Matrices

A skew symmetric matrix $\mathbf{A} \in SO(3)$ is given by [36, definition 2.2]

$$\mathbf{A}^T = -\mathbf{A},\ \mathbf{A} \in \mathbb{R}^{3x3} \tag{3.1}$$

Such skew symmetric matrices can be represented by the isomorphism $\wedge$ from the vectors $\mathbf{u} = (u_1, u_2, u_3) \in \mathbb{R}^3$ to $so(3)$ with [9, section 9.2]

$$\hat{\mathbf{u}} = \begin{pmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{pmatrix} \tag{3.2}$$

The inverse operation is denoted by $\vee : so(3) \to \mathbb{R}^3$. Skew symmetric matrices model the cross product:

$$\mathbf{u} \times \mathbf{v} = \hat{\mathbf{u}}\mathbf{v} \tag{3.3}$$

## 3.2 Rigid Body Motion

A rigid body motion is defined as a map:

$$g : \mathbb{R}^3 \to \mathbb{R}^3;\ \mathbf{u} \to g(\mathbf{u}) \tag{3.4}$$

which preserves norm and cross products of any two vectors [36, definition 2.3]:

$$\begin{aligned} \|g(\mathbf{u})\| &= \|\mathbf{u}\|,\ \forall \mathbf{u} \in \mathbb{R}^3 \\ g(\mathbf{u}) \times g(\mathbf{v}) &= g(\mathbf{u} \times \mathbf{v}),\ \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^3 \end{aligned} \tag{3.5}$$

Therefore, a rigid body motion can be represented by a motion of a Cartesian coordinate frame attached to the rigid body. Such a motion contains six degrees of freedom (DOF). In this thesis, we use two ways to represent rigid body motions:

a) Representation via transformation matrix $\mathbf{T} \in SE(3)$ of the special euclidean group [9, section 1.2.3]:

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tag{3.6}$$

where the rotation matrix $\mathbf{R} \in SO(3)$ is of the special orthogonal group

$$SO(3) = \{\mathbf{R} \in \mathbb{R}^{3x3} \,|\, \mathbf{R}^T\mathbf{R} = \mathbf{I},\, det(\mathbf{R}) = +1\} \tag{3.7}$$

and the translation vector $\mathbf{t} \in \mathbb{R}^3$ is a 3D vector describing the translation of the origin. The inverse motion can be described by the inverse of $\mathbf{T}$:

$$\mathbf{T}^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tag{3.8}$$

Throughout this thesis, the camera trajectory is described as a sequence of rigid body motions. $T_i$ refers to the transformation from world frame to frame $i$; $\mathbf{T}_{ji}$ refers to the relative transformation from frame $i$ to frame $j$.

$$\mathbf{T}_{ji} = \mathbf{T}_j\,\mathbf{T}_i^{-1} \tag{3.9}$$

b) Representation via Lie Algebra $se(3)$ [9, section 9.2]:
The rotational matrix $\mathbf{R}$ contains nine variables, but considering the constraints of $SO(3)$, it only has three DOF. It is desirable to set up an unconstrained optimization problem and directly optimize the three DOF instead of ensuring the constraints numerically. This can be done by formulating the problem in the Lie Algebra $se(3)$, because any element of $se(3)$ called twist $\hat{\xi}$

$$se(3) = \{\hat{\xi} = \begin{pmatrix} \hat{\mathbf{w}} & \mathbf{v} \\ \mathbf{0} & 0 \end{pmatrix} \mid \hat{\mathbf{w}} \in so(3),\, v \in \mathbb{R}^3\} \tag{3.10}$$

can be mapped to an element of the rigid body motions $T \in SE(3)$ by the unique *exponential map* [11]:

$$exp:\ se(3) \to SE(3);\ e^{\hat{\xi}} = \sum_{n=0}^{\infty} \frac{\hat{\xi}^n}{n!} = \mathbf{T} \tag{3.11}$$

The twist coordinates $\xi = (\mathbf{w}^T, \mathbf{v}^T)^T$ contain three variables for the rotational component $\mathbf{w}$ and three variables for the translational component $\mathbf{v}$. The inverse mapping,

the *logarithmic map*, exists for all transformations but is not unique: There exist many different twist coordinates describing the same transformation. This can be understood by considering that a rotation by 0 or by 360 degrees around the same axis results in the same position.

## 3.3 Landmark Representation

Landmarks are 3D points that can be described by three coordinates $\mathbf{p} \in \mathbb{R}^3$ in a reference frame. Inspired by [21], we use a different representation in this thesis: Each 3D point is associated with exactly one host frame. It is represented by its 2D pixel coordinates in the respective host frame and an associate inverse distance *id*. Unlike geometric BA, we keep the pixel location in the host frame fixed during optimization, and we only optimize the inverse distance parameter. As a consequence, every residual involves exactly two frames, i.e. host $i$ and target frame $j$. We refer to frame $i$ and $j$ as *host-target-pair ij* or *ji* (since in most cases the frame $j$ also hosts points which project to frame $i$).

A *bearing vector* $\mathbf{b}$ is a 2-DOF vector of unit-length describing a direction. We can use a bearing vector to connect the origin of a host camera with the 2D pixel coordinates. Therefore, an inverse distance *id* together with either a bearing vector or 2D pixel coordinates in a host frame can be used interchangeably to describe a landmark.

## 3.4 Parameter Optimization

We refer to the error terms as *residuals* $r$. In the case of PBA, the residuals are the difference of pixel intensities, possibly weighted by affine lightning transform parameters. We model landmarks sparsely, i.e. we only transform selected pixels of interest from the host to all *reachable* target frames. A target frame is reachable if the pixel can be projected from the host to the target image and maintain a safety margin to the boundary in order to allow meaningful interpolation. The set of all reachable observations is reduced by filtering as follows: We remove observations which are too close to the camera, have too large absolute residual value or where the ratio of target inverse distance $id_{target}$ over host inverse distance $id_{host}$ is either too large or too small. This later check is performed to remove observations with very different object scales in host and target frame. We investigate geometric and photometric occlusion detection algorithms to remove observations, see Section 4.12 and 4.13. In a second filtering stage, we remove landmarks if they contains too few (remaining) observations, their inverse distance is negative, or the host inverse distance $id_{host}$ is either too large or too small.

Each of the described filters requires to select thresholds. In this thesis, we keep all these thresholds constant throughout our experiments, except otherwise stated. For each residual pattern belonging to one landmark, we only estimate one scalar value *id*; for each keyframe we estimate six transformation parameters. Since we have many more residual patterns than frames, the system is generally over-determined.

### 3.4.1 Bayesian Derivation of Least Squares

From a Bayesian point of view [8, chapter 1.2], PBA tries to find the structure and motion parameters $\boldsymbol{\theta}$ which maximize the posterior probability $p(\boldsymbol{\theta}|\mathbf{r}) \propto p(\mathbf{r}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})$ given all residuals $\mathbf{r}$. Equality up to a constant is denoted by $\propto$. We assume a uniform prior on $\boldsymbol{\theta}$, i.e. $p(\boldsymbol{\theta}) = constant$, we can alternatively minimize the following negative log likelihood:

$$\arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathbf{r}) = \arg\min_{\boldsymbol{\theta}} -log(p(\boldsymbol{\theta}|\mathbf{r})) = \arg\min_{\boldsymbol{\theta}} -log(p(\mathbf{r}|\boldsymbol{\theta})) \tag{3.12}$$

A residual $\boldsymbol{r}_i$ can be composed of all residuals in the residual pattern, i.e. $\dim(\boldsymbol{r}_i) = N_p$ or we can formulate the equations pixel-wise, i.e. $\dim(\boldsymbol{r}_i) = 1$. We assume that every residual $\boldsymbol{r}_i$ is coming from an independent distribution giving the following joint likelihood:

$$p(\mathbf{r}|\boldsymbol{\theta}) = \prod_{i=1}^{N} p_i(\boldsymbol{r}_i|\boldsymbol{\theta}) \tag{3.13}$$

This likelihood models unknown error sources resulting in non-zero photometric residuals, given the structure and motion parameters. These error sources include false structure and motion estimates, noise in the images, occlusions, non-lambertian objects as well as varying exposure times and scene lightning. Minimization of the negative log likelihood results in the following optimization problem:

$$\arg\min_{\boldsymbol{\theta}} -log(p(\mathbf{r}|\boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} -log(p_i(\boldsymbol{r}_i|\boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \frac{1}{2}\rho_i(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2) \tag{3.14}$$

where $-log(p_i(\boldsymbol{r}_i|\boldsymbol{\theta})) = \frac{1}{2}\rho_i(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$ [1]. For this optimization problem, there exists an iteratively reweighted least squares (IRLS) optimization problem which shares the

[1] Our particular choice of using $\rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$ limits the class of possible residual distributions $p(\boldsymbol{r}_i|\boldsymbol{\theta})$. However, in practice many common distributions can be modelled. Our formulation is in accordance with [2], [10], [54]. Note how the robust problem is formulated slightly differently in [27], [53, p. 318] and [45] by using $\rho(\|\boldsymbol{r}_i\|)$. When changing between $\rho(\|\boldsymbol{r}_i\|)$ and $\rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$, the robust loss function must be adapted to still yield the same optimization problem.

same stationary points $\boldsymbol{\theta}^*$ [2, eq. 8], namely:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} w_i(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2) \ \|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2, \quad w_i(s) = \frac{d\rho_i(s)}{ds} \tag{3.15}$$

where we denote $\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2 = s$. In the special case when we set $\rho_i(s) = s$ (least squares loss) in our original problem, we have $w_i = 1 \ \forall \ i$ in the IRLS formulation. In the general case of $\rho_i(s)$, a solution to the IRLS problem can be obtained by the following iteration scheme, where the weights are constant during every *outer iteration* $k$. In every iteration, a NLS problem of the form:

$$\boldsymbol{\theta}^{k+1} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} w_i(\boldsymbol{\theta}^k) \ \|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2 \tag{3.16}$$

must be solved, where

$$w_i(\boldsymbol{\theta}^k) = \frac{d\rho_i(s)}{ds}\Big|_{s=\left\|\boldsymbol{r}_i(\boldsymbol{\theta}^k)\right\|^2} \tag{3.17}$$

Close to convergence, we have $\boldsymbol{\theta}^k \approx \boldsymbol{\theta}^{k+1}$. Therefore, for the intermediate problems in step $k$, stationary points close to convergence are given by

$$\sum_{i=1}^{N} 2 \ \rho_i'\left(\left\|\boldsymbol{r}_i(\boldsymbol{\theta}^{\boldsymbol{k+1}})\right\|^2\right) \boldsymbol{r}_i(\boldsymbol{\theta}^{\boldsymbol{k+1}})^T \ \nabla_{\boldsymbol{\theta}} \boldsymbol{r}_i(\boldsymbol{\theta}^{\boldsymbol{k+1}}) = \mathbf{0} \tag{3.18}$$

We use the identity $\frac{d\|\boldsymbol{x}\|^2}{d\boldsymbol{x}} = 2\boldsymbol{x}$ [47, equation 131] and the chain rule. The stationary points of our original problem 3.14 are determined by the same equation up to a factor of two. By casting our robust problem to an IRLS problem, we can use common minimization algorithms. In this thesis, we do not solve the NLS exactly. Instead, we employ the Gauss-Newton algorithm which linearizes the residuals at the current parameter state, see Section 3.4.4. By linearizing, we only solve an ordinary, linear least squares (OLS) problem in every iteration.

One important thing to note here: Since in VSLAM/VO we are continuously calculating our photometric residuals $\boldsymbol{r}_i$ for a given image sequence (given $\boldsymbol{\theta}$), we can model the likelihood $p_i(\boldsymbol{r}_i|\boldsymbol{\theta})$ by fitting a probability distribution to the observed photometric residual histogram. This allows for a data-adaptive, automatic selection of weights $w_i$ during the runtime of the algorithm. However, it is not clear what the best choices for histogram modelling are. First of all, the type of distribution must be defined. The work of [2] provides a quantitative methodology to chose among a class of pre-selected candidate distributions. Inspired by [27] and [63], we employ a t-distribution to model

the photometric residuals. In general, multimodal distributions could be an even better fit to the observed histogram. Secondly, it is unclear how to actually aggregate the residual histogram. For example, it is possible to compute one histogram for all residuals of the PBA problem and fit the distribution parameters to it. In this case, we would model only one distribution $p(\boldsymbol{r}_i|\boldsymbol{\theta})$ for all residuals and lose the dependence of index $i$ on $p$ in the notation. Similar to this idea, in this thesis we average the t-distribution's scale parameter across multiple datasets to obtain one global constant, see experiment 4.6. We compare this approach to the idea of computing one histogram per target frame. Modelling one distribution per target frame is similar to [63], but we derive a corrected formula presented in Section 3.4.2. In experiment 4.6, we analyze the effect of the correct weight formula. We furthermore re-implement the approach by Agamennoni et al. given in [2] to automatically determine the weights $w_i$ during the runtime of the algorithm for Huber as well as the t-distribution loss, see Section 4.9. In general, histograms could also be aggregated among a certain sliding-window of neighboring keyframes, which could improve the robustness of the parameter fitting algorithm.

### 3.4.2 Robustification by Using the t-Distribution

Assuming the photometric residuals are originating from a t-distribution is a popular choice in the literature [27], [63]. The parameters of a t-distribution are influenced far less by outliers than for example the Gaussian distribution. In our notation, we drop the dependency of $r_i(\boldsymbol{\theta})$ on $\boldsymbol{\theta}$:

$$p_i(r_i|\sigma, v) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\,\pi\,\sigma^2}\,\Gamma(\frac{v}{2})}\,(1+\frac{r_i^2}{\sigma^2\,v})^{-\frac{v+1}{2}} = \frac{A}{B}\,(1+\frac{r_i^2}{\sigma^2\,v})^{-\frac{v+1}{2}} \tag{3.19}$$

We additionally drop the dependency of the t-distribution parameters on the target frame to avoid cluttered notation, i.e. we write $v$ and $\sigma$ instead of $v_i$ and $\sigma_i$. Note that when fitting the t-distribution to $N_{target}$ target frames, we would have $N_{target}$ different $v$ and $\sigma$ to estimate.

Setting $r_i^2 = s$, taking the negative logarithm and multiplying by 2 (see Equation 3.14) of this expression gives us:

$$\rho_i(s) = -2log(A) + 2log(B)\ + (v+1)\cdot log(1+\frac{s}{\sigma^2\,v}) \tag{3.20}$$

Therefore the weight is given by:

$$w_i(s) = \frac{d\rho_i(s)}{ds} = (v+1)\frac{\frac{1}{\sigma^2\,v}}{1+\frac{s}{\sigma^2\,v}} = \frac{v+1}{\sigma^2\,v+s} = \frac{1}{\sigma^2}\frac{v+1}{v+(\frac{r_i}{\sigma})^2} \tag{3.21}$$

Note that our derivation differs to the weight presented in the literature [27], [28] by the factor $\frac{1}{\sigma^2}$. This is due to the fact, that in these systems only one constant scale parameter $\sigma$ is estimated for all residuals, i.e. $\sigma$ can then be factored out of the overall log likelihood. In our manual solver implementation, we estimate a $\sigma_t$ parameter during runtime for each target keyframe $t$ separately. Therefore, we need to keep this factor in our formulation. Similarly, it should be kept in the formulation given in [63].

The scale parameter $\sigma_t$ of the t-distribution represents uncertainty: For more unsure, spread out distributions, a higher $\sigma_t$ is estimated. Conversely, the factor $\frac{1}{\sigma_t^2}$ represents certainty. Assume the following scenarios:

- $r_i \gg \sigma_t \Rightarrow w_i \to 0$. If we have a very large residual compared to our scale estimation $\sigma_t$, we are in the outlier region and should weight the residual less, eventually converging to 0.
- $r_i \ll \sigma_t \Rightarrow w_i \to \frac{1}{\sigma_t^2} \frac{v+1}{v}$. If we have a small residual compared to $\sigma_t$, we consider this an inlier and weight it with the factor $\frac{1}{\sigma_t^2}$ (considering $\frac{v+1}{v} \approx 1$ for $v = 5$).

Consider one inlier residual $r_a \ll \sigma_a$ coming from a distribution with $\sigma_a$ and one inlier residual $r_b \ll \sigma_b$ from a distribution with $\sigma_b$. If $\sigma_a \ll \sigma_b$, we get $w_a \gg w_b$. So the residual $r_a$ is given a higher weight and contributes more to the optimization, because its estimated uncertainty $\sigma_a$ is smaller. The residual $r_b$ comes from a more unsure distribution and therefore should not contribute as much to the overall cost.

The parameters of the t-distribution $<\sigma, v>$ can be estimated by using an EM Algorithm. In this thesis we set $v = 5$ because fitting $v$ in an online, data-dependent fashion is not expected to bring major improvements, see [63]. We fit $\sigma$ using the derived maximization in [50, section 3, step 5], see Equation 4.31. The initial value for the EM algorithm is set to $\hat{\sigma}_{MAD}$ using:

$$\hat{\sigma}_{MAD} = 1.4826 \operatorname*{median}_i(|r_i - \operatorname*{median}_i(r_i)|) \tag{3.22}$$

MAD abbreviates Median Absolute Deviation. $\hat{\sigma}_{MAD}$ is a robust scale estimation.

### 3.4.3 Robustification by Using the Huber Loss

There exist multiple robust loss functions in the literature [62]. In this thesis, we employ the Huber loss function among others, given by:

$$\rho_\tau(s) = \begin{cases} s & \text{if } s < \tau \\ 2\sqrt{\tau s} - \tau, & \text{else} \end{cases} \tag{3.23}$$

If the residuals are in the inlier region, i.e. $s < \tau$, the Huber function is equivalent to assuming a Gaussian distribution. Residuals in the outlier region only have a linear influence on the cost. The Huber loss function can be motivated by the same Bayesian derivation as above, assuming an underlying Huber density for the residuals [39]. However, we do not explicitly model the parameters of the Huber density. The *efficiency* of an estimator describes the certainty of the parameter estimates. To keep 95% of the efficiency of the original least-squares formulation, we must set $\tau = 1.345\ \sigma$, where $\sigma$ is the standard deviation [58], [62]. Since $\sigma$ is unknown and needs to be estimated from the data, we investigate multiple strategies in Section 4.8: (1) setting $\tau$ to a constant value, (2) setting $\tau = 1.345\ \hat{\sigma}_{MAD}$, (3) setting $\tau = 1.345\ \hat{\sigma}_{sample}$ where $\hat{\sigma}_{sample}$ is the sample standard deviation [30, p. 1064], (4) setting $\tau = 1.345\ \hat{\sigma}_{tdist}$ where $\hat{\sigma}_{tdist}$ is estimated with the same EM algorithm as for the t-distribution. In Section 4.9, we determine the robust weights $w_i$ automatically for the Huber loss, using the proposed approach by Agamennoni et al. given in [2].

### 3.4.4 Levenberg-Marquardt Algorithm

The *general* Levenberg-Marquardt (LM) algorithm is based on the idea of Newton's method in optimization [54, section 4]: Newton's method starts at an initial guess $\boldsymbol{\theta}^{(0)}$ and iteratively approximates the non-linear cost function $C(\boldsymbol{\theta})$ by its second order Taylor expansion at the current evaluation point. The next evaluation point $\boldsymbol{\theta}^{(k+1)}$ is set to the minimizer of this second order approximation. The general update equations of Newton's methods are given by:

$$\boldsymbol{H}\,\Delta\boldsymbol{\theta} = -\nabla C(\boldsymbol{\theta}) \tag{3.24}$$

where $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}$ and $H_{ij} = \frac{\partial^2 C}{\partial x_i \partial x_j}$.

The Gauss-Newton (GN) approximation approximates the Hessian matrix for NLS problems by first order derivatives only. For NLS problems of the form:

$$\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_i w_i\, r_i(\boldsymbol{\theta})^2 = \min_{\boldsymbol{\theta}} \boldsymbol{r}(\boldsymbol{\theta})^T\ \boldsymbol{W}_k\ \boldsymbol{r}(\boldsymbol{\theta}) \tag{3.25}$$

with $\boldsymbol{W}(\boldsymbol{\theta}) = diag(w_1, ..., w_N)$, the Hessian matrix can be written as

$$H_{jk} = 2 \sum_i w_i \left(\frac{\partial r_i}{\partial x_j}\,\frac{\partial r_i}{\partial x_k} + r_i\,\frac{\partial^2 r_i}{\partial x_j\,\partial x_k}\right) \approx 2 \sum_i w_i \frac{\partial r_i}{\partial x_j}\,\frac{\partial r_i}{\partial x_k} \tag{3.26}$$

Writing above approximation in matrix form by using $J_{ij} = \frac{\partial r_i}{\partial x_j}$ and $\boldsymbol{W} = diag(w_1, ..., w_N)$, yields:

$$\boldsymbol{H} \approx 2\ \boldsymbol{J}^T \boldsymbol{W} \boldsymbol{J} \tag{3.27}$$

This approximation is valid for small residuals and close to linear residuals (second order terms are negligible) [2]. $\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J}$ is positive definite by construction. Since $\nabla C(\boldsymbol{\theta}) = 2\,\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{r}$, the Gauss-Newton update for NLS problem becomes:

$$(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J})\,\Delta\boldsymbol{\theta} = -\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{r} \tag{3.28}$$

Levenberg-Marquardt methods solve a regularized system which is controlled by a dampening parameter $\lambda$ and a diagonal matrix $\boldsymbol{M}$:

$$(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J} + \lambda\,\boldsymbol{M})\,\Delta\boldsymbol{\theta} = -\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{r} \tag{3.29}$$

If $\boldsymbol{M} = \boldsymbol{I}$, the dampening parameter $\lambda$ continuously interpolates between Gauss-Newton ($\lambda \to 0$) and standard gradient descent ($\lambda \to \infty$) with step size $\frac{1}{\lambda}$. Gradient descent can always produce a decrease in the error [43, p. 21] given a sufficiently small step size.

The choice of $\boldsymbol{M} = diag(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J})$ damps each component differently, which is useful if the parameters are of different scale. The Hessian matrix and its approximation contain curvature information of the objective function. Again if we chose a very large $\lambda$ in this case, the update is approximately given by $\Delta\boldsymbol{\theta} \approx \frac{1}{\lambda} diag(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J})^{-1}\,\boldsymbol{J}^T\boldsymbol{W}$, i.e. we update the parameter $\theta_i$ less (note the inversion) if the curvature along parameter dimension $i$ is high.

The LM-algorithm can also be cast as a *trust-region approach* [43, p. 258], where in every iteration we solve a subproblem of linearized residuals subject to a constraint limiting the maximum update step to the current *trust region radius*. In this thesis, we do not adopt a formal trust-region approach, but we update the value of $\lambda$ as discussed in the next section. Note that our updating $\lambda$ is analogous to the trust region approach and both share similar convergence properties [43, p. 268].

### 3.4.5 Levenberg-Marquardt for PBA

In Section 3.4.1 we have shown that we can cast our robustified non-linear cost function for PBA to an IRLS formulation. In every outer iteration $k$, we linearize the current NLS problem, see iteration scheme 3.16. We solve the resulting OLS problem by first applying a dampening matrix $\mathbf{M}$. Overall, we perform the following algorithm in our

[2] An alternative derivation of the Gauss-Newton update equation for NLS, is to linearize the residuals to $\mathbf{r_{lin}}^k = \mathbf{r}^k + \mathbf{J}^k\,\Delta\boldsymbol{\theta}^k$ in every iteration [43, page 255] and solve $\arg\min_{\Delta\boldsymbol{\theta}^k} \left\|\mathbf{r_{lin}}^k\right\|_2$.

manual solver:

**Algorithm 1:** Levenberg-Marquardt for PBA

**Result:** stationary point $\boldsymbol{\theta}^*$ for PBA cost $\sum_{i=1}^{N} \rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$

1 **Input:** load $\boldsymbol{\theta}^0$ from initialization (LDSO)
2 set initial $\lambda$, $\lambda_{increase} > 1$ and $\lambda_{decrease} > 1$;
3 compute the scales $\sigma^2$ per-frame or globally (using for example Equation 4.31 for the t-distribution case) or set constant;
4 **for** $k = 1 \ldots k_{max}$ **do**
5 compute Jacobians $\boldsymbol{J}_k^T$ at $\boldsymbol{\theta}^k$ using relative-absolute decomposition, [45, chapter 6.2];
6 compute diagonal matrix $\boldsymbol{W}_k$ using Equation 3.17 or alternatively using the self-tuning approach [2] (see Section 4.9);
7 linearize: compute $\mathbf{J}_k$ and $\mathbf{r}_k$ to obtain $\boldsymbol{r}_{lin} \approx \mathbf{r}_k + \mathbf{J}_k \Delta\boldsymbol{\theta}$, see Section 3.4.6 ;
8 **for** $lm = 1 \ldots lm_{max}$ **do**
9 solve $(\boldsymbol{J}_k^T \boldsymbol{W}_k \boldsymbol{J}_k + \lambda \boldsymbol{M}_k) \Delta\boldsymbol{\theta}_k = -\boldsymbol{J}_k^T \boldsymbol{W}_k \boldsymbol{r}_k$ (OLS problem);
10 update $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \Delta\boldsymbol{\theta}$;
11 **if** $C(\boldsymbol{r}(\boldsymbol{\theta}^{k+1})) < C(\boldsymbol{r}(\boldsymbol{\theta}^k))$ **then**
12 $lm = lm_{max}$ (accept step);
13 decrease $\lambda$: $\lambda = \frac{\lambda}{\lambda_{decrease}}$ (move towards Gauss-Newton);
14 **else**
15 $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k$ (restore old states);
16 increase $\lambda$: $\lambda = \lambda \cdot \lambda_{increase}$ (move towards Gradient Descent);
17 **end**
18 **end**
19 **end**

If the update decreases the cost, we update our state $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \Delta\boldsymbol{\theta}$ and we decrease the dampening parameter $\lambda^{k+1} = \frac{\lambda^k}{\lambda_{decrease}}$ with $\lambda_{decrease} > 1$. This increases the possible update step length in the next iteration and makes the update direction move closer to GN. If the update increases the cost, we keep the states by setting $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k$ and we increase the dampening parameter by setting $\lambda^{k+1} = \lambda^k \cdot \lambda_{increase}$.

Note how we check the decrease of the cost in line 11: In Section 4.10 we analyze the effect of evaluating the original PBA cost instead of the least squares cost $C(\boldsymbol{r}(\boldsymbol{\theta}))$ as discussed in [60].

A detailed description of how to efficiently solve the linear system in line 9 can be found in [45, chapter 6.2]. Because of the structure of the (approximated) Hessian matrix, we can use the Schur complement to make it computationally feasible [21]. The

Schur complement uses the fact that the update equations $\boldsymbol{H}\,\Delta\boldsymbol{\theta} = -\nabla C(\boldsymbol{\theta})$ can be decomposed into a block-matrix structure, where the bottom right element is easily invertible [54].

$$\underbrace{\begin{pmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{pmatrix}}_{\boldsymbol{H}}\underbrace{\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix}}_{\Delta\boldsymbol{\theta}} = \underbrace{\begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix}}_{-\nabla C(\boldsymbol{\theta})} \tag{3.30}$$

In the case of PBA, $\mathbf{A}$ contains the camera-camera correlations, $\mathbf{D}$ contains only diagonal entries since landmarks are uncorrelated. Matrix $\mathbf{B}$ and $\mathbf{C}$ store the camera-landmark correlations. The vector $\mathbf{x}$ comprises all $N_{cam}$ camera pose updates, and the vector $\mathbf{y}$ comprises all $N_{lmk}$ landmark inverse distance updates. $N_{cam}$ is the number of cameras multiplied by six, since each camera is parameterized with six absolute pose parameters. $\mathbf{f}$ and $\mathbf{g}$ are the gradients computed for the poses and landmarks, respectively. Note that we usually have many more landmarks than camera poses in PBA. Therefore, using the Schur complement helps a lot because we can first invert the *reduced camera system* [54] system of size $\{N_{cam} \times N_{cam}\}$ (if $\mathbf{D}$ contains only non-zero diagonal entries) to obtain $\mathbf{x}$:

$$\mathbf{x} = (\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\,(\mathbf{f} - \mathbf{B}\mathbf{D}^{-1}\mathbf{y}) \tag{3.31}$$

$\mathbf{S} = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}$ is called Schur complement of the block matrix $\boldsymbol{H}$. Given $\mathbf{x}$, $\mathbf{y}$ can easily be obtained by using:

$$\mathbf{y} = \mathbf{D}^{-1}(\mathbf{g} - \mathbf{C}\mathbf{x}) \tag{3.32}$$

Inverting a diagonal matrix can be performed by inverting all elements individually. In Section 4.11, we investigate the solver behaviour for using different dampening strategies for the LM steps. In particular, we apply dampening not just to the original Hessian matrix $\boldsymbol{H}$ but we validate if it is possible to alternatively dampen the Schur complement for efficiency reasons.

### 3.4.6 Hessian Matrix for Robust Loss Functions (Triggs Correction)

In Section 3.4.4, we provided an alternative derivation for the Gauss-Newton algorithm. This derivation performs a first-order linearization of the non-linear residual $\mathbf{r}_{lin}(\boldsymbol{\theta}) \approx \mathbf{r}(\boldsymbol{\theta}^k) + \mathbf{J}(\boldsymbol{\theta}^k)\,(\boldsymbol{\theta} - \boldsymbol{\theta}^k) = \mathbf{r}_k + \mathbf{J}_k\,\Delta\boldsymbol{\theta}$ function at the current evaluation point $\boldsymbol{\theta}^k$. Plugging the linearized residual back into the NLS cost:

$$C_{OLS}(\boldsymbol{\theta}) = \mathbf{r}_{lin}(\boldsymbol{\theta})^T\ \boldsymbol{W}\ \mathbf{r}_{lin}(\boldsymbol{\theta}) \tag{3.33}$$

gives the GN update equations:

$$(\boldsymbol{J}_k^T\boldsymbol{W}\boldsymbol{J}_k)\,\Delta\boldsymbol{\theta} = -\boldsymbol{J}_k^T\boldsymbol{W}\boldsymbol{r}_k \tag{3.34}$$

as before. Solving this system and re-linearizing the residuals at the new evaluation point is repeated until convergence.

In [60], [61], the same idea is applied to the robustified PBA cost function. First, we linearize the residuals and then plug them into the robust cost:

$$C_{pba}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \rho(\|\mathbf{r}_{lin,i}(\boldsymbol{\theta})\|^2) \tag{3.35}$$

Subsequently a second order expansion of the robust loss functions $\rho()$ around $\Delta\boldsymbol{\theta} = 0$ is performed, which leads to the Triggs correction [60]:

$$\rho(\|\mathbf{r}_{lin,i}(\boldsymbol{\theta})\|^2) \approx 2\rho'_{ik}\boldsymbol{r}_{ik}^T\boldsymbol{J}_{ik}\Delta\boldsymbol{\theta} + \Delta\boldsymbol{\theta}^T\boldsymbol{J}_{ik}^T \underbrace{(\rho'_{ik}\boldsymbol{I} + 2\rho''_{ik}\boldsymbol{r}_{ik}\boldsymbol{r}_{ik}^T)}_{\boldsymbol{H}_{ik}} \boldsymbol{J}_{ik}\Delta\boldsymbol{\theta} + const. \tag{3.36}$$

where $\rho'_{ik} = \rho'(\left\|\mathbf{r}_i(\boldsymbol{\theta}^k)\right\|^2)$ and $\rho''_{ik} = \rho''(\left\|\mathbf{r}_i(\boldsymbol{\theta}^k)\right\|^2)$. In contrast to the IRLS updates, we use a corrected Hessian matrix that takes the second order derivative of the robust loss function $\rho''$ into account. The second order derivative of the residual is ignored. We employ this correction only for positive definite $\boldsymbol{H}_{ik}$, i.e. when $\rho'_{ik} + 2\rho''_{ik}||\boldsymbol{r}_{ik}||_2 > 0$, analogous to the implementation in [4]. Alternatively, reverting to a positive definite approximation in the case of a negative definite Triggs Hessian $\boldsymbol{H}_{ik}$, converges to higher objectives according to [60].

According to [54], the Triggs correction can be implemented by re-weighting the Jacobian by

$$\boldsymbol{J}_{reweighted,i} = \sqrt{\rho'}(1 - \alpha\frac{\boldsymbol{r}_i\boldsymbol{r}_i^T}{\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2})\ \boldsymbol{J}_i \tag{3.37}$$

and the residuals by

$$\boldsymbol{r}_{reweighted,i} = \frac{\sqrt{\rho'}}{1-\alpha}\boldsymbol{r}_i \tag{3.38}$$

where $\alpha$ is given by

$$\alpha = \frac{1 + \sqrt{1 + 2\ \|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2\ \frac{\rho''}{\rho'}}}{2} \tag{3.39}$$

In this thesis, we implemented the Triggs correction according to the above formulas in our manual solver. We present the results in Section 4.7.

# 4 Experiments and Results

In the following sections, we present modifications to PBA and their effect on global consistency (ATE in meters), map quality and convergence behaviour. We discuss feasibility of implementation and effects on runtime. We provide an extensive evaluation on Kitti odometry 00-10, Euroc MAV and a simulated dataset using Carla, see Table 4.1. Additionally, we use a small *test dataset* to validate our implementations: It consists of Euroc MAV V101, Carla Circle and Carla 12 cam, a subset of Carla Circle.

| dataset | #l-c | #kf | #lm | #obs |
|---|---|---|---|---|
| carlacircle | 6 | 294 | 105604 | 833412 |
| eurocMH01 | 16 | 481 | 308039 | 2291053 |
| eurocMH02 | 7 | 382 | 249158 | 1919379 |
| eurocMH03 | 19 | 650 | 376060 | 2992960 |
| eurocMH04 | 3 | 413 | 198726 | 1532359 |
| eurocMH05 | 16 | 400 | 214337 | 1680738 |
| eurocV101 | 21 | 695 | 449646 | 3639173 |
| eurocV102 | 11 | 673 | 350979 | 2764613 |
| eurocV103 | 1 | 1251 | 538534 | 4469763 |
| eurocV201 | 7 | 412 | 246423 | 2013929 |
| eurocV202 | 22 | 868 | 429225 | 3322179 |
| eurocV203 | 0 | 1083 | 325252 | 2660546 |
| kitti00 | 501 | 3999 | 1705939 | 14263131 |
| kitti01 | 0 | 829 | 164757 | 1361659 |
| kitti02 | 29 | 4515 | 1848411 | 15415771 |
| kitti03 | 0 | 459 | 226726 | 1869337 |
| kitti04 | 0 | 248 | 92155 | 763672 |
| kitti05 | 229 | 2137 | 920247 | 7636899 |
| kitti06 | 82 | 848 | 339315 | 2817939 |
| kitti07 | 6 | 753 | 333366 | 2771703 |
| kitti08 | 0 | 3839 | 1749070 | 14416276 |
| kitti09 | 0 | 1474 | 619933 | 5176014 |
| kitti10 | 0 | 1010 | 404889 | 3316595 |

Table 4.1: **Information about dataset.** Number of: loop closures (l-c), keyframes (kf), landmarks (lm) and observations (obs) displayed. Euroc has loop closures for all but V203, on Kitti there are datasets with and without loop closures. Our *test dataset* used for developement contains Carla Circle and a slightly differently pre-processed eurocV101 sequence.

In most experiments, we report the geometric average of ATE over all sequences, relative to the LDSO initialization *init_pgo*, see Equation 1.3. To allow for a more detailed analysis, we have split our dataset into the following four sub datasets:

- **euroc-ok:** All Euroc sequences but euroc-fail. On these sequences, init_pgo yields lower ATE than init_odometry. Additionally, PBA yields lower ATE for most tested configurations.
- **euroc-fail:** eurocMH04, eurocV103 and and eurocV203. On these sequences, init_odometry has (local) failures, i.e the map deviates (locally) strongly from ground truth. PBA still improves the local consistency of these maps, but the global measure of ATE often increases compared to init_pgo. Hence, comparing ATE is not very informative on euroc-fail.
- **kit-no-loop:** All Kitti sequences but kit-loop.
- **kit-loop:** kitti00, kitti02, kitti05, kitti06 and kitti07.

The following list gives an overview over the conducted studies:

1. Landmark Parameterization
   a) **Inverse Depth vs. Inverse Distance** (Section 4.1) Can we interchange between distance and depth parameterization?
   b) **Normal Vector for Residual Pattern** (Section 4.2) Can we improve ATE or map quality results by optimizing a normal vector for each landmark's residual pattern?
2. Host - Target Frame Relationship
   a) **Interpolation Schemes in Target Image** (Section 4.3) How should we interpolate in the target image?
   b) **Residual Formulations** (Section 4.4) Which residual formulation is ideal for PBA?
   c) **Approximated Warp and Motion Jacobian** (Section 4.5) Which approximations are valid for the host-target transformation chain?
3. Robust Norms
   a) **Corrected Weight for t-Distribution** (Section 4.6) Does the correct weight formula improve PBA results?
   b) **Corrected Levenberg-Marquardt Hessian (Triggs Correction)** (Section 4.7) Does the second order (Triggs) correction of the Hessian for robust norms improve PBA results?

c) **Estimating Scale for Robust Huber Loss** (Section 4.8) How should the Huber parameter be selected for PBA? Can it be reliably estimated from data with existing approaches?

d) **Self-Tuning M-Estimators Approach** (Section 4.9) Can we find data-dependent tuning constant for the robust loss functions (Huber, t-distribution and others), such as in "Self-Tuning M-Estimators" by Agamennoni et al. [2]?

4. Levenberg-Marquardt Routine

   a) **Step Criteria** (Section 4.10) Which cost needs to be evaluated for validation of the LM update step?

   b) **Dampening Strategies** (Section 4.11) How do different dampening strategies affect PBA results?

5. Pre-Processing to Improve the Initialization

   a) **Geometric Occlusion Detection** (Section 4.12) Can a simple geometric occlusion detection algorithm improve PBA results?

   b) **Photometric Occlusion Detection using ZNCC** (Section 4.13) Can a simple photometric occlusion detection algorithm improve PBA results?

We use the following default configuration for all our experiments:

| | |
|---|---|
| Landmarks: inverse distance | Interpolation: bilinear_smooth |
| Landmarks: no normal vectors | Triggs correction: false |
| Residual: SSD | Warp: full (approximated) |
| Robust norm: global Huber $\tau = 5.0$ | LM step criteria: linearized cost $\mathbf{r}^T\mathbf{W}\mathbf{r}$ |
| landmark LM dampening: none | camera LM dampening: identity |
| Patch: DSO pattern | Solver: manual solver |
| No geometric occlusion detection | No photometric occlusion detection |

Table 4.2: **Default configuration for experiments.**

If an entire experiment deviates from these default settings, we list the difference under a separate Configuration subsection. If only some of the presented results or figures deviate from our default settings, we mention the difference locally under the respective figure or table. We do not mention if it is clear from the experimental description that a certain configuration changes, e.g. we do not explicitly state that the landmark representation changes when comparing inverse distance against inverse depth parameterization.

## 4.1 Landmark Parameterization: Inverse Depth vs. Inverse Distance

**Motivation**

It has been shown by [15] that using an *inverse depth* parameterization is advantageous over a standard Euclidean (x, y, z) coordinate parameterization. Points at infinity can be represented by using an inverse depth of zero. It is better suited to represent initial depth estimates from stereo over the whole depth range [15]. Inverse depth is used in DSM [63]. For camera models such as *fisheye lenses*, inverse distance should be used [37] to model points behind the camera. In the default PBA implementation, landmarks are represented using inverse distance, not inverse depth. To see the difference between these two parameterizations, we compare the averaged results of final $ATE_{rmse}$ for both parameterizations. In our implementation, if using an inverse *distance* parameterization, all 3D points of the residual pattern belonging to the same landmark are located on a spherical surface. If using an inverse *depth* parameterization, all 3D points of the same residual pattern are located on a planar surface which is parallel to the camera's xy image plane.

One scenario where the inverse depth could be slightly superior in terms of computational requirements is when a *pinhole camera model* is used. To see why, consider a pinhole camera model which projects the 3D world coordinates $(X, Y, Z)^T$ onto the image coordinates $(u, v)$ [36, p. 52]:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f\,s_x & f\,s_\theta & o_x \\ 0 & f\,s_y & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{K}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \underbrace{\mathbf{T}_{cw} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}_{(X_c, Y_c, Z_c, 1)^T} \tag{4.1}$$

The matrix $\mathbf{K}$ denotes the intrinsic calibration parameters. The rigid body motion from world to camera coordinate system is given by $\mathbf{T}_{cw}$. The parameter $\frac{1}{\lambda}$ represents *inverse depth* in the camera frame. Projecting a known 3D point $(X, Y, Z)^T$ yields the vector $(u\lambda, v\lambda, \lambda)^T$, where the depth can be read from the third component $\lambda$. Conversely, the unprojection of a chosen 2D point $\lambda(u, v, 1)^T$ with estimated depth $\lambda$, is $\lambda\ \mathbf{K}^{-1}\ (u, v, 1)^T$. Using *inverse distance* causes a slight computational overhead. The distance needs to be calculated explicitly with $\sqrt{X_c^2 + Y_c^2 + Z_c^2}$. Unprojecting a 2D image point to 3D is given by $\frac{distance}{\|\mathbf{K}^{-1}(u,v,1)^T\|_2}\mathbf{K}^{-1}(u, v, 1)^T$. It might be useful to represent pixel coordinates with bearing vectors in the inverse distance case, i.e. vectors of unit length. Converting from a $\lambda\ (u, v, 1)^T$ vector to a bearing vector requires a division by its norm.

When working with inverse distance or depth parameterizations, the warp function is

algebraically rearranged to prevent a division by zero in our implementation. For example, consider the case of inverse distance, where the unprojection function $\Pi_c^{-1}(\mathbf{p}, id_p)$ transforms the 2D pixel coordinates $\mathbf{p}$ to a bearing vector $\mathbf{b}$. Subsequently, $\mathbf{b}$ is scaled by its distance $\frac{1}{id_p}$ to yield the 3D point:

$$\mathbf{p}' = \Pi_c(\mathbf{R}_{ji}\Pi_c^{-1}(\mathbf{p}, id_p) + \mathbf{t}_{ji}) = \Pi_c(\mathbf{R}_{ji}\frac{\mathbf{b}}{id_p} + \mathbf{t}_{ji}) \tag{4.2}$$

We now use the fact that $\Pi_c(\mathbf{x}) = \Pi_c(k \cdot \mathbf{x})$, where $k \geq 0$ is a scalar value describing all points along the bearing vector between the target camera's origin and the unprojected 3D point. Multiplying by $id_p$ results in the modified warp function:

$$\mathbf{p}' = \Pi_c(\mathbf{R}_{ji}\mathbf{b} + \mathbf{t}_{ji} \cdot id_p) \tag{4.3}$$

whichs prevents the division by zero. Note that points at infinity are represented by zero inverse distance. The same holds for the case of inverse depth parameterization, where the unprojection function $\Pi_c^{-1}(\mathbf{p})$ transforms the 2D pixel coordinates $\mathbf{p}$ to a $z = 1$ vector $(x', y', 1)^T$ and then scales the vector by its depth value.

**Configuration**
We deviate from the default configuration in Table 4.2 by the following list:

- Solver: ceres-solver
- camera LM dampening: diagonal of Schur complement

**Results**
Even though the results indicate a minor advantage of using the inverse distance parameterization, we cannot observe a systematic advantage. We suspect that the ATE difference might be incidental, due to the different residual pattern geometry in 3D. However, note that this difference of residual pattern geometry becomes negligible if landmarks are far away from the host-target camera pair. We generally recommend to use inverse distance as default parameterization in our solver. If runtime is crucial and inverse depth could improve it, we could also use inverse depth since the ATE difference is quite small.

| | init_odometry | init_pgo | idist | idepth |
|---|---|---|---|---|
| all sequences | 2.131 | 1.0 | **0.675** | 0.684 |

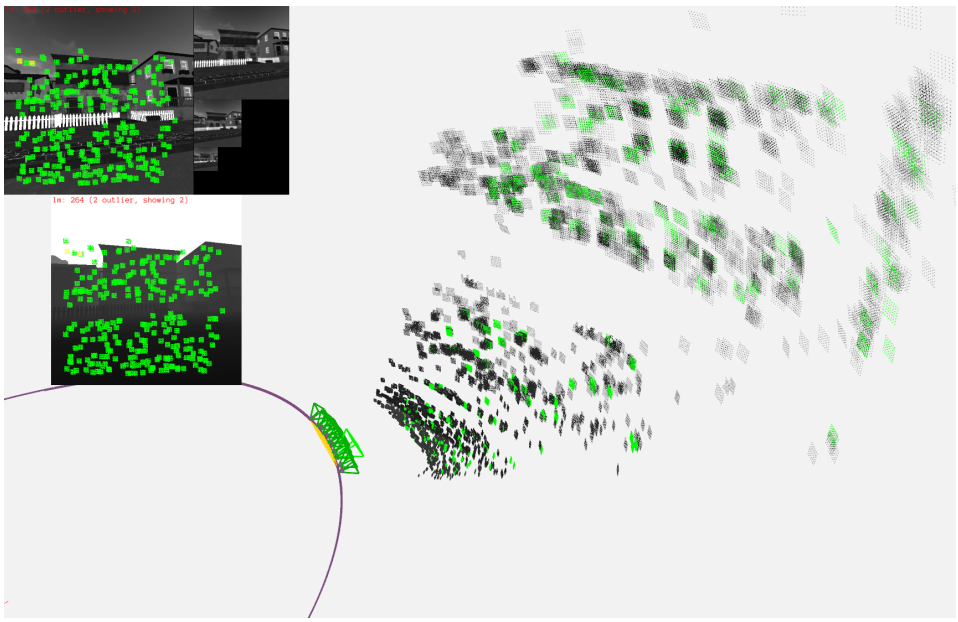
Table 4.3: **Averaged ATE for inverse distance (idist) vs. inverse depth (idepth).** We keep inverse distance as default parameterization because it has slightly lower averaged ATE. Only for an application where the pinhole camera model is used and runtime is crucial, one could equally switch to inverse depth. $ATE_{rmse,geo}$ is relative to init_pgo, geometric mean over all sequences. The smallest (best) ATE is in bold, see Equation 1.3.

Figure 4.1 shows the different residual pattern 3D geometry. The hosted residual patterns in the current frame are displayed in green. The twelve cameras are displayed on the lower left as green frustums. The ATE is quite different for a large residual pattern when switching between inverse distance (0.90 centimeters) and inverse depth (0.54 centimeters). The lower ATE for inverse depth can be explained by the fact that on the selected scene carla_12cam, the cameras are viewing a planar house wall, i.e. a planar residual pattern fits better to the underlying geometry.

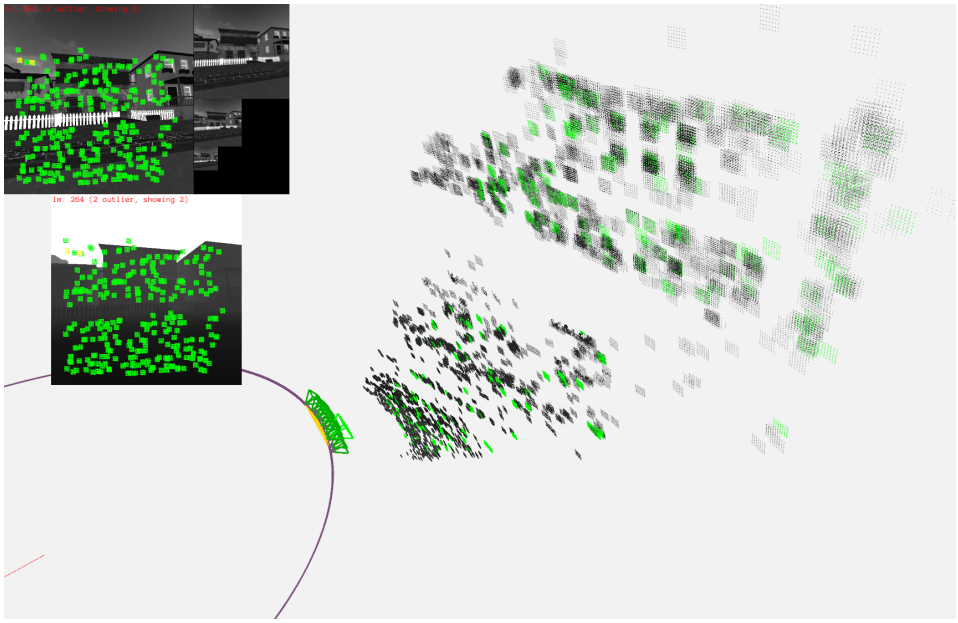
For a smaller residual pattern, the ATE difference between inverse distance and inverse depth is much lower, e.g. 0.45 centimeters and 0.42 centimeters for the DSO pattern. This can be explained by the fact that the difference of unprojected residual pattern geometry, i.e. a planar structure (inverse depth) vs. a spherical structure (inverse distance), becomes negligible if residual pattern are small compared to their distance from the host camera.

**Validation**

With the difference of a spherical vs. a planar 3D residual pattern in mind, we validate our implementation to make sure we do not introduce unintended errors. In particular, we expect to see the same results for both parameterizations if using a residual pattern consisting of only one central pixel. Indeed, we measured that in this case both parameterizations are equal up to $max\_diff = 5 \cdot 10\text{e}{-4}$ meters ATE difference on our *test data* set. We neglect this small numerical difference, which might be mostly due to different normalization strategies of the bearing vectors. As discussed above, we use bearing vectors $\mathbf{b}$ in the case of inverse distance and $z = 1$ vectors $(x', y', 1)^T$ in the case of inverse depth to represent pixel positions in the host image.



(a) Inverse distance - residual patterns are spherical, final ATE is 0.90 centimeter



(b) Inverse depth - residual patterns are planar, final ATE is 0.54 centimeter

Figure 4.1: **Spherical 3D residual pattern (inverse distance) vs. planar residual pattern (inverse depth).** Both plots display the dataset carla_12cam (a subset of carla_circle) after PBA. A residual pattern of size 7x7 is selected (see Figure 4.11) for better visualization. The inverse depth parameterization performs better here because the planar patches fit more accurately to the planar house fronts.

## 4.2 Landmark Parameterization: Normal Vector for Residual Pattern

**Motivation**
When using an inverse distance parameterization, the pixels of the residual pattern are located on a spherical surface centered in the host camera's origin. On the other hand, when using inverse depth parameterization, the unprojected pixels of the residual pattern are located on a planar surface which is parallel to the host camera's image plane. It is not clear which 3D geometry of the residual pattern is preferable, as discussed above. We expect that it is advantageous if residual patterns can exhibit varying, flexible 3D geometry according to the actual scene geometry. To achieve this in a computationally feasible way, we assign a normal vector which can be optimized for each residual pattern independently. The normal vector is perpendicular on the unprojected residual pattern which forms a plane. A normal direction has two DOF, hence at least two parameters are required for its description. We parameterize the normal vectors using the *stereographic projection model* [55, section III B]:

$$\mathbf{n} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \mu a \\ \mu b \\ \mu - 1 \end{pmatrix}, \ \mu = \frac{2}{1 + a^2 + b^2} \tag{4.4}$$

where $(a, b)^T$ are the two parameters representing the normal vector, and $\mathbf{n}^T = (x, y, z)^T$ is a unit-length bearing vector describing the normal vector in 3D space. The central pixel determines the distance $h_{plane}$ of the unprojected residual pattern plane from the camera's origin, given by:

$$h_{plane} = \mathbf{n}^T \mathbf{b}_{center} \frac{1}{id_p} \tag{4.5}$$

In contrast to scaling each bearing vector $\mathbf{b}$ for the pixels of the residual pattern by their common inverse distance parameter $id_p$, we now scale each bearing vector of the residual pattern individually by a distinct distance $h_b$. For each bearing vector, the distance $h_b$ is chosen such that the normal vector $\mathbf{n}$ is perpendicular to the unprojected residual pattern, i.e.:

$$h_b = \frac{h_{plane}}{\mathbf{n}^T \mathbf{b}} = \frac{\mathbf{n}^T \mathbf{b}_{center} \frac{1}{id_p}}{\mathbf{n}^T \mathbf{b}} \tag{4.6}$$

This results in the modified warp function:

$$\mathbf{p}' = \Pi_c(\mathbf{R}_{ji} \mathbf{b} h_b + \mathbf{t}_{ji}) = \Pi_c(\mathbf{R}_{ji} \mathbf{b} \, \mathbf{n}^T \mathbf{b}_{center} + \mathbf{t}_{ji} \, id_p \, \mathbf{n}^T \mathbf{b}) \tag{4.7}$$

where we multiply by $id_p$ to avoid a possible division by $id_p = 0$. Similarly, we multiply by $\mathbf{n}^T\mathbf{b} > 0$, since this expression can be very small when normal and bearing vector are close to orthogonal. In the following, we provide a qualitative analysis using a visualization of the photometric map as well as quantitative results.

**Experiments**
Introducing two parameters $(a, b)^T$ for each landmark increases the number of parameters. This is why we expect that a joint optimization of all camera poses and inverse distances and normals might be computationally very demanding and might not converge. We try the following optimization sequences: $[PBA + normals]$, $[PBA, normals]$ and $[normals, PBA]$, where $PBA$ denotes optimization of all camera poses and inverse distances with fixed normals, $normals$ denotes optimization of all normal parameters $(a, b)^T$ for each landmark with fixed camera poses and fixed inverse distances, and $PBA + normals$ denotes the joint optimization of all camera poses, inverse distances and normal parameters. For example, in $[normals, PBA]$, we first solely optimize normals and subsequently we perform PBA with fixed normals. In addition to the listed sequences, we have additionally tried other sequence orders but did not find major differences among them. In the following, we give a motivation for the chosen sequences:

1. $[PBA + normals]$: This sequence is used to show whether joint optimization is feasible. We compare this against the other sequences in terms of ATE.

2. $[PBA, normals]$: This sequence is used to test whether it is advantageous to post-process the normal vectors. In a real application, this could make the final point cloud better initialized for further processing steps, i.e. to generate dense surface reconstructions or to perform path planning. Since ATE is not affected by the subsequent normal refinement, we provide a qualitative comparison.

3. $[normals, PBA]$: This sequence is used to determine whether PBA can profit from having the residual patterns aligned as pre-processing step. In addition to qualitative comparison, we compare it in terms of ATE to the other sequences.

For larger residual patterns, the normal vector alignment is expected to work better, because there is more information determining the orientation. Therefore, we also experiment with larger pattern sizes on the test datasets. In the following, we initialize the normal vectors to be pointing away from the host frame, i.e. in $(0, 0, 1)^T$ direction by setting $a = b = 0$, except stated otherwise.

**Configuration**
We deviate from the default configuration in Table 4.2 by the following list:

- Solver: ceres-solver

**Results**

The initialization in Figure 4.2 shows how the patches are aligned parallel to the cameras image plane for $a = b = 0$. Qualitative results on carla_12cam and carla_circle in Figure 4.3, 4.4 and 4.5 confirm that our normal optimization does help to align the residual patterns with the underlying (planar) geometry, i.e. with the walls of the captured house and the road. The smaller dataset carla_12cam is a subset of carla_circle. The segment of carla_circle displayed in Figure 4.5 contains the same frontal house wall as captured in carla_12cam. Figure 4.4 shows the results for a larger patch, where the normal optimization seems work better for the alignment with the house wall. This can be explained by the fact, that the larger patch provides more information for its normal direction estimation. On the test dataset, we could notice no qualitative difference between $[PBA + normals]$, $[PBA, normals]$ and $[normals, PBA]$ in terms of map quality.

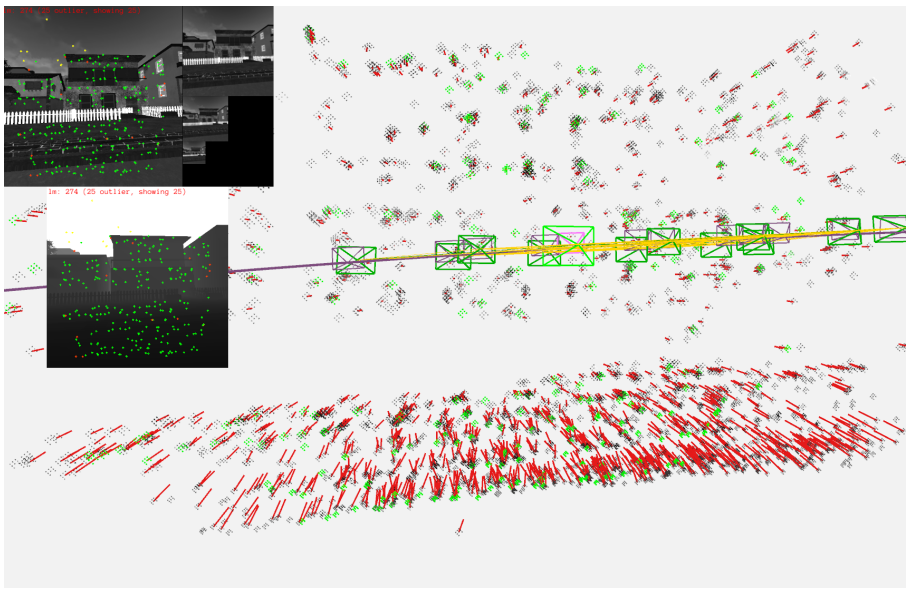
For all figures in this section, we use the following conventions: Normal vectors are displayed in red, hosted points in the current frame are highlighted in green. Unprojected landmarks contain the gray scale value as captured in the image, i.e. they are mostly gray and black. Outlier landmarks are displayed in yellow. Keyframe camera poses are displayed by the green camera frustums, the current active frame is visualized by an enlarged, light-green camera frustum. The ground-truth trajectory is displayed in purple. Estimated camera clusters are highlighted in yellow. The image on the top left corner shows the current camera view (enlarged, light-green camera frustum). On the right hand side of the camera view, the *image pyramid* is displayed. The image below the current camera view shows the ground-truth depth map, where lighter values represent distant and darker values represent close objects.

Table 4.4 shows the average ATE results for normal vector alignment. The ATE in column $[PBA, normals]$ is the same as for $[PBA]$ because the poses are fixed during post-processing of the normals. In the case of aligning the normals before PBA, the ATE degrades quite a lot on average. This is mainly due to a number of datasets where $[normals, PBA]$ performs significantly worse. Interestingly, the optimization $[normals, PBA]$ also performs better on a few sequences, namely Kitti00, Kitti02, Kitti05 and Kitt06. These sequences happen to be all part of kit-loop. However, $[normals, PBA]$ performs quite worse on euroc-ok than the others, even though it contains loop closures as well. By looking at the visualized map, we could not find a definite reason why $[normals, PBA]$ outperforms the other methods on these sequences. The joint optimization $[PBA + normals]$ performs worse than $[PBA, normals]$ on all sequences. This might be explained by the fact that for $[PBA + normals]$ the numbers of parameters is too large which makes the system less determined, especially for small patterns. Hence, the optimization is more likely to get stuck in local minima.

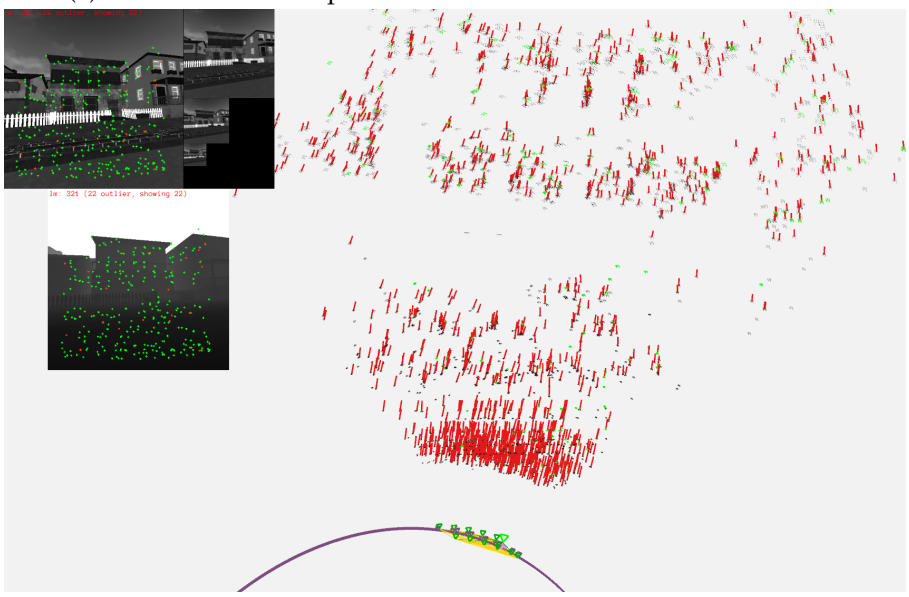
| | $[PBA, normals]$ | $[normals, PBA]$ | $[PBA + normals]$ |
|---|---|---|---|
| all sequences | **0.672** | 0.762 | 0.731 |
| euroc-ok | **0.668** | 0.837 | 0.707 |
| euroc-fail | **1.001** | 1.000 | 1.056 |
| kit-no-loop | **0.743** | 0.869 | 0.849 |
| kit-loop | 0.540 | **0.496** | 0.542 |

Table 4.4: **Averaged ATE results for normal vector optimization.** Joint optimization $[normals, PBA]$ as well as $[PBA + normals]$ yield worse ATE, hence $[PBA, normals]$ should be used. All three options qualitatively yield similar normal vectors on our test dataset. $ATE_{rmse,geo}$ is relative to init_pgo, geometric mean over respective sequences. The joint optimization $[PBA + normals]$ is without Kitti00 because the memory requirement is too high. The smallest (best) ATE is in bold.

There is no major difference among the strategies in terms of runtime, i.e. overall runtime is very similar for $[PBA, normals]$, $[normals, PBA]$ or $[PBA + normals]$. The total runtime for $[PBA, normals]$ increases by approximatively 80 percent compared to $[PBA]$ without normal optimization. The qualitative normal vector orientation relative to the structure is similar for all presented optimization modes for our test data set. As a conclusion, it is advisable to use $[PBA, normals]$ since it achieves the lowest ATE and the normal optimization is qualitatively similar compared to the other methods on the test dataset. In constrast to the other methods, $[PBA, normals]$ allows to actually only perform the normal optimization if it is really desired.

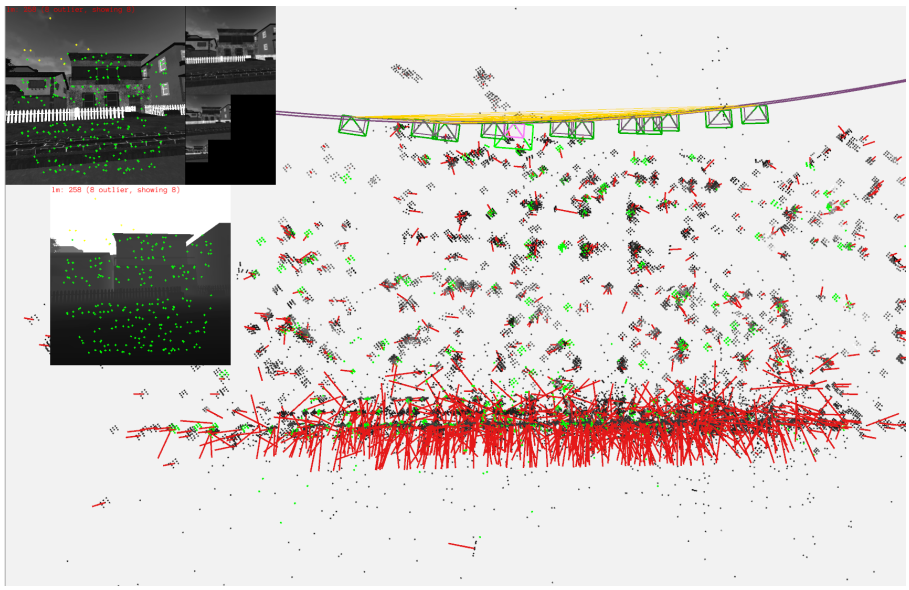


(a) Initialization DSO pattern - front view of house wall and road

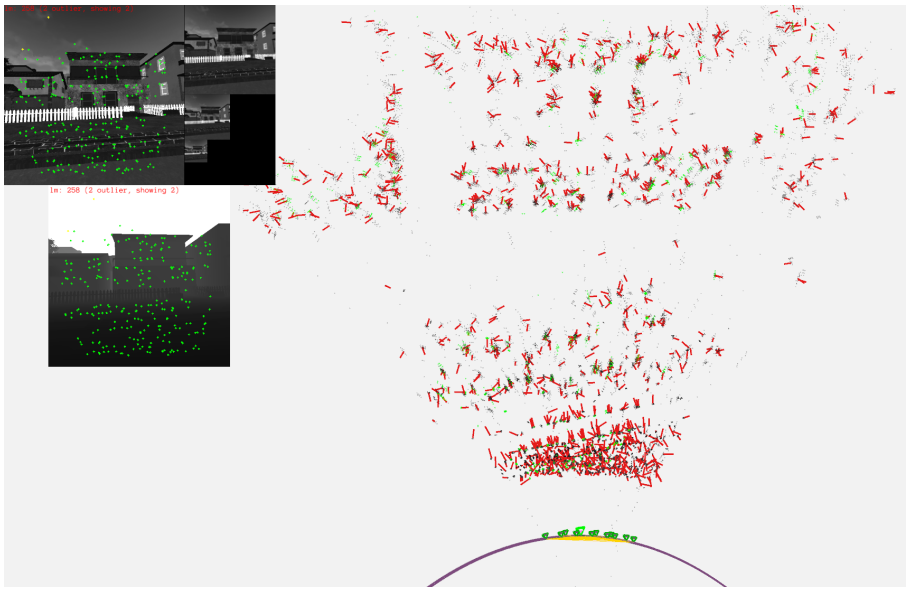


(b) Initialization DSO pattern - top view of house wall and road

Figure 4.2: **Front and top view after initialization on carla_12cam for DSO pattern.** The normals are initialized with $a = b = 0$ such that they point away from the host frame. The unprojected residual pattern is parallel to the host camera's image plane. Red: normal vectors. Green: hosted point in current frame. Gray-Black landmarks: Unprojected landmarks contain the gray scale value as captured in the original image.

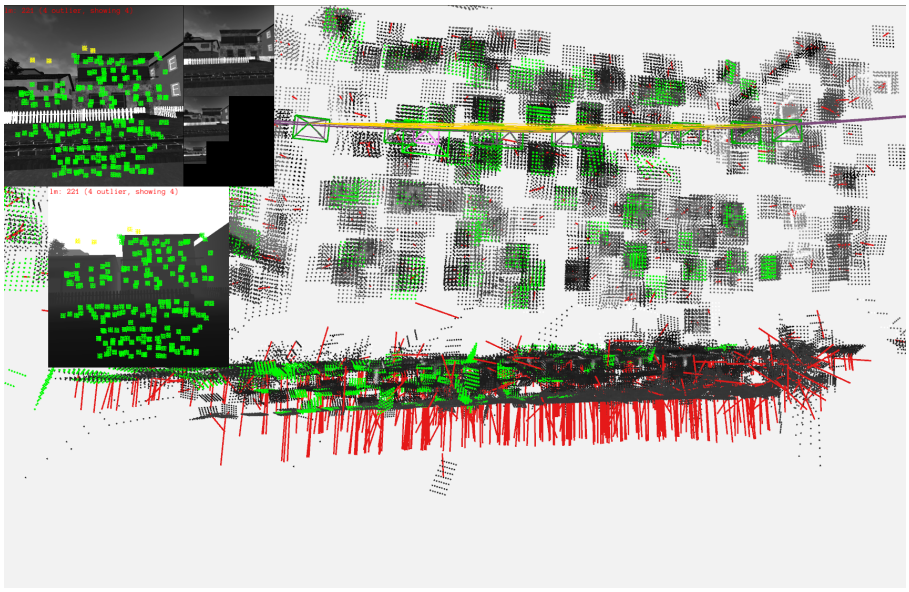


(a) $[PBA + normals]$ - front view of house wall and road

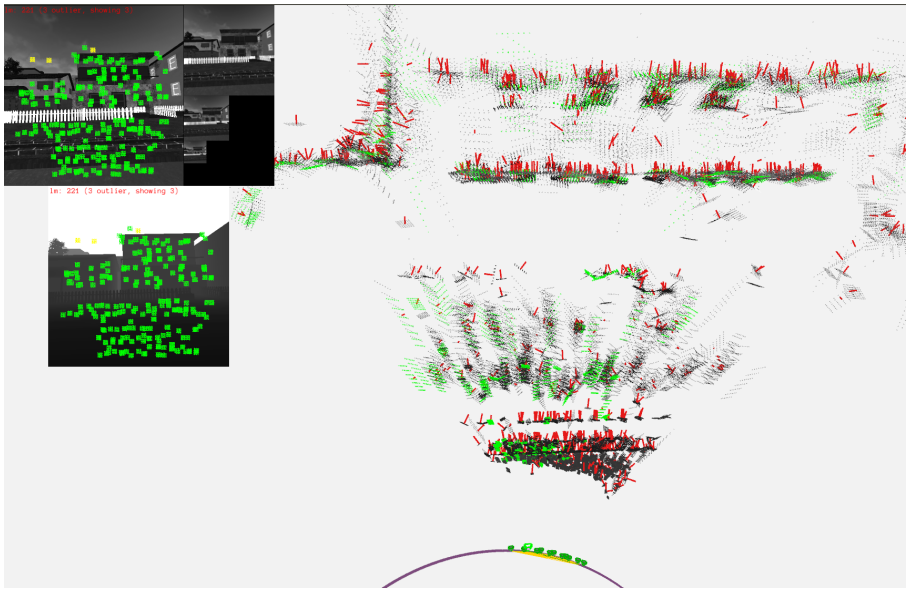


(b) $[PBA + normals]$ - top view of house wall and road

Figure 4.3: **Front and top view after PBA+normals on carla_12cam for DSO pattern.** The normals align very well with the road. The alignment with the house wall is not perfectly consistent, i.e. a large number of normal are not perpendicular to the underlying wall. The other presented optimization sequences, i.e. $[PBA, normals]$ and $[normals, PBA]$, look similar on the test dataset.

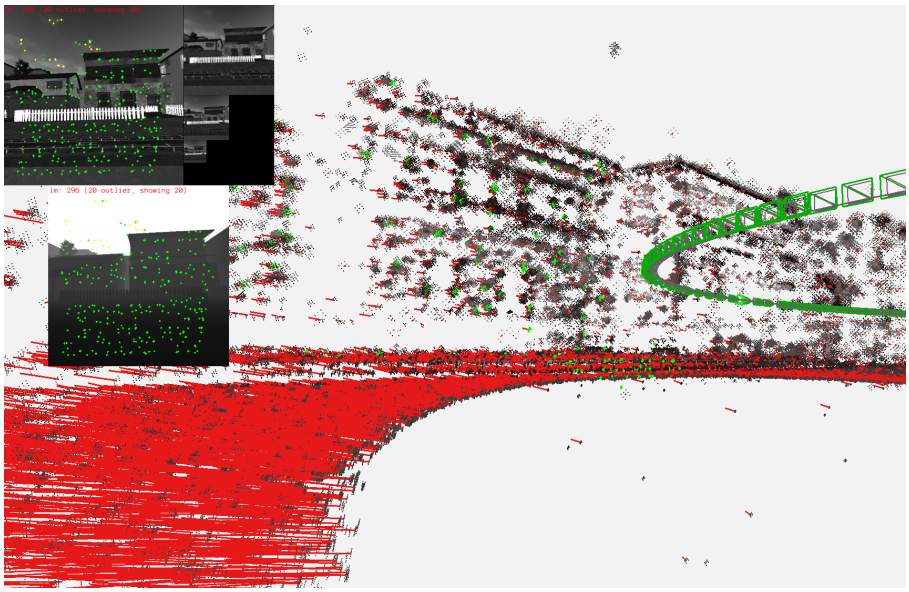


(a) $[PBA, normals]$ - front view of house wall and road

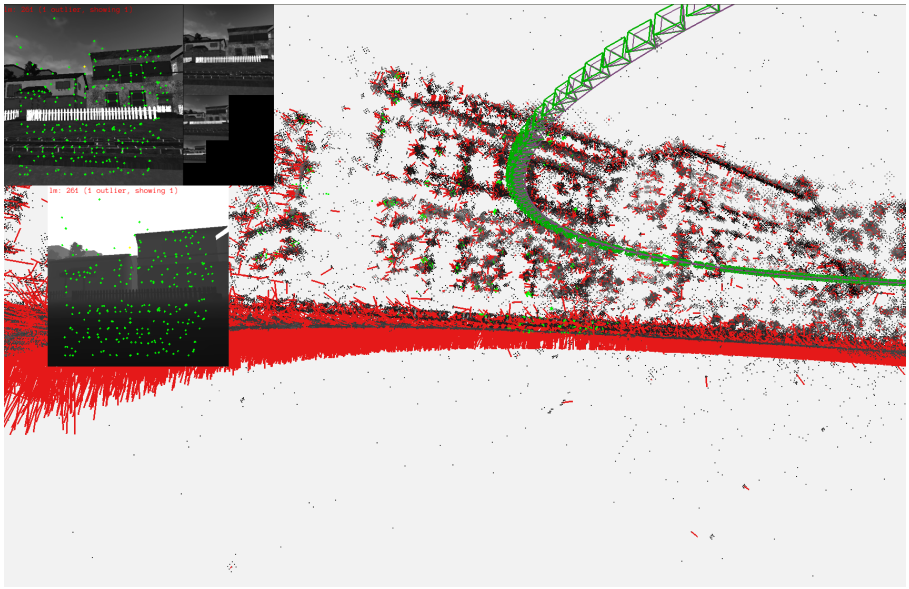


(b) $[PBA, normals]$ - top view of house wall and road

Figure 4.4: **Front and top view after PBA, normals on carla_12cam for a sparse 7x7 pattern.** Normal vectors are aligned more consistently with the house wall compared to Figure 4.3 b). This might be related to the fact that a larger patch provides more information for its normal estimation. The sparse 7x7 pattern is visualized in Figure 4.11. Results for $[PBA + normals]$ and $[normals, PBA]$ are not shown because they are visually similar.



(a) Initialization with $a = b = 0$ - front view of same house as in carla_12cam



(b) $[PBA + normals]$ - front view of same house as in carla_12cam

Figure 4.5: **Front view of initialization and after PBA+normals on carla_circle using DSO pattern.** Compared to carla_12cam, the normal optimization yields qualitatively similar results on the larger data set carla_circle. The displayed segment of carla_circle shows the same frontal house wall as in carla_12cam.

From Figure 4.5 to Figure 4.9, we show the alignment for init_pgo, $[PBA + normals]$, $[PBA, normals]$ and $[normals, PBA]$ on a few selected sequences. The aligned 3D trajectories are projected onto the xy-plane of the world coordinate system and visualized as 2D plot. On eurocV102 and eurocMH03, the alignment is qualitatively similar among the tested normal optimization sequences, see Figure 4.5 and 4.6. For the datasets eurocV202, kitti02 and kitti07, the trajectory is significantly better aligned using $[PBA, normals]$ compared to $[PBA + normals]$ or $[normals, PBA]$. This qualitative difference is also reflected by the ATE, displayed in Table 4.4. Therefore, after analyzing the trajectory alignments qualitatively, we still recommend to use $[PBA, normals]$. It performs mostly better or just slightly worse than the others, whereas $[PBA + normals]$ or $[normals, PBA]$ often perform significantly worse and only sometimes slightly better.

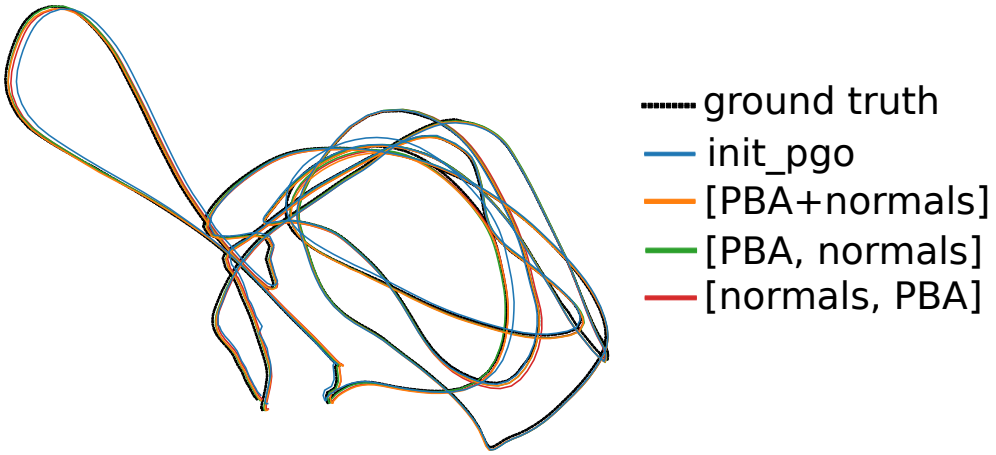



Table 4.5: **Aligned trajectories for the proposed normal vector optimization sequences on eurocMH03.** There is no large difference between the different normal vector optimization modes on eurocMH03. The trajectories are aligned to ground truth and their 2D projection onto the xy-world plane is shown.

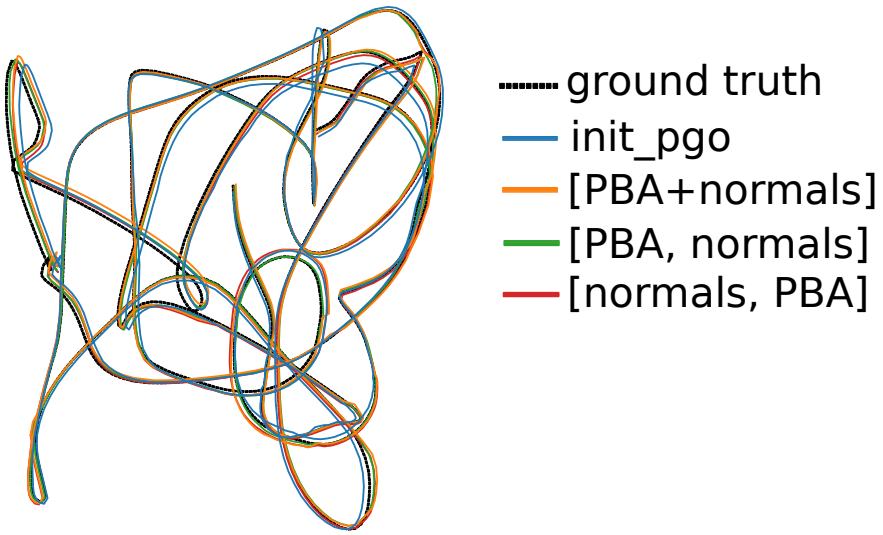



Table 4.6: **Aligned trajectories for the propose normal vector optimization sequences on eurocV102.** There is no large difference between the different normal vector optimization modes on eurocV102. The trajectories are aligned relative to ground truth and the 2D projection onto the xy-world plane is shown.

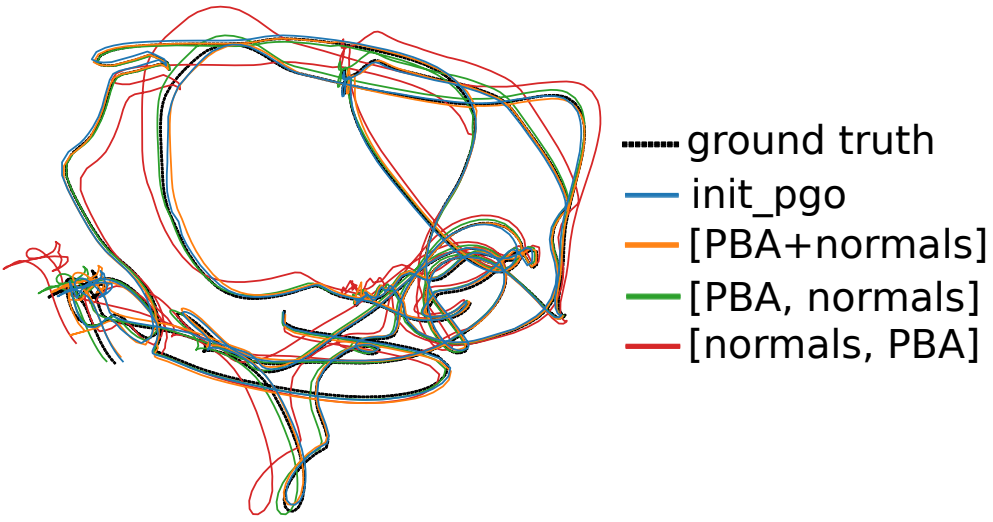



Table 4.7: **Aligned trajectories for the propose normal vector optimization sequences on eurocV202.** $[normals, PBA]$ performs the worst on eurocV202; $[PBA + normals]$ performs slightly worse than $[PBA, normals]$. The trajectories are aligned relative to ground truth and the 2D projection onto the xy-world plane is shown.

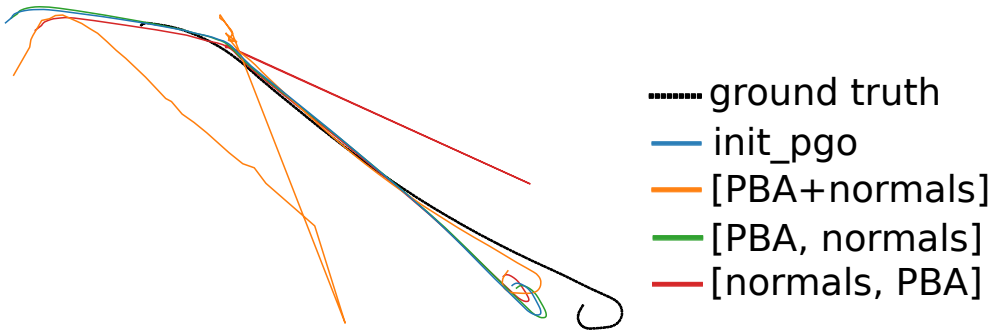



Table 4.8: **Aligned trajectories for the propose normal vector optimization sequences on kitti02.** The joint optimization $[PBA + normals]$ and $[normals, PBA]$ both fail on Kitti02, whereas $[PBA, normals]$ is similar to init_pgo. The trajectories are aligned relative to ground truth and the 2D projection onto the xy-world plane is shown.

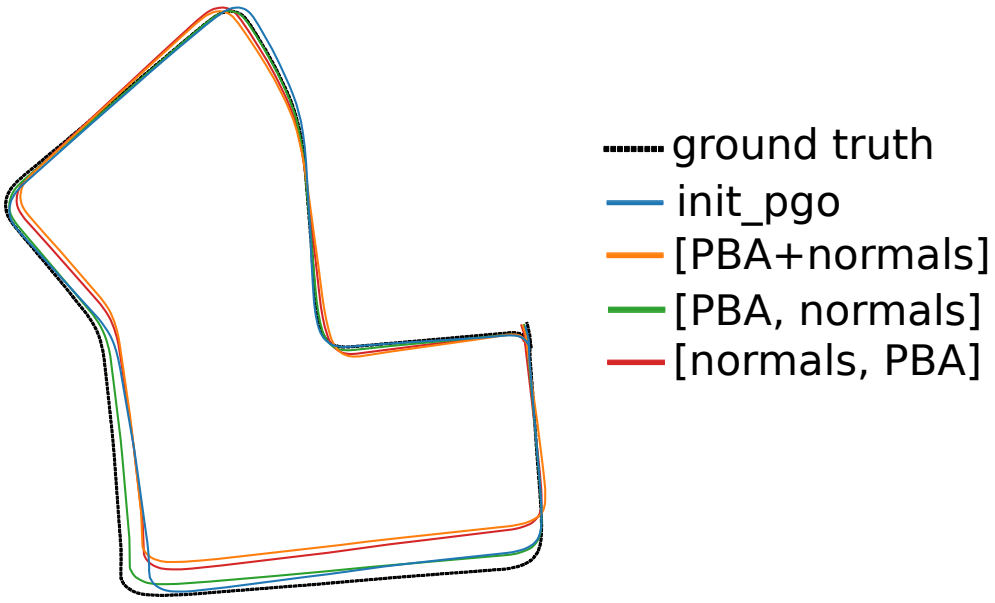



Table 4.9: **Aligned trajectories for the propose normal vector optimization sequences on kitti07.** The joint optimization $[PBA + normals]$ and $[normals, PBA]$ both perform slighlty worse than $[PBA, normals]$ on kitti07. The trajectories are aligned relative to ground truth and the 2D projection onto the xy-world plane is shown.

**Validation**

We perform two experiments to validate our normal vector implementation: In the first validation experiment, all normal vectors are initialized to $(0, 0, 1)^T$ by setting

$a = b = 0$, so that the residual pattern is parallel to the image plane for each hosted landmark. We perform PBA with constant normals. We expect that the final result must be exactly equal to the case of an inverse depth parameterization, when no patch normals are used. Indeed, we can find no measurable difference between the inverse depth parameterization and the inverse distance parameterization with $a = b = 0$ on our test dataset. Additionally, we can visually confirm that the residual patterns in 3D are planar, see Figure 4.3.

In the second validation experiment, we initialize the normals parallel to the bearing vectors of the hosted 2D points. We use inverse distance parameterization and perform PBA with constant normals. The results are expected to be close to an inverse distance parameterization without normals, i.e. there should be only a minor change of final ATE. Indeed, we found only minor numerical differences in the last decimal places of the ATE on our test dataset. We can visually confirm that in this case, the unprojected residual patterns are planar surfaces, which are normal to their bearing vector from the host camera, see Figure 4.6.

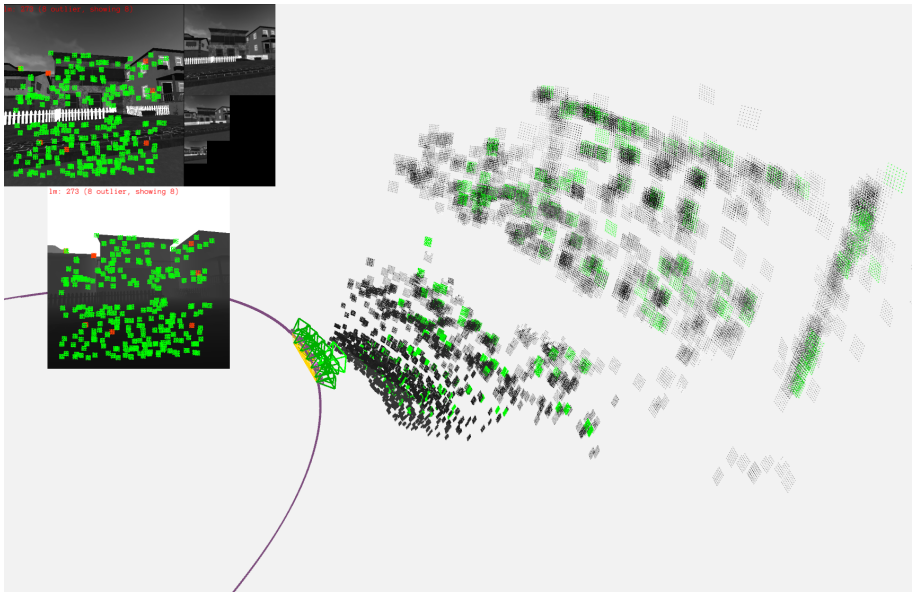


Figure 4.6: **Initializing the normal vectors of the patches parallel to their bearing vector in inverse distance parameterization on carla_12cam as validation.** We perform this initialization to validate our implementation. As expected, the structure resembles a mixture of inverse distance and inverse depth parameterization. The planar patches are headed towards the cameras. Green: hosted point in current frame. Gray-Black: Unprojected landmarks contain the gray scale value as captured in the image

## 4.3 Host - Target Frame: Interpolation Schemes in Target Images

**Motivation**
The points $\mathbf{p}$ in the host image are selected at constant pixel positions. The warp function which transforms host points $\mathbf{p}$ to target points $\mathbf{p}'$ is given by:

$$\mathbf{p}' = \Pi_c(\mathbf{R}_{ji}\Pi_c^{-1}(\mathbf{p}, id_p) + \mathbf{t}_{ji}) = \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{4.8}$$

Most target points $\mathbf{p}'$ end up at subpixel positions. Hence, for the evaluation of the photometric residuals, we need to compute an interpolated (estimated) image value $I(\mathbf{p}')$. Similarly, for the evaluation of the Jacobians, we need to interpolate the image gradient $\nabla I(\mathbf{p}') = [I_x(\mathbf{p}'), I_y(\mathbf{p}')]^T$ at those subpixel positions (see Section 4.5). If the interpolation mask around $\mathbf{p}'$ exceeds the image borders, we implement a *clamping strategy* which uses the closest pixel of the image border as pixel value for the out-of-bounds pixels. In many SLAM/VO systems, so-called *bilinear interpolation* is used due to its simplicity, e.g. in DSO [21] or in the work of Alismail et al. [5]. In constrast to that, the default interpolation scheme in the ceres-solver library is *bicubic interpolation* with exact derivatives, which we denote as *bicubic_exact* in our experiment.

**Experiments**
We compare four different combinations of image value and image gradient interpolation in terms of final ATE: *bilinear_exact*, *bilinear_smooth*, *bicubic_exact*, *bicubic_smooth*, where the first word denotes the image value interpolation and the second word denotes the image gradient interpolation. The methods with suffix *local* stand for a gradient interpolation which takes into account the same pixel neighborhood as the corresponding image value interpolation. Bilinear_exact computes the gradient by forward difference between intermediate linearly interpolated values; bicubic_exact computes the gradient by evaluating the exact derivatives of intermediate interpolated cubic spline functions, see the explanations below. The methods with suffix *smooth* refer to a gradient interpolation which takes into account a larger neighborhood of pixels. They first compute a gradient image in the x as well as in the y-direction by central differences. In the case of bilinear_smooth, this gradient image is then interpolated with bilinear image value interpolation to obtain the gradient value; in the case of bicubic_smooth, the gradient image is interpolated with bicubic interpolation. In order to perform bilinear or bicubic interpolation once, we take into account a neighborhood of 4 and 16 pixels, respectively. The gradient interpolation bilinear_smooth and bicubic_smooth take 12 and 32 pixels into account.

Bilinear interpolation is illustrated in Figure 4.7. To explain bilinear interpolation, we let $ix = int\lfloor x' \rfloor$, $iy = int\lfloor y' \rfloor$, where $int\lfloor x \rfloor$ takes a positive real number $x$ and outputs the closest integer smaller or equal to $x$. Furthermore, we set $dx = x - ix$, $dy = y - iy$ and $ddx = 1 - dx$, $ddy = 1 - dy$.

$$\begin{aligned} I_{bilin}(\mathbf{p}') = ddx \cdot ddy \cdot I(ix, iy) + ddx \cdot dy \cdot I(ix, iy+1) + \\ dx \cdot ddy \cdot I(ix+1, iy) + dx \cdot dy \cdot I(ix+1, iy+1) \end{aligned} \tag{4.9}$$

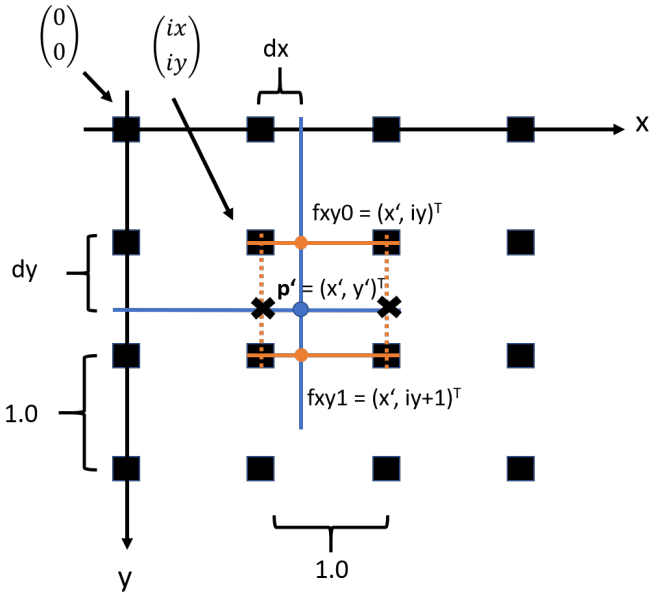



Figure 4.7: **Bilinear interpolation illustrated.** Black squares represent integer pixel coordinates. Blue lines represent subpixel positions of the target point $\mathbf{p}'$. Orange lines represent the intermediate interpolation along the x-direction first. Four pixel values are required.

The bilinear interpolation process can be interpreted as follows:

1. At first, we interpolate along the x-direction at $dy = 0$ and $dy = 1$ separately, which is illustrated by the orange lines in Figure 4.7. We assume a linear image function along these orange lines, with start and end point given by the actual image values. This linear image along the upper orange line is described by $I(x', iy) = ddx \cdot I(ix, iy) + dx \cdot I(ix+1, iy)$. Note that $0 <= dx <= 1$ and $ddx = 1 - dx$. We attain two intermediate interpolated values at $(x', iy)^T$ and $(x', iy+1)^T$, represented by two orange circles in Figure 4.7.

2. To obtain the image value, we linearly interpolate between $(x', iy)^T$ and $(x', iy + 1)^T$, i.e. we set $I(\mathbf{p}') = ddy \cdot I(x', iy) + dy \cdot I(x', iy + 1)$.

3. [***Only*** *in the case of bilinear_exact*] In the case of bilinear_exact gradient interpolation, we use *forward differences*. To obtain the gradient in y-direction, we set $I_y(\mathbf{p}') = I(x', iy + 1) - I(x', iy)$. For the gradient in x-direction $I_x$, we first perform the same linear interpolation as in (1) along the y-direction to obtain intermediate image values, denoted by the black crosses in Figure 4.7. Then, we calculate the forward difference between these intermediate interpolated values.

Performing this process first along the y-axis for $dx = 0$ and $dx = 1$ and then along the x-axis yields the same result. The main advantage of bilinear interpolation is that it is fast to compute since only 4 pixels are accessed and few multiplications and additions are required.

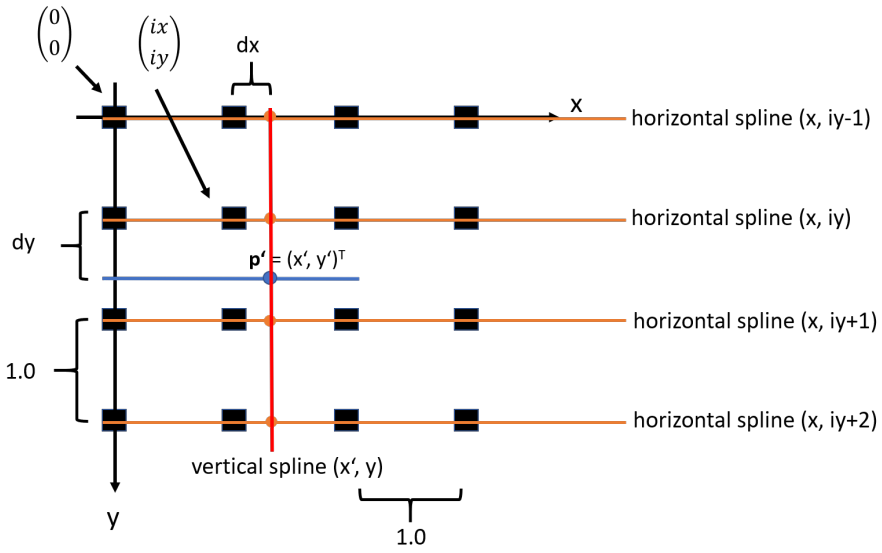



Figure 4.8: **Bicubic interpolation using splines illustrated.** Black squares represent integer pixel coordinates. The blue and red line represent subpixel positions of the target point $\mathbf{p}'$. Orange lines represent the intermediate splines along the x-direction, which are used to obtain image values and image derivatives along the x-direction at $(x', Y_{splines})$. The red line shows the vertical spline along the y-direction. 16 pixel values are required.

Similar to the ceres-solver, we implement the bicubic convolution algorithm in [29] to perform bicubic interpolation. There is a similar, multi-stage process which describes this interpolation:

1. At first, we fit 1D cubic spline functions [30, p. 820-825] horizontally along the x-direction for all y-values in the set $Y_{splines} = \{iy-1, iy, iy+1, iy+2\}$. We evaluate the fitted spline functions to attain intermediate image values at $(x', Y_{splines})$, illustrated by the four orange circles in Figure 4.8.
   [***Only** in the case of bicubic_exact*] In the case of bicubic_exact image gradient, we evaluate the exact image derivative along the x-direction $I_x(x', Y_{splines})$ at each orange circle by taking the analytical derivative of the spline functions.

2. We fit another vertical spline along the y-direction using the image values at $(x', Y_{splines})$, illustrated in red in Figure 4.8. This spline can be evaluated at $\mathbf{p}'$ to read off the image value $I_{bicubic}(\mathbf{p}')$.
   [***Only** in the case of bicubic_exact*] In the case of bicibuc_exact image gradient, the image derivative in y-direction $I_y(\mathbf{p}')$ is obtained by the derivative of this red spline equation.

3. [***Only** in the case of bicubic_exact*] We fit another vertical spline along the red line. This vertical spline is constructed from the derivative values $I_x(x', Y_{splines})^T$ along the x-direction. The spline value at $(x', y')^T$ is used as image derivative $I_x(\mathbf{p}')$.

The above procedure is equivalent to performing a convolution with a separable filter as shown in [29]. Therefore, if we first perform vertical interpolation in step (1) and then horizontal interpolation in step (2), we obtain the same result, which is experimentally validated in our unit tests.

**Results**

The smooth gradient interpolations (green, brown, purple) decrease the cost and ATE faster than exact gradients for EurocV101, as can be seen in Figure 4.9. This might be due to the fact that the gradient is composed of more neighboring pixels for the smooth interpolations. The difference in convergence behaviour between our manual solver and the ceres-solver implementation becomes quite small when the same interpolation method is used for both. Note that while the smooth interpolation decreases ATE faster on eurocV101, all schemes converge to similar final ATE. The final photometric costs differ slightly and they are the highest for exact derivative computations.

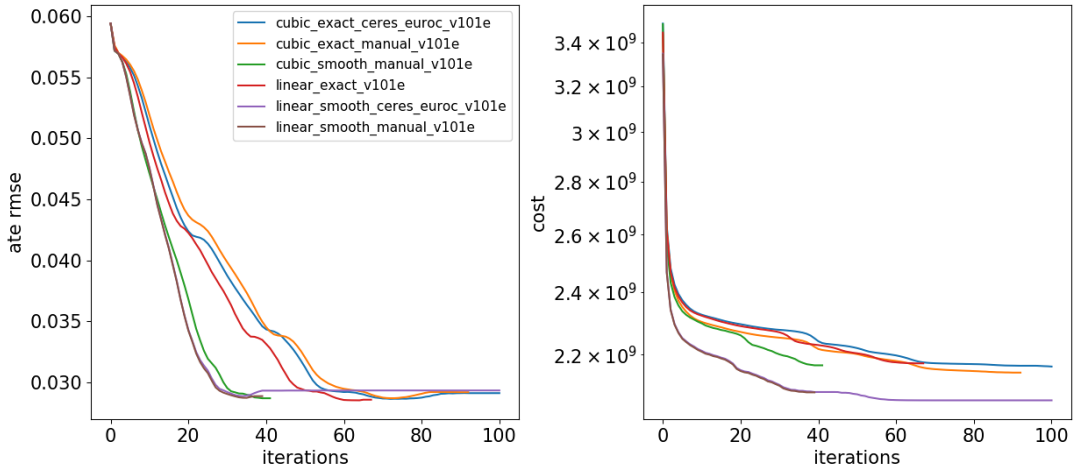



Figure 4.9: **Smooth interpolation results in faster convergence on Euroc V101.** On the left we see ATE over solver iterations. On the right, we see the total photometric cost over solver iterations (see Equation 1.1). When we perform *smooth* interpolation, we can achieve faster convergence compared to the exact interpolation. The difference in convergence behaviour between the manual solver and the ceres-solver implementation becomes very small when using the same interpolation method.

In Figure 4.10, it can be seen that our *smooth* interpolation schemes introduced above behave similarly in terms of ATE as interpolating exactly but on one pyramid level higher. This may be due to the fact that both the image pyramid and our smooth gradient interpolation take additional neighboring pixels into account. If we interpolate smoothly but on one pyramid level above, even more smoothing is introduced. An example is shown in in Figure 4.10, where the brown curve arises by interpolating the gradient with *bilinear_smooth* on pyramid level one. In this case, the ATE decreases the fastest; however, final ATE is higher. This can be explained by the fact that too many neighboring pixels are taken into acoount in this case and the gradient becomes less distinct.

The faster convergence of our manual solver compared to our ceres-solver implementation can be largely explained by the different interpolation schemes which are used. We assume that the remaining difference between the two implementations in terms of convergence is mainly due to numerical differences, which for example might arise from different evaluation orders, slightly different implementation details for the linear solver and different updating strategies of the dampening parameter $\lambda$ for LM.

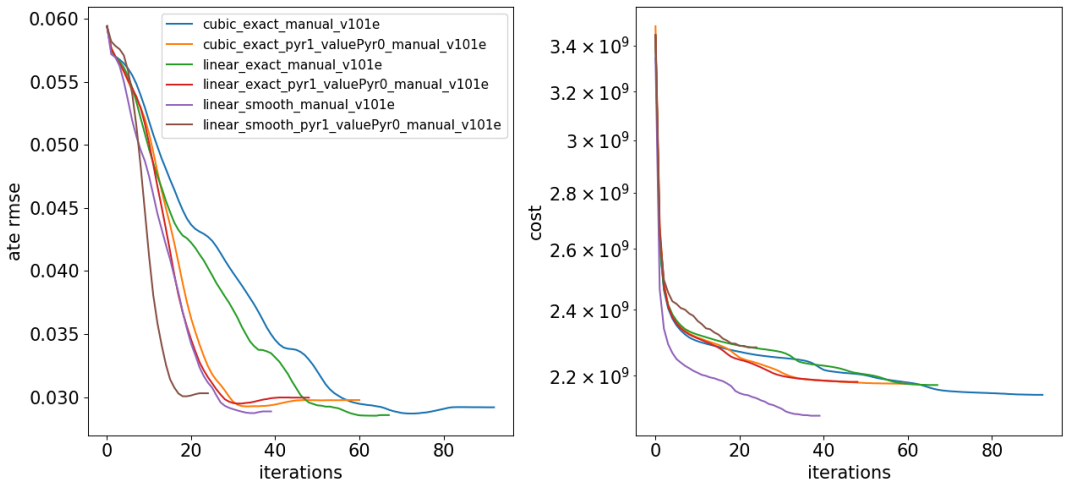



Figure 4.10: **Smooth interpolation behaves similar as interpolating on image pyramid on EurocV101.** In the left plot, we see ATE over solver iterations. In the right plot, we see the total photometric cost over solver iterations. Performing smooth image gradient interpolation on pyramid level 1 (brown curve) converges even faster; however, the final ATE is higher as for the other methods. The image pyramid is built using a convolution with a 5x5 Gaussian kernel, reducing both the image width and the image height by a factor of $\frac{1}{2}$ for each pyramid level.

Note that the reduction of cost and ATE is correlated, but they do not deterministically follow the same trajectories. Similar to performing PBA on an image pyramid which has been implemented in [45], we propose to initially use our smooth interpolation schemes and then switching back to the exact interpolations later. The results for using *bicubic_smooth* for the first 20 iterations and switching back to *bicubic* for the resulting 80 iterations is displayed in Table 4.10, rightmost column. This gives the most accurate results compared to keeping the same interpolation scheme for all iterations. A very similar final ATE is achieved if the first 40 iterations are used to interpolate smoothly. Choosing between bilinear and bicubic interpolation for the first 20 (or 40) iterations also does not influence the ATE significantly. We recommend to use bicubic_sm@20It as default interpolation method for PBA since it achieves the lowest ATE and converges faster than the exact methods.

| | bilin_ex | bilin_sm | bicub_ex | bicub_sm | bicub_sm@20It |
|---|---|---|---|---|---|
| all sequences | 0.671 | 0.673 | 0.669 | 0.666 | **0.657** |

Table 4.10: **Comparison of four interpolation methods.** bicubic_sm@20It denotes interpolation of bicubic_smooth (bilin_sm) for the first 20 iterations and bicubic_exact (bicub_ex) for the 80 remaining iterations. This is the interpolation method which we recommend for PBA because it achieves lowest ATE and converges quicker than the exact gradient computations. Very similar results are achieved when bicubic_smooth is used for the first 40 iterations or if we use bilinear_smooth instead of bicubic_smooth for the first 20 or 40 iterations. Bold numbers show the best configuration (lowest ATE). $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

**Validation**

We created unit tests to show equivalence of our bicubic interpolation and the ceres-solver interpolation method on three arbitrary test images. By performing these unit tests, we have confirmed that the difference between either implementation is at most 10e$-$6 for the image value as well as image gradient components. Furthermore, we confirmed that bilinear interpolation yields similar interpolated image values compared to bicubic interpolation. We also showed that a numeric image derivative $\nabla_{num} I[\mathbf{p}]$ of interpolated image values yields a similar gradient as the gradient $\nabla_{cub} I[\mathbf{p}]$ obtained with the bicubic_exact method. The numeric derivative is evaluated by central differences of the interpolated image values around the pixel of interest $\mathbf{p} = (u, v)^T$ using bicubic image value interpolation. For example, in the x-direction, we compute $\nabla_{cub,x} I[(u, v)^T] = 0.5\, I[(u + \varepsilon, v)^T] - I[(u - \varepsilon, v)^T] \,/\, (2\,\varepsilon)$. The stepsize is set to $\varepsilon =$ 10e$-$5. We require that $\|\nabla_{cub} I[\mathbf{p}] - \nabla_{num} I[\mathbf{p}]\|_2 \leq 0.01$ for each tested pixel $\mathbf{p}$ in the unit test.

## 4.4 Host - Target Frame: Residual Formulations

**Motivation**

In Equation 1.2, the residuals are formulated as in DSO [21]. In this section, we present and compare other common residual formulations. The residual arises from a pixel comparison between host frame $i$ and target frame $j$. For most cases, we use the residual $\mathbf{r}_{ssd}$ is this thesis if not otherwise stated. SSD abbreviates sum of squared differences. It does not account for exposure times, vignetting or illumination differences between the vantage points of frame $i$ and $j$. For one reprojected residual pattern, we have $N_p$ scalar

residual values:

$$\mathbf{r}_{ssd}^{(k)} = I_j[\mathbf{p}'_k] - I_i[\mathbf{p}_k],\ k = 1...N_p \tag{4.10}$$

where $\mathbf{p}_k = \mathbf{p}_{center} + \mathbf{u}_k$ with the offset vector $\mathbf{u}_k$ and $\mathbf{p}'_k = w(\mathbf{p}_k, id_p, \mathbf{T}_{ij})$ is given by the warp function. In this thesis, we define $\mathbf{u}_1 = (0,0)^T$, i.e. the center point is the first pixel of the residual pattern.

The residual $\mathbf{r}_{ab}$ uses the affine brightness transform parameters to correct for unknown exposure time changes, thus increasing the parameter space.

$$\mathbf{r}_{ab}^{(k)} = (I_j[\mathbf{p}'_k] - b_j) - \frac{e^{a_j}}{e^{a_i}}(I_i[\mathbf{p}_k] - b_i) \tag{4.11}$$

The residual $\mathbf{r}_{lssd}$ takes into account the arithmetic mean of all pixel values in the residual pattern. We denote the arithmetic mean of all pixel values in the host patch by $\bar{\mathbf{I}}_i$, in the target patch by $\bar{\mathbf{I}}_j$. By including the factor $\bar{\mathbf{I}}_j/\bar{\mathbf{I}}_i$, any multiplicative changes in the host or in the target patch brightness are neutralized. In contrast to $\mathbf{r}_{ab}$, where the brightness transform paramters $(a_i, a_j, b_i, b_j)$ apply to the whole image, these multiplicative changes are taken care of independently for each patch in $\mathbf{r}_{lssd}$. The reason for multiplicative brightness changes might be different exposure times for host and target image or non-uniform as well as changing scene illumination.

$$\mathbf{r}_{lssd}^{(k)} = I_j[\mathbf{p}'_k] - \frac{\bar{\mathbf{I}}_j}{\bar{\mathbf{I}}_i} I_i[\mathbf{p}_k] \tag{4.12}$$

Similarly, the residual $\mathbf{r}_{lnssd}$ [55] normalizes the pixel intensities by the mean of the residual pattern to account for lighting variations between host and target patch. Note that the other residuals measure a difference in pixel, whereas $\mathbf{r}_{lnssd}$ is without units and in the range $-N_p < \mathbf{r}_{lnssd}^{(k)} < N_p,\ k = 1..N_p$.

$$\mathbf{r}_{lnssd}^{(k)} = \frac{I_j[\mathbf{p}'_k]}{\bar{\mathbf{I}}_j} - \frac{I_i[\mathbf{p}_k]}{\bar{\mathbf{I}}_i} \tag{4.13}$$

In the work by Usenko et al. [55], the zero normalized cross correlation (ZNCC) is mentioned as a possible residual function. It is not employed in their systems because it is computationally too expensive. Since we perform a general investigation in this thesis, not majorly focusing on runtime, we additionally include the ZNCC in this experiment. It is defined as:

$$ZNCC(\mathbf{p}) = \frac{[\mathbf{I}_i[\mathbf{p}] - \mu(\mathbf{I}_i)]^T\ [\mathbf{I}_j[\mathbf{p}'] - \mu(\mathbf{I}_j)]}{\|[\mathbf{I}_i[\mathbf{p}] - \mu(\mathbf{I}_i)]\|_2\ \|[\mathbf{I}_j[\mathbf{p}'] - \mu(\mathbf{I}_j)]\|_2} \tag{4.14}$$

where $\mathbf{I}_i(\mathbf{p})$ is the vector consisting of all image values from the host patch $i$ in the neighborhood centered at $\mathbf{p}$ [38]. Similary, $\mathbf{I}_j(\mathbf{p}')$ contains all image values from the target image's neighborhood centered at the transformed pixel $\mathbf{p}'$. The arithmetic mean of the image patch stacked as vector of size $N_p$ is denoted by $\mu(\mathbf{I}_i)$, i.e. each entry $\mu(\mathbf{I}_i)^{(k)} = \bar{\mathbf{I}}_i$ for $k = 1...N_p$. If we write the equation with zero-mean image vectors, i.e. $\tilde{\mathbf{I}}_i(\mathbf{p}) = \mathbf{I}_i(\mathbf{p}) - \mu(\mathbf{I}_i(\mathbf{p}))$, we get:

$$ZNCC(\mathbf{p}) = \frac{\tilde{\mathbf{I}}_i[\mathbf{p}]^T \, \tilde{\mathbf{I}}_j[\mathbf{p}']}{||\tilde{\mathbf{I}}_i[\mathbf{p}]||_2 \, ||\tilde{\mathbf{I}}_j[\mathbf{p}']||_2} = cos(angle[\tilde{\mathbf{I}}_i[\mathbf{p}], \tilde{\mathbf{I}}_j[\mathbf{p}']]) \tag{4.15}$$

The ZNCC is the cosine of the angle between the zero-mean image vectors and therefore $-1 < ZNCC < 1$. From the definition via zero-mean vectors, it becomes clear that ZNCC is invariant to affine brightness changes in the images patches, i.e. if we multiply or add a constant to all image values in the patch, the ZNCC remains unchanged. Similar image values result in a ZNCC close to 1, whereas different image values results in a low ZNCC close to -1.

Note that the ZNCC is different from the other presented norms ($\mathbf{r}_{ssd}$, $\mathbf{r}_{ab}$ $\mathbf{r}_{lssd}$ and $\mathbf{r}_{lnssd}$) because it can not be written as squared sum of individual residual values $\mathbf{r}^{(k)}$ of the patch. All other presented norms compute one residual value for each individual pixel of the patch and their final value arises from a sum over the squared individual entries. For example, SSD is given by $\mathbf{r}_{ssd}^T \, \mathbf{r}_{ssd}$. Furthermore, ZNCC is a similarity measure, meaning that higher values are preferred. The other norms calculate a dissimilarity measure. Since we include ZNCC in our least-squares formalism, we need to use a dummy vector $\mathbf{r}_{zncc,i}$ to describe the residual values in patch $i$. The vector $\mathbf{r}_{zncc,i}$ consists of $N_p$ entries, where $N_p$ is the number of pixels in patch $i$. We include ZNCC into our least-square solver routine by setting each entry $\mathbf{r}_{zncc,i}^{(k)}$ with $k = 1...N_p$ as follows:

$$\mathbf{r}_{zncc,i}^{(k)} = \begin{cases} \sqrt{1 - \text{ZNCC}} & \text{if } k = 1 \\ 0 & \text{else} \end{cases} \tag{4.16}$$

If $N$ is the total number of residuals in our PBA problem, we have $i = 1...N$. We use $\sqrt{1 - ZNCC}$, because the residuals will be squared in our implementation, i.e. we calulate $\mathbf{r}_{zncc,i}^T \, \mathbf{r}_{zncc,i}$ for each patch $i$, see Section 3.4.4. This results in $1 - ZNCC$ to be minimized, which in turn means that we maximize the similarity measure ZNCC.

Furthermore, we use the residual function ZNSSD: There exists the following relation between ZNCC and ZNSSD, derived in [46] and given by:

$$ZNSSD = 2 \cdot (1 - ZNCC) \tag{4.17}$$

where ZNSSD is defined as $\mathbf{r}_{znssd,i}^T \, \mathbf{r}_{znssd,i}$. Minimizing $1 - ZNCC$ as suggested above, is therefore analogous to minimizing ZNSSD. Each element $k$ of the vector $\mathbf{r}_{znssd,i}$ is

computed as:

$$\mathbf{r}_{znssd}^{(k)} = \frac{\tilde{\mathbf{I}}_j^{(k)}}{\sqrt{\tilde{\mathbf{I}}_j^T \tilde{\mathbf{I}}_j}} - \frac{\tilde{\mathbf{I}}_i^{(k)}}{\sqrt{\tilde{\mathbf{I}}_i^T \tilde{\mathbf{I}}_i}} \tag{4.18}$$

where $\mathbf{p_k}$ is a point of the residual pattern, i.e. $k = 1...N_p$. ZNSSD is similar to $\mathbf{r}_{lssd}$, but in the case of ZNSSD we compute zero-mean vectors, and we divide by the scaled sample standard deviation instead of the mean.

**Configuration**
We deviate from the default configuration in Table 4.2 by the following list:

- Solver: ceres-solver
- Robust norm: global Huber with $\tau = 0.2$ for LNSSD, ZNCC and ZNSSD and $\tau = 5.0$ for all others
- camera LM dampening: diagonal of Schur complement

**Results**
Table 4.11 shows all presented formulations. The ATE improves significantly when modelling brightness changes, i.e. SSD performs worse than all others (except ZNCC).

Taking brightness changes only implicitly into account by using LSSD, LNSSD or ZNSSD performs similar or even better than explicitly optimizing for $a_i, b_i, a_j, b_j$ in the $\mathbf{r}_{ab}$ formulation (APOPT). Since the explicit optimization increases the parameter space, therefore requiring more memory and computational resources, we recommend to use LSSD which is the best choice on our dataset. The reason why it performs better than ABOPT might be related to the fact that it takes local brightness changes into account, individually for each patch. However, this can not explain why LNSSD and ZNSSD perform worse than LSSD. There might be other differentiating properties of LSSD, LNSSD and ZNSSD which can explain their different performances. One difference between LSSD in contrast to LNSSD and ZNSSD, is that LSSD computes residual values in units of pixels, whereas both LNSSD and ZNSSD are measures without units. Because of that, we also choose different Huber parameters, i.e. $\tau = 5.0$ for LSSD and $\tau = 0.2$ for LNSSD and ZNSSD, which is another reason why they could perform differently. LSSD is computationally more demanding than simple SSD since the mean needs to be calculated by summing over all pixels in the residual pattern. Additionally, the computation of motion Jacobians causes a small computational overhead for LSSD, see Section 4.5. Therefore, if runtime is the most crucial design factor of the PBA system, SSD should be used.

| residuals: | SSD | LSSD | LNSSD | ABOPT | ZNCC | ZNSSD |
|---|---|---|---|---|---|---|
| all sequences | 0.693 | **0.658** | 0.670 | 0.666 | 0.751 | 0.676 |
| euroc-ok | 0.691 | 0.554 | **0.491** | 0.556 | 0.535 | 0.548 |
| euroc-fail | 0.997 | 1.025 | 1.002 | 1.021 | 1.004 | **0.994** |
| kit-no-loop | **0.724** | 0.800 | 0.884 | 0.873 | 1.000 | 0.872 |
| kit-loop | 0.568 | 0.559 | 0.672 | **0.503** | 0.733 | 0.685 |

Table 4.11: **Comparison of $ATE_{rmse,geo}$ for different residual formulations.** We recomment using LSSD or ZNSSD, which in constrast to ABOPT take brightness changes only implicitly (but locally for each patch) into account. This avoids the introduction and optimization of new parameters such as in ABOPT. Huber loss function with $\tau = 0.2$ for LNSSD, ZNCC and ZNSSD and $\tau = 5.0$ for all others, see Equation 3.23. Kitti00 is excluded for abopt because the memory requirement is too large. $ATE_{rmse,geo}$ relative to init_pgo.

Interestingly, the naive formulation of ZNCC does not work well. However, the analogous measure ZNSSD, which is reformulated to fit into the least-squares formalism, performs quite well. A possible reason why ZNSSD performs better than ZNCC might be related to the fact that the contribution of each residual to the LM Hessian matrix is only a rank-one matrix for ZNCC. To see why, consider the example residual vector $\mathbf{r}_{zncc,i} = (1 - ZNCC, 0, 0, 0)^T = (a, 0, 0, 0)^T$, for $N_p = 4$. The Jacobian $\mathbf{J}_{ZNCC}$ for ZNCC can exclusively contain non-zero elements in the first row. Having for example three motion parameters $(x, y, z)^T$, we obtain:

$$\mathbf{J}_{ZNCC} = \begin{pmatrix} \frac{\partial a}{\partial x} & \frac{\partial a}{\partial y} & \frac{\partial a}{\partial z} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.19}$$

The rank of $\mathbf{J}_{ZNCC}$ can be at most one, because the rank is determined by the number of linearly independent columns [30, p. 283]. Since $rank(\mathbf{AB}) \leq min[rank(\mathbf{A}), rank(\mathbf{B})]$ [47, eq. 546], the rank of $\mathbf{J}_{ZNCC}^T \mathbf{J}_{ZNCC}$ is therefore at most one, as well. Overall, the LM Hessian is a sum of the Hessians computed for each residual. Hence, the final LM Hessian might be of full rank and invertible even without any dampening for both ZNCC and ZNSSD. Besides this difference, we expect that ZNSSD and ZNCC both perform better for larger patterns, because ZNCC is not well-defined by only a few pixels, see Section 4.13.

In a small follow-up experiment, we investigated the final ATE when the effect of the camera vignette is taken into account for $\mathbf{r}_{ssd}$ before computing the residuals, as in

DSO. We used the calibration tool from the basalt project [57] to compute the vignette for Euroc MAV. The output is a grayscale image containing values in the range $[0, 255]$. By normalizing this image, we obtain the vignetting function $V(\mathbf{x}) : \Omega \rightarrow [0, 1]$ as in DSO, where $\Omega$ is the image plane. We do not have access to the camera response $\gamma()$ and assume it is linear. We pre-process the loaded images $I_{loaded}$ by performing i.e. $I_{new}(\mathbf{x}) = \frac{I_{loaded}(\mathbf{x})}{V(\mathbf{x})}$ $\mathbf{x} \epsilon \Omega$ before setting up the PBA routine. There is almost no improvement in modelling the vignette, see Table 4.12.

| | vignette | no_vignette |
|---|---|---|
| eurocMH01 | **0.015** | **0.015** |
| eurocMH02 | **0.014** | **0.014** |
| eurocMH03 | **0.038** | **0.038** |
| eurocMH04 | **2.873** | *2.882* |
| eurocMH05 | **0.067** | **0.067** |
| eurocV101 | **0.033** | **0.033** |
| eurocV102 | **0.036** | **0.036** |
| eurocV103 | *1.233* | **1.232** |
| eurocV201 | **0.014** | **0.014** |
| eurocV202 | **0.053** | **0.053** |
| eurocV203 | **1.281** | *1.283* |

Table 4.12: **Absolute ATE values for Euroc with and without vignette for $\mathbf{r}_{ssd}$.** Bold numbers are the lowest (best) ATE for the respective sequence. Modelling the vignette function does not improve our results. When loading the images we divide them element-wise by the vignette image. We suspect to see a larger improvement when the camera response $\gamma()$ is modelled, but we do not have access to $\gamma()$ for the employed datasets and assume it is linear

## 4.5 Host - Target Frame: Approximated Warp and Motion Jacobian

**Motivation**

The quantities of interest, i.e. the motion $\xi$ and structure $id_p$ parameters, are included

in the warp function $w$:

$$\mathbf{p}'_k = \Pi_c(\underbrace{\mathbf{T}_{ji}\Pi_c^{-1}(\mathbf{p}_k, id_p)}_{\mathbf{p}_{3D,k}}) = w(\mathbf{p}_k, id_p, \mathbf{T}_{ji}) \tag{4.20}$$

where $\mathbf{T}_{ji} = \mathbf{T}_j\mathbf{T}_i^{-1} = exp(\hat{\xi}_j)\, exp(\hat{\xi}_i)^{-1}$. We fix the residual pattern in the host frame which is associated with a landmark of inverse distance $id_p$. In this section, $\Pi_c^{-1}(\mathbf{p}, id_p)$ maps 2D pixel coordinates $\mathbf{p}$ to homogeneous coordinates $\mathbf{p}_{3D} \in \mathbb{R}^4$ representing a 3D point in the coordinate system of the host; conversely, $\Pi_c(\mathbf{p}_{3D})$ maps homogeneous coordinates $\mathbf{p}_{3D} \in \mathbb{R}^4$ to 2D pixel positions $\mathbf{p}'$.

We employ two approximations in the current implementation: At first, to obtain the warped pixel positions $\mathbf{p}'_k$ for all pixels $k = 1...N_p$ of the residual pattern, we perform a first-order Taylor expansion of the warp function around the central pixel. The central pixel is denoted by $\mathbf{p}_{center}$; the offset to a pixel $k$ of the residual pattern in the host is given by $\mathbf{u}_k$. Hence, the first-order expansion of the warp is given by:

$$w(\mathbf{p}_{center} + \mathbf{u}_k, id_p, \mathbf{T}_{ji}) \approx w(\mathbf{p}_{center}, id_p, \mathbf{T}_{ji}) + \left.\frac{\partial w(\mathbf{p}, id_p, \mathbf{T}_{ji})}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_{center}} \cdot \mathbf{u}_k \tag{4.21}$$

The derivative of the warp function relative to the pixel position in the host can be written in terms of the projection Jacobian $\mathbf{J}_{proj}$ and unprojection Jacobian $\mathbf{J}_{unproj}$:

$$\left.\frac{\partial w(\mathbf{p}, id_p, \mathbf{T}_{ji})}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_{center}} = \underbrace{\left.\frac{\partial \Pi_c(\mathbf{p}_{3D})}{\partial \mathbf{p}_{3D}}\right|_{\mathbf{p}_{3D}=\mathbf{T}_{ji}\Pi_c^{-1}(\mathbf{p},id_p)}}_{\mathbf{J}_{proj}(\mathbf{p}_{3D})} \cdot \mathbf{T}_{ji} \cdot \underbrace{\left.\frac{\partial \Pi_c^{-1}(\mathbf{p})}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_{center}}}_{\mathbf{J}_{unproj}} \tag{4.22}$$

The second approximation that we make in our implementation is to approximate the *full Jacobians* ($\mathbf{J}_\xi$ or $\mathbf{J}_{id}$) of the residual function by only computing *motion Jacobians* $\mathbf{J}_{motion}$ for the central pixel. In this thesis, we define the motion Jacobian to be this part of the full Jacobian which does not include the image gradient. For example, in the case of $\mathbf{r}_{ssd}$, row $k$ of the approximated full Jacobian with respect to pose parameters $\xi$ is given by:

$$\mathbf{J}^{(k)}_{\xi,ssd} = \frac{\partial \mathbf{r}^{(k)}_{ssd}(w(\mathbf{p}_k, id_p, \mathbf{T}_{ji}))}{\partial \xi} = \nabla_{\mathbf{p}} I_j[\mathbf{p}]^T\Big|_{\mathbf{p}=\mathbf{p}'_k} \cdot \underbrace{\mathbf{J}_{proj}(\mathbf{p}_{3D,central}) \cdot \frac{\partial \mathbf{p}_{3D,central}}{\partial \xi}}_{\mathbf{J}_{motion}} \tag{4.23}$$

where $\frac{\partial \mathbf{p}_{3D}}{\partial \xi} = \begin{pmatrix} \mathbf{I}_{3x3} & -\mathbf{p}\hat{3}D \\ \mathbf{0}^T & 0 \end{pmatrix}$ [20]. $\nabla_{\mathbf{p}} I[\mathbf{p}]^T|_{\mathbf{p}=\mathbf{p}'_k}$ denotes the image gradient evaluated at the transformed pixel position $\mathbf{p}'_k$ in the target image $j$.

Similarly, row $k$ of the approximated Jacobian with respect to the point's inverse distance parameter $id_p$ is given by:

$$\mathbf{J}_{id,ssd}^{(k)} = \frac{\partial \mathbf{r}_{ssd}^{(k)}(w(\mathbf{p}_k, id_p, \mathbf{T}_{ji}))}{\partial id_p} = \nabla_{\mathbf{p}} I_j[\mathbf{p}]^T \Big|_{\mathbf{p}=\mathbf{p}'_k} \cdot \underbrace{\mathbf{J}_{proj}(\mathbf{p}_{3D,central}) \cdot \frac{\partial \mathbf{p}_{3D}}{\partial id_p}}_{\mathbf{J}_{motion}} \tag{4.24}$$

where $\frac{\partial \mathbf{p}_{3D}}{\partial id_p} = (\mathbf{t}_{ij}^T,\ 1)^T$ is independent of which pixel of the residual pattern is chosen because they all share the same $id_p$ parameter. In summary, in both cases the approximation is given by using the projection Jacobian $\mathbf{J}_{proj}$ of the central pixel $\mathbf{p}_{3D,central}$ for all pixels of the residual pattern. Additionally, in the case of pose parameters, we use the Jacobian $\frac{\partial \mathbf{p}_{3D,central}}{\partial xi}$ of the central pixel as approximation. Note that we can write $\mathbf{r}_{lssd}$ as:

$$\mathbf{r}_{lssd}^{(k)} = I_j[\mathbf{p}'_k] - \frac{\bar{\mathbf{I}}_j}{N_p\, \bar{\mathbf{I}}_i}\, I_i[\mathbf{p}_k] = I_j[\mathbf{p}'_k] - \frac{I_i[\mathbf{p}_k]}{N_p\, \bar{\mathbf{I}}_i} \sum_{n=1}^{N_p} I_j[w(\mathbf{p}_k + \mathbf{u}_n)] \tag{4.25}$$

where $\mathbf{u_n}$ is the offset to model the residual pattern in the host frame, for example: $\{\mathbf{u}_1, ..., \mathbf{u}_{N_p}\} = \{(0,0)^T, (1,-1)^T, (-1,1)^T, (-1,-1)^T, (2,0)^T, (0,2)^T, (-2,0)^T, (0,-2)^T\}$ for the DSO pattern. $\bar{\mathbf{I}}_i$ and $\bar{\mathbf{I}}_j$ denote the arithmetic mean of all pixel values in the host and target patch, respectively.
Therefore, row $k$ of the Jacobian $\mathbf{J}_{\xi|id,lssd}$ for LSSD can be expressed computed as:

$$\mathbf{J}_{\xi|id,lssd}^{(k)} = \mathbf{J}_{\xi|id,ssd}^{(k)} - \frac{I_i[\mathbf{p}_k]}{N_p\, \bar{\mathbf{I}}_i} \sum_{n=1}^{N_p} \mathbf{J}_{\xi|id,ssd}^{(n)} \tag{4.26}$$

Overall, for $\mathbf{r}_{lssd}$ we have to compute the same $N_p$ Jacobians as for the case of $\mathbf{r}_{ssd}$. Similarly, we can express the Jacobians for $\mathbf{r}_{lnssd}$ with the Jacobians of $\mathbf{r}_{ssd}$:

$$\mathbf{r}_{lnssd}^{(k)} = \frac{I_j[\mathbf{p}'_k]}{\bar{\mathbf{I}}_j} - \frac{I_i[\mathbf{p}_k]}{\bar{\mathbf{I}}_i} \tag{4.27}$$

By using the quotient rule we get the following expression for row $k$ of the Jacobian for LNSSD:

$$\mathbf{J}_{\xi|id,lnssd}^{(k)} = \frac{\mathbf{J}_{\xi|id,ssd}^{(k)}\, \bar{\mathbf{I}}_j - I_j[\mathbf{p}'_k]/N_p \cdot \sum_{n=1}^{N_p} \mathbf{J}_{\xi|id,ssd}^{(n)}}{\bar{\mathbf{I}}_j^2} \tag{4.28}$$

**Experiments**
To validate the above approximations, we compare the ATE between the approximated version and the exact version. Both the approximate and the exact version warp each pixel of the patch, hence we call this the **full warp**. The exact full warp projects each

pixel using the warp function; the approximated full warp projects each pixel using the first order Taylor expansion of the warp. Furthermore, the exact version uses the true projection Jacobian $\mathbf{J}_{proj}(\mathbf{p}_{3D})$ for each 3D point of the unprojected residual pattern. Accordingly, it uses the exact parameter Jacobians $\frac{\partial \mathbf{p}_{3D}}{\partial xi}$.

In a second experiment, we try an even stronger approximation by using the image gradient of only the central pixel for all pixels of the residual pattern. We compare two scenarios: (1) We use the central pixel's gradient $\nabla_{\mathbf{p}} I[\mathbf{p}'_{central}]^T$ but compute the warp and the motion Jacobians exactly. Image values and residual values are therefore exact. (2) We approximate the image gradient as well as the warp and the motion Jacobian by the central pixel. In both cases, we interpolate the image gradient with the bilinear_smooth interpolation, to take at least some neighborhood information into account. This experiment is done for $\mathbf{r}_{ssd}$.

In a third experiment, we apply the proposed warp and Jacobian approximations using larger residual pattern to determine in which region the proposed approximations are valid. The employed patterns are visualized in Figure 4.11. In all other presented experiments in this thesis, we use the standard residual pattern from DSO if not stated otherwise.

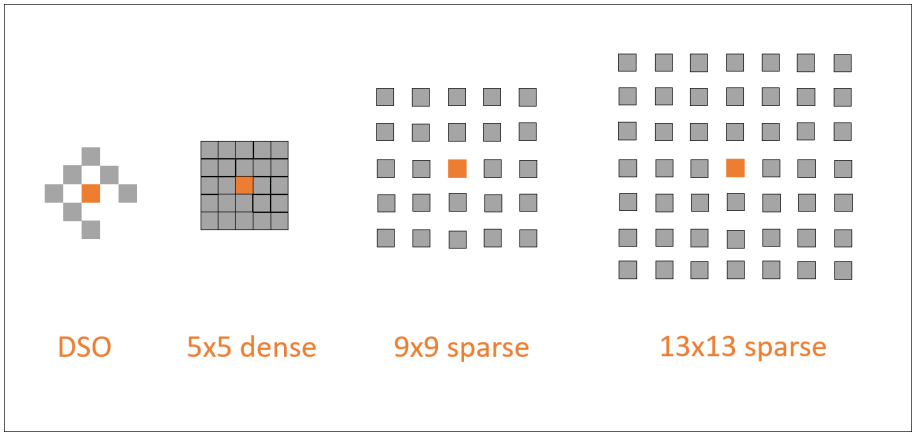



Figure 4.11: **Patterns used to test approximation radius.** The sparse patterns only occupy every second row and column of the pixel grid.

In a fourth experiment, we use a very **simple warp** function such as in [5]. The simple warp projects only the center pixel $\mathbf{p}_{3D,center}$ exactly and simply adds the host image offset $\mathbf{u}_n$ to reach pixels in the target image's residual pattern:

$$\mathbf{p}' = \Pi_c(\mathbf{p}_{3D,center}) + \mathbf{u}_n \tag{4.29}$$

**Configuration**

We deviate from the default configuration in Table 4.2 by the following list:

- Robust norm: global Huber with $\tau = 0.2$ for LNSSD and $\tau = 5.0$ for all others
- camera LM dampening: diagonal of Schur complement

**Results**
Table 4.13 shows the results of approximating the full warp by its first order expansion and the motion Jacobians by the central pixel. The difference between exact and approximated versions is very small for all selected norms. For LSSD and LNSSD, the approximated version is slightly better. From this we can conclude, that for the DSO pattern it is feasible to use the approximated full warp in contrast to the exact full warp function.

| | SSD | | LSSD | | LNSSD | |
|---|---|---|---|---|---|---|
| full warp and $\mathbf{J}_{motion}$: | exact | approx | exact | approx | exact | approx |
| no euroc-fail | **0.654** | 0.657 | 0.639 | **0.635** | 0.647 | **0.640** |

Table 4.13: **Full warp and motion Jacobian: approximate vs. exact for different norms.** We show the ATE difference between approximated and exact version of the full warp and motion Jacobian. For the DSO pattern, it is feasible to use the approximated version. The dataset euroc-fail is excluded from the table because it differs a lot when choosing between the approximated or exact version, hence it does not contribute to the average in a meaningful way. $ATE_{rmse,geo}$ relative to init_pgo, for geometric mean over all sequences except euroc-fail.

Table 4.14 shows the result of using the central image gradient in combination with exact motion Jacobians (left column) vs. the central image gradient with approximated motion Jacobians (right column). The results are similarly bad for both cases. This shows that the most *crucial component* of the full Jacobian ($\mathbf{J}_{\xi}$ or $\mathbf{J}_{id}$) is to evaluate the image gradient at a location close to its true location. However, it is sufficient if the evaluation point for the image gradient is only an approximated pixel position, using a first-order Taylor expansion of the warp function around the central pixel.

| <exact $\nabla_p I[\mathbf{p}']$, exact $\mathbf{J}_{motion}$> | <false, true> | <false, false> |
|---|---|---|
| all sequences | 0.863 | 0.864 |

Table 4.14: **Approximation of image gradient by central pixel for $\mathbf{r}_{ssd}$.** If the image gradient is approximated by the central pixel, it does not matter whether the motion Jacobians and the warp are exact or not. To obtain good results, the image gradient must be evaluated at the warped target position or close to it using an approximation such as a first order expansion of the warp function. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

The third experiment is summarized in Table 4.15. The difference between the exact and the approximated motion Jacobians becomes noticeable only for very large patterns, i.e. the 13x13 sparse pattern in our experiments. Therefore, for smaller pattern we can safely use the approximations without having to resort to the exact versions. Note that with such a large residual pattern the optimization starts to worsen, even if motion Jacobians are exact. This is because we assign the same inverse distance to all points of the residual pattern and in many real-world scenarios such large patterns do not accurately model the geometry.

| | 5x5 dense | | 9x9 sparse | | 13x13 sparse | |
|---|---|---|---|---|---|---|
| full warp and $\mathbf{J}_{motion}$: | exact | approx | exact | approx | exact | approx |
| all sequences | **0.692** | **0.692** | **0.739** | 0.742 | **0.785** | 0.796 |

Table 4.15: **Full warp and motion Jacobian: approximate vs. exact for varying residual patterns with $\mathbf{r}_{ssd}$.** For larger residual patterns, the difference between approximated and exact warping becomes more dominant. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences. Kitti00 is excluded for the 13x13 pattern, because the required memory is too large.

The results when using the simple warp given in equation 4.29 are displayed in Table 4.16. It is necessary to take a closer look at the sub datasets to understand how the different warping strategies perform compared to each other. The average over all sequences is better for larger patterns using the full warp; in the case of the DSO pattern, the approximated full warp performs equally well as the simple warp. However, their advantage over the exact full warp is mainly due to the fact that on euroc-fail, the exact version increases the ATE a lot. That is why we show results without euroc-fail in Table 4.13. Also note that the simple warp only outperforms the full warp on either kit-no-loop or euroc-fail for all tested pattern sizes. The captured scenes on kit-no-loop

contain mostly linear motion on straight roads, therefore it makes sense that the simple warp can outerpform the more complex full warp here. Furthermore, a meaningful comparison on euroc-fail is not possible, because PBA diverges in those parts of the trajectory where the initialization failed. For the small DSO pattern, it might still be feasible to use the simple warp for all sequences. For larger patterns, the results degrade a lot when using the simple warp in contrast to the full warp (approximated or exact), mostly on euroc-ok and kit-loop (compare with Table 4.15 and Table 4.16).

| | DSO pattern | | | 9x9 sparse | | | 13x13 sparse | | |
|---|---|---|---|---|---|---|---|---|---|
| warp: | exact | approx | simple | exact | approx | simple | exact | approx | simple |
| all | 0.707 | **0.693** | **0.693** | **0.739** | 0.743 | 0.787 | **0.785** | 0.796 | 0.847 |
| euroc-ok | **0.688** | 0.691 | 0.690 | 0.738 | **0.731** | 0.779 | **0.795** | 0.796 | 0.930 |
| euroc-fail | 1.183 | **0.997** | 1.006 | 0.996 | 0.996 | **0.986** | 1.015 | **1.013** | 1.010 |
| kit-no-loop | 0.719 | 0.724 | **0.702** | 0.778 | 0.781 | **0.766** | 0.736 | 0.774 | **0.699** |
| kit-loop | **0.548** | 0.568 | 0.579 | **0.566** | 0.566 | 0.603 | **0.579** | 0.582 | 0.640 |

Table 4.16: **Simple vs. full warp (exact and approximate version) for varying pattern sizes.** For smaller patterns, the simple warp yields similar results as the full warp; for larger patterns or datasets with more rotations in the image plane, the full warp should be used. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over respective sequences. Kitti00 is excluded for the 13x13 pattern, because the required memory is too large.

Note that on Kitti and Euroc, especially on kit-no-loop, there is no major rotational movement recorded. We expect that on certain datasets which contain a lot of rotational movements in the image plane, e.g. in mobile phone applications, the full warp (exact or approximate) must be used to achieve reasonable results, even for smaller patterns.

## 4.6 Robust Norm: Corrected Weight for t-Distribution

**Motivation**

Besides the robust Huber loss, we use the t-distribution to model our residual distribution. Since we estimate the scale $\sigma_t$ of the t-distribution for each target frame $t$ independently, we must use a corrected weight $w_{i,corrected}$ in the least-squares formulation (see Section 3.4.2). This corrected formulation differs by the factor $\frac{1}{\sigma_t^2}$ from the

literature in [27], [28], [63]:

$$w_{i,corrected} = \frac{1}{\sigma_t^2} \frac{v+1}{v + (\frac{r_i}{\sigma_t})^2} \tag{4.30}$$

We compare the ATE for the corrected and the uncorrected formulation. Additionally, we also look at the statistics of $\sigma_t$ for different datasets. Only if the scale parameters $\sigma_t$ vary a lot across target frames, we can expect to see a noticeable difference. The scale $\sigma_t$ for target frame $t$ is obtained by performing the following $k$ iterations (usually $k$ is around 3-5) [50, section 3, step 5]:

$$(\sigma_t^2)^{(k+1)} = \frac{1}{N_t} \sum_{i=1}^{N_t} r_i^2 \frac{v+1}{v + (\frac{r_i}{\sigma_t^{(k)}})^2} \tag{4.31}$$

where $N_t$ is the number of residuals residing in target frame $t$. We initialize $\sigma_t^{(0)} = MAD$, Equation 3.22. Note however, that the estimation of scale parameters require enough inlier residual data points to find the t-distribution's parameter robustly. In the case of not having enough (reliable) residuals, it might be better to use a global constant value for the $\sigma$ parameter, possibly averaged over multiple datasets.

**Results**

Table 4.17 shows detailed results for each dataset when fitting the scales $\sigma_t$ for each target frame during runtime. Note that on euroc-fail (eurocV103, eurocV203, eurocMH04) and also on eurocV202, our new formulation results in higher final ATE than_pgo. A numerical comparison of ATE does not make sense for these failed sequences, because when PBA yields a higher ATE, the initialization itself is likely not good enough and should be improved. Only if we can observe a meaningful PBA result for all compared methods, we can argue about which method works better than than the others, i.e. which method decreases the ATE more. Therefore, we additionally exclude eurocV202 for computing the averages in Table 4.19. For all other Euroc sequences, the new formulation provides an improvement compared to the original one, see Table 4.17.

Furthermore, by setting $\sigma_t = mean(\sigma_{all}) = 8.95$, we can use either our corrected formula or the old formula because they are equivalent up to scale for constant $\sigma_t$. This is similarly done in [26]. In this case, we achieve the lowest ATE, see Table 4.20. Therefore, it might not be worth spending the computational effort of fitting the t-distribution, at least for the datasets Euroc or Kitti. We have further tested using separate constants for Kitti where $mean(\sigma_{kitti}) = 7.91$ and Euroc where $mean(\sigma_{euroc}) = 10.00$. However, there is no advantage in doing so, the result does not vary strongly when choosing slightly different tuning constants. As a side-note, it is interesting to see that

eurocV203 actually does improve compared to init_pgo in the case of setting $\sigma_t = 8.95$, which is almost never the case and therefore eurocV203 is part of euroc-fail.

| | init_pgo | new_tdist | old_tdist | const_8p9536 |
|---|---|---|---|---|
| carlacircle | 0.024 | **0.012** | **0.012** | *0.014* |
| eurocMH01 | 0.034 | **0.014** | **0.014** | *0.015* |
| eurocMH02 | 0.019 | *0.014* | **0.013** | *0.014* |
| eurocMH03 | 0.087 | *0.044* | 0.057 | **0.041** |
| eurocMH04 | **2.669** | 2.963 | 3.408 | *2.825* |
| eurocMH05 | *0.070* | **0.063** | 0.071 | 0.072 |
| eurocV101 | 0.040 | **0.032** | 0.034 | *0.033* |
| eurocV102 | 0.058 | *0.044* | 0.048 | **0.038** |
| eurocV103 | **1.218** | 1.240 | 1.259 | *1.236* |
| eurocV201 | 0.026 | **0.012** | 0.016 | *0.014* |
| eurocV202 | *0.053* | 0.145 | **0.046** | 0.054 |
| eurocV203 | *1.318* | 1.431 | 1.402 | **1.225** |
| kitti00 | 11.217 | **2.989** | 7.260 | *5.225* |
| kitti01 | 115.157 | 112.645 | *103.061* | **99.815** |
| kitti02 | 27.671 | 25.925 | *24.399* | **23.248** |
| kitti03 | 2.916 | *1.139* | **1.121** | 1.240 |
| kitti04 | 1.149 | 0.764 | **0.705** | *0.753* |
| kitti05 | 5.123 | 3.957 | **3.058** | *3.362* |
| kitti06 | 14.593 | **4.835** | 6.125 | *5.736* |
| kitti07 | 3.079 | 1.839 | *1.462* | **1.447** |
| kitti08 | 130.317 | *104.249* | **97.242** | 124.707 |
| kitti09 | 75.914 | 63.914 | **61.454** | *61.779* |
| kitti10 | 17.238 | 13.983 | *13.735* | **13.388** |

Table 4.17: **Detailed ATE results for all sequences using the old weight, the corrected weight and globally constant sigma.** Even though the geometric average over all sequence is the same for both old_tdist and new_tdist (our corrected formula Equation 4.30), we see significant improvement on some scenes using new_tdist, e.g. Kitti00, Kitti06 and eurocMH05. Comparing these detailed results with a measure for the variation of estimated sigmas (CoV) given in table 4.18, shows that our corrected formula mainly improves those sequences with higher variation in sigma. Comparing all three displayed alternatives, the best option is choosing a global constant $\sigma_t = mean(\sigma_{all}) = 8.95$ because it yields the lowest averaged ATE and there is no need to compute the sigmas explicitly during runtime. The numbers display the absolute ATE in meters. Bold numbers indicate lowest (best) ATE.

Table 4.18 shows the *coefficient of variation* (CoV) $\frac{std(\{\sigma\})}{mean(\{\sigma\})}$, which allows a comparison of how much the sigmas deviate relative to their mean value [12]. The arithmetic mean of the estimated scales $\{\sigma\}$ across all target frames for a specific dataset is denoted by $mean(\{\sigma\})$. The sample standard deviation of sigmas across target frames is denoted by $std(\{\sigma\})$.

| | CoV of estimated sigmas |
|---|---|
| carlacircle | **0.269** |
| eurocMH01 | **0.663** |
| eurocMH02 | **0.633** |
| eurocMH03 | **0.749** |
| eurocMH04 | **0.749** |
| eurocMH05 | **0.638** |
| eurocV101 | **0.674** |
| eurocV102 | **0.600** |
| eurocV103 | **0.700** |
| eurocV201 | **0.529** |
| eurocV202 | **0.669** |
| eurocV203 | **0.838** |
| kitti00 | **0.458** |
| kitti01 | **0.664** |
| kitti02 | **0.405** |
| kitti03 | **0.589** |
| kitti04 | **0.366** |
| kitti05 | **0.425** |
| kitti06 | **0.361** |
| kitti07 | **0.421** |
| kitti08 | **0.485** |
| kitti09 | **0.481** |
| kitti10 | **0.441** |

Table 4.18: **Coefficient of Variation (CoV) for each dataset.** CoV is calculated as $\frac{std(\{\sigma\})}{mean(\{\sigma\})}$, where $mean(\{\sigma\})$ denotes the arithmetic mean of the scales $\{\sigma\}$ across all target frames for a specific dataset and $std(\{\sigma\})$ its sample standard deviation. We expect to see more improvement using new_tdist over old_tdist if a sequence contains a larger CoV. This is because the formulas for new_tdist and old_tdist only differ if the scale parameter sigma is non-constant across target frames.

Theoretically, if the factor $CoV$ is high, we expect to see more improvement because then the variation of sigmas across target frames is numerically higher and should not be neglected. This can be confirmed in our experiments: For the Euroc sequences where the new formulation converges on, $w_{i,corrected}$ shows consistent improvement, with a CoV between 0.53 and 0.75 (average 0.65). We also see a substantial improvement on kit-no-loop, where CoV is between 0.37 and 0.66 (average 0.51). We see no improvement using the corrected formulation $w_{i,corrected}$ on kit-loop, where CoV is between 0.36 and 0.46 (average 0.41).

| | TDist weight | | |
|---|---|---|---|
| weight: | $w_{i,corrected}$ | $w_{i,old}$ | average CoV |
| all sequences | **0.708** | **0.708** | 0.58 |
| euroc-ok | **0.627** | 0.702 | 0.65 |
| euroc-fail&eurocV202 | 1.355 | **1.053** | 0.74 |
| kit-no-loop | **0.520** | 0.584 | 0.51 |
| kit-loop | 0.720 | **0.683** | 0.41 |

Table 4.19: **Fitting the t-distribution scale parameter $\sigma_t$ per target frame $t$ during runtime.** Comparison of formula given in [63] and our corrected weight formula, see Equation 3.21. Right-most colum shows the average CoV on the respective sequence. We expect to see more improvement using our corrected formulation $w_{i,corrected}$ if the CoV is higher. We exclude eurocV202 from euroc-ok because here our new weight formulation increases ATE above init_pgo and a numerical comparison is not reasonable. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean the respective sequence.

| | all datasets | no euroc-fail& no eurocV202 |
|---|---|---|
| all sequences | 0.690 | **0.650** |

Table 4.20: **Using constant scale parameter $\sigma_t = mean(\sigma_{all}) = 8.95$ obtained as average over all Kitti and Euroc sequences.** In this case the corrected and the old weight formulation can both be employed since the constant factor does not alter the optimization results. We use the old formula. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

Figure 4.12 shows results on a *differently pre-processed* photometric map for the dataset EurocV101. The difference to *init_pgo* is that the photometric map consists of 710 cameras, 458.058 landmarks and 4.517.006 observations, whereas *init_pgo* consists of

695 cameras, 449.646 landmarks, 3.639.173 observations (see Table 4.1). Additionally, the photometric map is sparsified according to a different criterion than *init_pgo*, i.e. redundant observations are removed differently. The reason why there are more observation present in this example is because observations are added for all neighbouring frames. In the case of *init_pgo*, new observations are only added for frames connected by loop closures. This experiment shows the importance of using the correct weight formulation: For example, consider the map is pre-processed more elaborately for offline reconstruction, our formulation could decrease the final error drastically in some cases. In the example, the final ATE drops from 0.34 meters to 0.28 meters.

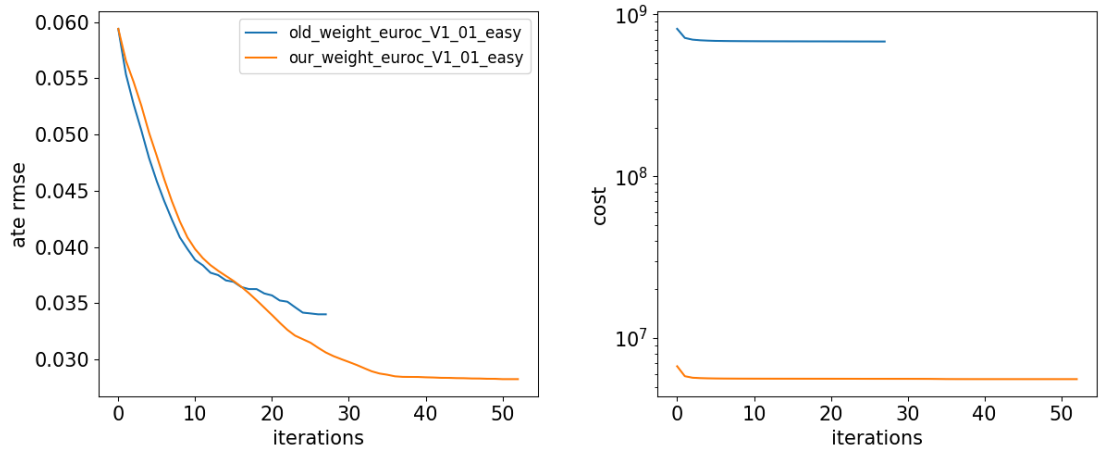



Figure 4.12: **ATE and photometric cost over solver iterations for our corrected t-distribution weight formula on a differently pre-processed eurocV101.** For this differently initialized eurocV101 map, our corrected formulation improves the final ATE a lot (orange) compared to the old formla (blue). The costs on the right have different scales because they differ by the factor of $\frac{1}{\sigma_t^2}$ for target frame $t$.

Figure 4.13 shows the alignment of the two formulation with the ground truth trajectory on eurocV101. Even though ATE is only 0.28 instead of 0.34 meters, there is no large qualitative difference between the new and the old formulation from this top view. However, note that some information about the trajectories is lost by the projection onto the xy-plane of the aligned trajectories and the alignment might have improved mostly in the xz or the yz-plane.

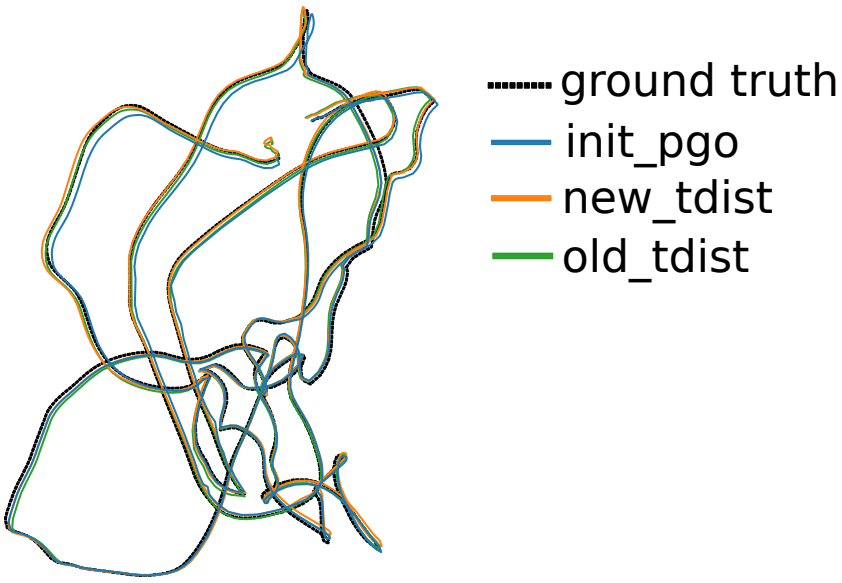



Figure 4.13: **Aligned trajectories for t-distribution (old vs. new) on eurocV101.** This figure shows the alignment for the differently pre-processes photometric map as in figure 4.12. The ATE is 0.28 for the new_tdist and 0.34 for old_tdist, however qualitatively there is no huge visible difference in alignment. This might be due to the fact that the aligmnent has mainly improved in the xz or the yz-plane, which is not shown. The trajectories are aligned to ground truth and their 2D projection onto the xy-world plane is shown.

On Kitti00, the ATE drops significantly from 7.26 meters using old_tdist to 2.989 meters using new_tdist. Accordingly, the alignment looks qualitatively a lot closer to ground truth with our new formulation, see Figure 4.15. On Kitti06, our formulation also yields lower ATE, i.e. 4.835 meters compared to originally 6.125 meters. However, this large numerical difference in ATE is not completely reflected in the alignment plots, i.e. the old and new formulation produce similar trajectories projected on the xy-plane of the world coordinate system, see Figure 4.16.

Figure 4.14 shows the alignment of the old_tdist and new_tdist with the ground truth trajectory on eurcoV202, where the ATE is worse for our formulation. Correspondingly, the alignment looks qualitatively worse in this case; especially in the left half of the trajectories there is a larger deviation from ground truth using our formulation.

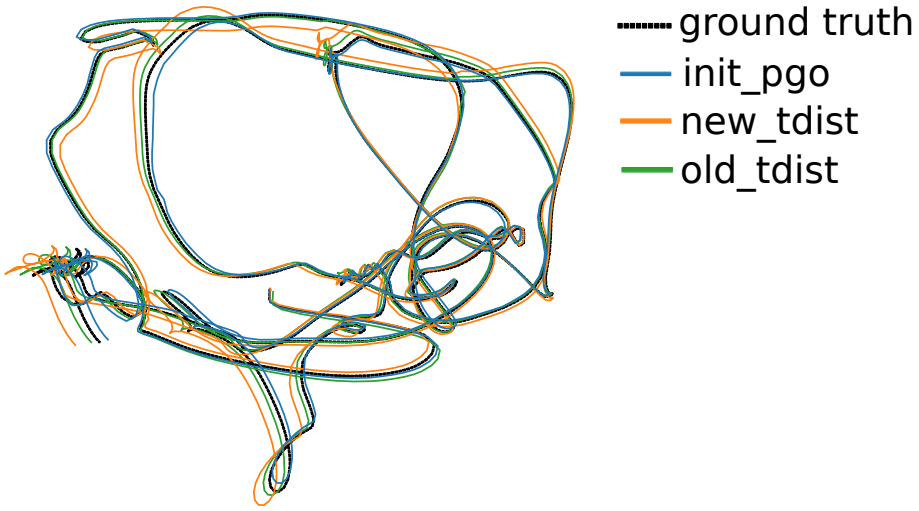



Figure 4.14: **Aligned trajectories for t-distribution (old vs. new) on eurocV202.** old_tdist yields an ATE of 0.046 meters, new_tdist yields an ATE of 0.145 meters. The numerical advantage of old_tdist is reflected in the plot because new_tdist deviates a lot more from ground-truth, mainly in the left half of the trajectory.

.

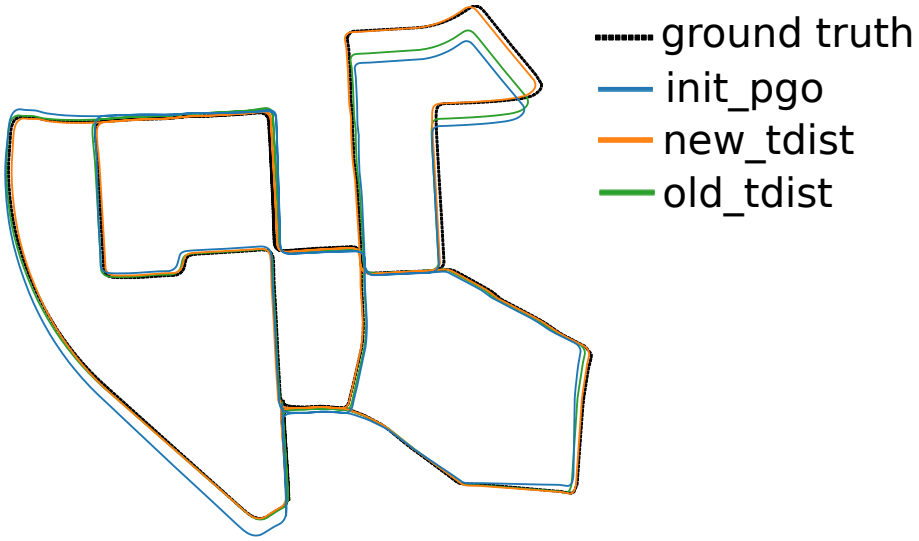



Figure 4.15: **Aligned trajectories for t-distribution (old vs. new) on kitti00.** old_tdist yields an ATE of 7.260 meters, new_tdist yields an ATE of 2.989 meters. The numerical advantage of new_tdist is reflected in the plot, mainly in the top-right and bottom-right loop of the trajectory new_tdist is closer to ground-truth.

.

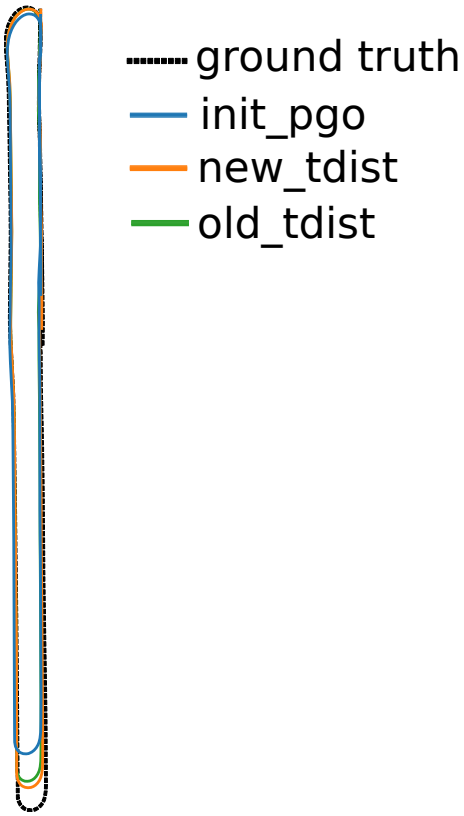



Figure 4.16: **Aligned trajectories for t-distribution (old vs. new) on kitti06.** old_tdist yields an ATE of 6.125 meters, new_tdist yields an ATE of 4.835 meters. The numerical advantage of new_tdist is only very sublte to notice in the plot, i.e. the trajectory of new_tdist forms a slightly larger circle on the bottom and on the top, which is closer to ground-truth.

.

## 4.7 Robust Norm: Corrected Levenberg-Marquardt Hessian (Triggs Correction)

**Motivation**

We implement the Triggs correction as described in Section 3.4.6. We employ the Triggs correction for the robust loss based on the t-distribution as well as the Huber loss. The

required first and second derivatives are presented here:

$$\rho'_\tau(s) = \begin{cases} 1 & \text{if } s < \tau \\ \frac{\tau}{\sqrt{\tau s}} & \text{else} \end{cases} \tag{4.32}$$

$$\rho''_\tau(s) = \begin{cases} 0 & \text{if } s < \tau \\ \frac{-\tau^2}{2\,(\tau s)^{1.5}} & \text{else} \end{cases} \tag{4.33}$$

For the t-distribution (TDist) with $v = 5$ we have:

$$\rho'_{TDist}(s) = \frac{3.0}{5\sigma^2 + s} \tag{4.34}$$

and

$$\rho''_{TDist}(s) = \frac{-3.0}{(5\sigma^2 + s)^2} \tag{4.35}$$

**Results**
When the robust loss function is based on the t-distribution, we show two different dampening strategies for completeness (see Section 4.11). The Triggs correction improves the ATE, even in the first 5 iterations. This is highly relevant if PBA is for example integrated into the back-end of a real-time SLAM system, where the number of iterations should be as small as possible.

| | TDist damp I | | TDist damp Schur | | TDist damp I @it5 | |
|---|---|---|---|---|---|---|
| Triggs correction: | true | false | true | false | true | false |
| all sequences | **0.698** | 0.714 | **0.705** | 0.717 | **0.701** | 0.711 |

Table 4.21: **Triggs correction for the t-distribution (TDist) using different camera dampening.** For both presented dampening strategies, the Triggs correction improves final ATE. Since it is computationally cheap to compute the Triggs correction, it is advisable to always use it in the case of a t-distribution loss. Furthermore, it is advantageous to use the Triggs correction even if only a few iterations are performed, see right-most column. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

In the case of a Huber robust loss function, the Triggs correction worsens the ATE. This can be explained by the fact that the second derivative $\rho''_\tau(s)$ of the Huber function is only non-zero for the outlier region. Therefore, only outliers contribute to the correction of the Hessian if the Triggs correction is used for Huber. If these residuals are very far in the outlier region, their contribution to the overall optimization should

instead be totally ignored. The optimization is only getting disturbed by the Triggs correction based on outliers and converges to a higher final ATE.

| | Huber damp $\boldsymbol{I}$ | | Huber damp $\boldsymbol{I}$ @it5 | | Huber damp $\boldsymbol{I}$ @it10 | |
|---|---|---|---|---|---|---|
| Triggs correction: | true | false | true | false | true | false |
| all sequences | 0.689 | **0.670** | 0.703 | **0.700** | 0.695 | **0.689** |

Table 4.22: **Triggs correction for the Huber loss function.** In the case of the Huber loss function, the Triggs correction should not be employed because it yields higher final ATE. We additionally show the ATE after 5 and 10 iterations, where the Triggs correction already worsens the results. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

In figures 4.17 and 4.18 we compare two extreme cases of the solver behaviour for the Triggs correction using the t-distribution on the dataset Kitti08. For Figure 4.17 where the identity is used for camera dampening, we see an improvement using the Triggs correction. Contrary to this, in Figure 4.18 we use the Schur complement for camera dampening and observe that the Triggs correction worsens the final ATE. We provide this example to emphasize that for seemingly small changes of the configuration, the ATE results can vary quite strongly in terms of absolute ATE change for some extreme cases.

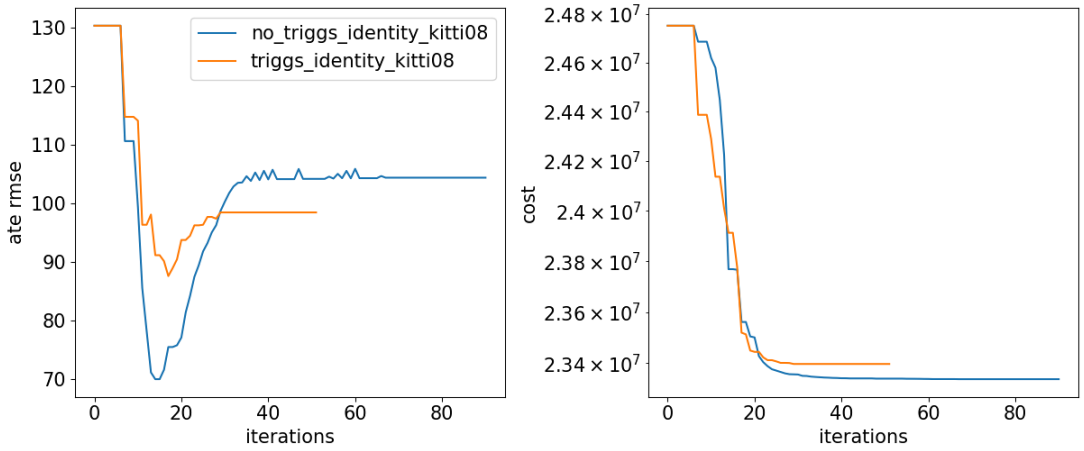



Figure 4.17: **Triggs correction for t-distribution using identity camera dampening on Kitti08.** Using the Triggs correction yields lower final ATE if the LM Hessian is dampened by the identity. The solver behaviour deviates substantially when using the The blue lines indicate the optimization without Triggs correction. The orange lines indicate the optimization using Triggs correction.

However, it should be noted that the absolute ATE for Kitti08 is generally quite high and the initial alignment by init_pgo deviates a lot from ground truth for the second half of the trajectory. On average, when switching between identity or Schur camera dampening, the results do not vary as strongly, see Table 4.21.

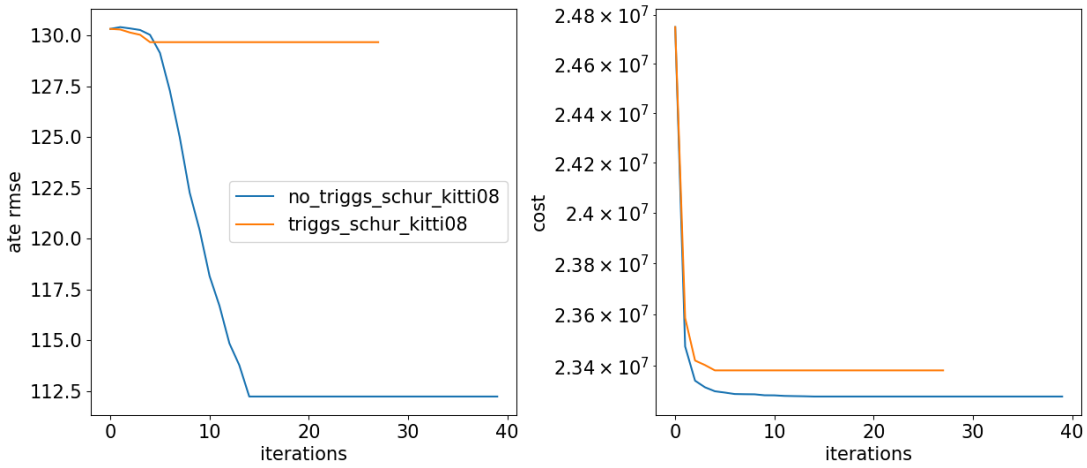



Figure 4.18: **Triggs correction for t-distribution using Schur camera dampening on Kitti08.** Using the Triggs correction yields higher final ATE if the LM Hessian is dampened by the diagonal of the Schur complement. The blue lines indicate the optimization without Triggs correction. The blue lines indicate the optimization without Triggs correction. The orange lines indicate the optimization using Triggs correction.

Furthermore, Figure 4.17 and Figure 4.18 show the general problem that ATE and total photometric error which we optimize are not the same but just correlated to a certain extend. The ATE does not deterministically follow the cost evolution. Most importantly, it is possible that the cost decreases during the optimization even though the global consistency measured by the ATE increases or stays constant. This might be related to the fact that the photometric error is a local constraint between host and target frames, wheres ATE is a global consistency measure. In a real application there is no ground truth trajectory available to compute ATE and only photometric cost can be measured.

## 4.8 Robust Norm: Estimating Scale for Robust Huber Loss

**Motivation**
As described in Section 3.4.3, we investigate the following strategies for selecting the Huber parameter $\tau$:

1. Setting a global $\tau = 5.0$, where 5.0 is an arbitrary constant value. Alternatively, setting $\tau = mean(\hat{\sigma}_{sample}) = 9.66$, where $mean()$ is the dataset mean of estimated scales over whole Kitti and Euroc
2. Setting $\tau = 1.345\ \hat{\sigma}_{MAD,t}$ for each target frame $t$ separately. $\hat{\sigma}_{MAD,t}$ is the initial scale estimate using MAD for each target frame $t$, see Equation 3.22.
3. Setting $\tau = 1.345\ \hat{\sigma}_{sample,t}$ for each target frame $t$ separately. $\hat{\sigma}_{sample,t}$ is the sample standard deviation [30, p. 1064] of all residuals in target frame $t$
4. Setting $\tau = 1.345\ \hat{\sigma}_{tdist,t}$ for each target frame $t$ separately. $\hat{\sigma}_{tdist,t}$ is estimated according to Equation 4.31.

Strategy (1) selects one global constant for all frames, whereas the other data-adaptive strategies (2)-(4) fit a Huber parameter for each frame $t$ individually during runtime. Because selecting a parameter is non-intuitive, we would prefer to use a data-adaptive strategy.

**Results**
Similar to the t-distribution scale estimation in Section 4.6, Table 4.23 shows that using a constant Huber parameter is the best strategy. Note however, that using a Huber constant of 5.0 yields the best result, even though the calulated mean of estimated scales is given by $mean(\hat{\sigma}_{sample}) = 9.66$. This is different to the case of the t-distribution, where the best results was found by setting $\sigma = mean(\sigma_{all}) = 8.95$. This shows that computing the global mean over estimated scales parameter is not the ideal strategy.

| | global constant for all frames | | data-adpative, per-frame | | |
|---|---|---|---|---|---|
| $\tau =$ | 5.0 | 9.66 | 1.345 $\hat{\sigma}_{MAD}$ | 1.345 $\hat{\sigma}_{sample}$ | 1.345 $\hat{\sigma}_{tdist}$ |
| all sequences | **0.672** | 0.689 | **0.680** | **0.680** | 0.681 |

Table 4.23: **Setting the Huber parameter with constant or data-adaptive strategies.** Data-adaptive strategies are generally preferred if the dataset characteristics are unknown beforehand. Therefore, despite the lowest ATE for a global constant of 5.0, in most applications we recommend to use 1.345 $\hat{\sigma}_{sample}$ or 1.345 $\hat{\sigma}_{tdist}$. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

Since the ATE difference is quite small between the global constant of 5.0 and the data-adaptive strategies using 1.345 $\hat{\sigma}_{sample}$ or 1.345 $\hat{\sigma}_{tdist}$, we still recommend to use one of the later data-adaptive strategies. Interestingly, 1.345 $\hat{\sigma}_{tdist}$ which estimates the scale in the same way as for the t-distribution works quite well for the Huber loss. This can possibly be explained by the fact that the scale estimation in Equation 4.31 is more stable than estimating the sample standard deviation of the residuals. Theoretically, we expect the sample standard deviation to be a better estimator for the sigmas of the robust Huber loss, see Section 3.4.3.

## 4.9 Robust Norm: Self-Tuning M-Estimators Approach

**Motivation**
As discussed in the previous sections on robust norms, it is non-trivial to select the best tuning constant $\phi$ for robust loss functions. Agamennoni et al. show how to estimate the tuning constant for a certain class of loss functions from data in [2]. In this experiment, we re-implement their proposed algorithm, which can be integrated into a weighted least-squares estimation process.

The main idea of Agamennoni et al. is to model the residual distribution by a so-called *elliptical distribution*, a generalized multivariate Gaussian. These elliptical distributions can give rise to many common M-estimators, including the Huber loss and the loss based on the t-distribution, as employed in this thesis. An elliptical distribution can be described as a mixture of Gaussians whose covariances are weighted by $w_{cov}$ [2]. Therefore, to estimate the parameters of the underlying elliptical distribution, the parameters of the Gaussian as well as the unknown covariance weights $w_{cov}$ need to be estimated. In order to achieve this, Agamennoni et al. employ an EM algorithm. Each residual $i$ is weighted by one $w_{cov,i}$. Note that this weight should not be confused with the weight $w_i$ of the weighted least squares algorithm, which arises from the robust loss function, see Equation 3.17.

In the expectation step of the EM algorithm, it is required to compute the expectation of the joint probability of covariance weights $w_{cov}$ and data distribution, see equation (6) in [2]. This expectation is approximated via so-called *adaptive importance sampling*. The resulting algorithm contains a number of internal parameters and implementation details, which are explained in the following:

1. A prior density $p(\phi)$ over the tuning constants $\phi$ needs to be selected. A zero-mean Gaussian is proposed in the paper. Both the proposal density $q(\phi)$ and the prior density $p(\phi)$ are modelled as multivariate Gaussians, parameterized by a

mean vector and covariance matrix. We adopt the implementation give in [49] to achieve this.

2. The number of iterations $m$ for calculating the proposal density $q(\phi)$ is typically between 3 and 5.

3. In each iteration $i = 1...m$, $|\Phi|$ tuning constants are drawn from the current proposal density $q(\phi)$. For each drawn tuning constant $\phi_j$, an intermediate variable $\rho_j$ is computed, with $j = 1...|\Phi|$. The set of $\{\phi_j, \rho_j\}$ is used to build the proposal density $q(\phi)$. The tuning constants can possibly be vector-valued. In our case, we have $dim(\rho_j) = 1$: For the Huber loss, we estimate the scalar Huber parameter $\tau$; for the t-distribution, we estimate the scale parameter $\sigma$.

4. To compute the intermediate variables $\phi_j$, equations (12) and (13) are employed in [2]. Equation (12) includes a multiplication of probabilities from $k = 1...n$, where $n$ is the number of data points in the PBA problem. This multiplication can quickly result in a numerical value of zero for large problems. This leads to a loss of precision. Even worse, a division by zero occurs if all $\phi_j$ are zero, because the set $\{\phi_j\}$ is normalized such that its sum is equal to one in every iteration $i$. Therefore, we randomly subsample only $n' < n$ out of all data points in the full PBA problem. We typically use a $n'$ between 500 and 2000.

5. Furthermore, the computation of the intermediate variables $\phi_j$ requires numerical integration from 0 to $\infty$. We employ the publicly available implementation [48] of the adaptive Simpson method [35], using the accuracy stopping criteria $\epsilon = 10e - 9$. We re-parameterize the integral by the following change of variables [16, p. 218], inspired by [1]:

$$\int_0^\infty f(x)dx \approx \int_{10e-6}^{1-10e-6} f\big(0 + \frac{t}{1-t}\big)\,\frac{1}{(1-t)^2}dt \tag{4.36}$$

6. The regularization constant $r$ ensures that the covariance matrix of the proposal density $q(\phi)$ is full rank, see equation (16b) in [2]. We use $r = 0.001$.

**Experiments**

As described above, there are many hyper-parameter choices to be made for the self-tuning approach. This is why in the first experiment, we provide an overview of parameter sets which work on our test dataset. We show the additional runtime required by the self-tuning approach.

In a second experiment, we use fixed hyper-parameters and run the self-tuning approach on the complete dataset.

**Results**

Table 4.24 and 4.25 show the ATE and runtime for a fixed tuning constant vs. using the self-tuning approach. In the following results table, we denote the paramter set as: $[|\Phi|, n', m, r, \mu_p, cov_p]$, where $|\Phi|$ is the number of samples taken from $q(\phi)$, $n'$ is the number of subsampled data points out of the total $n > n'$ residuals in PBA, $m$ is the number of outer iterations for calculating $q(\phi)$, $r$ is the regularization constant and $\mu_p$ and $cov_p$ are the prior mean and covariance, respectively.

The ATE or runtime results are not majorly influenced by choosing a prior on carla_12cam or carla_circle, see Table 4.24 and 4.25. This can be explained by the fact that the self-tuning approach fits the robust weight $w_i$ according to the underlying data and the prior is only relevant for the first few iterations. The authors of the original paper also simply chose a zero-mean prior with the identity as covariance matrix, see [2, sec. IV, D].

The runtime column shows the additional runtime required by the self-tuning approach to compute the set $\{\phi_j, \rho_j\}$, i.e. we measure the time to finish the loop over $m$ and divide by the total runtime. It mainly grows with $|\Phi|$ and $n'$ and $m$. The computation of weights in line 7 of algorithm 1 in [2] is neglected. We measure runtime on the same machine under similar loads. The runtime increase in percent is generally larger for carla_12cam because it consists of only 25088 observations for 3275 landmarks but the chosen hyper-parmeters are in a similar range as for carla_circle. The runtime difference when choosing a prior and leaving all other hyper-paramters constant in Table 4.25 (row three vs. row four) is explained by a slightly different workload on the machine during the time of measurement.

| | Huber | | TDist | |
|---|---|---|---|---|
| carla_12cam | ATE | runtime increase | ATE | runtime increase |
| $\tau_{Huber} = 5.0 \mid \sigma_{TDist} = 8.95$ | 0.514 | 0.0% | 0.416 | 0.0% |
| $[100, 100, 3, 0.001, 0, 1]$ | 0.428 | 4.6% | 0.405 | 3.7% |
| $[500, 500, 3, 0.001, 0, 1]$ | 0.429 | 15.1% | 0.400 | 11.5% |
| $[1000, 1000, 3, 0.001, 8.95, 8.0]$ | 0.448 | 24.0% | 0.451 | 22.2% |

Table 4.24: **ATE and runtime comparison of self-tuning approach vs. a global tuning constant on carla_12cam.** In case of Huber loss, we set of $\tau = 5.0$; in case of the t-distribution loss, we set $\sigma = 8.95$. ATE is largely agnostic to different hyper-parameters within the displayed order of magnitude in this table. The aditional runtime required for the computing the set $\{\phi_j, \rho_j\}$ in percent increases with larger $|\Phi|$ and $n'$. ATE in centimeters. The hyper-parameter vector in the left-most column describes: $[|\Phi|, n', m, r, \mu_p, cov_p]$.

| | Huber | | TDist | |
|---|---|---|---|---|
| carla_circle | ATE | runtime | ATE | runtime |
| $\tau_{Huber} = 5.0 \mid \sigma_{TDist} = 8.95$ | 8.95 | N.A. | 9.64 | N.A. |
| $[500, 500, 3, 0.001, 0, 1]$ | 13.1 | 0.64% | 12.9 | 0.43% |
| $[500, 500, 3, 0.001, 8.95, 8.0]$ | 13.0 | 0.83% | 13.7 | 0.52% |
| $[5000, 5000, 3, 0.001, 0, 1]$ | 13.1 | 0.94% | 12.9 | 0.48% |

Table 4.25: **ATE and runtime comparison of self-tuning approach vs. a global tuning constant on carla_circle.** In case of Huber loss, we set of $\tau = 5.0$; in case of the t-distribution loss, we set $\sigma = 8.95$. ATE is largely agnostic to different hyper-parameters within the displayed order of magnitude in this table. The aditional runtime required for the computing the set $\{\phi_j, \rho_j\}$ in percent increases with larger $|\Phi|$ and $n'$. ATE in centimeters. The hyper-parameter vector in the left-most column describes: $[|\Phi|, n', m, r, \mu_p, cov_p]$.

Table 4.26 and Table 4.27 show the self-tuning approach on the complete dataset for the Huber and t-distribution loss, respectively. We compare the self-tuning approach to a fixed tuning constant and to another data-adpative strategies, as described in Section 4.8 and Section 4.6.

| | Huber | | |
|---|---|---|---|
| | self-tune | $\tau_{Huber} = 5.0$ | $\tau = 1.345\ \hat{\sigma}_{sample,t}$ |
| all sequences | 0.689 | **0.672** | 0.681 |
| euroc-ok | **0.684** | **0.684** | 0.689 |
| euroc-fail | **1.010** | 1.018 | 1.020 |
| kit-no-loop | 0.693 | **0.652** | 0.691 |
| kit-loop | 0.575 | 0.540 | **0.537** |

Table 4.26: **Huber self-tuning approach on the complete dataset in comparison to a fixed tuning constant and another data-adaptive strategy.** Even though the self-tuning approach performs slightly worse in the case of Huber, it could be advisable to use it because it allows to find the most suitable robust loss function among a selection of candidate robust loss functions. We use $[|\Phi|, n', m, r, \mu_p, cov_p] = [2000, 2000, 3, 0.001, 0, 1]$ for the self-tuning approach and expect that better average ATE can be achieved by performing a hyper-paramter search. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over respective sequences.

The advantage of the self-tuning approach in comparison to the alternative data-

adaptive approaches for Huber and t-distribution loss, is that it is more general, i.e. it can easily be employed to other robust loss functions as well, see [2, Table I]. Furthermore, the self-tuning approach could be used to determine which robust loss function is the most suitable for the underlying data, see [2] for more details.

Interestingly, the results in Table 4.27 show that in the case of a t-distribution loss, the self-tuning approach works significantly better than the compared approaches. The final best ATE is now $ATE_{rmse,geo} = 0.679$ meters. This might be due to the fact that the global threshold of global $\sigma_{TDist} = 8.95$ is not ideal and if selected differently, it could possibly outperform the self-tuning approach as well. Similarly, the per target frame modelling of sigmas using our corrected formula might not be ideal because too few inlier residuals could be present in some frames to robustly estimate the sigmas. This approach is described in Section 4.6. However, since we prefer the self-tuning approach due to its generality, it might be worth searching for better hyper-parameters for the self-tuning approach; we expect that this could lead to even lower ATE for Huber and t-distribution. Before this experiment, the Huber loss was always more competitive than the t-distribution, because it achieves $ATE_{rmse,geo} = 0.672$ meters for a global Huber parameter of $\tau = 5.0$.

| | TDist | | |
|---|---|---|---|
| | self-tune | $\sigma_{TDist} = 8.95$ | new_tdist |
| all sequences | **0.679** | 0.690 | 0.708 |
| euroc-ok | 0.685 | **0.683** | 0.754 |
| euroc-fail | 1.019 | **1.000** | 1.071 |
| kit-no-loop | **0.680** | 0.726 | 0.720 |
| kit-loop | 0.547 | 0.544 | **0.520** |

Table 4.27: **T-distribution self-tuning approach on the complete dataset in comparison to a fixed tuning constant and another data-adaptive strategy.** The self-tuning approach should be used in the case of the t-distribution because it achieves lowest ATE. The column new_tdist fits residuals per target frame explained in Section 4.6. We use $[|\Phi|, n', m, r, \mu_p, cov_p] = [2000, 2000, 3, 0.001, 0, 1]$ for the self-tuning approach. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over respective sequences.

## 4.10 Levenberg-Marquardt: Step Criteria

**Motivation**
In order to use the LM algorithm we cast our non-linear function of robustfied residuals $\sum_{i=1}^{N} \rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$ to a NLS formulation. This gives rise to constant weights in every iteration. We additionally linearize the residuals to obtain $C(\boldsymbol{r}(\boldsymbol{\theta})) = \mathbf{r}_{lin}(\boldsymbol{\theta})^T \boldsymbol{W} \mathbf{r}_{lin}(\boldsymbol{\theta})$ as discussed in Section 3.4.5. The LM algorithm checks in each inner iteration (algorithm 1 line 11) if the cost $C(\boldsymbol{r}(\boldsymbol{\theta}))$ has decreased.

However, our ultimate goal in this thesis is to minimize the cost $\rho(\mathbf{r}) = \sum_{i=1}^{N} \rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$. Therefore, in this experiment we alternatively use $\sum_{i=1}^{N} \rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$ to accept or reject a LM step. Theoretically, we expect that this formulation could help in increasing global consistency since it considers the orignal loss function. According to [60], in practice it depends on the programming framework which cost is evaluated and therefore there might also be no huge difference.

**Configuration**
We deviate from the default configuration in Table 4.2 by the following list:

- camera LM dampening: diagonal of Schur complement

**Results**
There is no significant difference between using the original $\rho(\mathbf{r})$ vs. $C(\boldsymbol{r}(\boldsymbol{\theta}))$. Since there is only a slight disadvantage of $\rho(\mathbf{r})$ in terms of ATE, it is advisable to use the cost which is easier and more efficient to implement. Note that in practice our PBA implementation might run in the back-end of a SLAM system to continuously improve the current map. Therefore, we also showed the results after five iterations, where again no difference can be observed for choosing the LM cost difference check.

| | TDist | | TDist @it5 | | Huber | | Huber @it5 | |
|---|---|---|---|---|---|---|---|---|
| cost: | $\rho(\mathbf{r})$ | $C(\boldsymbol{r}(\boldsymbol{\theta}))$ | $\rho(\mathbf{r})$ | $C(\boldsymbol{r}(\boldsymbol{\theta}))$ | $\rho(\mathbf{r})$ | $C(\boldsymbol{r}(\boldsymbol{\theta}))$ | $\rho(\mathbf{r})$ | $C(\boldsymbol{r}(\boldsymbol{\theta}))$ |
| all sequences | 0.709 | 0.707 | 0.711 | 0.711 | 0.693 | 0.690 | 0.707 | 0.707 |

Table 4.28: **Using the original cost $\rho(\mathbf{r}) = \sum_{i=1}^{N} \rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$ vs. using the linearized cost $C(\boldsymbol{r}(\boldsymbol{\theta})) = \mathbf{r}_{lin}(\boldsymbol{\theta})^T \boldsymbol{W} \mathbf{r}_{lin}(\boldsymbol{\theta})$ after casting to a OLS problem.** We recommend to evaluate the cost which is more efficient to calculate in the respective implementation, since ATE differs only slightly on average. Dampening with Schur complement, Huber parameter is set to $\tau = 5.0$, t-distribution weights are determined per target frame with the corrected formulation. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

## 4.11 Levenberg-Marquardt: Dampening Strategies

**Motivation**
The update equations for LM are given by:

$$(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J} + \lambda\,\boldsymbol{M})\,\Delta\boldsymbol{\theta} = -\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{r} \tag{4.37}$$

see Section 3.4.4 for further details. We can split this large system into the part containing the camera parameters (index c) and in the landmark part (index l):

$$\underbrace{\begin{pmatrix}\mathbf{A}_{cc} & \mathbf{B}_{cl}\\ \mathbf{C}_{lc} & \mathbf{D}_{ll}\end{pmatrix}}_{\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J}+\lambda\,\boldsymbol{M}} \underbrace{\begin{pmatrix}\mathbf{x}_c\\ \mathbf{x}_l\end{pmatrix}}_{\Delta\boldsymbol{\theta}} = \left[\begin{pmatrix}\tilde{\mathbf{A}}_{cc} & \mathbf{B}_{cl}\\ \mathbf{C}_{lc} & \tilde{\mathbf{D}}_{ll}\end{pmatrix} + \lambda\cdot\begin{pmatrix}\mathbf{M}_{cc} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}_{ll}\end{pmatrix}\right]\begin{pmatrix}\mathbf{x}_c\\ \mathbf{x}_l\end{pmatrix} = \underbrace{\begin{pmatrix}\mathbf{f}_c\\ \mathbf{g}_l\end{pmatrix}}_{-\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{r}} \tag{4.38}$$

Since $\mathbf{D}_{ll}$ is a diagonal matrix, we apply the Schur complement instead of inverting the left hand side. We obtain the following two subsystems:

$$\mathbf{x}_c = (\underbrace{\mathbf{A}_{cc} - \mathbf{B}_{cl}\mathbf{D}_{ll}^{-1}\mathbf{C}_{lc}}_{\mathbf{S}})^{-1}\,(\mathbf{f}_c - \mathbf{B}_{cl}\mathbf{D}_{ll}^{-1}\mathbf{y}_l) \tag{4.39}$$

$$\mathbf{x}_l = \mathbf{D}_{ll}^{-1}(\mathbf{g}_l - \mathbf{C}_{lc}\mathbf{x}_c) \tag{4.40}$$

Applying LM dampening with the diagonal dampening matrix $\mathbf{M} = \begin{pmatrix}\mathbf{M}_{cc} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}_{ll}\end{pmatrix}$ affects the diagonal of matrix $\mathbf{A}_{cc} = \tilde{\mathbf{A}}_{cc} + \lambda\,\mathbf{M}_{cc}$ and the diagonal of $\mathbf{D}_{ll} = \tilde{\mathbf{D}}_{ll} + \lambda\,\mathbf{M}_{ll}$, respectively. Either the identity or the diagonal of the original Hessian $diag(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J})$ is used for dampening in the LM algorithm, see Section 3.4.4. However, we do not store the original Hessian $\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J}$ in the manual solver, but only the Schur complement $\mathbf{S}$ to save computational and memory resources. Therefore, in the current implementation we set $\mathbf{M}_{cc} = diag(\mathbf{S})$ and $\mathbf{M}_{ll} = \mathbf{0}$ (first column in Table 4.29). In this case, we can compute the Schur complement $\mathbf{S}$ only once and use it for all inner iterations, i.e. for varying $\lambda$ in the LM algorithm. This is possible because $\mathbf{M}_{cc}$ only affects $\mathbf{A}_{cc}$, i.e. $+\lambda\mathbf{M}_{cc}$ can simply be added to the (stored) Schur complement $\mathbf{S}$. However, this is not in accordance with the standard LM, described in Section 3.4.4. Furthermore, the influence of $\mathbf{M}_{ll}$ on the landmarks Hessian part $\mathbf{D}_{ll}$ is currently ignored.

**Experiment**
We want to verify if the above approximation is valid. We investigate four dampening strategies for the camera poses Hessian $\mathbf{A}_{cc} = \tilde{\mathbf{A}}_{cc} + \lambda\,\mathbf{M}_{cc}$, using:

1. The diagonal of the Schur complement as in the current implementation, i.e. $\mathbf{M}_{cc} = diag(\mathbf{S})$
2. The upper diagonal of the original Hessian matrix, i.e. $\mathbf{M}_{cc} = diag(\tilde{\mathbf{A}}_{cc})$
3. The identity $\mathbf{M}_{cc} = \boldsymbol{I}$
4. No dampening $\mathbf{M}_{cc} = \mathbf{0}$ (corresponds to Gauss-Newton (GN) but enforcing a cost decrease by the condition in line 11, algorithm 1)

For each of the presented camera dampening options, we either set the dampening of $\mathbf{D}_{ll} = \tilde{\mathbf{D}}_{ll} + \lambda \mathbf{M}_{ll}$ to $\mathbf{M}_{ll} = \mathbf{0}$ (no dampening) or we use use the original Hessian $\mathbf{M}_{ll} = diag(\tilde{\mathbf{D}}_{ll})$. In the special case of dampening the camera matrix by the identity, we also employ $\mathbf{M}_{ll} = \mathbf{I}$ for the pose Hessian. Furthermore, we compare convergence results of pure GN and LM with no dampening.

**Results**

| | no landmark dampen | | | | landmark dampen | | | |
|---|---|---|---|---|---|---|---|---|
| camera dampen: | Schur | Original | **I** | None | Schur | Original | **I** | None |
| all sequences | 0.688 | 0.691 | **0.671** | 0.683 | 0.691 | 0.691 | 0.677 | **0.668** |

Table 4.29: **Four different camera pose Hessian dampening $\mathbf{M}_{cc}$ options, with and without landmark dampening.** All dampening options yield very similar results for all sequences, except on Kitti03, Kitti08, Kitti09, where the identity outperforms the others. Hence, we recommend to use camera dampening by the identity or the Schur matrix if runtime and memory efficiency is important. Explicitly computing the diagonal of the Hessian is not advantageous. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

Dampening with the original matrix, as proposed in the traditional LM algorithm, performs similar compared to the Schur complement. It can be concluded that the approximation of using the Schur complement is quite reasonable. Using the identity matrix yields the best results by a small margin. The theoretical problem with the identity is that it dampens landmark and pose increments by the same unit, even though they might be of extremely different scale. Therefore, we would have expected a slight disadvantage of the identity vs. using the original matrix. Note that all dampening options yield very similar results for all sequences, except on Kitti03, Kitti08, Kitti09, where the identity outperforms the others. This might be related to the expected issue

of numerical instabilities in our solver as mentioned before. Without considering the mentioned Kitti sequences, we cannot find a systematic advantage in convergence behaviour of the identity over any other dampening matrix.

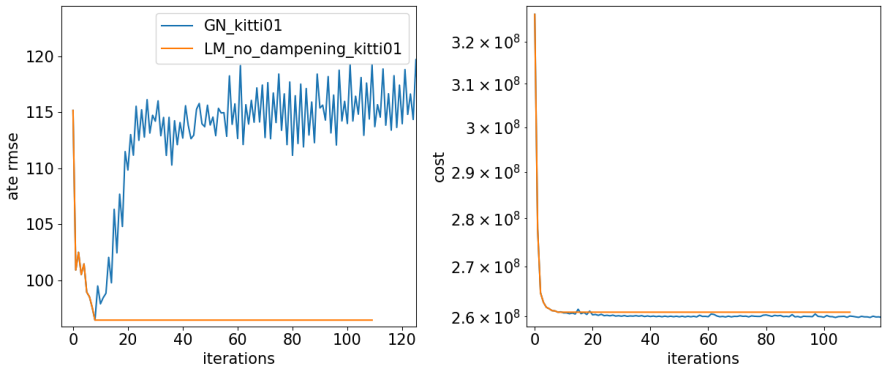
Whether or not the landmark part of the Hessian is damped does not play a significant role, except when no camera dampening is performed. This exception is due to the fact that in the case of no dampening on neither poses nor landmarks, we actually perform a modified GN algorithm: Since there is no $\lambda$ it is similar to GN, but we still enforce a cost decrease for every step as in LM. The pure GN method without enforcing a cost decrease, results in even higher ATE, namely $ATE_{rmse,geo} = 0.779$ if landmarks are dampened, $ATE_{rmse,geo} = 0.747$ without dampening. Detailed results for pure GN vs. LM without any dampening can be found in Table 4.30.

Figure 4.19 shows the convergence of cost and ATE for three examples from Table 4.30. For Kitti01 and Kitti08, the ATE start to completely diverge after the first 5-10 iterations in the case of pure GN (displayed in blue). Enforcing a decrease in the cost results in earlier termination of LM because it cannot find a cost decrease. This ensures that the cost and or ATE does not increase unboundedly. As an exception, GN performs better on Kitti02 than our preferred LM method. Note that the cost and ATE are generally noisier for GN. Overall, we prefer using the LM method since it performs better on average and does not result in huge final ATE because of wrong update steps.

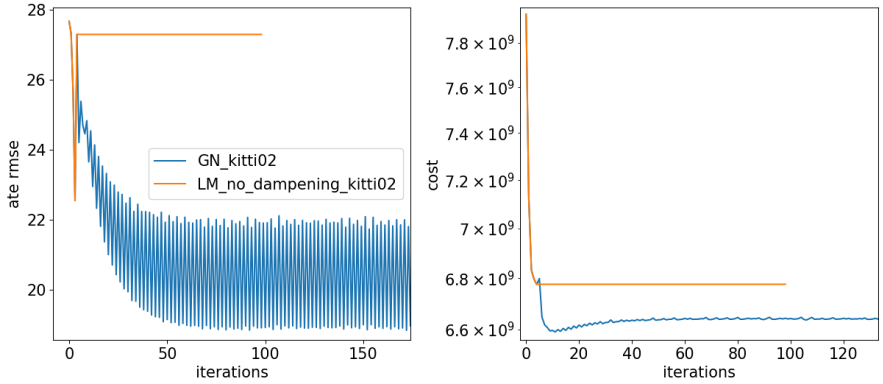
We can generally conclude, that the most important feature of our non-linear optimization algorithm is the cost decrease check. It does not matter whether the true non-linear cost or the intermediate quadratic costs are used for this check, see Section 4.10. We can also conclude that applying at least some dampening to the LM Hessian does help, but it is not very crucial which kind of dampening matrix is used. It is sufficient to use the Schur complement, or alternatively to only dampen the landmarks which is computationally very cheap as well. This behaviour can be explained by the fact that the dampening only plays a significant role once $\lambda$ is large, i.e. when many iterations have been performed. However, once we have performed between 10 to 30 iterations, in most cases the cost and ATE are very close to their final value. Therefore, using the dampening only plays a role in minor improvements at the end.

| | init_pgo | GN_no_dampening | LM_no_dampening |
|---|---|---|---|
| carlacircle | *0.024* | **0.014** | **0.014** |
| eurocMH01 | *0.034* | **0.015** | **0.015** |
| eurocMH02 | *0.019* | **0.014** | **0.014** |
| eurocMH03 | *0.087* | **0.042** | **0.042** |
| eurocMH04 | **2.669** | 2.879 | *2.828* |
| eurocMH05 | **0.070** | 0.074 | *0.073* |
| eurocV101 | *0.040* | **0.033** | **0.033** |
| eurocV102 | 0.058 | *0.042* | **0.038** |
| eurocV103 | *1.218* | 2.197 | **1.202** |
| eurocV201 | *0.026* | **0.014** | **0.014** |
| eurocV202 | **0.053** | 0.056 | *0.054* |
| eurocV203 | **1.318** | *3.055* | **1.318** |
| kitti00 | 11.217 | *5.330* | **5.223** |
| kitti01 | *115.157* | 121.917 | **96.376** |
| kitti02 | 27.671 | **18.764** | *27.297* |
| kitti03 | 2.916 | *1.106* | **0.896** |
| kitti04 | 1.149 | **0.737** | *0.752* |
| kitti05 | 5.123 | *3.371* | **3.352** |
| kitti06 | 14.593 | **5.690** | *5.736* |
| kitti07 | 3.079 | **1.311** | *1.321* |
| kitti08 | **130.317** | *262.354* | **130.317** |
| kitti09 | 75.914 | **57.538** | *62.812* |
| kitti10 | 17.238 | **10.098** | *12.713* |
| $ATE_{rmse,\,geo}$ | 1.00 | 0.747 | **0.683** |

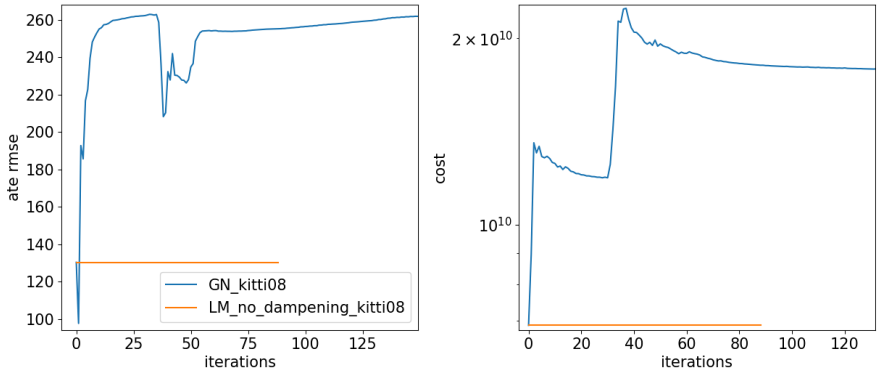
Table 4.30: **Detailed results for pure GN vs. LM without any dampening.** The only difference between the two versions is that the LM algorithm terminates if we cannot decrease the cost with the current update step. LM terminates after approximately 100 iterations on average, whereas GN reaches the maximum number of iterations. Both versions perform similar for most sequences, but GN produces significantly higher ATE on a few sequences, e.g. eurocV103, eurocV203, kitti01, kitti03, kitt08. Hence, we recommend using LM, preferably with some kind of dampening to achieve even better averaged ATE, see Table 4.29.




(a) Kitti01




(b) Kitti02




(c) Kitti08

Figure 4.19: **Convergence behaviour of pure GN (blue) vs. LM without any dampening (orange).** For Kitti01 and Kitt08, pure GN produces a large ATE, for Kitt02 it is slightly better. We prefer the LM method because it performs better on average and does not result in huge final ATE because of wrong update steps such as the GN method.

## 4.12 Pre-Processing: Geometric Occlusion Detection

**Motivation**
Most presented methods have not shown a major improvement in terms of final ATE. We assume that this is because PBA optimization is quite stable in itself, i.e. agnostic to the proposed changes. In general, PBA heavily depends on the initialization. In this experiment, we implement a basic occlusion detection algorithm as pre-processing step before PBA to improve its initialization.

**Experiment**
Our occlusion detection algorithm is visualized in Figure 4.20 for a PBA problem consisting of only two frames $A$ and $B$. The algorithm proceeds as follows:

1. For each frame, we determine the central pixel of all host and all target points. In Figure 4.20, the algorithm is currently working on frame $A$, where points $A1$, $A2$ are hosted and the target points $B1'$, $B2'$, $B3'$ are projected. We have implemented a parallelized version across frames in our manual solver.

2. Inside the current frame $A$, we check the 2D distance in pixels of each *host-target* and each *target-target* pair. A host-target pair consists of a hosted point in $A$ and a reprojected point $BX'$. A target-target pair consists of two non-identical target points which are reprojected from any frame into frame $A$. Pairs which are closer than the threshold *min_dist_pixels* proceed to next step, in the example pair $< A2, B3' >$ and pair $< B1', B2' >$. In Figure 4.20 these pairs are enclosed by a blue ellipse.

3. Now, we check the distance of the point in 3D relative to the host camera. In case of a close host-target pair $< AY, BX' >$, we mark the target point as occluded if its distance to the host camera $dist(BX')$ is greater by a factor of $dist\_factor$ than the hosted point's distance $dist(AY')$, i.e. $dist(BX') > dist\_factor * dist(AY')$.

   This scenario is illustrated in Figure 4.20 for $B3'$. We do not mark hosted points as occluded. Similarly, in the case of a close target-target pair such as $< B1', B2' >$, we mark the point which is more distant to the host camera by the factor $dist\_factor$. In the example, both $B1'$ and $B2'$ have similar distances, hence we do not mark any point as occluded.

4. All points marked as occluded are removed from the list of observations before running PBA. Since we require a minimum number of observations for a landmark when setting up our PBA routine, some landmarks might be completely removed due to our occlusion detection algorithm.

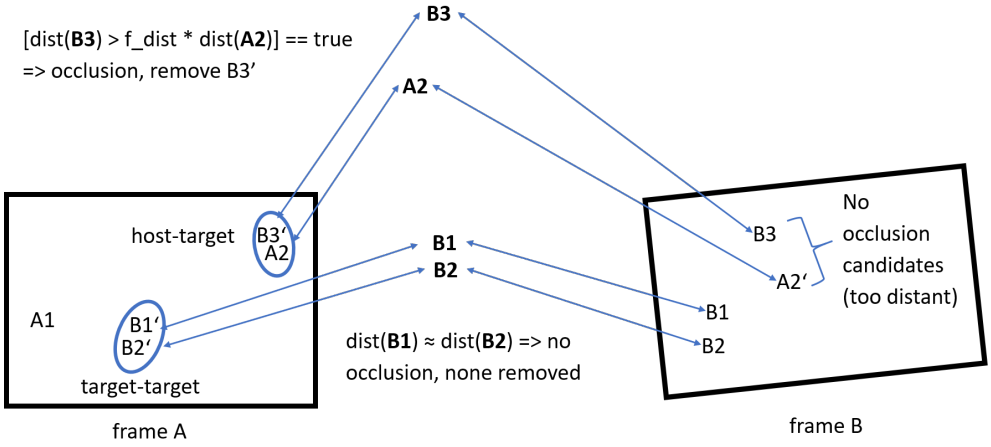



Figure 4.20: **Occlusion detection algorithm in frame A visualized.** Frame A hosts point A1 and A2, frame B hosts B1, B2, B3. Bold letters indicate points in 3D. The dash indicates reprojected points, for example B1′ is the projection of **B1** onto frame A. To decrease the runtime of our geometric occlussion detection, the algorithm is parallelized across frames.

**Results**

We found that using $dist\_factor \approx 1.04 \sim 1.07$ and $min\_dist\_pixels \approx 4 \sim 6$ gives the best averaged result. We use the DSO pattern. Note that we only calculate the pixel distance between central pixels. Therefore, the distance $min\_dist\_pixels$ needs to be increased for larger pattern sizes. Even when the central pixels are distant from each other, large patterns might still overlap.

| $[dist\_factor, min\_dist\_pixels]$ | None | $[1.05, 6]$ | $[1.05, 4]$ | $[1.07, 6]$ |
|---|---|---|---|---|
| all sequences | 0.672 | **0.661** | 0.662 | 0.665 |
| euroc-ok | 0.684 | 0.673 | 0.672 | **0.669** |
| euroc-fail | 1.015 | **1.013** | 1.02 | 1.020 |
| kit-no-loop | **0.652** | 0.701 | 0.684 | 0.701 |
| kit-loop | 0.540 | **0.508** | 0.516 | 0.531 |

Table 4.31: **Geometric occlusion detection results. None denotes the results without applying our occlusion detection algorithm.** Our occlusion detection algorithm can slightly improve the averaged ATE. We use the Huber loss function with global $\tau = 5.0$. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over respective sequences.

Looking at the result in Table 4.31, it becomes evident that our occlusion detection algorithm mostly improves the sequences containing loop closures. This can be explained by the fact that most sequences without loop closures usually do not contain as many simple occlusions of the kind described in this section above.

Figure 4.21 shows the alignment of trajectories with and without occlusion detection algorithm on Kitti00. Even though ATE varies a lot on Kitti00, qualitatively there is only a slight improvement visible in the alignment. It is a general issue, that the ATE is not a perfect measure for global consistency. This can further be seen in Figure 4.22, where both our occlusion detection algorithm and the comparison without occlusion detection fail. In both cases, the trajectory contains many more discontinuities which are neither present in the ground truth nor in init_pgo. By separating our dataset into the four categories of euroc-ok, euroc-fail, kit-loop and kit-no-loop, we alleviate this problem slightly. In further studies it would be very advisable to also include other metrics into the *global consistency* evaluation, e.g. a point to plane distance for the point cloud as proposed in DSM [63].

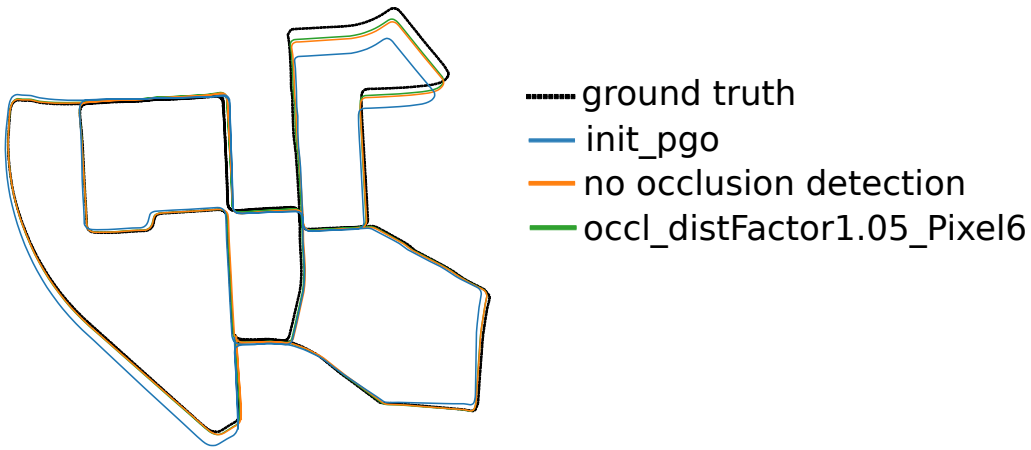



Figure 4.21: **Aligned trajectories after geometric occlusion detection on Kitt00.** Even though our occlusion detection algorithm yields only 4.563 meters final ATE instead of 5.216 meters without occlusion detection, both trajectories look similar after projection to the xy-world plane. From this we can conclude that it is not sufficient to simply compare ATE for different configurations. We use $dist\_factor = 1.05$ and $min\_dist\_pixels = 6$

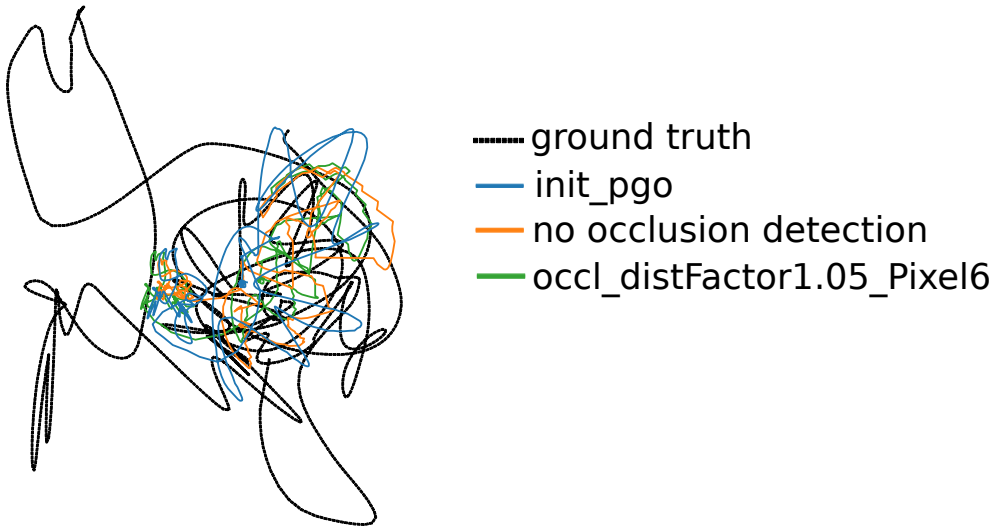



Figure 4.22: **Aligment after geometric occlusion detection on eurocV103.** We use $dist\_factor = 1.05$ and $min\_dist\_pixels = 6$. This plot shows the general problem that (averaged) ATE is not the only metric which should be used to determine global consistency: On eurocV103, using our occlusion detection algorithm yields lower ATE, even though both trajectories fail. By dividing our dataset into euroc-ok, euroc-fail, kit-loop and kit-no-loop, we can compensate for such cases to a certain extent.

## 4.13 Pre-Processing: Photometric Occlusion Detection Using ZNCC

**Motivation**

In this experiment, we use the ZNCC as a score between the the $N_p$ pixels of the residual pattern in the host and in the target. Based on the ZNCC, we decide whether or not to remove the observation before calling the PBA routine. If the ZNCC falls below *zncc_min*, we mark the target observation as occluded and remove it. Similar to the geometric occlusion detection algorithm in Section 4.12, we remove only the corresponding target observations.

**Results**

For our standard pattern from DSO consisting of 8 pixels (see Figure 4.11), it is very difficult to find a suitable threshold. After an extensive parameter search for *zncc_min* we can only present very minor improvements. The ATE is quite agnostic for any $0 < zncc\_min < 0.9$ as can be seen in Table 4.32.

| *zncc_min* | none | 0.1 | 0.5 | 0.75 | 0.9 |
|---|---|---|---|---|---|
| all sequences | 0.672 | 0.673 | 0.671 | 0.674 | **0.670** |

Table 4.32: **Photometric occlusion detection for DSO pattern with** $0 < zncc\_min < 0.9$**.** ZNCC between $0 < zncc\_min < 0.9$ barely influences the result. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

Interestingly, the ATE starts to deteriorate once the threshold is chosen above 0.9, see Table 4.33. This is because too many observations are removed and the optimization becomes less constrained.

| *zncc_min* | none | 0.925 | 0.95 | 0.975 | 0.985 | 0.995 |
|---|---|---|---|---|---|---|
| all sequences | **0.672** | 0.674 | 0.677 | 0.689 | 0.710 | 0.798 |

Table 4.33: **Photometric occlusion detection for DSO pattern with** $zncc\_min > 0.9$**.** ZNCC larger than 0.9 deteriorates the results. $ATE_{rmse,geo}$ relative to init_pgo, geometric mean over all sequences.

From these results, we suspect that for a residual pattern consisting of only 8 pixels, the ZNCC cannot be estimated reliably. Therefore, no suitable threshold can be found for such a small pattern. We additionally performed the photometric occlusion detection on a larger pattern of 25 pixels (9x9 sparse pattern in Figure 4.11). Here it has proven to be easier to find a suitable threshold. We can decrease the $ATE_{rmse,geo}$ from 0.738 to 0.722 by using $zncc\_min = 0.9$.

To conclude, our photometric occlusion detection algorithm only works on larger patterns. A pattern size of 8 pixels is too small to define a threshold, because the ZNCC estimate is not reliably enough. Note that the photometric occlusion detection is computationally very efficient, see Table 4.34. This table shows the average runtime per sequence for the photometric and geometric occlusion detection algorithms. It also shows the average runtime for the complete PBA procedure. Note that the total runtime contains all pre- and postprocessing steps which are not necessary when using our PBA pipeline in a real application. For example, we align our estimated poses in every iteration to the ground truth poses and compute the ATE. Also, we compute and output many intermediate debugging variables. Removing the runtime options which are used for development is also expected to decrease total runtime significantly. Furthermore, in a real SLAM application we can limit the number of iterations since in most scenarios the first few iterations decrease the cost and ATE by far the most. The total runtime is expected to drop at least by a factor of 3 to 4 when the proposed measures are taken.

| | total PBA | geometric (parallel) | photometric |
|---|---|---|---|
| ∅ runtime per sequence | 735 s | 10.13 s | 0.24 s |

Table 4.34: **Average runtime per sequence for our occlusion detection algorithms in seconds.** The geometric version is parallelized and still takes around 40 times longer than the simple photometric occlusion detection. The total PBA runtime includes all pre- and postprocessing steps as well as computation of ATE in every iteration.

# 5 Conclusions

In this thesis, we have evaluated numerous modifications of the PBA formulation. Overall, PBA has proven to be quite agnostic to many of the proposed changes. However, we could show that the interpolation schemes in the target image significantly alter the optimization. Our current recommendation is to use a smooth interpolation for the first few iterations followed by an exact interpolation. Bicubic interpolation has a slight advantage in terms of final ATE but is more expensive to compute than bilinear interpolation.

We could not improve final ATE by optimizing the 3D orientation of each residual pattern individually, parameterized by a normal vector. The ATE increase when first optimizing the normal vectors with fixed poses and landmarks, followed by an optimization of poses and landmarks with fixed normals. The ATE also increases when performing a joint optimization of normal vectors, poses and landmarks. However, we found that by first performing regular PBA followed by a normal vector optimization as post-processing step, the residual patterns align parallel to underlying planar geometry. This was qualitatively confirmed on the test data set, where the residual patterns align parallel to the road or the house walls.

In order to compensate for challenging scene illumination or unknown exposure times, it is clearly beneficial to choose a residual formulation which takes into account affine brightness changes to some extent. There is no noticeable advantage to optimize these affine brightness parameters explicitly using $\mathbf{r}_{ab}$ versus implicitly modelling them using for example $\mathbf{r}_{lssd}$. We expect to see further improvements if the camera response function $\gamma()$ is included as in DSO [21]. Modelling the vignetting has not shown a significant improvement.

We found that it is sufficient to use the first order Taylor expansion of the warp function as an approximation. Likewise, it is sufficient to approximate the motion Jacobians using the central pixel only. These approximations hold for a residual pattern of common size. The optimization degrades using these approximations once the residual pattern is unreasonably large, e.g. covering more than 13x13 pixels.

Finding suitable tuning constants for the robust loss functions is difficult. Satisfactory results can be achieved when averaging the estimated tuning constants over all datasets. This is relevant for applications where data set characteristics are known a priori. The lowest averaged ATE when using the robust Huber loss is achieved by setting a global

Huber parameter of $\tau = 5.0$. The lowest averaged ATE when using the t-distribution is achieved when determining the weights via self-tuning approach proposed in [2]. We generally recommend to use the self-tuning approach because it selects the robust weights in a data-adaptive manner and it can easily be used for different robust loss functions as well. For the case of estimating the tuning constants per target frame as in [63] or [45], we derived a corrected robust weight formulation for the t-distribution. We found that it makes a major difference to use our corrected formula in some scenarios, e.g. notably on a costly pre-processed photometric map of the EurocV101 sequence, see figure 4.12.

Furthermore, we showed that in the case of robust loss functions, the second order Triggs correction applied to the LM Hessian can improve the averaged ATE. If the robust loss is based on the t-distribution, the Triggs correction appears to be beneficial, even in the first few iterations of the optimization. In the case of a robust Huber loss, the Triggs correction degrades the results. We suspect this degradation is because the second derivative is zero in the inlier region and only outlier observations contribute to the correction of the Hessian. Therefore, the second derivative of the loss function needs to be analyzed before using the Triggs correction.

We could show that it is not extremely relevant which particular matrix is used to dampen the Hessian in the LM algorithm. It is sufficient to only dampen the landmarks based on the identity matrix or to alternatively use the diagonal of the Schur matrix to dampen the camera block. Most importantly, it is not required to compute and store the diagonal of the original Hessian. However, the check whether or not the current update step actually decreases the cost in the LM routine is crucial. It seems irrelevant for PBA whether the original non-linear cost $\rho(\mathbf{r}) = \sum_{i=1}^{N} \rho(\|\boldsymbol{r}_i(\boldsymbol{\theta})\|^2)$ or the intermediate OLS costs $C(\boldsymbol{r}(\boldsymbol{\theta})) = \mathbf{r}_{lin}(\boldsymbol{\theta})^T \boldsymbol{W} \mathbf{r}_{lin}(\boldsymbol{\theta})$ of the IRLS formulation are evaluated.

Lastly, we implemented simple occlusion detection algorithms before setting up the PBA. Our geometric occlusion detection algorithm improves performance slightly. The proposed photometric occlusion detection algorithm based on ZNCC only results in a performance improvement if larger residual patterns are used. If the residual pattern is too small, no meaningful threshold can be found since ZNCC is not estimated reliably from only a few pixels.

**Future work** can be divided into two directions: Similar to this thesis, further modifications which could improve the PBA formulation should be investigated. One idea is to systematically work out the numerical properties of our manual solver. We cannot fully explain the solver's behaviour in some cases. For example, in the case of using the Identity matrix for LM dampening, we have expected to see worse results compared to using the original Hessian matrix, because the Identity is scale agnostic. We suspect that this unexpected result might be due to numerical properties in the

solver. It could also be beneficial to improve the updating strategy of the dampening parameter $\lambda$ which is currently increased in every unsuccessful iteration. One possible starting point to investigate the numerical characteristics is to try closing the gap between our manual solver implementation and our ceres-solver implementation. Both solvers behave slightly different even though they optimize the same cost. By choosing the bicubic interpolation methods for either solver, the gap between them has already been reduced. Remaining differences include the updating strategy of $\lambda$, the linear solver implementation and the evaluation order when setting up the OLS equation system. Furthermore, it might be fruitful to systematically determine the relationship between the formulated cost function and our actual quantity of interest, i.e. the ATE (and map quality). Is it possible to only apply these cost updates which do not alter the ATE for the worse? How can the robustness of ATE be increased? To achieve this, it is possibly necessary to use more sophisticated metrics which also take local consistency and landmark accuracy into account.

Other promising research topics to improve PBA are more advanced occlusion and deduplication algorithms. We expect to see a substantial performance gain if occlusions are avoided. The proposed occlusion detection algorithms do not seem ideal yet. Deduplication algorithms could reduce the number of redundant landmarks in 3D space and make the optimization faster and possibly more accurate. To make the results more significant, the system should be evaluated on an even larger collection of different data sets. It is possible to evaluate our method on any kind of VSLAM data set where a sufficiently accurate initialization using VO and PGO can be achieved. It is also possible to extend our PBA routine to a stereo formulation for stereo data sets. Furthermore, it would be very useful to perform quantitative structure evaluation, for example using the metrics proposed in [63]. An analysis of motion **and** map accuracy could provide new insights into how well PBA performs on both tasks. Another line of work could be to compare the current PBA formulation or parts of it against alternative methods using for example deep learning or traditional feature-based BA.

The alternative direction of **future work** is to integrate our PBA solver into a real VSLAM system. To achieve this goal, a large focus needs to be put on runtime improvements. One major runtime boost could be achieved by implementing an *inverse compositional* version of the parameter update equations, see [6], [45]. More sophisticated data structures and new parallelization techniques could also result in a significant speedup. Furthermore, we make use of many runtime options to ease algorithmic development in the current implementation. By removing these runtime options, another major speedup is expected. To actually integrate PBA into a VSLAM, many research questions are open: How many iterations of PBA should actually be performed to reach an acceptable balance of accuracy and computational time? Can the PBA routine profit from multiple optimization rounds? Should additional information

be acquired from the front-end? Which components of pre-processing should run in the PBA back-end and which algorithms are also relevant in the front-end?

# List of Figures

# List of Tables

# Bibliography


[1] S. G. J. ( stevengj) @ Massachusetts Institute of Technology. "cubature. multi-dimensional adaptive integration (cubature) in C." In: GitHub, 2020.

[2] G. Agamennoni, P. Furgale, and R. Siegwart. "Self-tuning M-estimators." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 4628–4635. DOI: 10.1109/ICRA.2015.7139840.

[3] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard. "Robust map optimization using dynamic covariance scaling." In: *2013 IEEE International Conference on Robotics and Automation*. Ieee. 2013, pp. 62–69.

[4] S. Agarwal, K. Mierle, et al. *Ceres Solver*. http://ceres-solver.org.

[5] H. Alismail, B. Browning, and S. Lucey. "Photometric Bundle Adjustment for Vision-Based SLAM." In: *CoRR* abs/1608.02026 (2016). arXiv: 1608.02026.

[6] S. Baker and I. Matthews. "Equivalence and efficiency of image alignment algorithms." In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. IEEE. 2001, pp. I–I.

[7] P. Bergmann, R. Wang, and D. Cremers. "Online Photometric Calibration of Auto Exposure Video for Realtime Visual Odometry and SLAM." In: *IEEE Robotics and Automation Letters (RA-L)* 3 (2 Apr. 2018). This paper was also selected by ICRA 18 for presentation at the conference., pp. 627–634.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer Science+Business Media, LLC, 2006. ISBN: 0387310738.

[9] J.-L. Blanco. "A tutorial on se (3) transformation parameterizations and on-manifold optimization." In: *University of Malaga, Tech. Rep* 3 (2010).

[10] M. Bosse, G. Agamennoni, I. Gilitschenski, et al. "Robust estimation and applications in robotics." In: *Foundations and Trends® in Robotics* 4.4 (2016), pp. 225–269.

[11] C. Bregler and J. Malik. "Tracking people with twists and exponential maps." In: *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231)*. June 1998, pp. 8–15. DOI: 10.1109/CVPR.1998.698581.

[12] C. E. Brown. "Coefficient of variation." In: *Applied multivariate statistics in geohydrology and related sciences*. Springer, 1998, pp. 155–157.

[13] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. "The EuRoC micro aerial vehicle datasets." In: *The International Journal of Robotics Research* (2016). DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html.

[14] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age." In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[15] J. Civera, A. J. Davison, and J. M. Montiel. "Inverse depth parametrization for monocular SLAM." In: *IEEE transactions on robotics* 24.5 (2008), pp. 932–945.

[16] R. H. Crowell and W. E. Slesnick. *Calculus with Analytic Geometry*. Dartmouth CHANCE Project, 5 January 2008.

[17] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt. "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Reintegration." In: *ACM Transactions on Graphics 2017 (TOG)* (2017).

[18] A. Delaunoy and M. Pollefeys. "Photometric Bundle Adjustment for Dense Multi-view 3D Modeling." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1486–1493.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA: An Open Urban Driving Simulator." In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[20] E. Eade. "Lie groups for 2d and 3d transformations." In: (2013).

[21] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry." In: *arXiv:1607.02565*. July 2016.

[22] P. J. Fleming and J. J. Wallace. "How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results." In: *Commun. ACM* 29.3 (Mar. 1986), pp. 218–221. ISSN: 0001-0782. DOI: 10.1145/5666.5673.

[23] X. Gao, R. Wang, N. Demmel, and D. Cremers. "LDSO: Direct Sparse Odometry with Loop Closure." In: *CoRR* abs/1808.01111 (2018). arXiv: 1808.01111.

[24] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[25] B. K. P. Horn and B. G. Schunck. "Determining Optical Flow." In: *ARTIFICAL INTELLIGENCE* 17 (1981), pp. 185–203.

[26] C. Kerl. "Odometry from RGB-D Cameras for Autonomous Quadrocopters." MA thesis. Germany: Technical University Munich, Nov. 2012.

[27] C. Kerl, J. Sturm, and D. Cremers. "Dense Visual SLAM for RGB-D Cameras." In: *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*. 2013.

[28] C. Kerl, J. Sturm, and D. Cremers. "Robust Odometry Estimation for RGB-D Cameras." In: *International Conference on Robotics and Automation (ICRA)*. May 2013.

[29] R. Keys. "Cubic convolution interpolation for digital image processing." In: *IEEE transactions on acoustics, speech, and signal processing* 29.6 (1981), pp. 1153–1160.

[30] E. Kreyszig. *Advanced Engineering Mathematics, 10th Eddition*. 2009.

[31] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. "g 2 o: A general framework for graph optimization." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3607–3613.

[32] Y. Latif, C. Cadena, and J. Neira. "Robust Loop Closing Over Time." In: July 2012. DOI: 10.15607/RSS.2012.VIII.030.

[33] G. H. Lee, F. Fraundorfer, and M. Pollefeys. "Robust pose-graph loop-closures with expectation-maximization." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 556–563.

[34] B. D. Lucas, T. Kanade, et al. "An iterative image registration technique with an application to stereo vision." In: (1981).

[35] J. N. Lyness. "Notes on the Adaptive Simpson Quadrature Routine." In: *J. ACM* 16.3 (July 1969), pp. 483–495. ISSN: 0004-5411. DOI: 10.1145/321526.321537.

[36] Y. Ma, S. Soatto, J. KoseckÃ1/2, and S. S. Sastry. *An Invitation to 3-D Vision, volume 26 of Interdisciplinary Applied Mathematics*. 2004.

[37] H. Matsuki, L. von Stumberg, V. C. Usenko, J. Stückler, and D. Cremers. "Omnidirectional DSO: Direct Sparse Odometry with Fisheye Cameras." In: *CoRR* abs/1808.02775 (2018). arXiv: 1808.02775.

[38] S. Mattoccia, F. Tombari, and L. Di Stefano. "Reliable rejection of mismatching candidates for efficient ZNCC template matching." In: *2008 15th IEEE International Conference on Image Processing*. 2008, pp. 849–852.

[39] G. P. Meyer. "An Alternative Probabilistic Interpretation of the Huber Loss." In: *arXiv preprint arXiv:1911.02088* (2019).

[40] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System." In: *CoRR* abs/1502.00956 (2015). arXiv: 1502.00956.

[41] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[42] D. Nistér, O. Naroditsky, and J. Bergen. "Visual odometry." In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 1. Ieee. 2004, pp. I–I.

[43] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[44] E. Olson and P. Agarwal. "Inference on networks of mixtures for robust robot mapping." In: *The International Journal of Robotics Research* 32.7 (2013), pp. 826–840.

[45] P. R. Palafox. "Local Tracking and Mapping for Direct Visual SLAM." In: 2019.

[46] B. Pan, H. Xie, and Z. Wang. "Equivalence of digital image correlation criteria for pattern matching." In: *Appl. Opt.* 49.28 (Oct. 2010), pp. 5501–5509. DOI: 10.1364/AO.49.005501.

[47] K. Petersen, M. Pedersen, et al. "The Matrix Cookbook, vol. 7." In: *Technical University of Denmark* 15 (2008).

[48] username: PureFox. *Numerical integration/Adaptive Simpson's method*. 30. September 2018.

[49] I. Sarantopoulos. "Implementing a Multivariate Normal Distribution in C++." In: 2017.

[50] C. Scheffler. "A derivation of the EM updates for finding the maximum likelihood parameter estimates of the Student's t distribution." In: *Technical note. URL www. inference. phy. cam. ac. uk/cs482/publications/scheffler2008derivation. pdf* (2008).

[51] N. Sünderhauf and P. Protzel. "Switchable constraints vs. max-mixture models vs. RRR - A comparison of three approaches to robust pose graph SLAM." In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 5198–5203.

[52] N. Sünderhauf and P. Protzel. "Towards a robust back-end for pose graph slam." In: *2012 IEEE international conference on robotics and automation*. IEEE. 2012, pp. 1254–1261.

[53] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[54] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. "Bundle adjustment—a modern synthesis." In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[55] V. Usenko, N. Demmel, D. Schubert, J. Stueckler, and D. Cremers. "Visual-Inertial Mapping with Non-Linear Factor Recovery." In: *IEEE Robotics and Automation Letters (RA-L) And Int. Conference on Intelligent Robotics and Automation (ICRA)* 5.2 (2020). <a href="https://arxiv.org/abs/1904.06504" target="blank">[arxiv]</a>, pp. 422–429. DOI: 10.1109/LRA.2019.2961227.

[56] V. C. Usenko, N. Demmel, and D. Cremers. "The Double Sphere Camera Model." In: *CoRR* abs/1807.08957 (2018). arXiv: 1807.08957.

[57] N. D. Vladyslav Usenko (user: VladyslavUsenko). "basalt-mirror." In: GitHub, 2020.

[58] L. Wang, C. Zheng, W. Zhou, and W.-X. Zhou. "A new principle for tuning-free Huber regression." In: *Preprint* (2018).

[59] R. Wang, M. Schworer, and D. Cremers. "Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3903–3911.

[60] C. Zach. "Robust bundle adjustment revisited." In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[61] C. Zach and G. Bourmaud. "Descending, lifting or smoothing: Secrets of robust cost optimization." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 547–562.

[62] Z. Zhang. "Parameter estimation techniques: A tutorial with application to conic fitting." In: *Image and vision Computing* 15.1 (1997), pp. 59–76.

[63] J. Zubizarreta, I. Aguinaga, and J. M. M. Montiel. "Direct Sparse Mapping." In: *CoRR* abs/1904.06577 (2019). arXiv: 1904.06577.

# Glossary

**EM** The EM-algorithm is an iterative algorithm which is generally applied to find the maximum likelihood of probability models containing latent (un-observable) variables [8, page 439]. 11, 19, 74

**geometric mean** To compare performance measures relative to a given base performance, the geometric mean should be employed [22]. For example, in this thesis we compare the geometric average of $ATE_{rmse}$ over multiple sequences of different methods relative to our initialization. Refer to equation 1.4 for details. 6, 30, 35, 50, 81

**irradiance constancy** If two cameras observe a 3D point from different vantage points, the received irradiances (in Watts per square meter) in the cameras at the respective pixel positions are the same [21]. 3, 8

**lambertian** Objects which show the same brightness independent of the observer's viewing angle [7]. 4, 16

**photoconsistency** If two identical cameras observe a 3D point from different vantage points, the response (pixel value) in the cameras at the respective pixel position is the same [27]. The cameras need to use same intrinsic calibration as well as the same exposure time. This assumption only holds for lambertian objects which show the same brightness independent of the viewing angle [7]. 2, 3, 8

**residual pattern** The set of $N_p$ points which belong to the same landmark in 3D space. Points of the same residual pattern share the inverse distance (depth) parameter $id$. In the case of inverse distance parameterization, the unprojected residual pattern in 3D lies on a sphere. In the case of inverse depth parameterization, the unprojected residual pattern forms a plane which is parallel to the image sensor. We also refer to the residual pattern as patch. 4, 5, 10, 16, 28, 30, 32, 50, 51, 56, 57, 58, 88, 91

**Schur complement** A method for solving a linear system of a special structure, which decomposes the system into two smaller subsystems [54]. By applying the Schur complement to our PBA Hessian, we can solve the *reduced camera system* to obtain the update of the poses. To obtain the update of the landmarks, we still need to invert a large system. However, this is very efficient because inverting a diagonal matrix can be performed by inverting the all entries individually. 22, 23, 80

**Triggs correction** A method for solving a robust cost function more exact than standard least-squares solvers, introduced in [54]. In particular, the Hessian matrix deviates from the standard IRLS approach by an additive factor, which leads to different convergence behaviour. The modified Hessian can be implemented by rescaling of Jacobians and residuals. The method converges to the same minimzer as IRLS. Further details can be found in [60], [61]. An exemplary implementation with improved runtime is given in the Ceres solver [4]. 24, 69

**warp** In the context of this thesis, the warping function describes the transformation chain between 2D pixels of host and target image. This includes the unprojection from the host image to 3D, a rigid body transformation in 3D, and the reprojection from 3D to the target image [27]. We evaluate approximations to this warp function for PBA in section 4.5. 3, 5, 10, 28, 29, 32, 44, 51, 56, 91

# Acronyms

**ATE** absolute trajectory error. 6, 25, 26, 33, 47, 48, 53, 71, 72, 82

**BA** Bundle Adjustment. 2, 9, 10, 12, 15

**DOF** degree of freedom. 10, 14, 15, 32

**DSM** Direct Sparse Mapping. 9, 10, 28, 87

**DSO** Direct Sparse Odometry. 2, 4, 9, 44, 50, 55, 57, 58, 86, 88, 91

**GN** Gauss-Newton. 20, 22, 23, 81, 82

**IRLS** iteratively reweighted least squares. 16, 17, 21, 24, 92, 105

**LDSO** Direct Sparse Odometry with Loop Closure. 4, 9, 10, 22, 26

**LM** Levenberg-Marquardt. 20, 21, 23, 48, 54, 71, 72, 79, 80, 81, 82, 92

**MAD** Median Absolute Deviation. 19, 62, 73

**NLS** non-linear least squares. 17, 20, 21, 23, 79

**OLS** linear or ordinary least squares. 17, 21, 22, 79, 92, 93, 98

**PBA** Photometric Bundle Adjustment. 2, 3, 8, 9, 10, 15, 16, 18, 21, 22, 23, 24, 26, 28, 33, 43, 49, 52, 55, 62, 70, 75, 79, 85, 88, 89, 91, 92, 93, 105

**PGO** Pose Graph Optimization. 4, 6, 8, 9, 11, 93

**RMSE** root mean squared error. 6

**SLAM** Simultaneous Localization and Mapping. 1, 9, 44, 70, 79, 89

**VO** visual odometry. 1, 2, 8, 9, 11, 17, 44, 93

**VSLAM** visual SLAM. 1, 2, 8, 9, 17, 93

**ZNCC** zero normalized cross correlation. 10, 51, 52, 54, 88