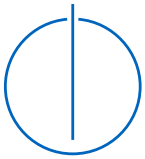
TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Master's Thesis in Informatics

Local and Global Mapping for Direct SLAM

Erkam Uyanik



TECHNICAL UNIVERSITY OF MUNICH

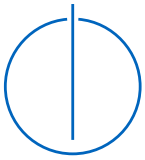
DEPARTMENT OF INFORMATICS

Master's Thesis in Informatics

Local and Global Mapping for Direct SLAM

Lokale und Globale Kartierung für Direktes SLAM

Author:	Erkam Uyanik
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Nikolaus Demmel, M.Sc.
Submission Date:	15.06.2021



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

Munich, 15.06.2021 Erkam Uyanik

Acknowledgments

I would like to thank my advisor Nikolaus Demmel for his guidance, support, and for the fruitful discussions we had. I am grateful to my supervisor Prof. Dr. Daniel Cremers for the opportunity to pursue my master's thesis in the Computer Vision Group.

I would like to extend my gratitude to everyone who has helped and supported me to reach this point in my studies. I am deeply indebted to Prof. Dr. Lale Akarun who introduced me to computer vision.

I am very grateful to my family and friends for their endless support.

Abstract

In this master's thesis, we propose Direct Sparse Mapping with Loop Closure (LDSM), a direct SLAM system with global mapping, loop closure detection, and global pose graph optimization as an extension of Direct Sparse Mapping (DSM) [1] by introducing loop closure detection and pose graph optimization.

The point selection procedure is modified to introduce repeatable feature points. Loop closures are detected with a feature-based bag of words (BoW) technique. We verify detected loop closures and estimate relative pose constraints by minimizing geometric errors. A global pose graph with relative pose constraints from loop closures and covisibility graph is optimized. The global map is then updated based on optimization results, enabling local tracking and mapping in the active window to utilize global corrections. An appearance-based verification step is added to the covisible window selection criteria to ensure reliability of covisibility graph in non-convex environments. We limit the search range of covisible window selection strategy to partly remedy the inefficiency of unbounded nature of the global map in long trajectories.

In our evaluations on public datasets, we demonstrate that the modified point selection strategy preserves the accuracy and pose graph optimization improves the performance significantly by reducing accumulated drifts in trajectories with large loops. LDSM achieves accurate results comparable to the state-of-the-art direct and indirect monocular methods.

Contents

1 Introduction

Localization of an agent given sensor inputs has been an important problem for computer vision and robotics applications such as autonomous driving and augmented reality with proposed solutions for visual odometry (VO) and simultaneous localization and mapping (SLAM) [2]. We are particularly interested in the monocular visual problem where only a single gray-scale camera data is available. While visual odometry systems estimate the trajectory, SLAM systems do this localization simultaneously with the construction of a map for the environment. Methods can be categorized as direct and indirect based on whether they use raw sensor measurements or not. Indirect methods create an intermediate representation for images to optimize a geometric error where usually correspondences of distinctive feature points are utilized. Direct methods, on the other hand, use raw pixel values and optimize a photometric error. While indirect methods have been more popular in the past owing to their speed and robustness to distortions, they suffer in scenes without rich texture and enough distinctive points. Direct methods do not depend on characteristic features and they are more robust in such scenes. Direct methods have gained popularity recently partly because of increased computational power and better visual sensors which provide well calibrated data with less geometric distortion. Direct methods model the sensor and the distortions that keypoints are robust against can be useful and provide more information if they can be modeled. Not losing information due to a feature extraction step also provides room for direct methods to utilize underlying information better.

Direct Sparse Odometry (DSO) [3] is a direct visual odometry system that jointly optimizes motion and structure using photometric bundle adjustment (PBA). DSO combines advantages of direct approaches with sparse data. By preferring to use sparse data, DSO eliminates the computational infeasibility that would be introduced by a geometry prior in a dense approach. It also benefits from a more precise sensor model with photometric calibration. DSO was one of the early direct applications that showed direct methods can be used in real-time. As a visual odometry method, DSO only predicts the trajectory incrementally in a sliding window. By marginalizing frames and points that leave the local window, it forgets previously visited areas and cannot use the rich information from revisited scenes. This results in accumulated drift in global translation, rotation, and scale which prevents successful applications to long trajectories.

Detecting revisited scenes and updating the trajectory to better reflect correspondences between frames help reduce accumulated drift in visual odometry systems. Direct

Sparse Odometry with Loop Closure (LDSO) [4] extends DSO to a visual SLAM system with loop closure detection and pose graph optimization as a correction technique. While preserving robustness of DSO, LDSO achieves better performance by significantly reducing accumulated drift. This is partly achieved by increasing repeatability of map points and computing distinctive descriptors for a small subset of map points. Loop closure detection and verification, relative pose computation, as well as pose graph optimization steps in LDSO are similar to a typical indirect SLAM system such as ORB-SLAM [5].

Although LDSO detects and corrects loop closures, existing map points are not reused to reflect reobservations and thus reobservations are ignored in the sliding window optimization. Instead of using a temporary map and marginalizing old keyframes, retaining keyframes and maintaining a global map allows to detect reobservations and also to reuse existing map points, and thus prevents duplication of points. Direct Sparse Mapping (DSM) [1] is a visual SLAM system with a persistent map based on photometric bundle adjustment. DSM proposes a local map covisibility window (LMCW) criteria to select active keyframes from both temporally close frames and frames that are covisible with the temporal keyframes. While DSM uses the main ideas of indirect visual SLAM techniques, it lacks loop closure correction. Despite having a global map, DSM still suffers from drifts due to larger loops and reuse of keyframes and map points is limited to local mapping and smaller loops.

1.1 Contribution

DSO provides a good foundation for direct, sparse systems. LDSO extends DSO with loop closure detection and pose graph optimization but still lacks a global map. DSM proposes a direct visual SLAM system with a global map but it is prone to accumulated drifts in long trajectories without loop closure correction. Combining the main ideas of these previous methods, we propose a direct SLAM system called Direct Sparse Mapping with Loop Closure (LDSM) with global mapping, loop closure detection, and global pose graph optimization, which is an extension of DSM.

We introduce repeatable keypoints to the existing point selection mechanism and use a feature based bag of words (BoW) approach for visual place recognition. For detected loop closure pairs, we estimate a relative pose in $Sim(3)$ using 3D geometry information. This step also acts as a loop closure verification step. We apply pose graph optimization with constraints from verified loop closures and covisibility graph connections. We update the global map based on PGO results, so that local mapping also uses the optimized global state.

Our contributions are as follows:

- We propose LDSM, a direct SLAM system with global mapping, loop closure detection, and pose graph optimization by extending DSM with

  - loop closure detection based on a feature-based BoW approach
  - relative pose estimation in $Sim(3)$ using 3D geometry information
  - pose graph optimization with relative pose constraints from loop closures and covisibility graph
  - merging optimized global poses to covisibility graph
- We modify
  - point selection strategy of DSM to include repeatable feature points
  - geometry based covisible window selection strategy of DSM with addition of an appearance based verification step to prevent false covisibility connections especially for non-convex environments
  - the search range of covisible window selection strategy of DSM to reduce candidate keyframes from all inactive keyframes to the inactive neighborhood of active temporal keyframes
- We build visual debugging tools for
  - detected loop closures with aligned point clouds based on estimated relative poses or relative poses from local mapping (see Fig. 5.2)
  - pose graph optimization problem, with options to show graph before and/or after the optimization with frame connections from and/or ideal positions of the graph constraints (see Fig. 5.4)
- On commonly used datasets, we show that modified point selection strategy does not deteriorate the performance and loop closure correction significantly decreases scale drift for long trajectories with large loops.

1.2 Outline

We first provide preliminary concepts which will be relevant for the remainder in Chapter 2. These include 3D transformations and Lie groups, feature points, visual place recognition, and pose graph optimization. We also define our notations. Next, we introduce state-of-the-art for direct and indirect monocular VO/SLAM systems in Chapter 3. Chapter 4 details Direct Sparse Mapping method, including its local window selection strategy, as our work is directly related to it. Then, we explain our proposed method LDSM in Chapter 5. We detail modified point selection strategy, loop closure detection and correction steps, changes to LMCW, and pose graph optimization. We share results in Chapter 6. We conclude the thesis in Chapter 7 and discuss possible future work.

2 Preliminaries

2.1 Notation

Light lower-case letters s represent scalars. Light upper-case letters I represent functions. Bold lower-case letters $\mathbf{v}$ represent vectors and bold upper-case letters $\mathbf{J}$ represent matrices.

Camera poses are represented as transformation matrices $\mathbf{T}_i \in \mathrm{SE}(3)$ from the camera coordinate system to the world coordinate system. We denote poses with letter $\mathbf{T} \in \mathrm{SE}(3)$ or letter $\mathbf{S} \in \mathrm{Sim}(3)$. The transformation from camera i to camera j is denoted by $\mathbf{T}_{ji} = \mathbf{T}_j^{-1}\mathbf{T}_i$ or $\mathbf{S}_{ji} = \mathbf{S}_j^{-1}\mathbf{S}_i$.

$\Pi(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ denotes projection function of a 3D point $\mathbf{p}$ to image plane and $\Pi^{-1}(\mathbf{p}, d_\mathbf{p}) : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^3$ denotes unprojection function of a 2D point $\mathbf{p}$ given its inverse depth $d_\mathbf{p}$.

The set of all keyframes is denoted by K. The set of active keyframes is denoted by K_a, the set of keyframes in the temporal window is denoted by K_t while the set of keyframes in the covisible window is denoted by K_c.

Points in a frame i are denoted by ${}^i\mathbf{p}_k$ in general where index k is optionally used to denote multiple points in frame i. Optimized candidates points are denoted by ${}^i\hat{\mathbf{p}}_k$ and active points are denoted by ${}^i\tilde{\mathbf{p}}_k$ when we need to differentiate between different types of points. Points with estimated depth are denoted with ${}^i\mathbf{p}_k^*$. P_i is used to denote all active points of a keyframe I_i.

2.2 3D Geometry

2.2.1 SE(3): Rigid Body Transformations

A rigid body motion preserves the distance and orientation between any two points on a rigid body and it can be described with a rotation and then a translation on any point of the rigid body. Translation can be represented with a vector $\mathbf{t} \in \mathbb{R}^3$. We define rotation $\mathbf{R}$ as an element of special orthogonal group $SO(3)$:

$$SO(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\} \tag{2.1}$$

A rigid body motion $\mathbf{T}$ can be represented as an element of special Euclidean group $SE(3)$:

$$SE(3) = \{\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3\} \tag{2.2}$$

$\mathbf{T} \in SE(3)$ has 6 degrees of freedom, 3 for rotation and 3 for translation.

The inverse of $\mathbf{T} \in SE(3)$ has the form

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.3}$$

We can formulate the rigid body transformation for a point $\mathbf{p} \in \mathbb{R}^3$ as

$$\mathbf{p}' = \mathbf{R}\mathbf{p} + \mathbf{t} \tag{2.4}$$

Alternatively, we can use homogeneous coordinates to represent this transformation linearly as

$$\begin{bmatrix} \mathbf{p}' \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{2.5}$$

2.2.2 Sim(3): Similarity Transformations

Similarity transformations $\mathbf{S} \in Sim(3)$ are the combination of rigid body transformations $SE(3)$ with a scaling factor s:

$$Sim(3) = \{\mathbf{S} = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3, s \in \mathbb{R}_{>0}\} \tag{2.6}$$

The inverse of $\mathbf{S} \in Sim(3)$ has the form

$$\mathbf{S}^{-1} = \begin{bmatrix} s^{-1}\mathbf{R}^T & -s^{-1}\mathbf{R}^T\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.7}$$

2.2.3 Lie Algebra

$SO(3)$, $SE(3)$, and $Sim(3)$ are Lie groups which form differentiable manifolds. These manifolds have corresponding tangent spaces, called Lie algebras, derived by differentiating manifolds at the identity. This relation between Lie groups G and corresponding

Lie algebras $\mathfrak{g}$ is useful to perform some operations in tangent space. Especially, Lie algebra is used to define derivatives in Lie groups which requires a plus operator.

There are mappings $exp_G : \mathfrak{g} \to G$ and $log_G : G \to \mathfrak{g}$. There are closed forms of all $exp_{SO(3)}$, $exp_{SE(3)}$, and $exp_{Sim(3)}$. Refer to [6] for more detail with mathematical definitions.

2.3 Feature Points

Features, keypoints, or interest points, are "interesting" points in images such as corners. A corner can be defined with the property of having a neighborhood with multiple dominant directions in image gradient. Indirect methods usually use feature points as an intermediate representation for images. Feature points should be local, invariant to geometric and photometric changes, distinctive, and robust to noise. We want *repeatable* feature points, that is we want to detect mostly the same set of points for a scene in different images.

There are many feature detectors with various mechanisms, such as FAST [7] or Shi-Tomasi [8].

FAST is a computationally efficient corner detection algorithm which only checks pixels on a circle of radius r around a candidate point. If there are n contiguous pixels that are all brighter or all darker than the candidate point by at least t, it is selected as a corner. Unlike other detectors, FAST features do not have orientation information. FAST features are also not scale invariant. Shi-Tomasi is also a simple and fast extraction method.

Feature descriptors distinctively describe the region around a feature point. Descriptors are needed to compare extracted features. Ideally, descriptors should be invariant to scale, rotation, viewpoint, and illumination changes.

BRIEF [9] is a binary descriptor with simple intensity difference tests. It initially smooths the image using a Gaussian kernel. For a number of randomly selected pairs of pixels from the neighborhood of the feature point, pixel intensities are compared to create the binary descriptor. It is very easy to compare binary descriptors with hamming distance, i.e. the number of different bits, rather than using Euclidean distance. BRIEF has a big advantage in terms of speed both for descriptor computation and for matching compared to other descriptors such as SIFT [10] and SURF [11]. It also gives similar or better recognition performance compared to other methods unless invariance to large rotations is required. BRIEF is not designed to be rotation invariant. It only tolerates small amounts of rotation.

Oriented FAST and rotated BRIEF (ORB) [12] is a rotation invariant binary descriptor based on FAST detector and BRIEF descriptor. ORB computes FAST features using image pyramids to have scale invariance. ORB computes feature orientations using

intensity centroids. ORB introduces a rotation-aware version of BRIEF. ORB provides a really fast alternative to popular descriptors SIFT and SURF. The feature detector in ORB can be replaced with another fast detector such as Shi-Tomasi if wanted.

2.4 Visual Place Recognition

Recognizing a previously visited place is important to correct trajectories and reduce accumulated drift. Using appearance-based approaches is one way of place recognition [13]. For this, local feature descriptors can be used. Bag of words (BoW) based techniques such as [14] provide an efficient way to compare images using their feature descriptors. This is done by discretizing descriptor space and using data structures such as a hierarchical vocabulary tree for more efficient searches. BoW based techniques are commonly used for loop detection ([15] [5] [4]).

DBoW3 [16] is an open source library which converts a set of feature descriptors for a given image into a bag of words vector and implements a database which can be queried to get similar images for a given image representation. This is performed efficiently using the inverse index of a given word. Inverse indices keep record of keyframes in which words are seen. See Fig. 2.1 for a visualization of a vocabulary tree and direct and inverse indices that compose the database.

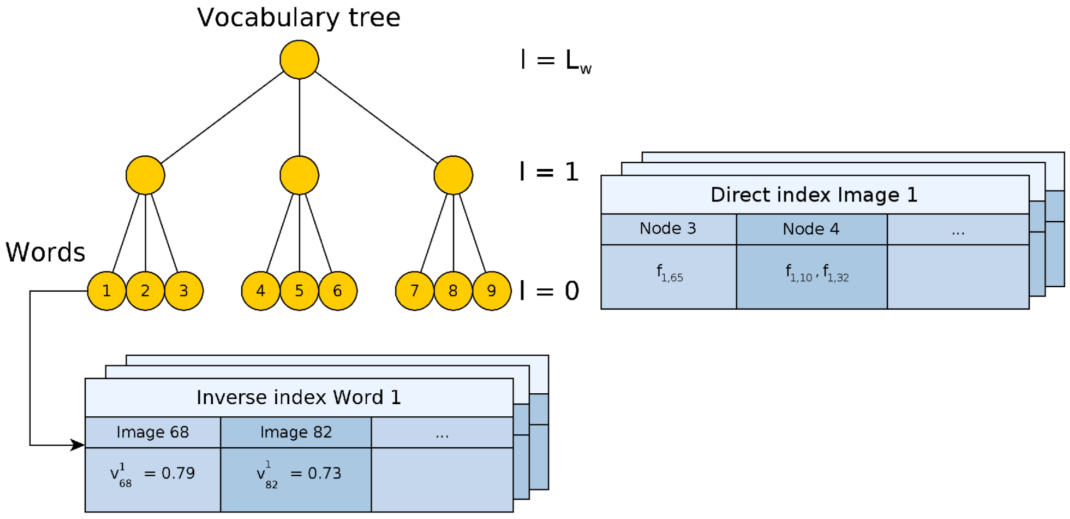


Figure 2.1: An example vocabulary tree and direct and inverse indices that compose the database. (Source [14])

A loop closure validation step usually follows loop closure detection, which applies geometric verification steps. A relative pose is also computed for the detected loop closure pair to utilize loop closure information in an optimization step such as pose

graph optimization (see Section 2.6).

2.5 Nonlinear Least Squares Optimization

Least squares method is used to approximate the solution of *overdetermined systems* where there are more observations than unknown parameters:

$$arg \min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{n} \|\mathbf{y}_i - F_i(\mathbf{x})\|^2 \tag{2.8}$$

where $F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^k$ is the underlying model, $\mathbf{y} \in \mathbb{R}^{k\times m}$ is the vector of m observations, $\mathbf{x} \in \mathbb{R}^n$ is the vector of model parameters, and $m \geq n$.

A minimum is reached when the gradient is zero. The problem has a closed form solution for linear models, but we are interested in more complex, nonlinear models. Solutions to nonlinear least squares problems use iterative approaches starting from an initial guess state $\mathbf{x}^0$, where parameters are updated in each iteration with an update $\Delta\mathbf{x}$ as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x} \tag{2.9}$$

Gradient descent is a first-order method which iterates in the opposite direction of gradient with

$$\Delta\mathbf{x} = -\gamma\nabla F(\mathbf{x}^k) \tag{2.10}$$

Optimization heavily depends on selection of step size γ.

Gauss-Newton is a second-order method where the second-order derivatives of the optimized cost are approximated using first-order derivatives of the measurement function F, with the update

$$\Delta\mathbf{x} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\Delta\mathbf{y} \tag{2.11}$$

where $\Delta\mathbf{y}_i = \mathbf{y}_i - F(\mathbf{x}^k)$.

Levenberg-Marquardt combines gradient descent and Gauss-Newton methods with a damping parameter λ with the update

$$\Delta\mathbf{x} = -(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T\Delta\mathbf{y} \tag{2.12}$$

2.6 Pose Graph Optimization

We will use pose graph optimization (PGO) to correct detected loop closures using computed relative poses from both loop closure pairs and from local odometry.

Pose graph optimization is a problem of optimizing camera poses $\mathbf{S}_i \in Sim(3)$ with relative pose constraints $\mathbf{S}_{ij} \in Sim(3)$. Camera poses form nodes while relative poses form graph edges.

This problem can be formulated with residuals

$$log_{Sim(3)}(\mathbf{S}_{ij} * \mathbf{S}_j^{-1} * \mathbf{S}_i) \tag{2.13}$$

for each relative pose constraint $\mathbf{S}_{ij}$ where camera poses $\mathbf{S}_i$ and $\mathbf{S}_j$ are optimized and $log_{Sim(3)}$ maps value from $Sim(3)$ to $\mathbb{R}^7$.

This is a nonlinear optimization problem commonly solved using iterative optimization techniques such as Gauss-Newton or Levenberg-Marquardt. Although we use Ceres [17] to solve this graph optimization problem, the paper of g2o library [18] is a good reference about general graph optimization problem with details such as use of graph sparsity.

The system is under-determined, i.e. the total error is invariant to any $Sim(3)$ transformation that is applied to all poses. To solve this, one or more nodes should be fixed during the optimization. In our method, we fix the pose of the first camera during PGO.

2.6.1 Effect of Scale Changes to Relative Poses

When we run pose graph optimization, camera poses $\{\mathbf{S}_i\}$ are optimized and this includes their scales $\{s_i\}$ as well. Since we keep the camera poses in the global map in $SE(3)$ as $\{\mathbf{T}_i\}$, after PGO we need to convert the optimized $Sim(3)$ poses back to $SE(3)$ and reflect scale changes to other data, namely the points $\{{}^i\mathbf{p}_k\}$ and also the relative pose measurements $\{\mathbf{S}_{ij}\}$.

Ideally, the changes due to scales should not change the PGO problem since we only move scale information from camera poses to points and relative pose constraints. For this, we check how we can keep the PGO residual the same after the changes.

We have the original measurements $\{\mathbf{S}_{ij}\}$, optimized $Sim(3)$ poses $\{\mathbf{S}_i\}$, and final residuals after PGO in Eq. 2.14. We use hat notation $\hat{}$ for the variables after scale

normalization.

$$\begin{aligned}
\mathbf{S}_{ij} * \mathbf{S}_j^{-1} * \mathbf{S}_i &= \begin{bmatrix} s_{ij}\mathbf{R}_{ij} & \mathbf{t}_{ij} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} s_j^{-1}\mathbf{R}_j^T & -s_j^{-1}\mathbf{R}_j^T\mathbf{t}_j \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} s_i\mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0}^T & 1 \end{bmatrix} \\
&= \begin{bmatrix} s_{ij}\frac{s_i}{s_j}\mathbf{R}_{ij}\mathbf{R}_j^T\mathbf{R}_i & \frac{s_{ij}}{s_j}\mathbf{R}_{ij}\mathbf{R}_j^T(\mathbf{t}_i - \mathbf{t}_j) + \mathbf{t}_{ij} \\ \mathbf{0}^T & 1 \end{bmatrix} \\
&\overset{!}{\approx} \begin{bmatrix} \hat{s}_{ij}\frac{\hat{s}_i}{\hat{s}_j}\hat{\mathbf{R}}_{ij}\hat{\mathbf{R}}_j^T\hat{\mathbf{R}}_i & \frac{\hat{s}_{ij}}{\hat{s}_j}\hat{\mathbf{R}}_{ij}\hat{\mathbf{R}}_j^T(\hat{\mathbf{t}}_i - \hat{\mathbf{t}}_j) + \hat{\mathbf{t}}_{ij} \\ \mathbf{0}^T & 1 \end{bmatrix} \\
&= \begin{bmatrix} \hat{s}_{ij}\hat{\mathbf{R}}_{ij}\mathbf{R}_j^T\mathbf{R}_i & \hat{s}_{ij}\hat{\mathbf{R}}_{ij}\mathbf{R}_j^T(\mathbf{t}_i - \mathbf{t}_j) + \hat{\mathbf{t}}_{ij} \\ \mathbf{0}^T & 1 \end{bmatrix}
\end{aligned} \tag{2.14}$$

After scale normalizations, we have $\hat{s}_i = \hat{s}_j = 1$ since for next PGO camera poses $\{\mathbf{T}_i\}$ are converted to $Sim(3)$ poses $\mathbf{S}_i$ with scale $s_i = 1$. Other components of camera poses $\mathbf{S}_i$ do not change due to scale normalization. Relative rotation measurements $\hat{\mathbf{R}}_{ij}$ are also not affected by scale changes.

$$\begin{aligned}
\hat{s}_{ij}\hat{\mathbf{R}}_{ij} &\overset{!}{\approx} s_{ij}\frac{s_i}{s_j}\mathbf{R}_{ij} \\
\hat{\mathbf{R}}_{ij} &= \mathbf{R}_{ij} \\
\hat{s}_{ij} &\leftarrow s_{ij}\frac{s_i}{s_j}
\end{aligned} \tag{2.15}$$

$$\begin{aligned}
\hat{s}_{ij}\hat{\mathbf{R}}_{ij}\mathbf{R}_j^T(\mathbf{t}_i - \mathbf{t}_j) + \hat{\mathbf{t}}_{ij} &= s_{ij}\frac{s_i}{s_j}\mathbf{R}_{ij}\mathbf{R}_j^T(\mathbf{t}_i - \mathbf{t}_j) + \hat{\mathbf{t}}_{ij} \\
&= s_i\left(\frac{s_{ij}}{s_j}\mathbf{R}_{ij}\mathbf{R}_j^T(\mathbf{t}_i - \mathbf{t}_j\right) + \frac{\hat{\mathbf{t}}_{ij}}{s_i}) \\
&\overset{!}{\approx} \frac{s_{ij}}{s_j}\mathbf{R}_{ij}\mathbf{R}_j^T(\mathbf{t}_i - \mathbf{t}_j) + \mathbf{t}_{ij} \\
\hat{\mathbf{t}}_{ij} &\leftarrow \mathbf{t}_{ij}s_i
\end{aligned} \tag{2.16}$$

We derive $\hat{s}_{ij}$ using first component of the residual in 2.15. We cannot remove s_i, s_j terms completely from the translation part of the residual, since $\hat{s}_{ij}$ is already derived and value of $\hat{\mathbf{t}}_{ij}$ cannot remove s_j from the term. Although we cannot keep the translation of the residual the same, with $\hat{\mathbf{t}}_{ij}$ we derive in Eq. 2.16, we rescale the translation term of the residual with a scale factor of s_i after normalization. This way, residuals point in the same direction as before. This is an intuitive result. During PGO, the optimization changes the scale of frame i from the initialization value of 1 to s_i. Assume $s_i > 1$ w.l.o.g.

This means that the PGO optimization "realizes" that the current scale of local frame i is shorter than the world frame scale. Since the relative pose constraint $\mathbf{S}_{ij}$ is expressed in that local frame i, when we normalize the scale of frame i back to 1, we need to also scale up the relative translation $\mathbf{t}_{ij}$ by factor s_i. When we computed the relative pose constraints $\mathbf{S}_{ij}$, we did not know that the frame's scale was off compared to the world frame scale, so we computed the translation in the "wrong" scale. Now after PGO, where we have figured out the right scale compared to the other frames, we can correct the translational part of the relative poses $\mathbf{t}_{ij}$ and thus also properly scale the residual to be in the same scale as the other frames. So, we can argue that this is a better outcome than keeping the translational part of the residual the same.

3 Related Work

3.1 Indirect Approaches

For the SLAM problem, many feature-based indirect systems have been proposed such as monoSLAM [19], PTAM [20], and more recently ORB-SLAM [5]. While initially filter based solutions were preferred, keyframe based solutions are demonstrated to be more accurate than filtering [21]. We focus on ORB-SLAM as it includes good parts from previous literature.

3.1.1 ORB-SLAM

ORB-SLAM is a sparse, indirect SLAM system with local bundle adjustment, loop closure detection with bag of words, global pose graph optimization, and global bundle adjustment. An overview of the system is shown in Fig. 3.1a. Same features are used for all tasks; tracking, mapping, relocalization, and loop closure. It focuses on map maintenance with a strategy where it quickly grows the global map during exploration but also aggressively removes redundant keyframes and map points. It also suggests an automatic map initialization strategy which was missing in PTAM, based on a heuristic selection between a fundamental matrix for non-planar scenes and a homography for planar scenes.

ORB-SLAM builds a covisibility graph. A local covisible area is used for tracking and mapping. The pose graph, named *essential graph*, contains a spanning tree, loop closure connections, and strong edges from covisibility graph. Spanning tree is built by connecting each new keyframe to the keyframe with most number of covisible points. See Fig. 3.1b for examples of covisibility graph, spanning tree, and essential graph. Essential graph is optimized to correct loop closures. An additional global bundle adjustment is also applied after pose graph optimization.

3.2 Direct Approaches

With direct methods, actual sensor measurements are used. This way, the sensor can be modeled better where factors such as exposure time can be used as model parameters instead of being unknown noise sources.

For direct methods, initially dense or semi-dense approaches are preferred.

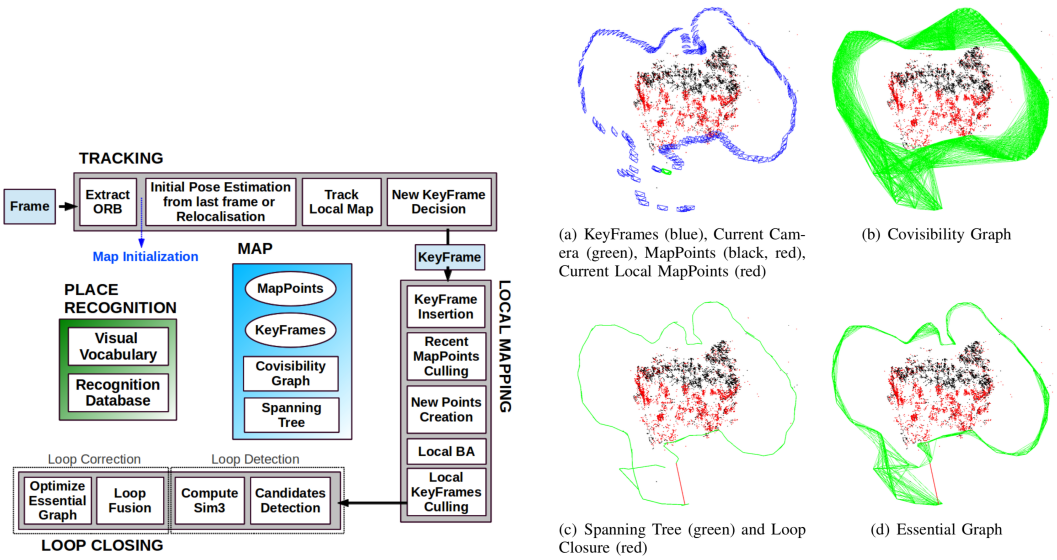


Figure 3.1: a) Overview of ORB-SLAM b) An example for reconstruction and graphs ([5], Fig. 1, 2)

Dense methods use all points in frames and they use the neighborhood information with a geometry prior which usually assumes neighborhoods are smooth. Sparse methods use only a set of independent points without any geometry prior.

DTAM [22] and LSD-SLAM [23] are some examples for direct, dense/semi-dense systems. The disadvantage of dense approaches is the infeasibility of the joint optimization of poses and geometry, due to the geometry prior they use with correlations between parameters.

3.2.1 Direct Sparse Odometry

Later, Direct Sparse Odometry (DSO) [3] is proposed as a direct, sparse visual odometry system where a photometric error is optimized without any geometry prior. DSO does point selection based on intensity gradients with a dynamic grid search. This way, edges or sparsely textured regions can be sampled as well in contrast to feature-based approaches.

DSO uses a photometric camera calibration model which includes lens attenuation, gamma correction, and known exposure times.

For a point $\mathbf{p}$ hosted in keyframe I_i, the photometric cost of the observation of $\mathbf{p}$ in

keyframe I_j is

$$ {}^{j}E_p = \sum_{\mathbf{u}_k \in N_p} w_k \| (I_i[\mathbf{u}_k] - b_i) - \frac{t_i e^{a_i}}{t_j e^{a_j}} (I_j[\mathbf{u}'_k] - b_j) \|_\gamma \tag{3.1} $$

where N_p is the set of pixels over a small patch around $\mathbf{p}$; $\mathbf{u}'_k$ is the projection of $\mathbf{u}_k$ to I_j; t_i, t_j are exposure times of I_i, I_j; a_i, b_i, a_j, b_j are affine brightness parameters for I_i and I_j; $\| \cdot \|_\gamma$ is the Huber norm; and $w_k = \frac{c^2}{c^2 + \|\nabla I\|_2^2}$ is a gradient dependent weight reducing effect of high gradient pixels.

Affine brightness transfer model $e^{-a_i}(I_i - b_i)$ is used for sequences without known exposure times.

Photometric cost function 3.1 depends on geometric and photometric parameters.

DSO jointly optimizes all model parameters including camera poses, camera intrinsics, and geometry parameters. Affine brightness parameters are also optimized. It employs a sliding window approach where old camera poses and points are marginalized.

Although DSO is an important work since it was the first direct sparse approach with a non-linear optimization framework, DSO is a visual odometry system and cannot reuse the rich information from the revisited parts of the environment.

3.2.2 Direct Sparse Odometry with Loop Closure

Direct Sparse Odometry with Loop Closure (LDSO) [4] extends DSO to a visual SLAM system with a loop closure detection and correction module similar to ORB-SLAM.

LDSO modifies the point selection strategy of DSO to select some of the points as repeatable feature points using Shi-Tomasi algorithm [8] and computes their ORB descriptors [12]. All selected points are still used for visual odometry. Having these feature points, LDSO uses common practices from indirect approaches. Loop closures are detected with a feature-based bag of words approach. For each loop closure candidate I_j for the current keyframe I_i, an $SE(3)$ relative pose T_{ij} is estimated by solving the PnP problem using 3D-2D feature point matches. Then, a $Sim(3)$ relative pose S_{ij} is optimized using Gauss-Newton method with 3D and 2D geometric constraints, initialized from the estimated relative pose T_{ij} with relative scale s_{ij} being 1.

A depth map D_i is created for the current keyframe I_i by projecting active map points to I_i. Using this depth map, depth values can be estimated for some feature points. Let $P = \{{}^{j}\mathbf{p}_k\}$ and $Q = \{{}^{i}\mathbf{p}_k\}$ be the sets of matched feature points for I_j and I_i respectively with matching indices. Let $Q_1 \subseteq Q$ be the set of points without an estimated depth and

$Q_2 = Q \setminus Q_1$. The cost function for the optimization of relative pose $\mathbf{S}_{ij}$ is

$$E_{loop} = \sum_{{}^i\mathbf{p}_k \in Q_1} w_1 \|\mathbf{S}_{ij}\Pi^{-1}({}^j\mathbf{p}_k, d_{j_{\mathbf{p}_k}}) - \Pi^{-1}({}^i\mathbf{p}_k, d_{i_{\mathbf{p}_k}})\|_2 + \sum_{{}^i\mathbf{p}_k \in Q_2} w_2 \|\Pi(\mathbf{S}_{ij}\Pi^{-1}({}^j\mathbf{p}_k, d_{j_{\mathbf{p}_k}})) - {}^i\mathbf{p}_k\|_2 \tag{3.2}$$

where w_1 and w_2 are weights for balancing the 3D error and the 2D reprojection error. By optimizing $\mathbf{S}_{ij}$, relative scale s_{ij} is also estimated.

Relative pose estimation step also acts as a loop closure verification step with multiple thresholds.

LDSO creates a covisibility graph from the sliding window optimization. It combines relative poses from the covisibility graph with the estimated relative poses for loop closures and uses these as constraints for pose graph optimization.

LDSO uses DSO as its SLAM frontend where odometry is estimated and applies loop closure detection and pose graph optimization in the backend.

With loop closure correction, LDSO recovers accumulated drift in the unobservable degrees of freedom, i.e. global translation, rotation, and scale in the monocular case. Although LDSO uses detected loop closures for pose graph optimization, reobservations cannot be utilized in local photometric bundle adjustment due to the absence of a global map.

3.2.3 Direct Sparse Mapping

Direct Sparse Mapping (DSM) [1] is a visual SLAM system with a persistent map based on local photometric bundle adjustment (PBA). Having a global map enables DSM to detect and utilize reobservations. It proposes a local window selection strategy which uses both temporally close keyframes and covisible keyframes. DSM uses a coarse-to-fine optimization scheme and employs a robust influence function to decrease the effect of outliers in the optimization based on T distribution. We give details of DSM in Chapter 4.

4 Direct Sparse Mapping

Direct Sparse Mapping (DSM) [1] is a visual SLAM system with a persistent map based on local photometric bundle adjustment (PBA). The persistent map enables retention of all keyframes and reuse of information in PBA.

DSM consists of a tracking frontend and an optimization backend. The tracking frontend tracks frames and points and also handles coarse initialization for the optimization. The optimization backend determines which keyframes should be included in the local window and jointly optimizes all active keyframes and map point parameters.

There are two threads for tracking and mapping. The tracking thread obtains a camera pose for each frame and decides when to add a new keyframe. The mapping thread processes new frames to track points from active keyframes and handles new keyframe creation by recalculating the local window, activating new points, and invoking PBA. The mapping thread also maintains the map locally consistent by removing outliers, detecting occlusions, and avoiding duplicating points. DSM uses the pinhole camera model. All images are first undistorted.

4.1 Local Map Covisibility Window

DSM uses a local map covisibility window (LMCW) to select active keyframes K_a which is a combination of temporal and covisibility criteria with respect to the latest keyframe. Figure 4.1 depicts LMCW.

The temporal window K_t is selected in a similar approach as in DSO where the most recent keyframes are chosen. Each new keyframe I_0 is added to the temporal window and another keyframe is removed. The two most recent keyframes I_1, I_2 are always preserved for the odometry accuracy. The removed keyframe is selected from the rest, based on a criterion which favors observations that render high parallax. This results with a temporal window with keyframes that are evenly distributed in space. This part is crucial during exploration as new points are initialized and it helps to maintain odometry accuracy.

In the covisible window K_c, keyframes that are covisible with those in the temporal part are selected. Keyframes with active map points that project into depleted areas (with respect to active map) in the latest keyframe I_0 are favored to get a more representative active map. This helps to avoid getting duplicate points when selecting new active

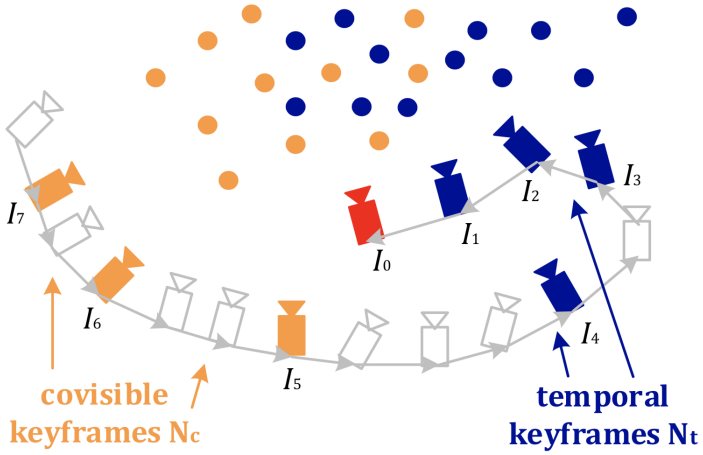


Figure 4.1: A LMCW example with 4 temporal keyframes (blue), 3 covisible keyframes (orange), and the latest keyframe I_0 (red). $K_c = N_c$ and $K_t = N_t$ for this figure. (Source [1])

points for the latest keyframe I_0. While this works to avoid duplicate active points in the active window K_a, it does not prevent duplicate map points in the global map K. Covisible window selection strategy needs to process all inactive keyframes which can take substantially longer with increasing number of keyframes, especially for long trajectories.

By using covisible keyframes, the system benefits from scene reobservations, decreasing motion drift and structural inconsistencies. Since there is no explicit compensation of the drift for larger loops, keyframe reuse is effectively limited to local mapping and smaller loops.

LMCW does not take photo-consistency into consideration, it only considers geometry. As a result, keyframes with camera poses that point to the same part of the map can be labeled as covisible even if they do not actually observe the same scene, e.g. due to occlusion. This is an important issue for non-convex environments such as in the Kitti dataset [24]. We address this issue in Section 5.6.

4.2 Point Visibilities

Point visibilities on the latest keyframe are computed for the active points of all keyframes K when selecting temporal and covisible windows. When the active points are selected for the new keyframe from the candidate points, visibility information of the active points is set for all active keyframes. Point visibilities are also updated in local BA, e.g. when observations are removed as outliers.

4.3 Covisibility Graph

DSM builds a covisibility graph by updating keyframe connections for active keyframes after every local BA. Covisibility connections are based on the total number of points from the keyframe pairs that are visible on the other frame. Point visibility information is not computed here, previously computed information is used.

Interestingly, this covisibility graph structure is not used anywhere in the code. We modify and utilize the covisibility graph in our work.

4.4 Photometric Bundle Adjustment

DSM uses the photometric cost function proposed by DSO (Eq. 3.1) with some modification.

For a point $\mathbf{p}$ hosted in keyframe I_i, the cost of the observation of $\mathbf{p}$ in keyframe I_j is

$$ {}^{j}E_p = \sum_{\mathbf{u}_k \in N_p} w_{r_k} w_{g_k} \left((I_i[\mathbf{u}_k] - b_i) - \frac{e^{a_i}}{e^{a_j}} (I_j[\mathbf{u}'_k] - b_j) \right)^2 \tag{4.1} $$

where N_p is the set of pixels over a small patch around $\mathbf{p}$, $\mathbf{u}'_k$ is the projection of $\mathbf{u}_k$ in I_j, a_i, b_i, a_j, b_j are affine brightness functions for I_i and I_j, w_{r_k} is the robust influence function, and $w_{g_k} = \frac{c^2}{c^2 + \|\nabla I\|_2^2}$ is a gradient dependent weight reducing effect of high gradient pixels.

Affine transfer model is used to handle camera automatic gain control and changes in scene illumination.

Using widely separated covisible keyframes can cause spurious observations resulting in occlusions and reflections which violate the photo-consistency assumption. The robust influence function w_{r_k} is used to handle this. It is based on photometric error distributions which are computed for each keyframe using all observations. T distribution is used as it is observed to explain sparse photometric error well.

Photometric cost function is used for geometry initialization, PBA, and map reuse.

The total error for LMCW is

$$ E = \sum_{I_i \in K_a} \sum_{\mathbf{p} \in P_i} \sum_{j \in obs(\mathbf{p})} {}^{j}E_p \tag{4.2} $$

where K_a is the set of active keyframes, P_i is the set of map points of I_i, $obs(\mathbf{p})$ is the set of observations for point $\mathbf{p}$.

Optimization is affected by estimation drift as a result of covisible keyframes which are temporally distant. To compensate for this drift, a multi-scale, coarse-to-fine scheme is employed to increase photometric convergence radius. Estimated geometry from each level is used as an initialization for the next level.

Eq. 4.2 is minimized after each new keyframe using iteratively reweighted Levenberg-Marquardt algorithm.

4.5 Map Points

In DSM, points are hosted in keyframes, represented with an inverse depth. New points are first initialized, then become ready for activation when their depth has been refined, and finally are activated to be optimized in the local PBA.

When a frame becomes a keyframe, candidate points are extracted for this frame. Image gradients are used to select salient points with an adaptive gradient threshold. DSM uses a dynamic grid approach for point selection. A frame is first divided into cells, then for each cell it tries to select the best point. Depending on the number of selected points compared to the desired number of points n_p, the window size w for the grid is heuristically adjusted, i.e. if we get more points than n_p the window size w is increased and vice versa.

With each new keyframe I_i, candidate points $\mathbf{p}$ of the active keyframes K_a are tracked to this new keyframe I_i to initialize or refine their depth estimates. Then, "good candidates" (which are initialized successfully, can be projected into the last keyframe, etc.) of the temporal keyframes K_t are refined further based on all active keyframes K_a. When a candidate point is initialized, it has an initial depth estimate, but this depth estimate only becomes reliable after it is successfully refined. Note that neither can every candidate point be initialized successfully nor every initialized candidate point be refined successfully. After candidate points are refined, eligible optimized candidate points $\hat{\mathbf{p}}$ of the temporal keyframes K_t are activated. When a candidate point becomes active, its depth is not updated again with this procedure. So, after a candidate point is optimized, there is no difference between whether it is a candidate point or an active point in terms of depth (before local bundle adjustments). If an active point later becomes an outlier, it is deleted.

5 Direct Sparse Mapping with Loop Closure

In our work, we use DSM method and its open source code [1] as a baseline and extend it to have loop closure detection and pose graph optimization. Before going into the details of each subsystem in the next sections, we present a high-level overview of the entire system.

We use a feature based bag of words approach to detect loop closure candidates, and for this we need distinctive feature points. We modify the point selection procedure of DSM to ensure that a fraction of all selected points are corners. We also compute ORB descriptors of detected corners.

We convert the corner descriptors of each keyframe to a bag of words representation to build a database of the past keyframes. To select possible loop closure candidates for the latest keyframe I_r, we query the frame database for similar frames. After filtering the results, we use the resulting frames as loop closure candidates for the frame I_r. For a loop closure candidate I_c, we estimate an initial relative pose $\mathbf{S}_{rc}$ between the candidate frame I_c and the reference frame I_r by solving the PnP problem. Then, we optimize this relative pose $\mathbf{S}_{rc}$ and use it as a constraint in pose graph optimizations.

To correct the global map using the detected loop closures, we run pose graph optimizations where we use relative poses $\{\mathbf{S}_{ij}\}$ between frames based on the detected loop closures and the covisibility graph as constraints and we optimize the keyframe poses $\{\mathbf{S}_i\}$.

Differently from previous methods such as LDSO which applies pose graph optimization as a post processing step, we use pose graph optimization results in tracking and mapping. We update keyframe poses $\{\mathbf{T}_i\}$, map points $\{{}^i\tilde{\mathbf{p}}_k\}$, and relative pose measurements $\{\mathbf{S}_{ij}\}$ based on the optimization results.

We make modifications to the covisible window selection procedure. Only geometry is considered for covisible window selection in the original approach. We add an appearance-based verification step to prevent false covisibility connections especially for non-convex environments where a geometry-based control is not enough to verify visibility. Covisible window search considers every inactive keyframe in the original approach. We reduce this search range to the inactive neighborhood of the active temporal

[1] github.com/jzubizarreta/dsm

keyframes to partially prevent the search range from growing unbounded.

We maintain a graph of connected keyframes, namely the covisibility graph. We use *neighborhood* N_i to refer to the neighborhood of a frame I_i which includes both covisible frames and temporally connected frames. *Temporally connected frames* are the frames that have been in the temporal window at the same time for at least one local bundle adjustment. We have a covisibility threshold to define *covisible frames*, which requires at least 100 active points that are visible from the other frame in total for the both frames. We do not have such a threshold for temporally connected frames.

For loop closure verification, we depend on 3D points and for this we want as many points to be optimized for a querying keyframe as possible. For this, we have a small delay between creation of a keyframe I_i and loop closure detection for it. This helps us to get more 3D points with more accurate depth estimates after a number of local bundle adjustments. We wait 5 keyframes, i.e. until keyframe I_i leaves the temporal window (maximum number of temporal keyframes is 5 by default), but this is a heuristic parameter that can be adapted.

When there is a loop in a trajectory, we expect to detect multiple loop closure pairs for that segment of the map and we prefer to use all loop closure pairs for a segment in PGO. For this, we employ a delay mechanism where we wait 5 keyframes without a loop closure to run PGO after we verify a loop closure. We also wait at least 20 keyframes between consecutive PGO runs as a cooldown period. This way, we run PGO with more loop closure edges and thus get more accurate corrections. But more importantly, we decrease the number of PGO calls and thus runtime significantly. As a result, the map gets updated less frequently with some delay. So, there is a trade-off between increasing runtime performance and keeping the map more up-to-date. For Kitti sequence 00, on average, we get 304.6 loop closures while we run PGO only 9.4 times (see Table 6.1). We assess the effect of this strategy in Section 6.3.4.

For optimization problems we use Ceres [17]. We use OpenCV [25] implementations of various computer vision algorithms.

5.1 Feature Point Selection

To detect loop closure candidates with a feature based bag of words approach, we first need to have some repeatable points. DSM as a direct method extracts points based on image gradients without taking repeatability into consideration and these points are not necessarily distinguishable, repeatable points, e.g. points from weakly textured regions or edges. We cannot use these points to detect loops, we need to introduce repeatable feature points such as corners. While we cannot use non-feature points for loop closure detection, we can use both feature and non-feature points for tracking and mapping, as the feature point selection algorithms also favor pixels with high gradients. So, we extract feature points as a fraction of all points. All points are used for tracking and

mapping as usual but in addition we compute descriptors for the feature points and use them for loop detection.

We get feature points by using either FAST or Shi-Tomasi corner detectors. We use FAST threshold of 30 for FAST detector and quality level (minimum accepted quality of corners relative to the best corner) of 0.01 for Shi-Tomasi detector. Our method is agnostic towards corner detector choice. DSM selects 1500 points for a frame by default. We increased this to 2000. We discuss the effect of this change in Section 6.3.2. Out of all points, we select 1/3.5, i.e. 571 points by default, as feature points.

DSM uses a dynamic grid approach for point selection. While we preserve this logic to select non-feature points, we also detect repeatable feature points using a similar grid approach. Since the window size w is adjusted to get the desired number of points n_p, we still get the same number of points in total and the adjusted window size w affects the selected corners the same way as other points.

Grid approach helps us to get a homogeneous selection across the frame. We do not want to get all corners from a small, highly textured region of the scene, which may not be visible from another perspective. With a homogeneous selection, we increase the probability of detecting mostly the same corners for a scene from different perspectives, even if there is some degree of occlusion.

Since what we call a corner changes depending on the resolution of an image, we need to consider corners for each level of the image pyramid, similar to how non-feature points are selected for each image level. We detect corners for each pyramid level, without limiting the total number of corners. After detecting all suitable corners, we select one corner per grid cell, regardless of the detected pyramid level. Then, we filter nearby corners (with a radius of 4 pixels). The remaining steps are applied to this initial selection.

We want to select more corners for pyramid levels with higher resolution, since we have more information and thus we get more corners for those levels. For this, we split the total number of corners for detection levels inversely proportional to detection level, e.g. $3n$, $2n$, and n points for a pyramid with 3 levels. For the default pyramid level of 3 and number of corners of 571, we try to select 285, 190, and 95 corners from level 1 (original image resolution), level 2, and level 3 respectively. If we cannot complete the total number of corners this way, we select from the remaining corners regardless of their detection level. We refine corner locations by finding their subpixel accurate locations [26]. We limit the maximum offset during refinement to 0.5 pixel in order to preserve the repeatability of points.

After finalizing the selection of corners, we select non-feature points with the original selection approach of DSM. Since not all candidate points can be initialized and activated, in the point selection step DSM selects a high number of redundant points. For our needs, we want as many of the corners to be activated as possible to get their estimated depth to be optimized. Although for some candidate points we also get optimized

depth, all active points have an optimized depth, which is also optimized in local bundle adjustment. To increase probability of corners getting activated and also to prevent selection of duplicate points, we remove non-feature points that are too close to selected corners (with a radius of 5 pixels). This way, we do not get adjacent points that are very similar, which can hinder some corners from getting activated. Fig. 5.1 shows the difference between feature points and non-feature points.

Figure 5.1: An example of the original (top) and the modified (bottom, shows only feature points) point selection procedure. With feature points we get repeatable points such as window corners.

After selecting all points, we also compute feature orientations, which we need for descriptor computation. For this, we use the intensity centroid approach as is usually done for ORB descriptors [12]. We then compute ORB descriptors, which are rotation invariant binary descriptors. As a binary descriptor, ORB is computationally very efficient compared to alternative methods such as SIFT [10]. This is especially important for us since we do not want to increase computational complexity too much for the point selection part with additional feature extraction and descriptor computation steps.

5.2 Depth Map Creation and Depth Estimation

For loop closure verification, we depend on depth information of points. We can use active points $\tilde{\mathbf{p}}$ and optimized candidate points $\hat{\mathbf{p}}$ as 3D points, since both of which

have depth information. We do not use candidate points $\mathbf{p}$ with initialized (but not optimized) depths as they are not reliable. We rely on optimized (but not activated) candidate points $\hat{\mathbf{p}}$ since they can become active points at any time without further changes, i.e. they have reliable depth information.

Since we select only a fraction of candidate points as corners and only a portion of all candidate points become active or have an optimized depth estimation, the number of 3D corners can be the bottleneck for loop closure verification, which in turn decreases the number of accepted loop closures. To prevent this, we create depth maps for frames if needed, and estimate depths for points without depth information using these depth maps.

We project all active points $\{\tilde{\mathbf{p}}_k\}$ from the neighborhood N_i of a frame I_i to I_i. For successfully projected points, we keep their positions and corresponding depth information in the depth map. Additionally to active points from the neighborhood N_i, we also record active points $\{{}^i\tilde{\mathbf{p}}_k\}$ of I_i in its depth map, which have precedence over projected points.

For a given point $\mathbf{p}$, we estimate its depth as the depth of the closest point to it in the depth map. The search area is limited to a circle of radius 5 in ℓ_1 norm, i.e. in Manhattan geometry.

For next steps, when we refer to *3D points* of a frame I_i, they include active points ${}^i\tilde{\mathbf{p}}_k$, optimized candidate points ${}^i\hat{\mathbf{p}}_k$, and points with estimated depths ${}^i\mathbf{p}^*_k$.

Figure 5.2: A loop closure pair (left) from Kitti sequence 00 with the point cloud of their neighborhoods (right) aligned with the estimated relative pose. In the point cloud, we see the street from the top. Orange map points are from the neighborhood of the querying keyframe (top left) and blue map points are from the neighborhood of the candidate keyframe (bottom left). Well aligned points indicate the accuracy of the estimated relative pose.

5.3 Loop Closure Detection

For loop closure detection, we use a feature based bag of words approach, namely DBoW3 [16]. We use the vocabulary which is prepared for our problem from ORB-SLAM, which is also used by LDSO.

We first compute the bag of words representation for a reference keyframe I_r using its corner descriptors. Then we query the frame database for the keyframes that are similar to the keyframe I_r. After the query, the keyframe I_r is added to the image database. We do not want to get keyframes that are close to the reference keyframe in time and place, i.e. if they are in the same segment of the trajectory it is not considered as a loop. This filtering is usually done (e.g. [4]) by using temporal information, e.g. excluding temporally adjacent keyframes from the results. We keep track of covisibility information and we know that covisible keyframes will not form loop closures. So, in addition to temporally connected keyframes, we also exclude covisible keyframes of the keyframe I_r from the results. This way we apply a broader filter than just using the temporal information.

We query the 20 most similar frames. After filtering temporal and covisible connections, we keep at most 10 candidates. Here, we do not filter query results further based on similarity scores, but an adaptive threshold based on similarity score could be used to filter out bad candidates and in turn increase runtime performance similar to what ORB-SLAM does.

Loop closure verification happens at next steps during relative pose estimation and optimization. We stop if we do not meet necessary conditions and thresholds at any step. This is supposed to ensure that no false positive loop closure is inserted into the global map.

We run next steps for each loop closure candidate I_c. We keep all verified candidates. So, for a query keyframe I_r, we can add multiple loop closure constraints to the global map.

5.4 Relative Pose Estimation

5.4.1 Feature Point Matching

Let us consider a single loop candidate I_c for reference keyframe I_r based on feature points. We want to estimate relative pose $\mathbf{S}_{rc}$ from I_c to I_r using the feature points. For this, first we need to get feature point matches between the two frames. At this step, we only use point similarities for matching based on point descriptors.

We can use a brute-force approach to match descriptors by checking hamming distances between all possible matches. This would result with the best possible matching between two sets of descriptors. As an alternative to brute-force approach, we can use DBoW3,

which would be approximate but faster than brute-forcing. DBoW3 builds a vocabulary tree that discretizes the descriptor space and we can only check the possible matches for features that correspond to the same word in the tree, instead of comparing all possible matches. In our method, we can opt to use either of these approaches. We provide comparison for these two approaches in Section 6.3.3.

For the best match for a descriptor, i.e. the match with minimum hamming distance, we have two more conditions. We check if the distance is less than or equal to a fixed threshold, which is 50, to prevent matches with weak similarity. We also expect the best match to be substantially better than the second best match. For this, we check if the minimum hamming distance is less than 60% of the second best hamming distance. This way, we do not accept matches with high uncertainty.

We expect to get at least 10 corner matches to proceed with relative pose estimation.

5.4.2 Relative Pose Estimation

We estimate the relative pose $\mathbf{S}_{rc}$ between the two frames by solving the Perspective-n-Point (PnP) problem using an iterative method based on a Levenberg-Marquardt optimization and Random Sample Consensus (RANSAC) [27]. By solving the PnP problem, we estimate a relative pose $\mathbf{T}_{rc}$ with rotation and translation from 3D-2D point matches. RANSAC helps us to use only the inliers for finding the relative pose since we might get wrong matches from the previous step. To solve the PnP problem, we expect at least 10 3D corners.

We want to get a similarity transformation $\mathbf{S}_{rc}$ with rotation, translation, and scale but we cannot estimate scale from the PnP problem. In order to estimate the relative scale s_{rc}, we get pose estimates both ways, i.e. we get both $\widetilde{\mathbf{T}}_{rc}$ and $\widehat{\mathbf{T}}_{cr}$. $\widetilde{\mathbf{T}}_{rc}$ is computed using 3D points from frame c and 2D observations from frame r. $\widehat{\mathbf{T}}_{cr}$ is computed using 3D points from frame r and 2D observations from frame c. We expect to get at least 10 inliers for both solutions.

We derive components of $\mathbf{S}_{rc}$ as follows:

$$\begin{aligned} s_{rc} &= \frac{\|\widehat{\mathbf{t}}_{cr}\|}{\|\widetilde{\mathbf{t}}_{rc}\|} \\ \mathbf{t}_{rc} &= s_{rc} * \widetilde{\mathbf{t}}_{rc} \\ \mathbf{R}_{rc} &= \widetilde{\mathbf{R}}_{rc} \end{aligned} \tag{5.1}$$

In our formulation the relative translation $\mathbf{t}_{rc}$ is in the scale of frame r. Since $\widetilde{\mathbf{t}}_{rc}$ is in the scale of frame c, we rescale it to frame r.

$\mathbf{S}_{rc}$ we derived here serves just as an initialization of the subsequent optimization.

5.5 Relative Pose Optimization

5.5.1 Projected Point Matches

Now that we have an initial relative pose estimation $\mathbf{S}_{rc}$, we try to find a new set of corner matches for pose optimization. Unlike the previous set of matches for which we only use descriptor similarity without considering spatial positions, we now match 3D points by relying mainly on their spatial position as a constraint. We project 3D points from the candidate keyframe I_c to the reference keyframe I_r. Then, for each projected point, we get nearby 3D points of the reference frame I_r (with a radius of 5 in ℓ_1 norm) and select the best match from these nearby points. Here, we filter points that do not have similar corner angles (larger than 0.2 rad) and points that have a hamming distance larger than 50. Since we only consider a small neighborhood for each point, we can include matches that are not unique across the whole image, which were filtered by the second-best distance check in the initial whole image matching we apply for relative pose estimation. We expect at least 10 matches.

2D-2D Point Matches with Epipolar Line Search

We use the number of projected point matches as a validity check for covisible keyframe candidates (see Section 5.6). In this use case, we do not use depth maps for runtime considerations but it results with a small number of 3D points.

To remedy this issue, we utilize epipolar geometry for the corners ${}^{c}\mathbf{p}_i$ without depth and we search for the best match with the same steps as before, we just make the search around the epipolar line of the point ${}^{c}\mathbf{p}_i$ on reference frame I_r instead of around its projection on reference frame I_r.

Note that we do not require matched corners ${}^{r}\mathbf{p}_i$ from the reference frame I_r to have depth information, since we only need the number of matches and we will not use points themselves.

Since the reliability of 2D-2D matches is lower than 3D-2D matches, we have stricter conditions for them. We filter points that have a hamming distance larger than 30. We also require that we have at least 2 3D-2D matches. We expect to get only 5 matches (instead of 10) for this use case.

5.5.2 Relative Pose Optimization

Using these matches we now refine the relative pose between the two keyframes. We have matched points which almost coincide when one point is projected to the other frame. Assuming that most of these matches actually point to the same 3D point in the world (we applied many filters until this point to support this assumption), ideally with a perfect relative pose (and assuming perfect points) they need to coincide exactly. That

is, we can optimize the relative pose by checking the correspondence of these matched points.

Let ${}^{c}P = \{{}^{c}\mathbf{p}_k\}$ be the 3D points from the candidate frame I_c and ${}^{r}P = \{{}^{r}\mathbf{p}_k\}$ be the matched 3D points from the reference frame I_r with matching indices.

The similarity transformation $\mathbf{S}_{rc}$ from candidate frame I_c to reference frame I_r is optimized by minimizing the error

$$E = \sum_{{}^{c}\mathbf{p}_k \in {}^{c}P} (\| {}^{r}\mathbf{p}_k - \mathbf{S}_{rc} * {}^{c}\mathbf{p}_k \|_{\gamma_1} + \| \Pi({}^{r}\mathbf{p}_k) - \Pi(\mathbf{S}_{rc} * {}^{c}\mathbf{p}_k) \|_{\gamma_2}) \tag{5.2}$$

where $||\cdot||_{\gamma_1}$, $||\cdot||_{\gamma_2}$ are the Huber norm functions. We use Huber loss parameter of 1 for both 3D Euclidean distance and 2D reprojection error terms.

After the optimization, we check residuals to get inliers using fixed thresholds for 3D and 2D Euclidean errors (we expect both errors to be less than 1), then remove the outliers from the problem. We expect to have at least 10 inliers. We solve the optimization problem again with only using the inliers. At this point, we check the average cost per point for the optimized problem to determine whether we have a good case for loop closure. If the normalized cost is small enough (less than 1), we save the optimized relative pose for these keyframes and consider the loop closure verified.

See Fig. 5.2 for an example of a loop closure pair with aligned point clouds of their neighborhoods.

We have manually tuned many of the parameters and thresholds in this section to get good performance. In future work, a more systematic parameter search might yield even better results across different datasets.

5.6 Changes to LMCW

To select covisible window for newly added keyframe I_r, DSM traverses all inactive keyframes I_i, project active points ${}^{i}\tilde{\mathbf{p}}_k$ to the newly added keyframe I_r and set their visibilities. Then, it selects the optimal covisible window. There are two problems with this approach.

For long trajectories, this search grows linearly with the number of keyframes. As a result, covisible window selection starts to take increasingly longer part of the runtime. We decrease this search window from all previous keyframes to only the covisible frames of the temporal connections of I_r, i.e. only the second degree of connections on the covisibility graph. As a downside, this would inevitably prevent some of the true covisibility connections from being added to covisibility graph. We also prevent visibility updates for the map points of old keyframes which we do not consider. As a result, this change can result with fewer covisibility connections, fewer map point observations, and

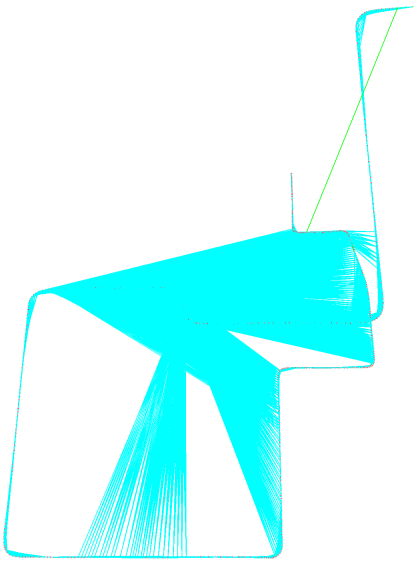


Figure 5.3: An example pose graph for Kitti sequence 00 without modified LMCW and covisibility threshold. Blue lines show covisibility constraints. Green lines show loop closure constraints.

therefore with a drop in BA accuracy if we use it without the loop closure module. But we expect the loop closure module to detect some of the connections that we possibly missed during covisible window selection. So, we are interested in the performance of this change with the enabled loop closure module.

The original covisible window selection only checks if points can be geometrically projected to the keyframe I_r. This works fine in a convex environment such as a room. But in a non-convex environment such as a neighborhood with non-covisible streets, we get many false covisible frame matches from LMCW. This prevents us from successfully correcting loops with pose graph optimization, since each part of the map is essentially connected to every other part.

To solve this issue, in addition to the geometric check, we also apply a feature-based check. We check the number of projected point matches (see Section 5.5.1). Since we need to make this control for many keyframes every time, we do not use depth estimation from depth map and instead we also use 2D-2D matches by utilizing epipolar geometry (see 5.5.1). For the projection check, we require at least 5 corner matches, 2 of which should be 3D-2D matches. Compare Fig. 5.3 with Fig. 5.4 for the effect of the added feature-based verification step. In Fig. 5.3, frames from non-crossing streets surrounded by buildings in a city center are wrongly labeled as covisible, which prevents pose graph optimization to correct the scale drift.

5.7 Pose Graph Optimization

We detected loop closures with optimized relative poses which conflict with the information from local mapping. We run a global pose graph optimization with all keyframes to decrease this conflict by distributing large errors between loop closure pairs to all keyframes and consequently correct loops.

We fix the first keyframe I_1 and optimize the rest of the keyframes. The non-fixed frame poses $\mathbf{S}_i$ are optimized by minimizing the error

$$E_{PGO} = \sum_{i,j \in C} w_{ij} \log_{Sim(3)}(\mathbf{S}_{ij} * \mathbf{S}_j^{-1} * \mathbf{S}_i) \tag{5.3}$$

where C is the set of connected keyframe pairs that are derived from loop closures or covisibility graph, $\log_{Sim(3)}$ is the logarithm function to map $Sim(3)$ to $\mathbb{R}^7$, and the weight $w_{ij} = 1$ for constraints from covisibility edges and $w_{ij} = 100$ for loop closure constraints to increase the importance of the latter.

For each keyframe I_i, we add relative poses $\mathbf{S}_{ij}$ for the keyframes I_j in its neighborhood N_i. We also add a relative pose $\mathbf{S}_{ij}$ for the parent keyframe I_j of I_i in the spanning tree. Spanning tree follows the same logic in ORB-SLAM. Although the parent node I_j of I_i in the spanning tree is also connected to I_i in covisibility graph, this connection is not guaranteed to be in N_i due to the covisibility threshold on the number of covisible points, e.g. if it decreases over time.

We need to derive relative poses $\mathbf{S}_{ij}$ for covisible keyframes I_i, I_j. This can be computed before each PGO using the current camera poses $\mathbf{T}_i$, $\mathbf{T}_j$ where we convert $\mathbf{T}_{ij}$ to $\mathbf{S}_{ij}$ by using the scale of 1, which is the scale of the fixed keyframe pose $\mathbf{S}_1$. This was our first approach. The problem with this approach is that we update camera poses based on PGO results and use camera poses to set up the PGO problem. So, if we end up deteriorating camera poses, then we would not be able to correct it with later PGO calls because we permanently corrupted the camera poses.

The solution to this, and also our current approach, is to use relative poses based on local bundle adjustment. After each local bundle adjustment, we compute relative poses between active keyframes and store it. We do not update these relative poses after PGO calls (other than reflecting scale changes). So, these relative poses do not correspond to current camera poses which we set based on PGO results, but rather correspond to the last local bundle adjustment where both keyframes were active.

The pose graph we optimize is similar to the essential graph of ORB-SLAM. ORB-SLAM uses a subset of covisibility graph to decrease the optimization complexity. This is not our main concern. We only take strong covisibility connections to filter false covisible keyframe pairs since covisibility information is not photometrically verified in local window.

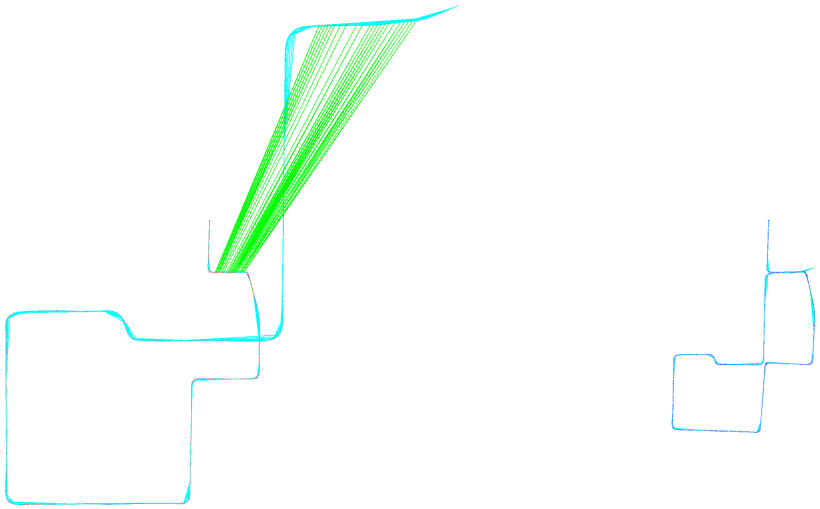


Figure 5.4: An example of a pose graph before (left) and after (right) the optimization for Kitti sequence 00. Blue lines show covisibility constraints. Green lines show loop closure constraints.

See Fig. 5.4 for an example of a pose graph before and after the optimization.

5.8 Global Map Updates

After a pose graph optimization, we update the frame poses $\mathbf{T}_i$ and inverse depths ${}^i d_k$ of their points ${}^i \mathbf{p}_k$ (2D positions of points are always fixed, only the depth changes when the frame pose changes). We absorb the optimized scale s_i into the points and drop it from the camera pose. This way, we get the camera pose $\mathbf{T}_i$ by simply using rotation matrix $\mathbf{R}_i$ and translation $\mathbf{t}_i$ of the optimized pose $\mathbf{S}_i$

$$\begin{aligned} \mathbf{S}_i &:= \begin{bmatrix} s_i \mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0}^T & 1 \end{bmatrix} \\ \mathbf{T}_i &\leftarrow \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0}^T & 1 \end{bmatrix} \end{aligned} \tag{5.4}$$

For every point ${}^i \mathbf{p}_k$ hosted on keyframe i with optimized scale factor s_i, inverse depth ${}^i d_k$ is updated as

$${}^i d_k \leftarrow \frac{{}^i d_k}{s_i} \tag{5.5}$$

We update relative poses $\mathbf{S}_{ij}$ to reflect scale changes as (see Section 2.6.1 for details)

$$\begin{aligned} \mathbf{t}_{ij} &\leftarrow \mathbf{t}_{ij} s_i \\ s_{ij} &\leftarrow s_{ij} \frac{s_i}{s_j} \end{aligned} \tag{5.6}$$

We also update covisibility graph for loop closure pairs, so that next local bundle adjustments could benefit while selecting covisible window. For each loop closure pair $\{I_i, I_j\}$ and for each pair of keyframes $\{I_k \in N_i, I_l \in N_j\}$, we update visibilities and add a new edge to the covisibility graph for the keyframe pair, if the keyframe pair do not already have a connection in the covisibility graph.

6 Results

We evaluate our proposed method LDSM on Kitti Odometry [24] and EuRoC MAV [28] datasets and provide comparisons with DSM, LDSO, and ORB-SLAM methods in monocular settings. We use the reported experiment results from [4] for LDSO and ORB-SLAM. We also share ablation and parameter studies in the last section.

We compare LDSM in detail with DSM and a DSM variant with modified point selection strategy (named DSM-corner or DSM-c in short) in our experiments. We use the same configuration of DSM parameters for all DSM variants (same as the public implementation other than number of candidate points for which we use 2000). Unless stated otherwise, for LDSM and DSM-c we use Shi-Tomasi corners, use the modified LMCW; and for LDSM we run PGO with a delay of 5 keyframes, and have a cooldown period of 20 keyframes. We also use the updated search range for covisible window selection for DSM method to get feasible runtime on long trajectories. DSM fails for Kitti dataset sequences in real-time configuration. We provide results where tracking and mapping processes are sequentialized. We report RMS absolute trajectory error (ATE) after aligning estimated trajectories with ground truths in $Sim(3)$.

6.1 Kitti Odometry Dataset

The Kitti odometry dataset contains 11 sequences from a driving car in its training set with ground truth trajectories. Sequences contain many large loops and it is ideal to see the effect of loop closure correction. There are 6 sequences with one or more loops: 00, 02, 05, 06, 07, 09. For sequences 07 and 09, there is only one large loop encompassing the entire trajectory.

We provide detailed comparisons of LDSM, DSM, and DSM-c methods on the Kitti dataset. For these experiments, we use the default parameters. We run each method on the Kitti sequences 5 times. We provide normalized cumulative error plots in Fig. 6.2a which show the percentage of successful runs with an error below a certain threshold. This way we visualize both accuracy and robustness of the methods at the same time. With higher thresholds we see the number of sequences the methods do not fail and with lower thresholds we see the number of sequences the methods perform well. We also provide heatmaps for all experiments in Fig. 6.2b for LDSM and DSM.

With loop closure correction, we see significant improvements for sequences with loops other than sequence 09. LDSM detected the loop for sequence 07 in 3 out of 5 times but

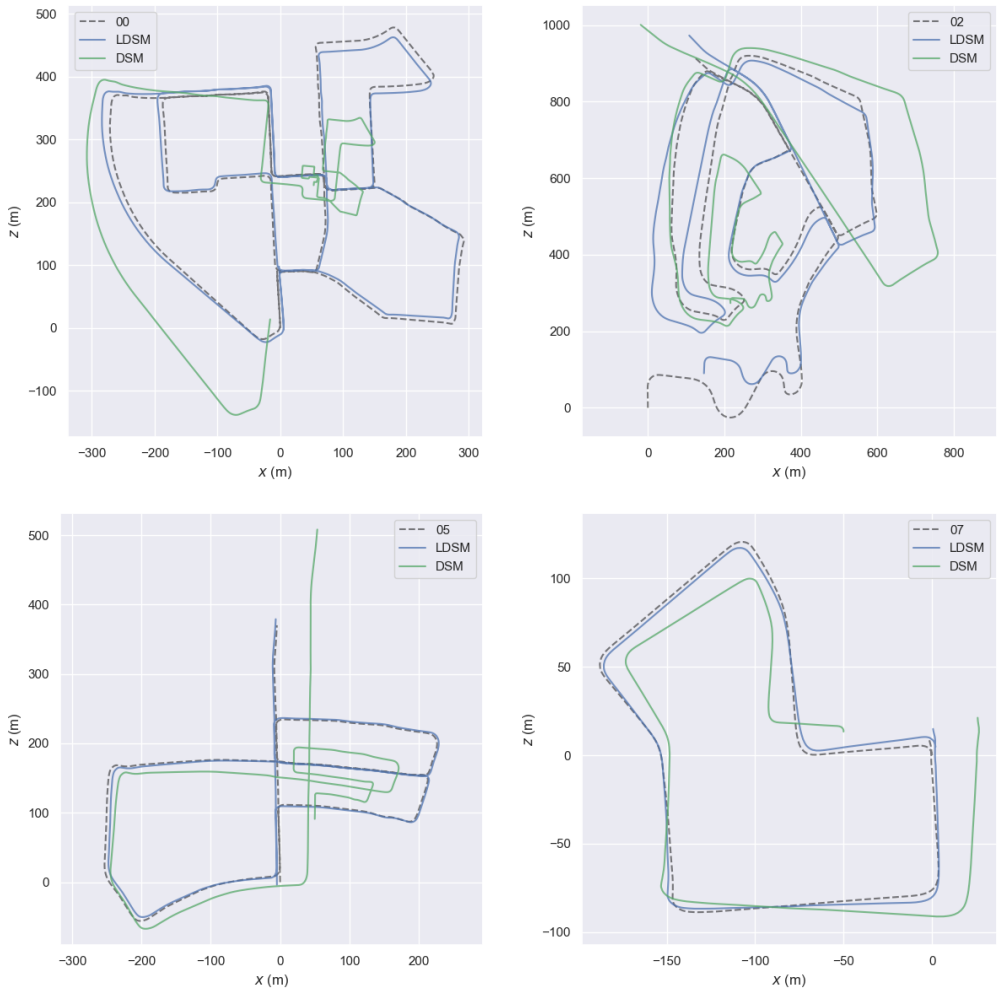
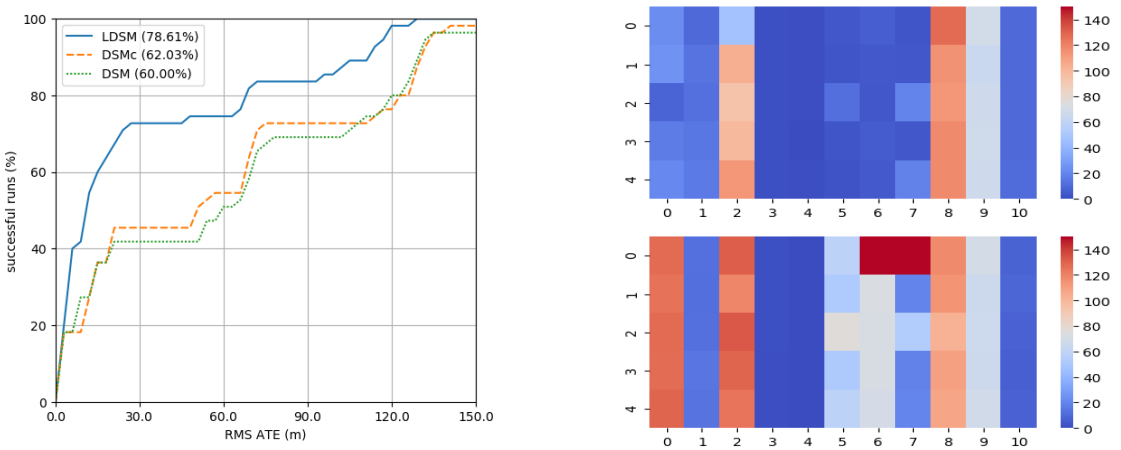


Figure 6.1: Sim(3) aligned trajectories of Kitti sequences 00, 02, 05, and 07 ([29])

failed to detect the loop for sequence 09 in all runs. For sequence 02, LDSM can correct scale drift using detected loops, but that is not enough to fully correct the trajectory. LDSM does not produce robust results for Sequence 02. See Fig. 6.1 for some qualitative results.

Table 6.1 shows average errors for each sequence in the dataset. It also includes results from LDSO and ORB-SLAM methods. For most of the sequences, our method achieves comparable accuracy to LDSO and ORB-SLAM. For sequences with loops, LDSM has the best accuracy for sequence 05 and 06. DSM also shows its quality for sequences without any loop, such as 03 and 04, which translates to good performance for LDSM as well.



(a)

(b) LDSM (top), DSM (bottom)

Figure 6.2: Result on Kitti dataset. a) Normalized cumulative errors. Numbers in the legend give the area under the curve. b) Full evaluation results. Columns represent sequences and rows represent iterations.

Seq.	DSM	LDSM	LDSO	ORB-SLAM	# loops	# PGOs
00	127.87	18.87	9.32	8.27	403.6	9.4
01	13.16	13.50	11.68	-	2.2	1.0
02	127.82	91.65	31.98	26.86	138.2	5.6
03	1.27	1.34	2.85	1.21	14.0	2.6
04	0.35	0.67	1.22	0.77	9.6	2.2
05	59.60	5.00	5.10	7.91	222.8	5.8
06	71.62	4.85	13.55	12.54	102.2	2.8
07	28.27	9.89	2.96	3.44	1.0	1.0
08	110.63	118.50	129.02	46.81	5.6	2.8
09	67.13	67.21	21.64	76.54	1.0	1.0
10	7.71	10.42	17.36	6.61	0.4	0.4

Table 6.1: RMS ATE (m) for Kitti dataset. ORB-SLAM fails for seq. 01. *# loops* and *# PGOs* columns give the average number of detected loop closures and the average number of PGO calls for the sequences for LDSM method respectively.

6.2 EuRoC MAV Dataset

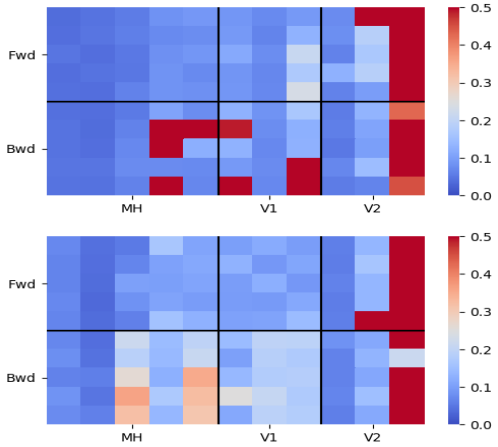
The EuRoC MAV dataset [28] contains 11 stereo sequences in 3 different indoor environments. In [1], DSM is only evaluated on the EuRoC MAV dataset. It is a widely used dataset to evaluate VO and SLAM methods and many state-of-the-art methods perform quite well on it.

We run each method on all the EuRoC MAV sequences forwards and backwards, 5 times on left camera images, for 110 runs in total. Fig. 6.3 contains the results. We

omit sequence V2_03 from the results as we do not get consistent results to make useful comparisons. Although DSM already works really well for the EuRoC dataset and the EuRoC dataset does not have large loops, LDSM gets better results for all machine hall (MH) sequences. For V1 and V2 environments, LDSM has similar performance to DSM.

Sequence	DSM	DSM-c	LDSM
MH_01	0.0713	0.0685	0.0434
MH_02	0.0446	0.0520	0.0430
MH_03	0.1718	0.1838	0.0606
MH_04	0.1356	0.1885	0.0840
MH_05	0.1951	0.2124	0.0834
V1_01	0.1284	0.1110	0.1813
V1_02	0.1497	0.1368	0.0736
V1_03	0.1466	0.2139	0.1575
V2_01	0.0632	0.0798	0.0702
V2_02	0.1326	0.1792	0.1330

(a)



(b) LDSM (top) and DSM (bottom)

Figure 6.3: Results for EuRoC MAV dataset using left camera images. We omit V2_03 from the results. a) RMS ATE (m) b) Full evaluation results.

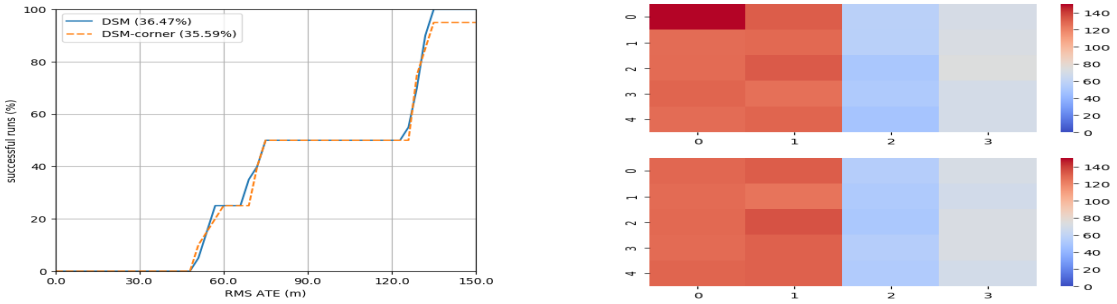
6.3 Ablation and Parameter Studies

6.3.1 Point Selection Strategy

We want to verify that modified point selection strategy does not deteriorate the accuracy of DSM. For this, we compare DSM variants with and without modified point selection strategy, without any other changes we propose, on a subset of Kitti sequences (00, 02, 05, 06). We provide normalized cumulative error plots and heatmaps in Fig. 6.4. Based on these results, we can claim that modified point selection strategy does not affect the performance of DSM.

6.3.2 Number of Candidate Points

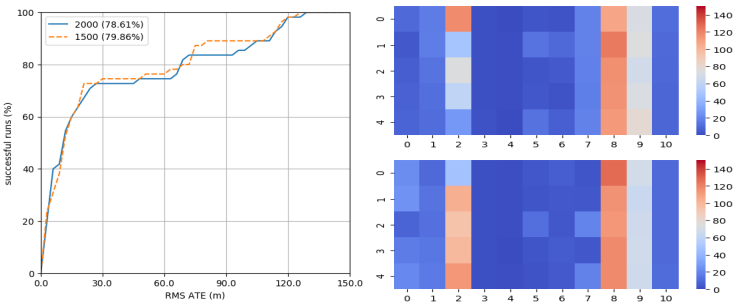
By default, DSM selects 1500 candidate points for each keyframe. We assess the effect of using a larger number of candidates. Fig. 6.5 shows heatmaps, normalized cumulative errors, and average number of detected loop closures for two LDSM experiments on Kitti dataset with 1500 and 2000 candidate points. With more candidates, LDSM detects more loop closures for all Kitti sequences. It also achieves to detect the single loop in Kitti sequence 07 and have a better accuracy for sequence 05, but it loses some accuracy for



(a) (b) DSM-c (top) and DSM (bottom)

Figure 6.4: Results of point selection strategy experiments. a) Normalized cumulative errors. b) Full evaluation results.

sequences 00 and 02. Overall they perform similarly. The fact that we tuned parameters for the loop closure verification part using 1500 candidate points might explain the worse performance for some sequences with loops despite having more loop closures when we use more candidate points. We might get better results with 2000 candidate points after retuning parameters for the loop closure verification part.



(a) (b) 1500 (top) and 2000 (bottom)

Seq.	1500	2000
00	362	403.6
01	0.6	2.2
02	101.2	138.2
03	4.8	14
04	2.2	9.6
05	161.4	222.8
06	60.4	102.2
07	0	1
08	0.2	5.6
09	0	1
10	0.2	0.4

Figure 6.5: Effect of number of candidate points. Kitti results for LDSM with 1500 and 2000 candidates. a) Normalized cumulative errors. b) Full evaluation results. c) Average number of loop closures

6.3.3 Matching Based on Brute Force and BoW

To match feature points between two frames, using BoW is a more efficient approach than brute force. On the other hand, brute force provides the best possible matching. See Table 6.2 for a comparison based on Kitti experiments. With BoW, we get slightly worse results compared to brute force.

Seq.	BoW	Brute
00	21.53	18.87
02	98.35	91.64
05	4.90	5.00
06	5.77	4.85

Table 6.2: Effect of descriptor matching algorithm. Kitti results for LDSM with BoW (left) and brute force (right) matching

6.3.4 Optimization Frequency

We run PGO with a delay of 5 keyframes, and have a cooldown period of 20 keyframes by default. For some sequences we detect hundreds of loop closures, but we run PGO less than 10 times for a sequence on average (Table 6.1). While this help us significantly in terms of speed, instead of waiting for loop closures to accumulate, running PGO more frequently would provide better results and also help by reflecting global corrections to local mapping earlier.

We have not observed a big improvement when we run PGO more frequently on Kitti sequences as shown in Table 6.3. The effect of optimization frequency depends on the trajectories as well. Since in Kitti sequences, we get loop closures in well separated groups, the effect might be different on another dataset.

Seq	delay: 0, cooldown: 5	delay: 5, cooldown: 20
00	13.35	18.87
02	50.62	91.65
05	7.13	5.00
06	7.36	4.85

Table 6.3: Effect of PGO frequency. Kitti results for LDSM with more frequent (left) and less frequent (right) optimizations.

7 Conclusion

In this master's thesis, we have proposed Direct Sparse Mapping with Loop Closure (LDSM), a direct SLAM system with global mapping, loop closure detection, and global pose graph optimization. Benefiting from the advancements of previous methods, we have built our direct SLAM system as an extension of DSM, by modifying the point selection procedure and introducing loop closure detection and pose graph optimization. We have made changes to DSM to prevent addition of incorrect connections to covisibility graph and to reduce search range for covisible window, which are critical for the performance and feasibility of pose graph optimization. We have also built visual debugging tools (see Fig. 5.2, 5.4) which were very useful for tuning loop closure verification steps and understanding issues with the pose graph optimization step.

In our evaluations on public datasets, we showed that the modified point selection procedure which now includes repeatable feature points does not negatively affect the performance. With loop closure correction, we achieved significant improvements for sequences which contain large loops. On the Kitti Odometry dataset, LDSM shows comparable performance to LDSO and ORB-SLAM in monocular settings. We also assessed different parts of our system and shared parameter studies.

In our work, we have mainly focused on the localization accuracy. For a SLAM system, runtime performance is also very crucial. One of the main drawbacks regarding runtime is the unbounded nature of our global map. DSM does not have a map maintenance strategy to remove redundant keyframes and map points from the global map. It avoids duplicating active points in the local window and creating redundant keyframes in previously visited scenes, but accumulating drift limits these to a local scale. With pose graph optimization we successfully correct accumulated drifts and correct the global map, but we do not have a strategy to remove redundancies after correcting the map. Keeping the global map bounded with a map maintenance strategy would significantly improve the runtime performance.

Another aspect we can improve regarding runtime performance is the implementation. We have not utilized a separate thread for loop closure detection and global pose optimization yet. We can also parallelize various subtasks such as the map updates after local and global optimizations. We have not spent time to implement algorithms efficiently and instead opted to use third-party implementations. We can detect time consuming parts and replace them with more efficient implementations, e.g. we can implement a custom PBA solver.

Bibliography

[1] J. Zubizarreta, I. Aguinaga, and J. M. M. Montiel, "Direct sparse mapping," *IEEE Transactions on Robotics*, 2020 (pg. v, 2, 16–18, 37).

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016 (pg. 1).

[3] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017 (pg. 1, 14).

[4] X. Gao, R. Wang, N. Demmel, and D. Cremers, "Ldso: Direct sparse odometry with loop closure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 2198–2204 (pg. 2, 8, 15, 26, 35).

[5] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015 (pg. 2, 8, 13, 14).

[6] E. Eade, "Lie groups for 2d and 3d transformations," *URL http://ethaneade. com/lie. pdf, revised Dec*, vol. 117, p. 118, 2013 (pg. 7).

[7] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, Springer, 2006, pp. 430–443 (pg. 7).

[8] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, IEEE, 1994, pp. 593–600 (pg. 7, 15).

[9] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, Springer, 2010, pp. 778–792 (pg. 7).

[10] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, pp. 1150–1157 (pg. 7, 24).

[11] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, Springer, 2006, pp. 404–417 (pg. 7).

[12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, Ieee, 2011, pp. 2564–2571 (pg. 7, 15, 24).

[13] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2015 (pg. 8).

[14] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012 (pg. 8).

[15] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008 (pg. 8).

[16] R. Muñoz-Salinas, *DBoW3*, https://github.com/rmsalinas/DBoW3, 2017 (pg. 8, 26).

[17] S. Agarwal, K. Mierle, *et al.*, *Ceres solver*, http://ceres-solver.org (pg. 10, 22).

[18] 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 (pg. 10).

[19] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007 (pg. 13).

[20] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, IEEE, 2007, pp. 225–234 (pg. 13).

[21] H. Strasdat, J. M. Montiel, and A. J. Davison, "Visual slam: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012 (pg. 13).

[22] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*, IEEE, 2011, pp. 2320–2327 (pg. 14).

[23] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*, Springer, 2014, pp. 834–849 (pg. 14).

[24] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 3354–3361 (pg. 18, 35).

[25] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000 (pg. 22).

[26] W. Förstner and E. Gülch, "A fast operator for detection and precise location of distinct points, corners and centres of circular features," in *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*, Interlaken, 1987, pp. 281–305 (pg. 23).

[27] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981 (pg. 27).

[28] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016 (pg. 35, 37).

[29] M. Grupp, *Evo: Python package for the evaluation of odometry and slam.* https://github.com/MichaelGrupp/evo, 2017 (pg. 36).