Visual-Inertial Odometry for 3D Pose Estimation and Scene Reconstruction using Unmanned Aerial Vehicles

by

Dylan Bryce Gareau

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

As Unmanned Aerial Vehicles (UAVs) become increasingly available, pose estimation remains critical for navigation. Pose estimation is also useful for scene reconstruction in certain surveillance applications, such as surveillance in the event of a natural disaster. This thesis presents a Direct Sparse Visual-Inertial Odometry with Loop Closure (VIL-DSO) algorithm design as a pose estimation solution, combining several existing algorithms to fuse inertial and visual information to improve pose estimation and provide metric scale, as initially implemented in Direct Sparse Odometry (DSO) and Direct Sparse Visual-Inertial Odometry (VI-DSO). VIL-DSO utilizes the point selection and loop closure method of the Direct Sparse Odometry with Loop Closure (LDSO) approach. This point selection method improves repeatability by calculating the Shi-Tomasi score to favor corners as point candidates and allows for generating matches for loop closure between keyframes. The proposed VIL-DSO then uses the Kabsch-Umeyama algorithm to reduce the effects of scale-drift caused by loop closure. The proposed VIL-DSO algorithm is composed of three main threads for computing: a coarse tracking thread to assist with keyframe selection and initial pose estimation, a local window optimization thread to fuse Inertial Measurement Unit (IMU) information and visual information to pose scale and pose estimate, and a global optimization thread to identify loop closure and improve pose estimates. The loop closure thread also includes the modification to mitigate scale-drift using the Kabsch-Umeyama algorithm. The trajectory analysis of the estimates yields that the loop closure improves the pose estimation, but causes to scale estimate to drift. The scale-drift mitigation method successfully improves the scale estimate after loop closure. However, the estimation error level struggles to exceed the other state-of-the-art methods, namely VI-DSO and VI-ORB SLAM. The results were evaluated on the EuRoC MAV dataset, which contains fairly short sequences. VIL-DSO is expected to show more advantages when used on a longer dataset, where loop closure is more useful. Lastly, using the odometry as a feed, scene reconstruction and the effects of various factors regarding mapping are discussed, including the use of a monocular camera, camera angle and resolution in outdoor settings.

Acknowledgements

I would like to thank everyone who made this these possible. I would like to thank my supervisors, Dr. William Melek and Dr. Baris Fidan, for their feedback and giving me this opportunity to learn and to grow. I would also like to thank my readers, Dr. Stephen Smith and Dr. Arash Arami, for their feedback and helping me to improve my thesis.

Thank you to my colleagues who supported me, especially David Murdoch for his help with software implementation of the software, and Barry Gilhuly for his help with troubleshooting.

Lastly, I would like to my family for supporting me through this time my life and always being there when I need them.

Dedication

This is dedicated to my family, for supporting me in all my endeavors.

Table of Contents

List of Figures

List of Tables

Abbreviations

ATE Absolute Trajectory Error 40, 42

BoW Bag of Words 15, 21, 24, 26, 30

DSO Direct Sparse Odometry iii, 3, 13, 15–17, 19–21, 24, 29, 30

DVSO Deep Virtual Sparse Odometry 14

EuRoC European Robotics Challenges 33, 39–41, 48, 50

FBoW Fast Bag of Words 30

FOV Field of View 19, 21, 34

g2o General Graph Optimization 8, 26

GSD Ground Sample Distance 34, 35, 50

IMU Inertial Measurement Unit iii, 1, 3, 5–7, 14–20, 22, 24, 27–31, 33, 40, 48, 50, 51

LDSO Direct Sparse Odometry with Loop Closure iii, ix, 3, 13, 17, 21, 24, 26, 29, 30, 40, 50, 51

MAP maximum a posteriori probability 10

RE Relative Error 40

REMODE REgularized MOnocular Depth Estimation 4, 5, 15, 32, 33, 37, 38, 50

RMSE Root-Mean-Square Error 40, 43, 46

ROS Robot Operating System 33

SE(3) Special Euclidean Group in Three Dimensions 7, 39, 42, 43, 46

Sim(3) Similarity Transformation Group in Three Dimensions 8, 27, 30, 39, 43

SLAM Simultaneous Localization and Mapping xiii, 6, 12, 13, 15, 27, 31, 49, 51

SO(3) Special Orthogonal Group in Three Dimensions 7

SVO Semi-Direct Visual Odometry 13, 15, 32

UAV Unmanned Aerial Vehicle 1, 3, 4, 6, 7, 12, 16, 24, 32, 33, 36, 40

UAVs Unmanned Aerial Vehicles iii, 1, 2, 6, 36

VI Visual-Inertial 43

VI-DSO Direct Sparse Visual-Inertial Odometry iii, 3, 13, 14, 17, 18, 24, 29, 30, 40, 43, 47–50

VIL-DSO Direct Sparse Visual-Inertial Odometry with Loop Closure iii, 3, 5, 13, 15, 17, 27–30, 32, 33, 37, 39–41, 43, 47–51

VIO Visual-Inertial Odometry 2, 6, 7, 11, 14, 50

VO Visual Odometry 6, 8, 12–15, 21, 40

VSLAM Visual Simultaneous Localization and Mapping (SLAM) 6, 12, 15

Chapter 1

Introduction

Robots are becoming increasing accessible and more common today. In particular, Unmanned Aerial Vehicles (UAVs), especially quadcopters, can be easily utilized in various applications. With the growing number of UAVs, the number of applications also grows. A common configuration for a UAV is to attach a camera for filming or surveillance purposes, or even to provide perspective to help the pilot navigate the drone through its environment. For autonomous vehicles, it becomes critical that the drone is able to locate its own position, as well as determine its orientation, in order to navigate. The combination of position and orientation is referred to as pose. In environments without access to GPS, the camera feed can be used to determined the camera's location by generating structure from motion. UAVs are also often equipped with an Inertial Measurement Unit (IMU) that can be used to further gather information about the pose of the vehicle.

In addition to pose estimation, it can be advantageous to compile the collected data in the form of a three-dimensional (3D) map because extensive visual data is received by the UAV. By reconstructing the entire scene, the user is able to see far more of the situation than the camera's current perspective, and can further evaluate the scenario. In order to accomplish 3D scene reconstruction, the pose of the UAV must be known. This can be accomplished using visual odometry. By comparing images temporally, depth estimates can be generated from a monocular camera, which in turn are used to generate the 3-D model of the environment.

This thesis presents a method for generating pose estimation when using UAVs in outdoor mapping applications. In particular, it focuses on improving visual odometry with inertial data fusion, as well as using loop-closure for better pose estimation. Subsequently, these pose estimates are intended to be fed into 3-D scene reconstruction algorithms to

generate detailed dense maps.

1.1 Motivation

There are events that can rapidly change the face of the planet. For example, landslides or avalanches can change the environment and create situations where timely operations are needed to save lives, while rending existing maps useless. Having information about the scenario can allow for quick and critical decisions in planning rescue operations, as well as other damage mitigation techniques. Often, if the situation is unknown, the rescuers cannot freely move around without jeopardizing their own safety, resulting in slower progress. Statistics show that 93% of avalanche victims survive if dug out within 15 minutes [10], then the survival rates drop fast to 30% at 35 minutes. Scouting ahead with UAVs is much more time-efficient, allow the rescuers to navigate the area and prioritize victims.

Improved pose estimation is also critical to navigation for robots and unmanned vehicles. GPS-denied environments can be challenging to traverse, such as in mountainous, cavernous or underwater regions. Utilizing Visual-Inertial Odometry (VIO) can provide the location of the vehicle, which can be useful in local and global path planning.

The more detailed the map, the better the decisions that can be made. A three-dimensional map would be able to highlight which parts of the terrain are passable. The map could also be input to path planning algorithms to suggest possible routes for rescue crews. In more advanced cases, autonomous rescue vehicles could used to navigate areas without requiring human operators.

Alternatively, UAVs can be used for defense-based systems for monitoring an area. Pose estimation is needed to navigate the environment and mapping can be used to note changes. If there are any significant changes to the scene, ongoing surveillance can quickly detect and report the differences.

1.2 Key Problems

Many of the existing algorithms for pose estimation and mapping are developed in indoor environments. When transferred to outdoor environments, there are various issues with these algorithms. For example, when the scale of the environment increases, it becomes more difficult to ascertain depth. Stereo cameras require larger effective baselines and infrared cameras can be flooded out by sunlight. Outside, there are also issues when the

skyline becomes visible to the camera. The sky view is typically quite uniform, making it difficult to identify features in the video feed. Algorithms are also unable to determine depth for the sky, which is effectively present at an infinite distance. The finite flight time of the UAV is another limiting factor for large outdoor areas, which needs to be taken in consideration when maximizing the coverage.

When using visual odometry as the sole source of pose information, there are various sources of error. The pose estimate can accumulate error and drift from the ground truth. Additionally, when the camera is in fast motion or in low-light scenarios, visual algorithms may struggle to detect features. This can result in loss of tracking and infeasibility of maintaining pose estimates. Monocular camera-based methods also struggle with determining scale for the scene.

1.3 Contributions

Visual-inertial odometry from [36] is able to provide metric estimates while providing advantages of both visual and inertial information. Similarly, loop-closure, as implemented in [15], is able to reduce drift errors over long datasets. In this thesis, the methods of Direct Sparse Odometry (DSO), Direct Sparse Visual-Inertial Odometry (VI-DSO) and Direct Sparse Odometry with Loop Closure (LDSO) are proposed to be integrated into a single method called Direct Sparse Visual-Inertial Odometry with Loop Closure (VIL-DSO), aiming to provide the advantages of both loop-closure and visual-inertial odometry. As the key novel contribution of this thesis, the features of the aforementioned methods are embedded in a single source code and the functions are unified in a single framework. One of the major challenges in achieving this contribution is the variable types for points and keyframes are managed differently in LDSO than in DSO and DSO. This is required to guarantee the point selection repeatability needed for loop closure. In addition to combining the existing methods, scale estimates are not maintained through the loop closure in LDSO. As part of VIL-DSO, a scale-drift mitigation method is proposed involving generating a scale factor between the sets of pose estimates generated before and after loop closure using the Kabsc-Umeyama algorithm. This helps to retain the scale estimates generated from the factor graph optimization used in pose estimation. This scale mitigation approach also requires less computation than recalculating a scale estimate after loop closure, such as by fusing IMU information at a global scale. The resulting pose estimates can then be used for navigation, or as input for other algorithms for mapping or path planning.

One proposed use of the estimated poses from VIL-DSO is to be fed into mapping

algorithms, such as REgularized MOnocular Depth Estimation (REMODE), to generate a dense map. A proposed system combining these is discussed as part of this thesis. Due to the nature of outdoor mapping, modifications to the mapping approach are needed to ensure map quality and coverage. These are intended to help maintain map quality while ensuring that the UAV can cover more area, and are discussed in regards to future work on mapping. This portion will also discuss parsing information more efficiently for depth estimates over large passes.

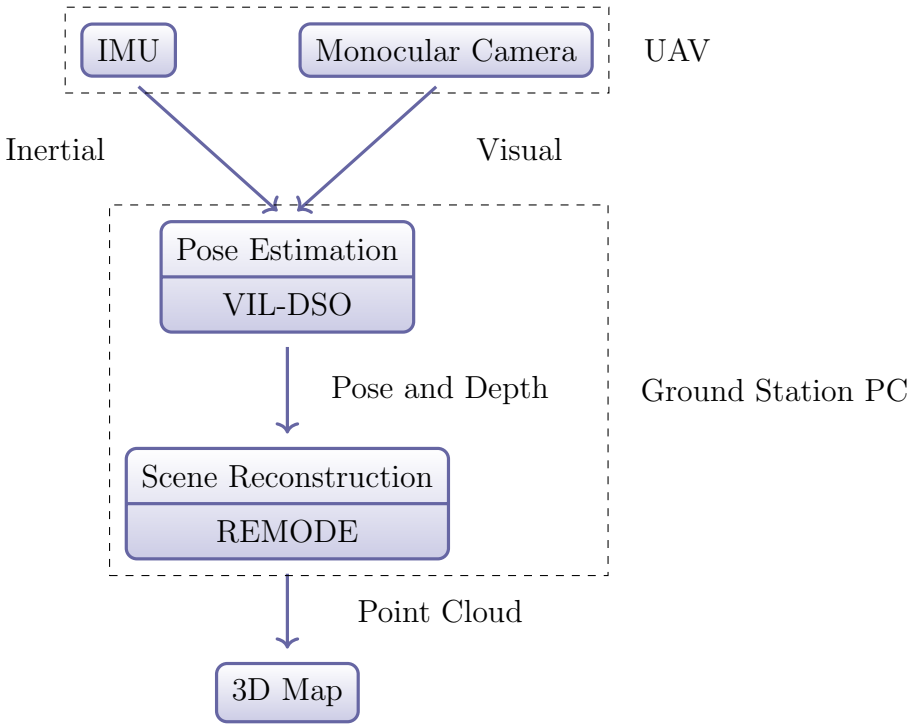


Figure 1.1: Simple layout of the scene reconstruction system.

1.4 Thesis Outline

The background information and the current state-of-the-art are presented in Chapter 2. Chapter 3 will presents the algorithm design for visual odometry and UAV pose estimation.

Chapter 4 covers the proposed methodology of used for 3D scene reconstruction using the pose information and the camera feed, as well as some discussions on for outdoor implementation issues and evaluation of the map. Chapter 5 contains experimental results and comparisons to state-of-the-art methods, and Chapter 6 presents the concluding remarks.

A brief layout of the proposed scene-reconstruction system is illustrated in Figure 1.1. VIL-DSO acts as the front-end pose estimation algorithm, taking in the IMU and camera data to generate pose and depth estimates, which are passed to REMODE, which acts a dense mapping algorithm.

Chapter 2

Literature Review

2.1 Visual Odometry and Visual SLAM Overview

The problem of scene reconstruction is often closely tied to the Simultaneous Localization and Mapping (SLAM) problem in robotics. According to [38], SLAM can be defined as trying to localize the robot or UAV within an unknown environment, while building a map of this environment as the same time. Since UAVs have limited payloads, sensor inputs available focus primarily on visual information, i.e. cameras, as well as inertial information, such as from an IMU, while avoiding the use of costly depth sensors such as RGB-D cameras and LIDAR. Due to the computational intensity of SLAM methods, finding the position and orientation of the robot through odometry is often implemented as the front-end feed for another dense mapping algorithm. These odometry methods can be classified as Visual Odometry (VO) or VIO depending on the information used. Similar, SLAM methods which depend on visual information are often referred to as Visual SLAM (VSLAM)

The combination of position and orientation is often called the UAV's pose. One of the key requirements for pose estimation is generating depth estimates of the scene. Stereo cameras can generate depths estimates, but are ineffective at long distances due to relative baselines. Typically in stereo photography, a general rule of thumb for the baseline is 1:30, where the baseline is 1/30 the length of the furthest object. This means for a UAV at 90 meters, a three meter baseline would be needed. Alternatively, RGB-D camera that utilize infrared reflections may be used, but are not effective in outdoor scenarios due to ambient lighting flooding the scene. Therefore, for large-scale outdoor mapping, monocular cameras are a more practical choice. When implemented as a standalone solution, monocular VO is

able to effectively generate pose estimates of the UAV, but is unable to observe scale. These methods use an arbitrary scale is assigned when the algorithm is initialized. Conversely, integrating inertial data from an IMU can generate pose measurements in the metric scale, but also can accumulate drift errors due to the accumulation of noise. By combining the visual and inertial information, the resulting odometry method can have both estimated metric scale and reduced drift error.

In scene reconstruction, a number of dense mapping algorithms rely on accurate pose estimates. When precise locations are not available through either Real-Time Kinematic (RTK) positioning or a camera system such as VICON, VIO can provide relatively accurate positioning with little hardware cost.

Before discussing these algorithms, important conventions and background information is covered.

2.2 Transformations and Pose Representations

This section summarizes the pose representations and the reference frame conventions used in VIO and pose estimation.

2.2.1 Lie Groups for 3D Transformations and Poses

To represent pose estimates, transformations in 3D space are used, and represented using Lie groups [6]. The pose of an object refers to both its position and orientation together, which correspond to translation and rotation respectively in terms of rigid-body motions.

Rotations in 3D form a special orthogonal group of order 3, SO(3), and are represented by 3x3 orthogonal matrices

$$\mathbf{R} \in \mathrm{SO}(3). \tag{2.1}$$

Translation in 3D can be represented by a vector in $\mathbb{R}^3$. Any arbitrary rigid-motion can be represented as a combination of a rotation and a translation. Therefore, all rigid-body motions in 3D form a special Euclidean group of order 3, SE(3), and are represented using homogeneous transformation matrices

$$C = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right) \in \mathrm{SE}(3) \tag{2.2}$$

which consist of a rotation matrix and a translation vector

$$\mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3. \tag{2.3}$$

Finally, similarity transformations are a combination of rigid-body motions and a scaling factor, s. In 3D space, they are are represented by the group, Sim(3), as follows:

$$\mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3, s \in \mathbb{R}, \tag{2.4}$$

$$T = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0} & s^{-1} \end{array}\right) \in \mathrm{Sim}(3). \tag{2.5}$$

The definitions and representations above allow us to manage combining transformations in different scales, such as visual odometry in its own arbitrary scale, and inertial odometry in the metric scale. Both SE(3) and Sim(3) transformations are often used in VO depending on how the particular VO method deals with scale.

2.2.2 Visual-Inertial Transformation Conventions

When working with different frames of references, frame notations and conventions need to be consistent. Typically the inertial reference frame is associated with the body of the robot, while the visual frame of reference is aligned with the camera. Thus, T_{bc} (analagous to T_c^b) refers to the transformation from the camera to the body. Similarly, T_{cw} (or T_w^c) represents the transformation from the world frame to the camera frame, which is also called the camera's pose in the world frame. T_{cw} is often used to represent the pose of the drone, since the camera's location is effectively the same the drone's location.

2.3 Factor Graphs and Optimization

Rather than estimating only the current pose, many algorithms optimize the pose estimation across multiple time steps using factor graphs. A brief review of factor graph optimization, based on notes from Dellaert [4], and Dellaert and Kaess [5], as well as the General Graph Optimization (g2o) framework [21], is described below.

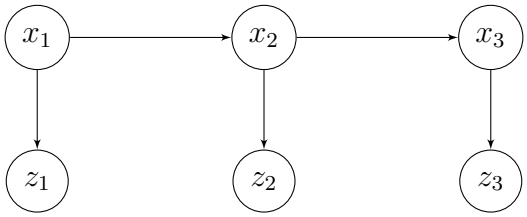


Figure 2.1: Hidden Markov Model, modeled as a Bayesian network

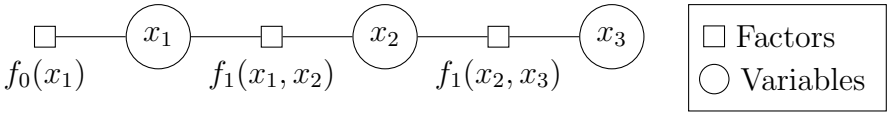


Figure 2.2: Hidden Markov Model, modeled as a factor graph

2.3.1 Factor Graph Models

A dynamic system can be often modeled as a Markov chain, in which the sequence of possible states is entirely dependent on the previous state values. In the context of these examples, the relevant states are pose $x \in \mathbb{R}^6$. A hidden Markov model assumes that in each time step t, the states x_t are not observable, and must be determined from the observations $z_t \in \mathbb{R}^6$, as shown in Figure 2.1. Note that the set of states x_t from all time steps is represented by X, and the set of observations z_t from all time steps is represented by Z. The observations Z are dependent on the hidden states X, resulting in the conditional probability density, $P(Z|X)$. In the example shown in Figure 2.1, given a series of known measurements z_1, z_2, z_3, the task is to find the hidden state sequence (x_1, x_2, x_3) which maximizes the posterior probability $P(x_1, x_2, x_3|z_1, z_2, z_3)$.

The dynamic system and Markov chain illustrated in Figure 2.1 can be alternatively represented using a factor graph. A factor graph is a bipartite graph that consists of two kinds of vertices: factors and variables. Variables represent the unknown states in the estimation, while the factors represent the probabilistic information on the variables that is gained from the observations. Edges exist only between factors and variables, connecting each factor to the variables that the factor provides information on. The alternative factor graph to the estimation problem in Figure 2.1 is illustrated in Figure 2.2. The factors f_i correspond to the relative poses, as generated from the observations z_i.

The value of the graph can be generated by taking the product of the factors

$$f(X) = \prod_i f_i(x_i). \tag{2.6}$$

The task is to estimate the subset of state variables $x_i \in X$ that maximizes the value of the graph. In other words, the factor graph is maximized by finding the values of each variable that maximizes the probabilities of receiving the given measurements Z. This yields the maximum a posteriori probability (MAP) state as

$$X^{\text{MAP}} = \arg\max_X \prod_i f_i(x_i). \tag{2.7}$$

The state estimates are generated through optimization. The residual function $r(x_t, z_t)$ of the factor graph can be used to generate the cost function used in optimization. A simple example of a residual function is

$$r(x_i, z_i) = h(x_i) - z_i, \tag{2.8}$$

where $\mathbf{h}(x)$ is the observation or sensor model that maps the observations Z, from the states, X.

$$h : X \to Z \tag{2.9}$$

The goal of the estimation is to find the values of X such that the likelihoods of observations Z are maximized, as in

$$p(Z|X) \propto exp\left(-\frac{1}{2}||h_i(x_i) - z_i||^2_{\Sigma_i}\right). \tag{2.10}$$

where Σ_i is the covariance matrix and $||\cdot||_{\Sigma_i}$ is the Mahalanobis norm. Since the logarithm of 2.10 is monotonic, maximizing the probability is equivalent to minimizing the negative log-likelihood function (as well as dropping the $\frac{1}{2}$ factor), which results in

$$X^{\text{MAP}} = \arg\min_X \sum_i ||h_i(x_i) - z_i||^2_{\Sigma_i}. \tag{2.11}$$

This is the same as minimizing the sum of nonlinear least squares. The cost function $g(X)$ is subsequently defined as

$$g(X) := \sum_i ||h_i(x_i) - z_i||^2_{\Sigma_i}. \tag{2.12}$$

Optimizing the cost function yields estimates for the entire sequence of states, which tends to yield a more smooth trajectory, as opposed to methods that only estimate the current state. Two common methods for solving the non-linear least squares problems include Gauss-Newton optimization and Levenberg-Marquardt optimization.

2.3.2 Marginalization

Due to the nature of continuous VIO, additional data is constantly being added to the system and constantly increases the complexity of the factor graphs. While it is possible to simply omit variables, and subsequently any factors tied to them, a more accurate optimization result can be obtained by marginalization. Marginalization is done using the Schur complement, as presented in [7]. If a system of equations can be expressed a block matrix, the Schur complement decomposes the block matrix into components, which allows for elimination of variables while retaining the information from the system of equations.

Assuming the cost function used to minimize the residuals is in the form

$$E(x) = 2x^T\mathbf{b} + x^T\mathbf{H}x + c, \tag{2.13}$$

the constant c can be dropped and it can be rewritten as a linear system (Gauss-Newton), which results in

$$\mathbf{H}\mathbf{x} = \mathbf{b}, \tag{2.14}$$

$$\begin{bmatrix} \mathbf{H}_{\alpha\alpha} & \mathbf{H}_{\alpha\beta} \\ \mathbf{H}_{\beta\alpha} & \mathbf{H}_{\beta\beta} \end{bmatrix} \begin{bmatrix} \mathbf{x}_\alpha \\ \mathbf{x}_\beta \end{bmatrix} = \begin{bmatrix} \mathbf{b}_\alpha \\ \mathbf{b}_\beta \end{bmatrix}, \tag{2.15}$$

where α are the variables we are retaining, and β refers to the variables that are being marginalized. Applying the Schur complement yields

$$\widehat{\mathbf{H}_{\alpha\alpha}}\mathbf{x}_\alpha = \widehat{\mathbf{b}'_\alpha}, \tag{2.16}$$

where

$$\widehat{\mathbf{H}_{\alpha\alpha}} = \mathbf{H}_{\alpha\alpha} - \mathbf{H}_{\alpha\beta}\mathbf{H}_{\beta\beta}^{-1}\mathbf{H}_{\beta\alpha} \tag{2.17}$$

$$\widehat{\mathbf{b}'_\alpha} = \mathbf{b}'_\alpha - \mathbf{H}_{\alpha\beta}\mathbf{H}_{\beta\beta}^{-1}\mathbf{b}'_\beta. \tag{2.18}$$

This is analogous to taking a system of equations and eliminating a variable, except with matrices. This gives us a new energy function related to x_α alone,

$$E'(x_\alpha) = \widehat{\mathbf{b}'_\alpha} + x_\alpha^T\widehat{\mathbf{H}_{\alpha\alpha}}x_\alpha. \tag{2.19}$$

Ultimately, this allows for the elimination of variables from the factor graph, while still retaining information from the edges (or factors) that were connected to it.

2.3.3 Pose Graph for Loop Closure

A simple example of a factor graph application is its use in loop closure. In this application, the poses of the UAV can be modeled as the graph variables, while the VO measurements are the factors, or edges, tying the variables together. When a loop is detected, an edge is generated between the corresponding poses. This kind of loop closure is used in LDSO [15] and is discussed in more detail in chapter 3.

2.4 Existing VO and VSLAM Methods

VO and VSLAM methods can be categorized by the formulation used. Direct methods use the actual sensor values, such as the brightness of each pixel. In the case of vision systems, it is using the pixels in the frames and trying to minimize photometric error. Alternatively, indirect methods use some level of pre-processing to generate an intermediate representation [7]. For example, this could be creating features based of various criteria such as corners or line-segments, and then minimizing the geometric error between these features. In addition to this, the sparse methods are used on only a certain selection of points, often intended for navigational purposes, while dense method method attempt to construct using all of the pixels available. These dense methods are often more precise, but also more computationally expensive. Dense methods also use the formulation of a geometry prior by utilizing the connections between pixels in the image, which can help yield smoothness in the results. Since in the context of search-and-rescue and real-time navigation, the goal is to reach real-time processing speeds, sparse methods are often more favorable. There are also differences in the formulations used to estimate the states. Some frameworks use extended Kalman filters, while others use non-linear optimization methods that can be modeled as factor graphs.

Since dense and spare methods are not mutually exclusive to direct and indirect methods, combinations can and do exist.

2.4.1 Sparse and Indirect Methods

Sparse and indirect methods tend to be some of the most widely-used techniques. These algorithms estimate 3D poses through matching keypoints or features, and utilize geometric error without generating a geometry prior. Some examples include Jin et al. [16], PTAM [20], and monoSLAM [3]. This also includes ORB-SLAM [24], and its successor, ORB-SLAM2 [25], which are key examples of SLAM algorithms that utilize the keyframe-based

approach proposed in [23] and developed further in [26]. These complete SLAM solutions yield some of the best results for pose estimation in the state-of-the-art.

2.4.2 Dense and Indirect Methods

The combination of dense and indirect is not as common in literature. These methods utilize geometric errors and the formulation of a geometry prior. Most indirect methods use features that reduce the amount of information, resulting in sparse formulation, but dense and indirect algorithms work from a dense, regularized flow field instead of features. Examples include [29] and [35]. These methods use the optical flow from consecutive frames for the generation of dense depth maps.

2.4.3 Dense and Direct Methods

Dense and direct methods optimize photometric error without using features, while also creating geometry prior to help determine the structure of the scene. Examples include DTAM [27] and its precursor [32]. LSD-SLAM [8] is also an example of this type of algorithm, working directly on the pixels in the images to form a semi-dense map for areas with significant intensity gradients. DPPTAM [2] is another method that combines odometry for pose estimation with dense scene reconstruction to produce a map.

2.4.4 Sparse and Direct Methods

Sparse and direct methods optimize photometric error directly on camera images without requiring intermediate features, and do not use a geometry prior to formulate the structure. The method from Jin et al. [11] uses an extended Kalman filter, while other examples such as DSO [7] and subsequent works, LDSO [15] and VI-DSO [36], use non-linear optimization, which can be modeled as a factor graph. The VIL-DSO method described in chapter 3 and presented in this thesis is also based on this formulation of VO.

2.4.5 Hybrid Methods and Deep Learning

There are also methods that do not fit exactly into any of the above categories. For example, Semi-Direct Visual Odometry (SVO) [12] and SVO2.0 [13] uses a semi-direct method, which includes an indirect formulation as part of its optimization.

In addition to this, some newer methods are starting to integrate machine learning into their algorithms. One such example is Deep Virtual Sparse Odometry (DVSO) [37], which uses training data to help with depth estimation. However, these algorithms are reliant on training data, and are less effective in scenarios that differ significantly from the training set. In the case of drastically shifted terrain in a natural disaster, it may be difficult to obtain sufficient training samples. Urban environments are easily repeatable and more appropriate for these applications.

2.4.6 Visual and Inertial Fusion

A number of VO methods also incorporate inertial information, resulting in VIO. These algorithms are able to observe scale without using RGB-D cameras. For example, there are implementations of ORB-SLAM that fuse visual and inertial information, as proposed in [26]. LearnVIORB-SLAM [17] is one such implementation that is later used in this thesis for algorithm comparisons.

Visual-inertial data fusion can be classed as either tightly-coupled or loosely-coupled methods. Tightly-coupled methods jointly optimize the motion information from the IMU with the visual information as part of the energy function used in the optimization. Conversely, loosely coupled methods apply the estimates from the VO as inputs into the combination algorithm used for optimization. This is typically easier to implement as it does not require modification of the VO algorithm. An example of loose-coupling is presented in [19], where an extended Kalman Filter is used to fuse IMU information with odometry measurements. VI-DSO is a tightly-coupled method, as it use IMU information directly in the VO portion of the estimation.

2.4.7 Gaps and Contributions in Visual Odometry

The motivation for using a direct and sparse formulation is analagous to DSO. Direct avoids the issues with features and lighting while sparse negates the issues related to the complexity of solving for a geometry prior, making real-time calculations feasible.

Visual odometry methods that only utilize monocular camera do not have observable scale The scale used is arbitrarily initialized, which make it difficult to do path planning with the pose estimates. A common solution to this is to gather depth information using an RGB-D camera, which may use either stereo or infrared reflection to estimate depth. Unfortunately, stereo cameras are only typically effective within a distance of 10 to 15 times

the baseline, and infrared radiation is often flooded by ambient light in outdoor scenarios. This makes these solutions nonideal for large-scale surveillance in outdoor scenarios.

Alternatively, metric scale can be obtained by including inertial information from an IMU. This is independent of what the camera sees, and as a result does not have the same weaknesses as RGB-D cameras. The main drawback to integrating inertial information is that it can be noisy and lead to accumulated drift, especially over long flights. To remedy this, loop closure is used. Loop closure identifies features that have been seen before, and includes them as part of the pose-graph optimization to reduce the drift error when a drone revisits an area.

Ultimately, the odometry algorithm used in this thesis is called Direct Sparse Visual-Inertial Odometry with Loop Closure, which is based off the visual odometry from DSO, while utilizing dynamic marginalization and Bag of Words (BoW) loop closure. Due to the loop-closure causing scale drift, a method is introduced to reduce this drift by matching the loop-closed pose estimates with their estimates from before loop closing.

2.5 Mapping after Pose Estimation

The algorithms discussed in the previous section include both odometry and SLAM solutions. In VSLAM solutions, the visual information is used to determine the location of features or pixels and allow for the generation of the map. In contrast, the primary goal of VO is to find the pose of the camera, and the map is not optimized. Some dense mapping algorithms struggle with reconstruction in real-time, while lighter odometry methods are able to generate pose with less computation. Once pose has been estimated, it can be used as the reference point for a scene reconstruction algorithm. Some complete scene-reconstruction system used odometry as front-end estimation for states, while using a dense mapping method at the back-end for scene reconstruction. In particular, REMODE [28] is a method that relies on robust pose estimates to form a dense map. A combined implementation using REMODE and SVO is described in [9]. However, these methods are more suitable when scene reconstruction quality is priority. In contrast, the priority of this thesis is surveying a large area quickly and efficiently in the event of a natural disaster. This thesis proposes VIL-DSO combining with REMODE to create a complete scene-reconstruction system, as well as noting specific considerations for outdoors mapping.

Chapter 3

Visual-Inertial Odometry for Pose Estimation

In order to generate accurate scene reconstructions, accurate pose estimation for the UAV is required. The accuracy of the pose estimation is critical in scene reconstruction since the pose of the drone is used as the reference point for the visual information used in reconstruction, as noted in algorithms such as [28]. The design goal of this chapter focuses on is to generate good pose estimates for the camera, and subsequently the UAV, using only monocular camera and IMU as sensors.

As part of the visual-inertial odometry implementation, a keyframe-based framework is used, similar to the one developed in [23] and used in [26]. The system states, such as the pose and the velocity, are tied directly to each keyframe. Keyframes are often used to manage visual information in classical visual-only systems [23]. To extend this framework, the IMU data and poses are aligned to these keyframes for simple management of the states. Keyframes with arbitrary intervals are still useful in computations, as long as the critical information, such as timestamps, is retained. This prevents the need for a constant rate of keyframes.

The odometry solution proposed in this thesis is based on DSO [7], which is available as open-source code. The keyframe-based framework considered in this thesis has three main threads to cover coarse tracking, local keyframe optimization, and global loop closure, as summarized in Figure 3.1. The tracking thread focuses on managing tracking features in order to generate rough pose estimates between keyframes. The local optimization thread forms pose estimates by minimizing the photometric error between the points and the frames, while also fusing IMU information with the visual information. This thread utilizes

an implementation of VI-DSO [36] as implemented by Sun in [33], as the original source code for VI-DSO was unavailable. The global optimization thread manages keyframes at a global scope, where it matches the current keyframe to a database and completes loop closures to improve the final pose estimates, as initially proposed in LDSO [15]. The combined algorithm is called Direct Sparse Visual-Inertial Odometry with Loop Closure (VIL-DSO). The layout between the threads for VIL-DSO is illustrated in Figure 3.2.

In terms of implementation, the DSO code of [7] forms the basis for the coarse tracking thread and the local keyframe optimization thread. VI-DSO adds IMU information to these two threads, and also implements dynamic marginalization. LDSO extends DSO by modifying the point selection method and adding the global optimization thread for loop closure. The proposed VIL-DSO algorithm integrates all these features, and adds on scale-drift mitigation to improve post loop-closure scale estimates. This chapter explains the details of this integration and the scale-drift mitigation add-on.

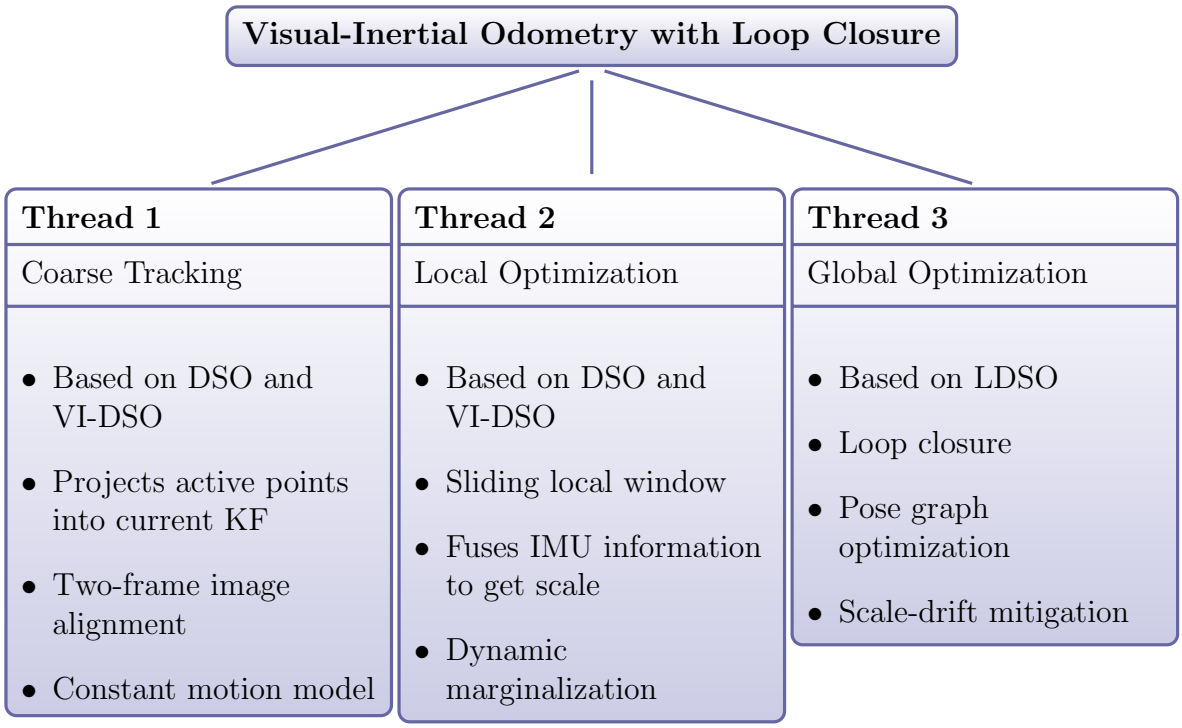


Figure 3.1: Main threads in VIL-DSO.

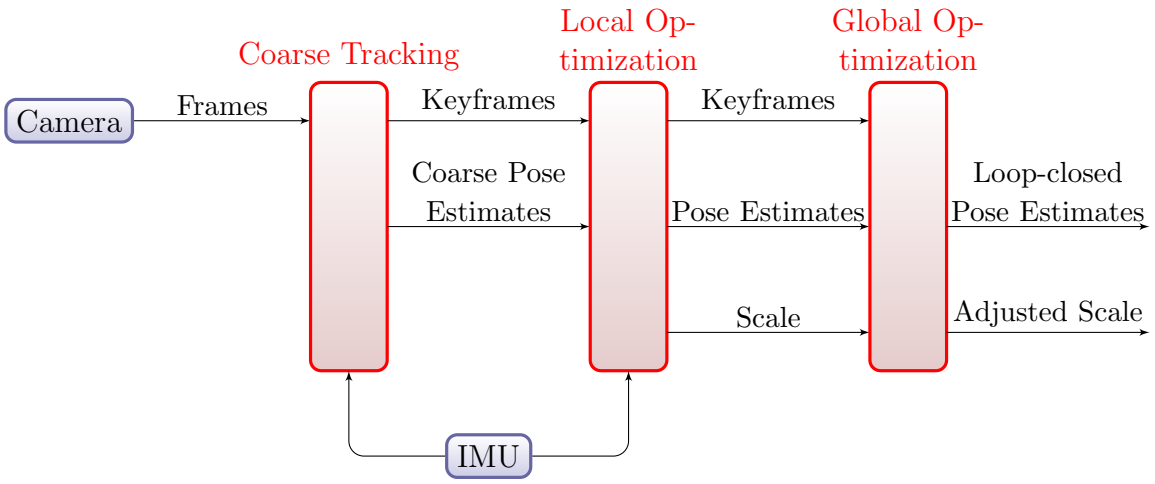


Figure 3.2: Layout of VIL-DSO

3.1 Initialization and Pre-integration of IMU

A number of key parameters need to be estimated prior to fusing IMU information into the factor graph. These include the gyroscope and accelerometer biases, as well as the magnitude of gravity. The initialization implementation used is based on VI-DSO [36], where initial estimates for the biases, b_0, are set to zero and the scale is set to 1.0. Gravity direction and magnitude are determined by averaging the first 30 measurements from the IMU. Note that scale is defined as the ratio between distances in the odometry reference frame and distances in the world reference frame.

Since IMU information is generated at a higher frequency than camera frames, the IMU data is pre-integrated to form single instances of data that can be fused at each time step. These time steps correspond to the time when each visual frame is taken, thus the pre-integrated IMU information represents all IMU readings taken between consecutive frames or keyframes, depending on where the IMU information is used. Pre-integration is calculated using a simple constant-acceleration model.

3.2 Keyframe and Point Management

Each frame that is input into the odometry program passes through a number of steps to determine if it will be used as a keyframe for the pose estimation in the local optimization

thread. First, all frames are tracked relative to the latest keyframe, as part of the coarse tracker. Next, a check is performed to see if a new keyframe is needed. A keyframe is generated from the current frame if it is needed. Lastly, old keyframes are marginalized as required.

3.2.1 Coarse Tracking Thread

The coarse tracking thread utilizes visual information to generate rough pose estimates between frames. This allows for tracking in between keyframes, where optimization is not used. It also provides an initial rough estimate for the local-window optimization to start at. This thread comes from DSO's codebase [7] with additions from [36] and [15]. The coarse tracker first projects all active points into the current keyframe and dilates them. Then, it uses conventional direct image alignment between the current frame and the active keyframe, during which the scale is fixed. As part of this alignment, multi-scale pyramid representation and a constant motion model are used. In addition, inertial residuals are calculated from the IMU pre-integration and included between subsequent frames. After the local-window optimization is completed, a new keyframe is generated and the coarse tracking is reinitialized, using the new scale, gravity, bias and velocities as references for the visual factors. The resulting poses from the coarse tracker are used as initial estimates in the local optimization thread.

3.2.2 Keyframe Generation

Keyframe generation is based on a criteria check, as proposed in DSO [7]. If a weighted summation of the following criteria exceed a predefined threshold, then a new keyframe is generated:

1. The Field of View (FOV) changes significantly, as measured by the mean square optical flow,

$$f := \left(\frac{1}{n} \sum_{i=1}^{n} ||\mathbf{p} - \mathbf{p}'||^2 \right)^{\frac{1}{2}}, \tag{3.1}$$

   where n is the number of frames since the last keyframe and $\mathbf{p}$ is the position.

2. The translation of the camera causes occlusion of the points, which is indicated by the mean flow without rotation,

$$f_t := \left(\frac{1}{n} \sum_{i=1}^{n} ||\mathbf{p} - \mathbf{p}'_t||^2 \right)^{\frac{1}{2}}, \tag{3.2}$$

where $\mathbf{p}_t$ is the warped point position with $\mathbf{R} = \mathbf{I}_{3x3}$

3. There is significant change in brightness, as indicated by a relative brightness factor between the keyframe and the current frame,

$$a := |log(e^{a_j - a_i} t_j t_i^{-1})| \tag{3.3}$$

The weighted summation of the above criteria is defined as

$$b := w_f f + w_{f_t} f_t + w_a a \tag{3.4}$$

where w_f, w_{f_t}, w_a are the relative weights. If this measure, b, exceeds a predefined threshold, T_{kf}, then a new keyframe is generated. The threshold, T_{kf}, is arbitrarily defined and set in DSO as $T_{kf} = 1$. The adjustment of the weights and this threshold was not explored.

After the weighted criteria is checked, an additional check against the IMU timestamps is calculated. If the time between IMU timestamps for the current frame and the previous keyframe exceeds a threshold (0.45 seconds is used) and the cumulative measure, b, in (3.4) exceeds another threshold, T_{imu}, then a new keyframe is taken. This is to ensure that keyframes continue to be taken when IMU data is absent. Similarly to the previous threshold T_{kf}, T_{imu} is also arbitrarily defined and was not adjusted from the source code, which was set as $T_{imu} = 0.45$. A new keyframe is also generated if it has been too long time-wise since the previous keyframe was generated.

3.2.3 Point Generation

In pose estimation, having excessive information is redundant, so only a limited number of points are used, which is also noted in [7] and [28]. A limited number of point candidates are proposed in each keyframe. The default value is 1500 in the original implementation of DSO. Similarly, the total number of active points retained across all active frames is limited to N_p. An arbitrary number, $N_p = 2000$, is used to keep computation minimal. Points are selected using a combination of two different methods. The first method is from

DSO [7], which uses a dynamic grid search. This is used to ensure that enough pixels are selected from portions of the image that may be poorly textured. The second method utilized is from LDSO, which selects corners that are detected from the Shi-Tomasi score [31]. Selecting corners improves repeatibility in point selection and improves the likelihood of achieving matches required for the BoW loop closure.

3.2.4 Marginalization

The goal of marginalization is to remove the old states, reducing the complexity of calculations, while retaining some a priori information about the previous states. The criteria to decide if certain data is marginalized or not is based on frame and pose management from the VO. Once the data from a frame is marginalized, associated factors are removed from the factor graph and a new factor is formed, acting as a priori information for the window. Marginalization is completed using the Schur-Complement, as detailed in preliminaries.

In order to reduce the amount of data being processed in the factor graph, older data is marginalized in a sliding-window approach, accordingly to a list of criteria from DSO [7], where I_1 is the current keyframe:

1. Keep the latest two keyframes, I_1 and I_2.
2. Marginalize a keyframe if it shares less than 5% of its points shared with I_1.
3. Marginalize a keyframe if the number of keyframes exceeds the set threshold, N_I. A distance score between keyframes is calculated to select the frame to remove,

$$s(I_i) = \sqrt{d(i,1)} \sum_{j \in [3,n] \setminus \{i\}} (d(i,j) + \epsilon)^{-1}, \tag{3.5}$$

   where $d(i,j)$ is the Euclidean distance between keyframes I_i and I_j, and ϵ is a small constant.
4. Marginalize a point if it is no longer in the FOV.
5. Marginalize a point if its host frame is marginalized.

VI-DSO [36] introduces a new method for marginalization called dynamic marginalization. In this, multiple marginalization factors are calculated and maintained, and are reset when the scale estimate deviates too much. The three factors used are:

1. M_{visual}, which contains the scale-independent information generated from the visual information, and no IMU information.

2. M_{curr}, which contains all information since the scale linearization point was set.

3. M_{half}, which contains only the recent states with scale estimates similar to the current scale estimate.

Whenever the current scale estimate deviates too much from the scale estimate at the linearization point, the marginalization priors are reset by setting M_{curr} to M_{half} and M_{half} to M_{visual}. This results in the optimization always retaining some level of information from prior states regarding the scale estimate.

3.3 Local Optimization Thread

In the local graph optimization thread, the factor graph minimizes photometric error between keyframes by optimizing energy residuals, as implemented in [7]. This is calculated within a sliding window to minimize the amount of information in the computation. Frames and points that leave the field of view are marginalized. The remaining keyframes form a window that is local to the current keyframe. Only these keyframes and their associated poses are optimized. The factor graph detailed in Figure 3.3 shows how the error terms used in the optimization are created. An error term is generated between the host keyframe of a point, and every keyframe that shares that point. For example, $E_{\mathbf{p}1^3}$ is the error term generated by Point 1 ($\mathbf{P}1$) hosted by Keyframe 1 and overlapped with Keyframe 3. These error terms are further defined in [7]. Edges between points and their host keyframes are indicated with blue lines, while edges between points and other overlapping keyframes are indicates with red lines.

The sum of these energy terms represents the full photometric error for all of the keyframes and points involved in the factor graph. This is defined in [7] as

$$E_{photo} := \sum_{i \in F} \sum_{\mathbf{p} \in P_i} \sum_{j \in obs(\mathbf{p})} E_{\mathbf{p}j}, \tag{3.6}$$

where i runs over all keyframes F in the local window, $\mathbf{p}$ is over all points P in keyframe i, and j is over all keyframes in which point $\mathbf{p}$ is observable. Optimization of the factor graph is executed using the Gauss-Newton algorithm. It is unlikely to converge to an incorrect

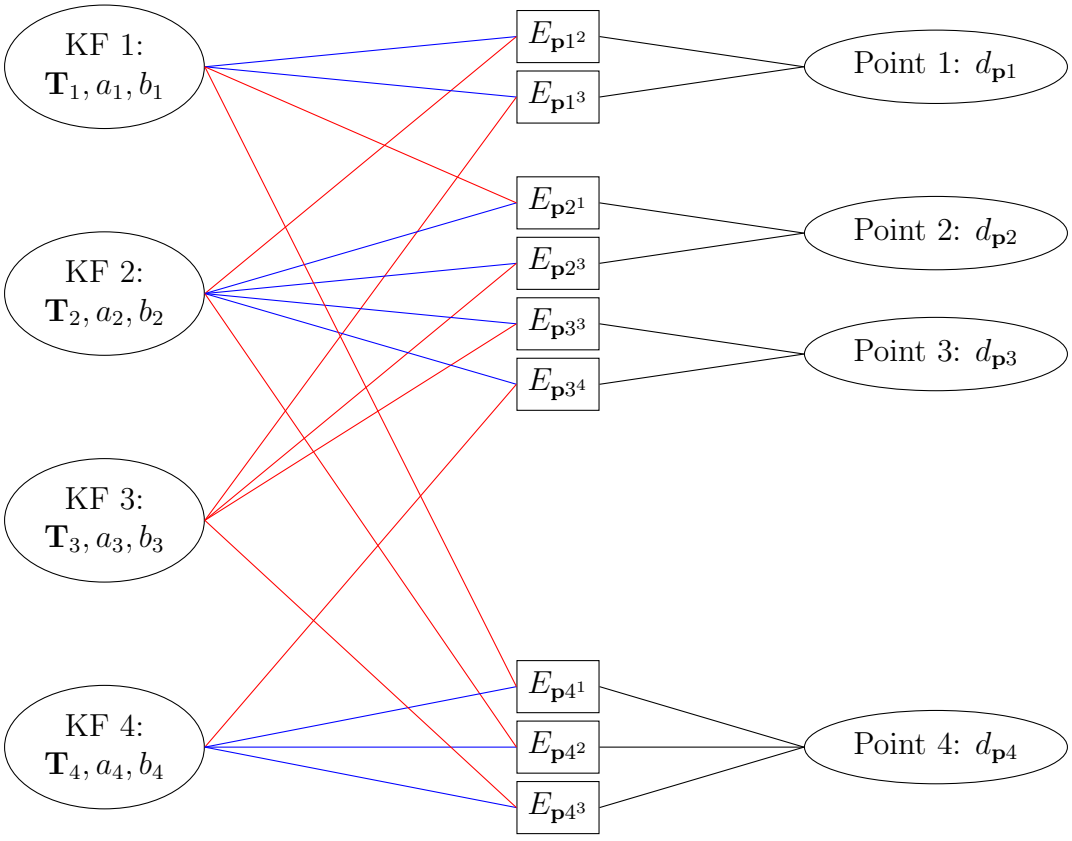


Figure 3.3: Factor graph of bundle adjustment to minimize photometric error, from DSO [7].

local minimum because initial pose estimates from tracking are used. The Levenberg-Marquardt algorithm is not used because it adds an additional level of computation while not substantially improving the results, as noted in [7].

To improve on the resulting state estimates from DSO and add metric scale, VI-DSO [36] proposes a factor graph that includes pose, IMU biases, velocity, visual variables, gravity direction and scale. Visual variables include camera intrinsics and brightness transfer function parameters. Inertial and visual measurements are used to generate factors in the graph that relate the unknown variables using the known probabilistic information. The resulting factor graph is illustrated in Figure 3.4.

3.3.1 Scale Estimation

As part of the local window optimization, a transformation from the DSO reference frame to the world reference frame is estimated. This transformation is expressed as $T_{WD} \in Sim(3)$, and includes both rotation and translation. It also includes a scalar scale factor that represents the ratio between distances in the DSO reference frame and the world reference frame. This transformation is continuously iterated, and then applied to poses when they are read by the visualizer or output from the program.

3.4 Global Keyframe Bundle Adjustment and Loop Closure

While each of the poses is calculated in a local window, these pose estimates do not account for when the UAV returns to previously visited locations. When returning to these locations, it is possible to reduce drift error by utilizing loop closure. A pose graph for loop closure is generated using all of the keyframes. This is different from the sliding local window, which only uses keyframes near the current frame. Instead, it uses a global scope which uses keyframes from the entire dataset. The factors between keyframes in the global pose graph are generated from relative poses that are calculated from the pose estimates of the local sliding window. The initial implementation of this thread is from LDSO [15]. The difference between the local and global scopes is illustrated in Figure 3.5.

In order to detect loop closure, the ORB descriptors [30] are calculated from each keyframe. These are repeatable due to using the corner features generated by the point selection method from LDSO [15]. The descriptors are then packed into a BoW database

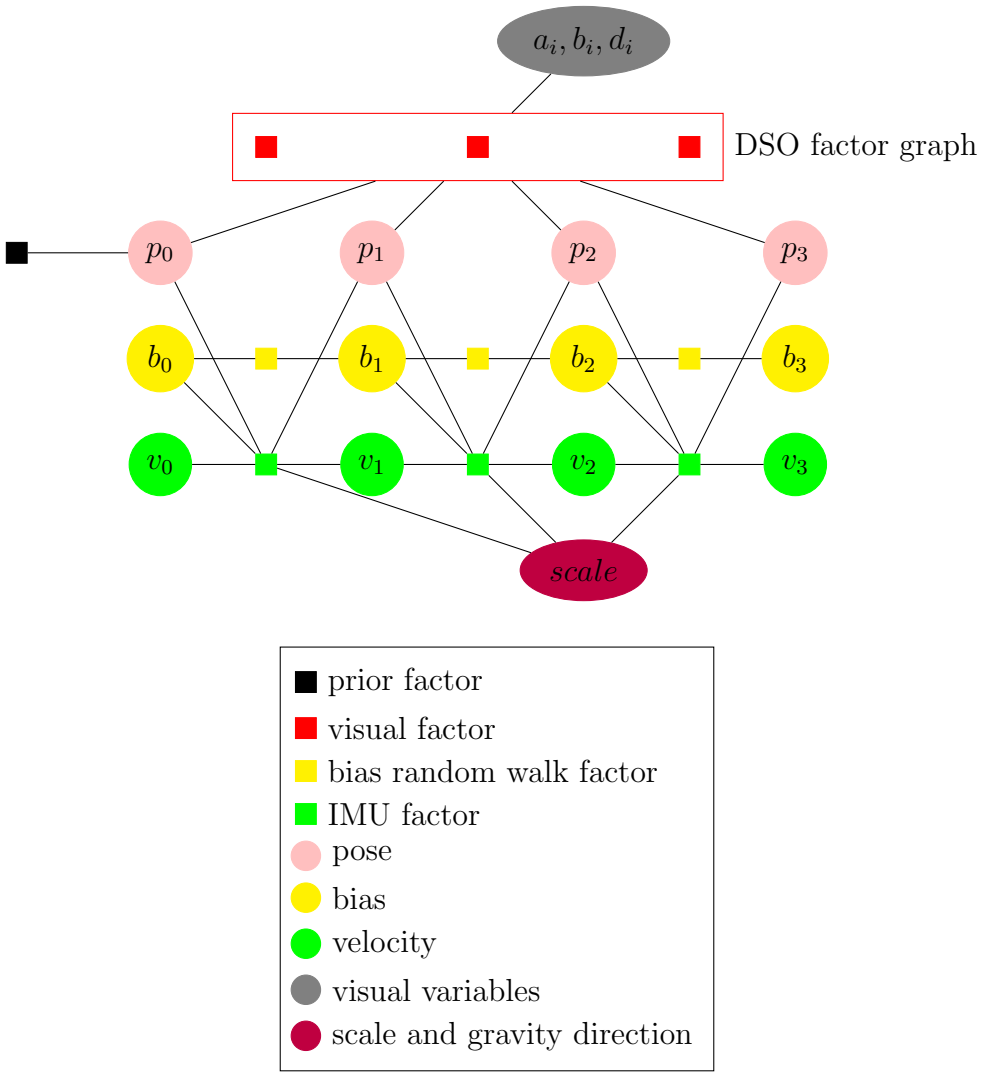


Figure 3.4: Factor graph for VI-DSO [36].

using DBoW3 [14]. When a new keyframe is generated, matching candidates are proposed by referencing the BoW database. When there are enough matches, a loop is detected and a Sim(3) transformation between the matched frames is calculated. This transformation is used to generate an edge in the global pose graph, as illustrated in Figure 3.6.

Factor graphs are implemented using the g2o framework [21]. Optimization is done using the Gauss-Newton method instead of the Levenberg-Marquardt method. The pose estimates from the local optimization thread are used as initial estimates to ensure convergence.

3.4.1 Scale Drift and Loop Closure

In the local optimization thread, pose and scale are estimated. Unfortunately, only poses are passed to the global optimization thread for loop closure, which means that the scale

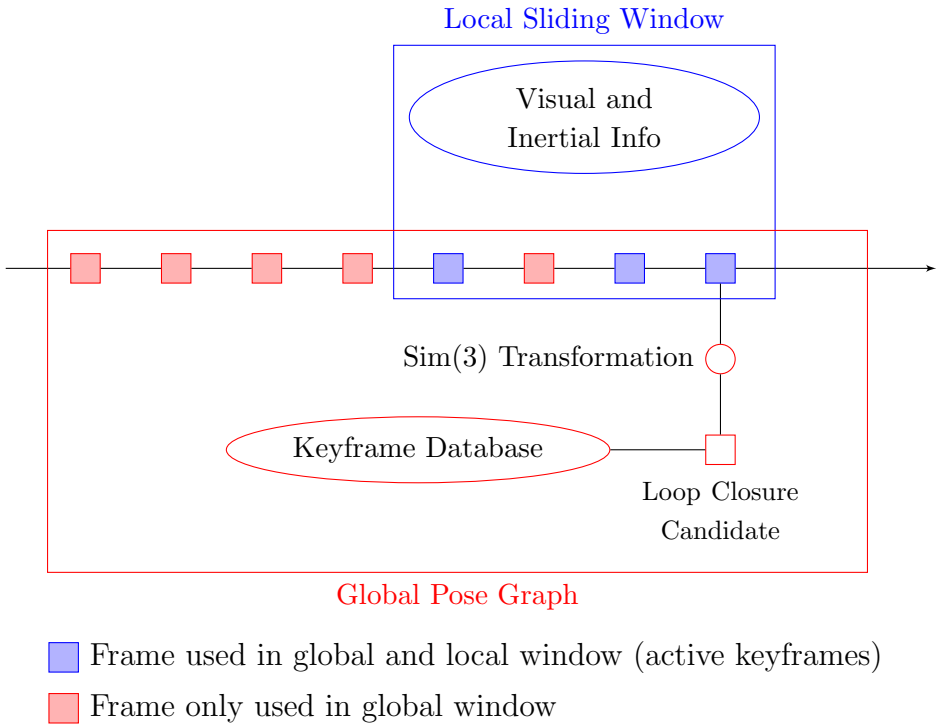


Figure 3.5: Local window versus global window modeled from LDSO [15].

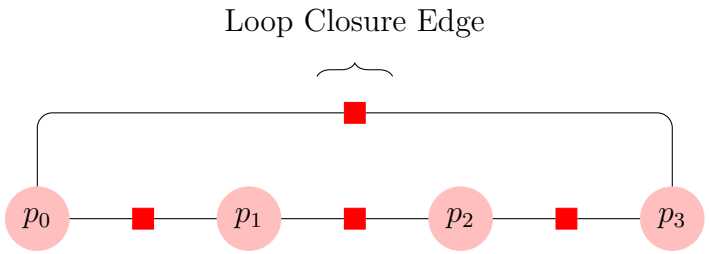


Figure 3.6: Loop closure represented by a pose graph.

is no longer directly observable. In order to close loops, the original pose estimates need to be stretched to fill the gaps, which can result in the scale drifting. Without using IMU information, it is impossible to recalculate a new scale estimate. Many algorithms are visual only and are not interested in the scale, or the scale drift, while other SLAM methods are able to fuse IMU and loop closure at a global scope. However, with VIL-DSO, fusing IMU at the global scope would counteract the advantages of using a local sliding window and a global loop closure separately.

In order to mitigate the scale drift without fusing IMU information at the global scope, the proposed method is to calculate a rotation and scale factor between the loop-closed trajectory and the original trajectory estimates using Kabsch-Umeyama alignment. The Kabsch algorithm [18] finds the optimal rotation matrix between the two paired sets of points by finding the minimal root-mean-square deviation, while Umeyama [34] extends this formulation to include a scale factor as well. This algorithm minimizes the squared error between the two trajectories by finding the optimal Sim(3) transformation between the two sets. This method is often used for aligning trajectories for error analysis.

The Kabsch-Umeyama algorithm is as follows: Given two sets of corresponding points $X = x1, x2, ..., x_n$ and $Y = x1, x2, ..., x_n$ The optimum transformation from X to Y can be decomposed into a rotation R, a translation t and a scale c. These parameters can be calculated as

$$R = USV^T \tag{3.7}$$

$$t = \mu_y - cR\mu_x \tag{3.8}$$

$$c = \frac{tr(DS)}{\sigma_x^2} \tag{3.9}$$

where,

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.10}$$

$$\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{3.11}$$

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mu_x\|^2 \tag{3.12}$$

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n} \|y_i - \mu_y\|^2 \tag{3.13}$$

$$\Sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \mu_y)(x_i - \mu_x)^T \tag{3.14}$$

and the singular value decomposition of Σ_{xy} is UDV^T. Subsequently,

$$S = \begin{cases} I & \text{if } det(\Sigma_{xy}) \geq 0 \\ diag(1, 1, ..., 1, -1) & \text{if } det(\Sigma_{xy}) < 0 \end{cases} \tag{3.15}$$

In the context of VIL-DSO, only the 3D positions of the points are used, such that S is a 3×3 matrix. The sets of points are the before and after loop closure position estimates. Only the scale factor is used from the Kabsch-Umeyama algorithm. This scale is then multiplied with the existing scale factor from the local optimization thread to form a new scale value. Using this method only mitigates the scale drift, as it shifts the scale value to be equivalent to the estimated scale from the local optimization thread. If scale drifts closer to its true value during loop closure, this method can actually makes results worse. This effect can be seen in some results in chapter 5. The algorithm is also only set up to compute the new scaling factor whenever there is a loop closure. If there is no loop closure near the current pose, scale error can accumulate through the remaining pose estimates.

Alternative Methods Explored for Loop Closure and Scaling Problems

Three other methods were proposed and explored to find a new scale factor after loop closure. The first method aligned the estimated trajectory with the pre-integrated IMU trajectory using Kabsch-Umeyama alignment to find a scale factor. The second method explored calculated relative distances from the IMU trajectory and plotted them against the relative distances from the estimated trajectory. The slope of this plot represents scale. Unfortunately, neither of these method were effective. The IMU information was too noisy to form a clear trajectory. The IMU information needs to be fused incrementally to prevent accumulation of too much noise.

The third method proposed was to calculate T_{b_i} as the relative pose for the body and IMU for frame i in the world reference frame as well as T_{c_i} as the relative pose for camera for frame i in the odometry reference frame. Next, convert T_{c_i} to the world frame by left multiplying it with T_{WD}, the transformation from odometry frame to world frame.

$$T_{WD}T_{c_i} = \left(\begin{array}{c|c} \mathbf{R}_{WD}\mathbf{R}_{c_i} & \mathbf{R}_{WD}\mathbf{t}_{c_i} + \mathbf{t}_{WD} \\ \hline \mathbf{0} & s_{WD}^{-1} \end{array}\right) \tag{3.16}$$

Since the scale factor is not equal to one in the bottom right, we need to right multiply by A, defined as

$$A = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & s_{WD} \end{array}\right). \tag{3.17}$$

This yields

$$s_{WD}T_{WD}A = \left(\begin{array}{c|c} \mathbf{R}_{WD}\mathbf{R}_{c_i} & s_{WD}\mathbf{R}_{WD}\mathbf{t}_{c_i} \\ \hline \mathbf{0} & 1 \end{array}\right) \tag{3.18}$$

Then, find T_{WD} such that

$$T_{WD} = \arg\min \sum_i \left(T_{b_i}^{-1}[s_{WD}T_{WD}T_{c_i}] - I\right), \tag{3.19}$$

where T_{WD} yields the rotation, translation and scale factor between the odometry and the world frame. This provides a scale factor that accounts for both orientation and translation. Unfortunately, it also requires a clear trajectory from pre-integrating the IMU, so it was not implemented.

Alternative methods for pre-integration could be explored to enhance trajectory generation from the IMU information. Then, these three methods could be more fully explored for generating a new scale estimate after loop closure.

3.5 Code Implementation

The implementation of the proposed VIL-DSO algorithm involved integrating the code sources of DSO [7], LDSO [36], and VI-DSO [36] [33], using LDSO as the code base, and applying some significant modifications as briefly summarized below:

- Functions and sections of code related to the IMU were transferred to LDSO from [33].

- Point and frame types were converted to match with LDSO, since LDSO uses different point and frame types than VI-DSO and DSO.
- Flags in the code for using the algorithm with and without IMU were fixed.
- Dynamic marginalization functions were adjusted to correctly follow formulations from [36] and allow for adjustments to scale.
- Functions were added to implement scale-drift mitigation.
- Functions were adjusted and added to output trajectory estimates for analysis.
- Topics in the visualizer for LDSO were separated and adjusted to correctly appear in the user interface.
- New topics were added to visualizer for the estimated trajectory with scale-drift mitigation.

3.6 Extensions and Limitations

A key drawback of VIL-DSO is that it does not have re-localization. It is unable to redetermine its location if tracking is lost. Many of the requirements to implement re-localization are present already in the algorithm. The IMU could be used for coarse tracking when tracking is lost visually, and then matches could be found using the existing BoW database, using a Sim(3) transformation between the match. However, points would need to be saved in the memory for each frame in the BoW.

Another drawback is that the software has not been optimized for computational speed. It runs at near real-time, at approximately 15 frames per second, but lags a bit behind the video source. Runtime information was not available for VI-DSO, but VIL-DSO is expected to be slower due to the addition of the loop closure thread. It is noted in [15] that LDSO's point selection method is slower than DSO, which would contribute to a slower runtime. A new version of DBoW3 has also been released, which is called Fast Bag of Words (FBoW). Implementation of this would likely improve computational speeds. Implementing GPU acceleration would also be ideal.

The pose estimates from the loop closure output only need to be re-scaled to be more accurate. Thus, loop closure pose estimates and the final scale estimates using the mitigation method were calculated in different steps in VIL-DSO. An alternative solution would

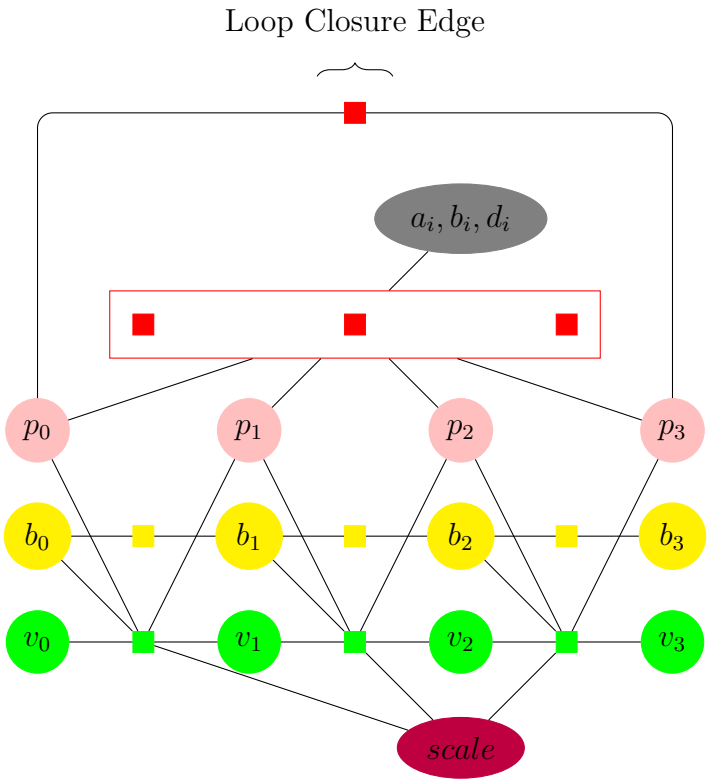


Figure 3.7: Loop closure with IMU data.

be to fuse loop closure and IMU data simultaneously, as shown in the factor graph in Figure 3.7. Unfortunately, this method requires a bundle adjustment in the global scope, which defeats the computational advantages of using a sliding window and omitting IMU information from the loop closure. The entire factor graph would be recalculated every loop closure. This solution would be more in the scope of full SLAM solutions. To further pursue this as a SLAM solution, inertial data could be tied to each of the points and the map itself could be optimized, rather than just the poses.

Chapter 4

Scene Reconstruction

After pose estimates for the UAV are generated, the next goal would be to generate more information about the scene. This can be accomplished by reconstructing the environment using a dense mapping algorithm. The dense map can be used to more easily evaluate the scenario and decide the course of action. The goal of this section is to produce a map with a visible surface for interpretation by human user. The map can also be used as a navigation tool for ground-based vehicles.

In particular, this chapter discusses hardware setup and considerations for future work with outdoor mapping. It explores using REMODE [28] as a dense mapping algorithm using the feeds of VIL-DSO

4.1 Dense Mapping using REMODE

One of the real-time mapping algorithms is REgularized MOnocular Depth Estimation, which is discussed throughout in this chapter. The goal of REMODE is to generate dense depth maps from a monocular camera in motion. To accomplish this, it utilizes Bayesian depth estimation to generate depth estimates for each pixel. As part of the algorithm, camera pose must be known. This can be provided by various means, such as with SVO as proposed in [9], or with another odometry program such as VIL-DSO. The odometry passes images, camera poses and coarse depth estimates to REMODE. REMODE also uses a smoothing method to help reduce the effects of noise from the camera pose estimates by providing spatial regularity.

4.1.1 Convergence Criteria

REMODE selects the current frame as the active keyframe. It then uses subsequent frames to converge the depth information on the pixels of the keyframe. When a sufficient number of points have converged depth estimates within a threshold, the keyframe is released and the converged depths are published into the visualizer. The algorithm then selects the next frame as a new keyframe.

4.2 Hardware and System Setup through ROS

The scene-reconstruction system is proposed to be implemented through the Robot Operating System (ROS) framework. A Matrice 210 RTK quadcopter is to be used as the UAV for gathering visual and inertial information. To record the data, a small computer was attached to the Matrice, which ran the DJI SDK ROS node to convert the information into ROS topics. Next, these ROS topics for the camera and the IMU are passed to VIL-DSO. A ROS wrapper is used to subscribe to the ROS messages, and to publish the odometry outputs as ROS topics. The RTK pose from the drone is also recorded to be used as a ground truth comparison.

In European Robotics Challenges (EuRoC) datasets, the DJI SDK node is replaced with ROS bags, which allow for playback of data. A diagram of the proposed setup is illustrated in Figure 4.1.

4.3 Outdoor Mapping Considerations

This section discusses considerations for outdoor mapping, including coverage, camera angles and flight times. While these generally apply to all applications for mapping, these concerns are typically exacerbated at larger scales.

4.3.1 Stereo and Monocular Comparisons

One of the key parameters of the scene that needs to be estimated is depth. Stereo cameras are able to generate depth estimates by utilizing geometry and a fixed baseline, which results in fusion of information spatially, as illustrated in Figure 4.2a. In contrast, a monocular camera is unable to generate depth information using space alone. Instead,

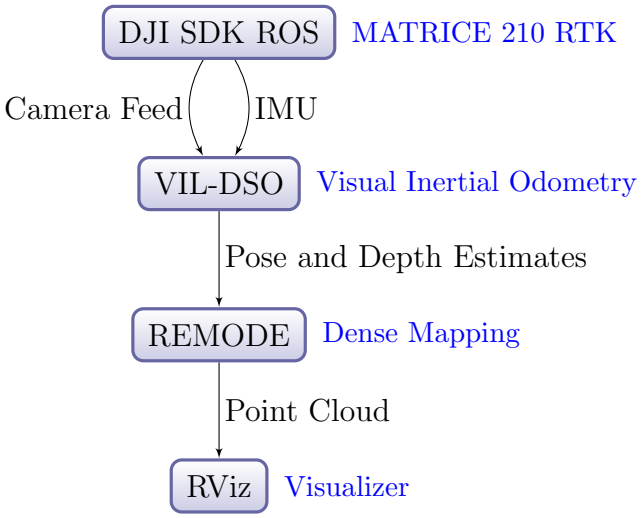


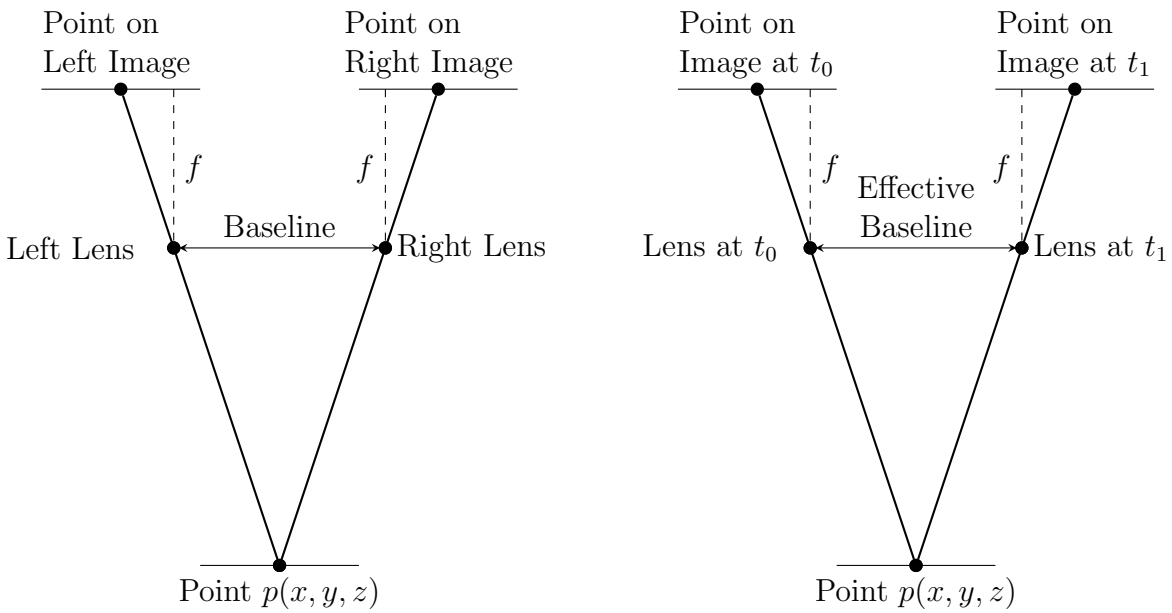
Figure 4.1: Proposed scene reconstruction system in ROS.

using monocular video requires fusion of the images both temporally and spatially, as illustrated in Figure 4.2b.

Thus for monocular camera, keyframes need to be sufficiently far apart to generate depth information, but must also overlap sufficiently so that enough frames are shared between keyframes. If a point does not exist in both keyframes, no depth information can be determined. Taking keyframes too closely together does not generate a wide enough effective-baseline. In fact, feeding too many of these nearby frames as keyframes results in noise and does not help converge the estimate. This is echoed in the literature, such as in [7]. With wide FOVs, it can take a long time to travel sufficient distance for the baseline, which can result in a delay in map generation.

4.3.2 GSD and Lens Selection

Another concern is the quality of the map itself. Often, higher resolution is required to resolve small details in map. Ground Sample Distance (GSD) is measure that can assist in understanding how certain parameters affects map quality. GSD is sampling metric that measures the distance between the centers of pixels on the ground. For example, a GSD of 2.4 $cm/pixel$ means that for every pixel in the image, 2.4 cm is covered. Figure 4.3 illustrates the parameters for calculating GSD. The equation for GSD [22] can be calculated



(a) Stereo cameras.

(b) Monocular camera.

Figure 4.2: Stereo cameras versus monocular camera for depth estimation.

as follows

$$GSD = \frac{hw_s}{fw_i \cos\theta}, \tag{4.1}$$

where h is the height from the ground, or the distance from the surface being mapped, w_s is the width of the sensor, f is the focal length, w_i is the width of the image and θ is the angle between the line of sight and the ground/surface. Note that the widths can be swapped with the heights of the image/sensor to get another GSD value. Usually, the larger (and worse) of the two values is taken.

Therefore, the goal is to maximize GSD while still maintaining enough overlap for depth estimation and tracking. Using a wide-angle lenses creates the most overlap, and was used in the experiments as it was the most robust for tracking, but at the cost of decreased resolution.

As altitude increases, there is more overlap in the images, but also greater GSD. This means that in order to maintain the same resolution at higher altitudes, a more narrow FOV (or longer focal distance) is required. Unfortunately, disturbances at higher altitudes are amplified to the image on the ground, which can make tracking more difficult. The effects of disturbances can be mitigated by adding camera stabilization, such as a gimbal.

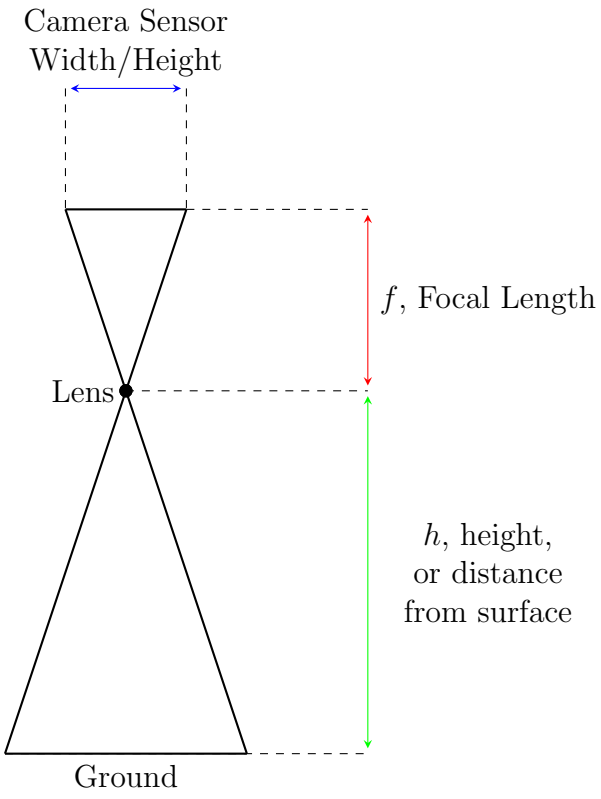
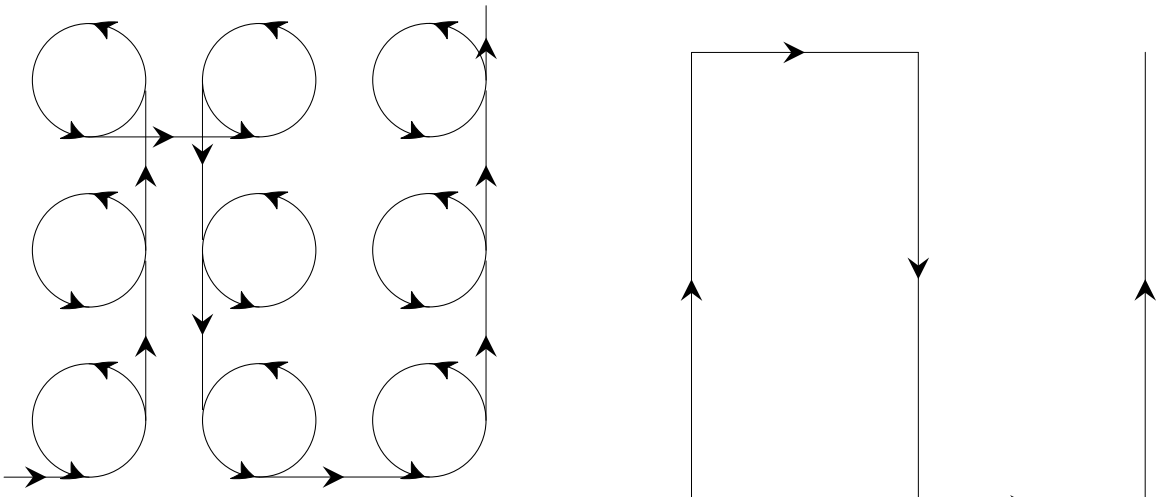


Figure 4.3: Diagram of GSD calculation.

4.3.3 Battery Life and Coverage planning

One of the major drawbacks with modern UAVs is the duration of the battery life. Many quadcopter UAVs only maintain flight for 15 to 30 minutes. The Matrice 210 RTK used in testing is only rated to fly a maximum 24 to 38 minutes depending on the payload, but this is conservative estimate that does not consider additional equipment and aggressive maneuvers. In reality, the available flight time is much less. In order to maximize the usage of the battery, the UAV needs to quickly and efficiently cover the areas that need mapping. One possible path is proposed in [9], where the drone flies in small circles over the area of interest, which is illustrated in Figure 4.4a. This circling yields more angles of the same subjects, and makes it easier to converge depth in scene reconstruction. However, it provides a lot of redundant information as well. It is more effective for the UAV to fly in straight lines to preserve momentum. To accommodate this, a simple pattern was used, as illustrated in Figure 4.4b.



(a) Circular path from [9]

(b) Straight path.

Figure 4.4: Some potential flight paths.

4.3.4 Camera Angles and Horizon Issues

When the camera is able to view above the horizon, pixels associated with the sky have infinite depth and cannot converge. If one of these frames is selected as a keyframe, they may not be published into the point cloud due to being unable to meet REMODE's convergence criteria. For example, if the keyframe image is 50% sky, then there is only 50% of the image that can converge, making it difficult to reach the threshold required for publishing the point cloud. An image with the sky visible is illustrated in Figure 4.5. Also, as can been seen in Equation 4.1, the line of sight is ideally perpendicular to the surface being mapped. Using a camera pointed straight at the ground most directly fulfills this, but then vertical surfaces are not mapped. Decreasing θ improves the mapping of vertical surfaces, but sacrifices resolution on the ground. If θ decreases too much, there is a risk of viewing the horizon, which will affect depth estimation.

4.4 Limitations and Extensions

Using the proposed mapping setup with REMODE will not allow the adjustment of poses after they are generated. When loop closures are detected in VIL-DSO, the pose adjustments are not passed onto the scene reconstruction algorithm. A potential way to address

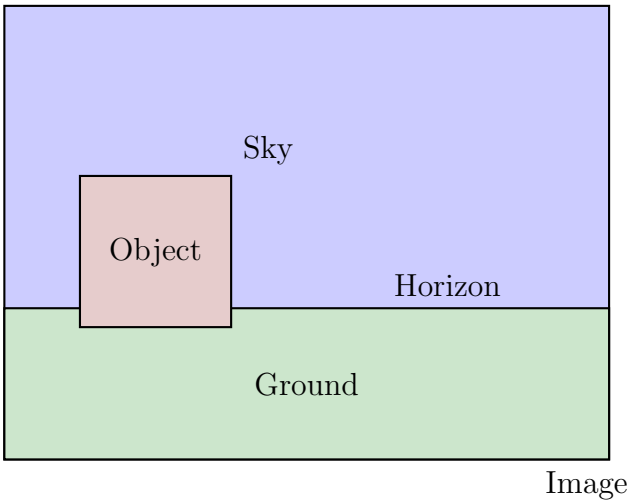


Figure 4.5: Image with the horizon visible.
Depth cannot being determined above the horizon and the object

this is to associate points in the point cloud with the keyframes that they are generated with. When the loop closure adjust the poses associated with the keyframe, the associated points should also have their positions adjusted accordingly. This would require significant modification of REMODE.

There are other mapping methods available in the literature that could be explored as a more in-depth survey of scene reconstruction. REMODE was used due to the availability of open-source code and its relevance in the context of scene reconstruction, as presented in [9].

Work could also be done on testing and quantifying parameters for ideal mapping. The parameters used in testing were primarily a combination of trial-and-error, and the equipment that was available.

Chapter 5

Experimental Results

VIL-DSO was tested on various sequences, and then compared with the available ground-truths for evaluation of error. The methods used for data alignment and error evaluation are discussed in this chapter. The results were generated using the using the EuRoC dataset.

To implement the trajectory evaluation, a tutorial and a program written by Zhang [40] were utilized. The program is available on GitHub [39].

5.1 Alignment and Error Types

Before calculating estimation error, the estimated trajectory and the ground truth trajectory need to be aligned. This alignment was completed using all the available frame position information, as opposed to only using initial frames. Sim(3) transformations are used to generate the ground truth-scaled results. The error comparisons align the datasets while also modifying the scale of the estimate in order to see how the estimates compare when independent of scale. The Sim(3) transformation is used to compare estimation methods that do not have observable scale, such as with monocular camera alone. Results using the Sim(3) transformation are referred to as "gt-scaled", as the estimates are scaled to match the ground truth. Alternatively, rigid body transformations, SE(3), are used to generate comparisons without manipulating the scale. The transformations are used to find error in translation and scale. As part of Zhang's implementation [40], the Kabsch-Umeyama algorithm is used for alignment of datasets.

Absolute Trajectory Error (ATE) refers to a direct comparison between the states of the estimated trajectory and the ground truth. To quantify ATE as a single term, the Root-Mean-Square Error (RMSE) is often used. RMSE is easy to use for comparison, but can be sensitive to when the error occurs. As an alternative to ATE, Relative Error (RE) is also used for comparison. RE looks at the changes in trajectory at different times, which multiple instances of error. However, RE is more difficult to compare as it does not produce a single number for evaluation. RE also does not show the accumulated drift error in the pose estimates, which can be improved with loop closure. For these reasons, only ATE is used for comparisons with the state-of-the-art methods. .

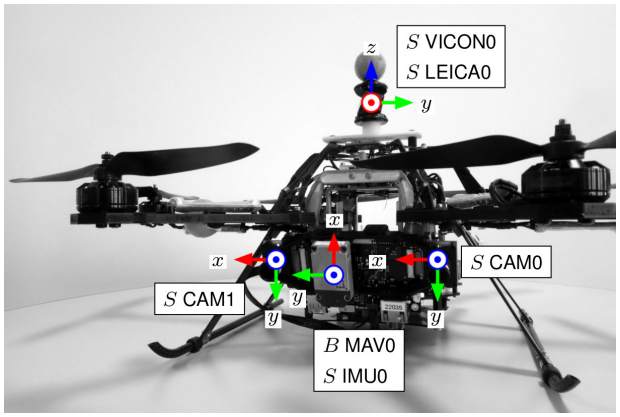
5.2 IMU Weights and Active Point Parameters

IMU weights were kept at their default values for the local optimization thread and for the coarse tracker. It was noted that increasing these weights introduces more noise into the system, since the IMU data is relatively noisy. This noise was visible in the estimate when the IMU weights were set too large. The number of point candidates was also increased to 3000 from 1500 per keyframes, and active points retained across the local sliding window was increased to 4000 from 2000. Typically, with only 1500 point candidates, approximately 800 were rejected with either VI-DSO or LDSO, but combined in VIL-DSO, nearly 1200 are rejected. Increasing the points improved tracking, but optimization of these values was not fully explored.

5.3 EuRoC Dataset

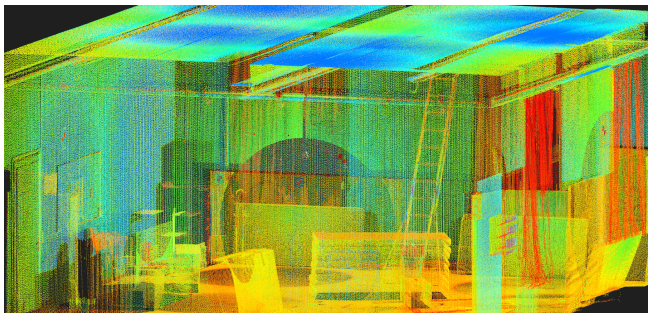
Many VO algorithms are tested on the publicly-available EuRoC dataset [1]. The ground-truth and results for other methods are readily available, which allows for straightforward comparisons. This dataset was recorded using an Asctec Firefly hex-rotor helicopter, as illustrated in Figure 5.1a. It carried visual-inertial sensor unit, with a stereo camera (Aptina MT9V034 global shutter, WVGA monochrome, 220 FPS) and a MEMS IMU (ADIS16448, angular rate and acceleration, 200 Hz).

To record poses for ground truth, a Vicon motion capture system (6D pose) was used, as well as a Leica MS50 laser tracker. The UAV was flown through two different environments, the Eidgenssische Technische Hochschule (ETH) Machine Hall (MH) and the Vicon Room (V1 and V2), which are illustrated in Figure 5.1



(a)

(b)



(c)

Figure 5.1: EuRoC Dataset, from [1]: (a) Asctec Firefly used in EuRoC data collection; (b) ETH Machine Hall; (c) 3D scan of the Vicon Room.

VIL-DSO was run on these sequences for trajectory evaluation. There were issues with initialization on some of the Vicon hall sequences, which are omitted. The ground truth and estimated trajectories for the first Machine Hall (MH_01) sequence from EuRoC are illustrated in Figure 5.2. The trajectories are colour-coded as follows:

- Green: ground truth trajectory
- Red: Estimated trajectory before loop closure
- Cyan: Estimated trajectory after loop closure
- Magenta: Estimated trajectory after loop closure and using scale drift mitigation

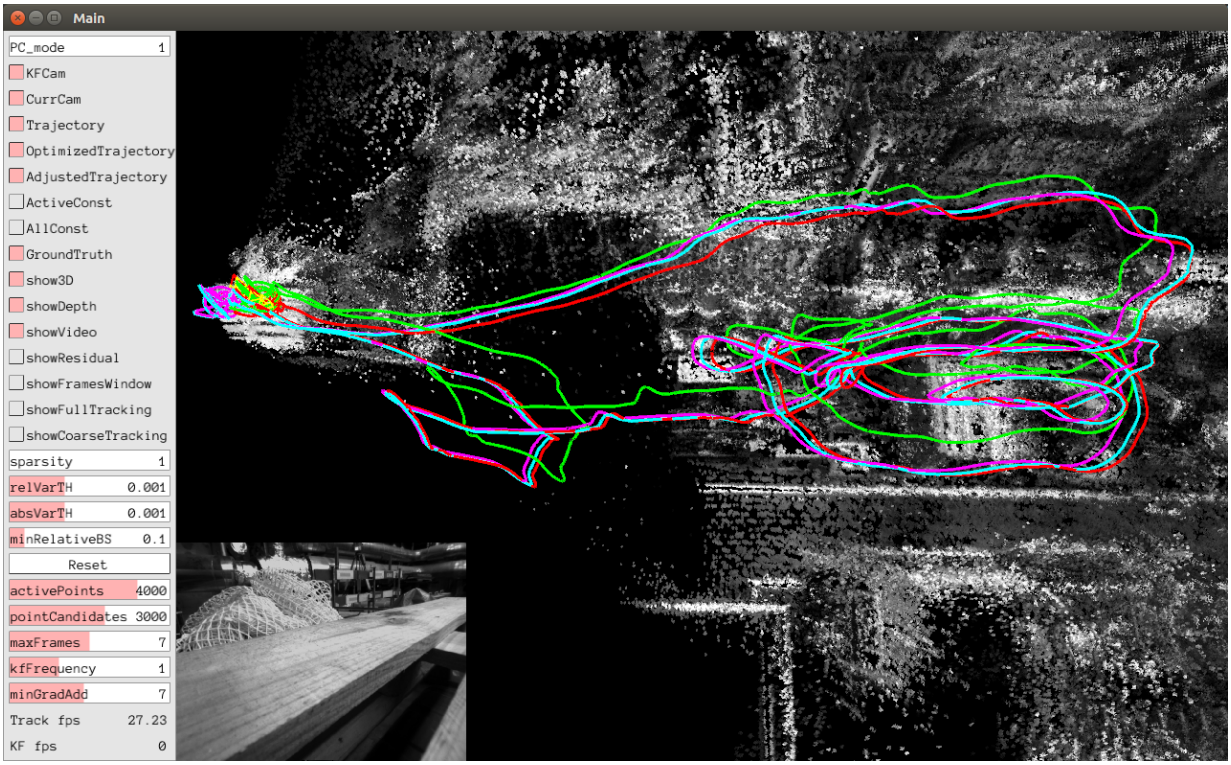


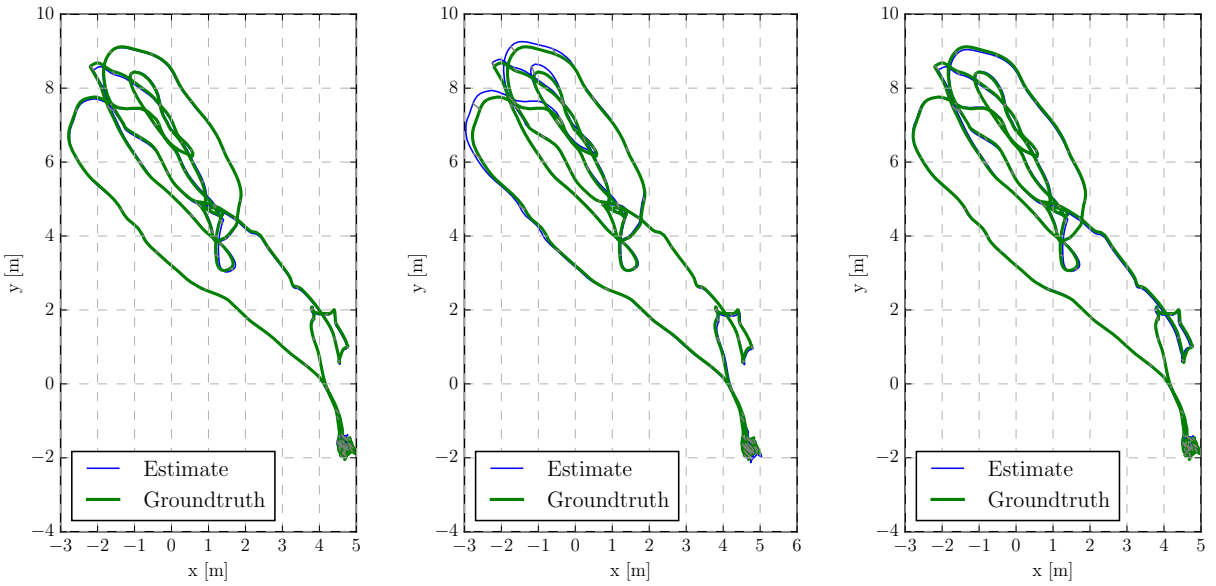
Figure 5.2: VIL-DSO test results: All estimated trajectories and ground truth for the MH_01 sequence.

5.3.1 EuRoC Trajectory Analysis

The trajectories from the visualizer shown in Figure 5.2 are not aligned, which makes it difficult to see the discrepancies. To illustrate the error, plots showing the aligned trajectories, relative and absolute errors were generated as part of the trajectory analysis.

Figure 5.3 illustrates the top view of the estimated trajectories compared with the aligned ground truth using SE(3) transformations. In Figure 5.3b, there is a larger gap between the estimated trajectory and the ground truth, which is caused by the scale drifting during loop closure. In Figure 5.3c, this error is fixed by the scale adjustment applied to mitigate the drift.

Figure 5.4 illustrates the same trajectories from a side view. Figure 5.5 shows the relative error at different stages in the trajectory. Because it is difficult to quantify differences from relative error, ATE is also included. Figure 5.6 shows the ATE for the translation in



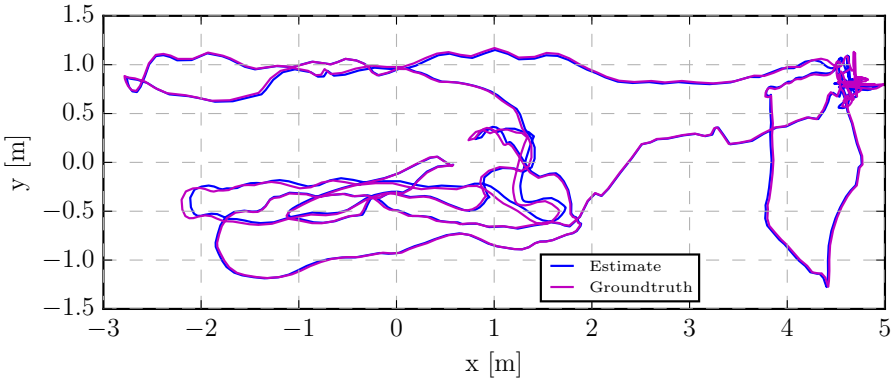
(a) (b) (c)

Figure 5.3: Top view of estimated trajectories aligned with ground truth for MH_01: (a) Without loop closure; (b) With loop closure; (c) With loop closure and scale-drift mitigation.

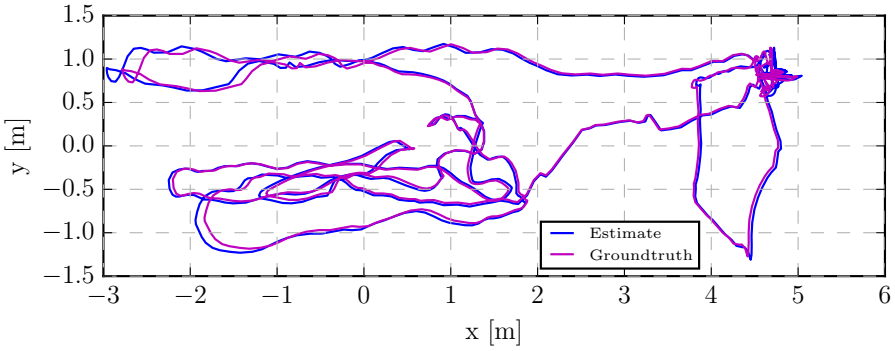
the x-,y- and z-directions. The peaks of the error increase after loop closure, approaching 200mm, and are decreased by the scale mitigation method to around 100mm.

5.3.2 EuRoC RMSE

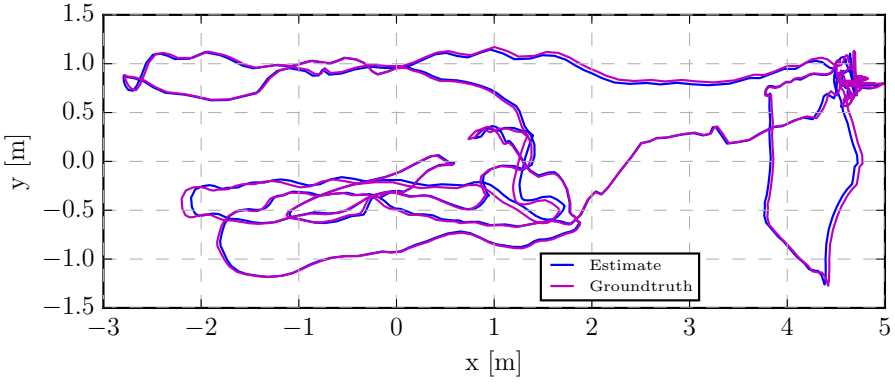
The plots in the Section 5.3.1 only show results for a single run of the MH_01 sequence. To show the effectiveness of VIL-DSO, RMSE was calculated for ten runs on each sequence. The medians of these values are shown in Table 5.1. For comparison to state-of-the-art methods, the resulting RMSE values for the estimated trajectories from VIL-DSO are compared with existing Visual-Inertial (VI) pose estimation techniques. These methods include VI-DSO, and VI-ORB SLAM. Table 5.1 shows the RMSE for translation and scale from the pose estimates using SE(3) transformations, as well as translation RMSE from ground truth-scaled trajectories using Sim(3) alignment. The results for VI-DSO are taken from [36] and are the medians of ten runs for each sequence. The VI-ORB SLAM results are taken from [36] as well, and are assumed to be representative of the algorithm. The latter



(a)

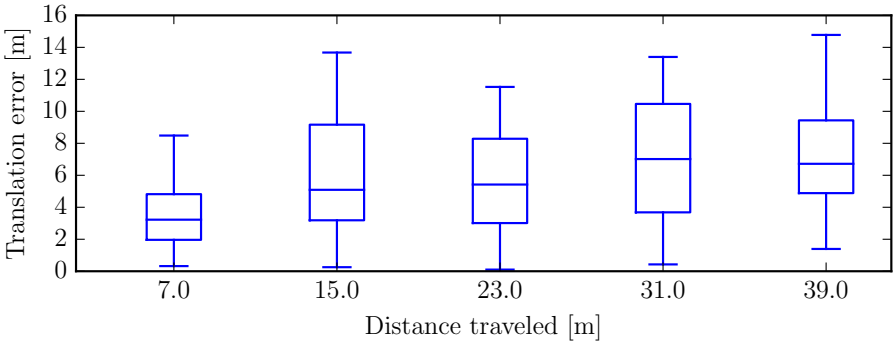


(b)

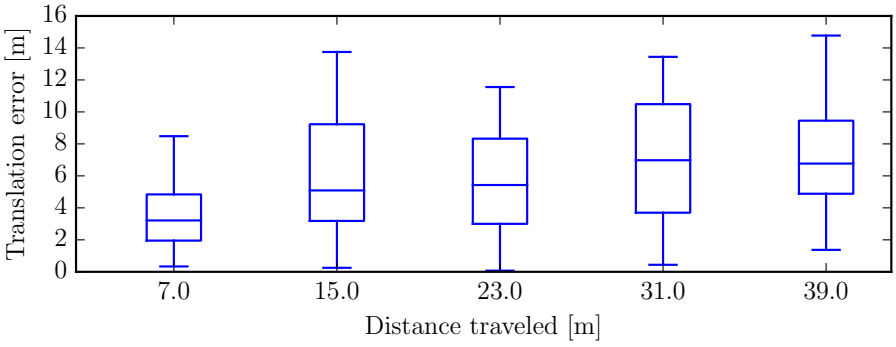


(c)

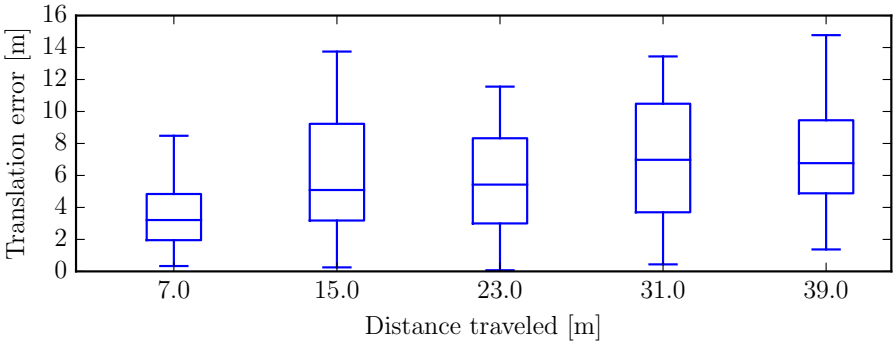
Figure 5.4: Side view of estimated trajectories with ground truth for MH_01: (a) Without loop closure; (b) With loop closure; (c) With loop closure and scale-drift mitigation.



(a)

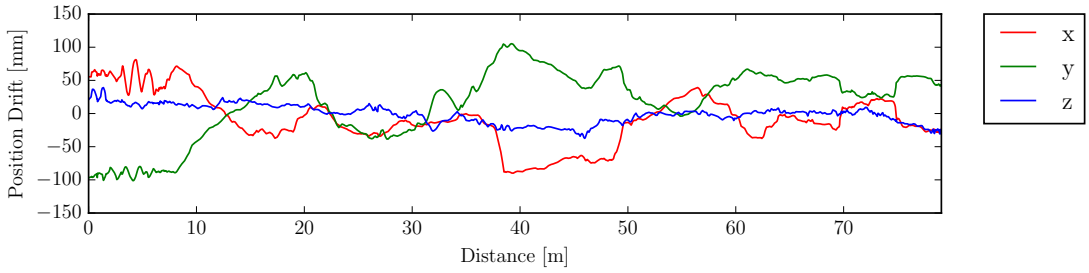


(b)

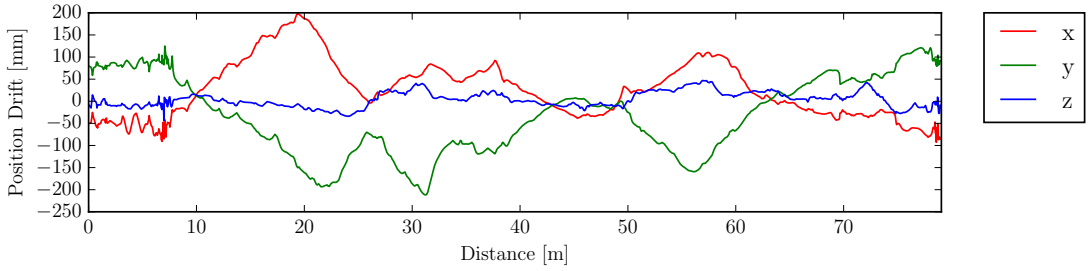


(c)

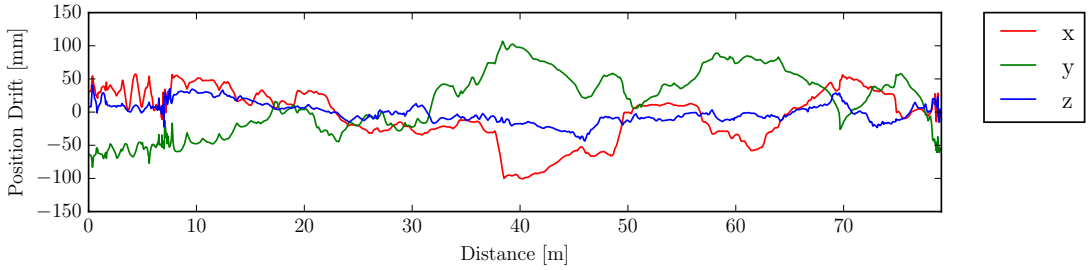
Figure 5.5: Relative error for MH_01: (a) Without loop closure; (b) With loop closure; (c) With loop closure and scale-drift mitigation.



(a)



(b)



(c)

Figure 5.6: Translation error for x-,y- and z-directions for MH_01: (a) Without loop closure; (b) With loop closure; (c) With loop closure and scale-drift mitigation.

sequences from the Vicon rooms (V1_02, V1_03, V2_02, V2_03) were difficult to initialize algorithms on, and were not included.

In Table 5.1, it can be seen that loop closure improves the ground truth-scaled translation RMSE in most of the sequences. In the first Vicon Room (V1_01) sequence, the errors are very similar, but loop closure did not improve the error, which may indicate either that the loop closure are unable improve this sequence significantly, or incorrect matches occurred during loop closure. However, the SE(3) translation and scale errors are some-

Method	Sequence	MH_01	MH_02	MH_03	MH_04	MH_05	V1_01	V2_01
VIL-DSO No Loop Closure	Translation RMSE (m)	0.075	0.145	0.517	0.933	0.267	0.117	0.091
	Translation gt-scaled (m)	0.066	0.057	0.448	0.751	0.252	0.092	0.085
	Scale RMSE (%)	1.1	2.0	5.4	8.2	2.7	3.2	2.2
VIL-DSO With Loop Closure	Translation RMSE (m)	0.089	0.137	0.411	0.596	0.167	0.118	0.068
	Translation gt-scaled (m)	0.047	0.042	0.083	0.160	0.102	0.093	0.062
	Scale RMSE (%)	1.2	1.8	3.8	4.4	1.4	3.2	1.6
VIL-DSO With Loop Closure and Scale Drift Mitigation	Translation RMSE (m)	0.065	0.064	0.268	0.425	0.138	0.112	0.066
	Translation gt-scaled (m)	0.047	0.042	0.083	0.160	0.102	0.093	0.062
	Scale RMSE (%)	0.8	0.9	2.9	3.7	1.1	2.9	1.5
VI-DSO Median of 10 runs	Translation RMSE (m)	**0.062**	**0.044**	0.117	**0.132**	0.121	0.059	0.040
	Translation gt-scaled (m)	0.041	0.041	0.116	0.129	0.106	0.057	0.031
	Scale RMSE (%)	1.1	0.5	0.4	0.2	0.8	1.1	1.2
VI-ORB-SLAM	Translation RMSE (m)	0.075	0.084	**0.087**	0.217	**0.082**	**0.027**	**0.028**
	Translation gt-scaled (m)	0.072	0.078	0.067	0.081	0.077	0.019	0.031
	Scale RMSE (%)	0.5	0.8	1.5	3.4	0.5	0.9	0.2

Table 5.1: VIL-DSO RMSE compared to other visual-inertial methods: Median of ten runs. Smallest translation errors are in bold font.

times better or worse for each sequence. This is due to the scale drift error during loop closure, as described in Chapter 3, which may drift either away or towards the true scale with respect to the ground truth trajectory. This indicates that the new pose estimates are more accurate in terms of direction and orientation, but not always in terms of scale.

The scale drift mitigation method decreased both scale and translation errors on all of the sequences. The resulting error is less than with and without loop closure. In reality, the mitigation method is not effective on all runs, but appears to be effective in majority of the cases. In 100 runs of the MH_01 sequence, the mitigation method reduced the translation and scale error on 92 runs. There was no discernible trend as to why it did not reduce the errors in the remaining 8 runs. A potential source of errors is that there may not have been significantly scale drift in these runs. In this case, scale drift would not be easily differentiated from the changes in pose estimates needed for loop closure. Alternatively, there may have not been a loop closure match near the end of the sequence and since scale drift mitigation is only called during loop closure, there may be accumulated error towards the end of the sequence. It is also important to note that the scale drift mitigation method does not calculate a new scale, but attempts to reduce the scale drift generated during loop closure. Calculating scale during loop closure would be a better approach.

The results shown in Table 5.1 show that VIL-DSO is comparable to existing state-of-the-art methods. Theoretically, the VIL-DSO results without loop closure are expected to be the same as VI-DSO, but there is a discrepancy, which is likely due to variations in the implementation, as source code was not available. In particular,a major difference is the

point selection method implemented in VIL-DSO, which generates more point candidates that favor corners, but are more likely to be rejected when combined with IMU information.

The best results for each sequence were also tabulated and are shown in Table 5.2. From these results, it can be seen that the estimates from VIL-DSO have the potential to be better than VI-DSO, but more testing and modifications would be required to ensure that this is consistent. Note that for MH_01, the scale drift mitigation did not improve the result. This is one of the few cases discussed where the scale drift was likely not significant and not observable relative to the pose estimate changes due to loop closure. Worst case trajectory errors are not included, as these are when the algorithm fails entirely and the error cannot be calculated.

Method	Sequence	MH_01	MH_02	MH_03	MH_04	MH_05	V1_01	V2_01
VIL-DSO No Loop Closure	Translation RMSE (m)	0.057	0.104	0.300	0.367	0.187	0.088	0.080
	Translation gt-scaled (m)	0.045	0.042	0.242	0.262	0.173	0.085	0.075
	Scale RMSE (%)	0.7	1.4	2.5	3.0	2.4	2.2	1.9
VIL-DSO With Loop Closure	Translation RMSE (m)	**0.039**	0.063	0.092	0.260	0.099	0.095	0.061
	Translation gt-scaled (m)	0.038	0.036	0.066	0.107	0.071	0.088	0.057
	Scale RMSE (%)	0.4	0.8	0.9	1.9	0.9	2.3	1.4
VIL-DSO With Loop Closure and Scale Drift Mitigation	Translation RMSE (m)	0.043	**0.043**	0.097	**0.115**	**0.072**	0.094	0.063
	Translation gt-scaled (m)	0.038	0.036	0.066	0.107	0.071	0.088	0.057
	Scale RMSE (%)	0.5	0.5	0.9	1.0	0.7	2.3	1.4
VI-DSO Median of 10 runs	Translation RMSE (m)	0.062	0.044	0.117	0.132	0.121	0.059	0.040
	Translation gt-scaled (m)	0.041	0.041	0.116	0.129	0.106	0.057	0.031
	Scale RMSE (%)	1.1	0.5	0.4	0.2	0.8	1.1	1.2
VI-ORB-SLAM	Translation RMSE (m)	0.075	0.084	**0.087**	0.217	0.082	**0.027**	**0.028**
	Translation gt-scaled (m)	0.072	0.078	0.067	0.081	0.077	0.019	0.031
	Scale RMSE (%)	0.5	0.8	1.5	3.4	0.5	0.9	0.2

Table 5.2: VIL-DSO RMSE compared to other visual-inertial methods: Best of ten runs. Smallest translation errors are in bold font.

5.4 Summary of Experimental Results

Using loop closure in VIL-DSO reduced the translation error when scaled to the ground truth VIL-DSO, as seen in Table 5.1. However, it sometimes decreases or increases the unscaled translation error due to the scale drift. The scale mitigation method that was proposed in Chapter 3 worked on the majority of runs on the EuRoC sequences, and is shown to improve consistency for maintaining scale through loop closure. When compared to the existing methods in the state-of-the-art, the resulting estimated trajectories from VIL-DSO are found to carry error levels similar to VI-DSO and VI-ORB SLAM. VIL-DSO

without loop closure was expected to perform similar to VI-DSO, but performed worse. This is hypothesized to be due to implementation discrepancies and the implementation of a different point selection method. The best cases of VIL-DSO did not consistently perform better than VI-ORB SLAM. However, VI-ORB SLAM is a full-SLAM solution and optimizes the map simultaneously with the pose estimate.

Chapter 6

Conclusion and Future Work

With the aim of generating accurate pose estimates, this thesis has studied the design of a Direct Sparse Visual-Inertial Odometry with Loop Closure (VIL-DSO) scheme, which combines existing algorithms for inertial and visual information from VI-DSO, as well as loop closure to reduce drift error from LDSO. The experimental test results demonstrate that the proposed scheme provides scaled readings in the world reference frame. The loop closure is shown to reduce drift errors and improve pose estimates, but also cause the scale estimate to drift. VIL-DSO introduces a scale-drift mitigation method, which improves scale estimates after loop closure, without requiring the fusion of IMU information again. As a VIO algorithm, VIL-DSO is effective in GPS-denied environments, and can be used for both outdoor search-and-rescue scenarios and indoor navigation.

For the short sequences in the EuRoC dataset, the results are close to results from existing methods in the state-of-the-art, but generally has more error. VIL-DSO is expected to perform better on longer sequences, where there is more time for drift error to accumulate when loop closure is not used. More testing on longer sequences is recommended to confirm this. Many parameters in the code were set at arbitrary values. Exploration into tuning the parameters could yield better results, especially in terms of the IMU weights and the number of point candidates proposed. Future work would be required to calculate scale during loop closure, instead of only mitigating the scale drift. This would most likely be best achieved by fusing IMU information at the global scope, instead of only in the local sliding window.

In the context of mapping, a proposed mapping system using VIL-DSO and REMODE was discussed. Key implementation aspects for outdoor mapping have been discussed, including the trade-offs between selecting focal length, the flight altitudes and the GSD.

Using VIL-DSO as the basis for a more complete SLAM method is expected to provide a more effective mapping solution without requiring a back-end dense mapping algorithm. Since the point cloud outputs of VIL-DSO scheme form a visible, but sparse map, this is the next step to developing it as a stand-alone system. Improved depth estimation would also help with reducing the noise of the map. These improvements to the depth estimation could be achieved using additional sensors, such as a radar.

Because the loop closure proposed in LDSO [15] does not take scaling into account, a method use Kabsch-Umeyama was proposed and implemented. A more complex method, as detailed in Chapter 3, accounts for aligning the camera and IMU sensors, but requiring generating a trajectory from IMU measurements. Alternatively, the local window could be eliminated and the algorithm could use a global scope for optimization as part of a SLAM solution, while also including loop closure within such optimization.

Implementing the complete proposed scene construction process remains in the realm of future work. However, it has also been highlighted that more in-depth exploration on scene reconstruction should be done. There are many other alternatives for mapping, although few operate in real-time. Finally, applying a surface to the mapped point cloud allows for straightforward transition to using the map for navigation.

References

[1] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[2] Alejo Concha and Javier Civera. DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5686–5693, 2015.

[3] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[4] Frank Dellaert. Factor Graphs and GTSAM: A Hands-on Introduction. Technical report, Georgia Tech College of Computing, 2012.

[5] Frank Dellaert and Michael Kaess. Factor Graphs for Robot Perception. *Foundations and Trends ® in Robotics*, 6(2):1–139, 2017.

[6] Ethan Eade. Lie Groups for 2D and 3D Transformations. http://ethaneade.com/lie.pdf, 2013.

[7] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018.

[8] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *European Conference on Computer Vision*, pages 834–849, 2014.

[9] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. Autonomous, Vision-based Flight and Live Dense 3D Map-

ping with a Quadrotor Micro Aerial Vehicle. *Journal of Field Robotics*, 33(4):431–450, 2016.

[10] Markus Falk, Hermann Brugger, and Liselotte Adler-Kastner. Avalanche survival chances. *Nature*, 368(6466):21–21, 1994.

[11] Paolo Favaro, Hailin Jin, and Stefano Soatto. A semi-direct approach to structure from motion. In *Proceedings - 11th International Conference on Image Analysis and Processing, ICIAP 2001*, pages 250–255, 2001.

[12] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 15–22. Institute of Electrical and Electronics Engineers Inc., 2014.

[13] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2017.

[14] Dorian Gálvez-López and Juan D Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[15] Xiang Gao, Rui Wang, Nikolaus Demmel, and Daniel Cremers. LDSO: Direct Sparse Odometry with Loop Closure. *IEEE International Conference on Intelligent Robots and Systems*, pages 2198–2204, 2018.

[16] Hailin Jin, Paolo Favaro, and Stefano Soatto. Real-time 3D motion and structure of point features: a front-end system for vision-based control and interaction. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, pages 778–779, 2000.

[17] Wang Jing. LearnVIORB. https://github.com/YoujieXia/VI_ORB_SLAM2, 2017.

[18] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923, 1976.

[19] Jonathan Kelly, Srikanth Saripalli, and Gaurav S Sukhatme. Combined visual and inertial navigation for an unmanned aerial vehicle. In *Springer Tracts in Advanced Robotics*, volume 42, pages 255–264, 2008.

[20] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007.

[21] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.

[22] Jon C. Leachtenauer and Ronald G. Driggers. *Surveillance and reconnaissance imaging systems : modeling and performance prediction*. Artech House, 2001.

[23] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[24] Raul Mur-Artal, J. M.M. Montiel, and Juan D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[25] Raul Mur-Artal and Juan D. Tardos. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[26] Raul Mur-Artal and Juan D. Tardos. Visual-Inertial Monocular SLAM with Map Reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.

[27] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. DTAM: Dense tracking and mapping in real-time. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2320–2327. Imperial College London, UK, 2011.

[28] Matia Pizzoli, Christian Forster, and Davide Scaramuzza. REMODE: Probabilistic, monocular dense reconstruction in real time. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2609–2616, 2014.

[29] Rene Ranftl, Vibhav Vineet, Qifeng Chen, and Vladlen Koltun. Dense Monocular Depth Estimation in Complex Dynamic Scenes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4058–4066, 2016.

[30] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.

[31] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[32] Jan Stühmer, Stefan Gumhold, and Daniel Cremers. Real-time dense geometry from a handheld camera. In *32nd DAGM Conference for Pattern Recognition*, pages 11–20, 2010.

[33] Jiaming Sun. VI-Stereo-DSO: Visual Inertial - Stereo - Direct Sparse Odometry. https://github.com/RonaldSun/VI-Stereo-DSO, 2019.

[34] Shinji Umeyama. Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[35] Levi Valgaerts, Andrs Bruhn, Markus Mainberger, Joachim Weickert, A Bruhn, M Mainberger, J Weickert, and J Weickert. Dense versus Sparse Approaches for Estimating the Fundamental Matrix. *Int J Comput Vis*, 96:212–234, 2012.

[36] Lukas Von Stumberg, Vladyslav Usenko, and Daniel Cremers. Direct Sparse Visual-Inertial Odometry Using Dynamic Marginalization. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2510–2517, 2018.

[37] Nan Yang, Rui Wang, Jrg Stückler, and Daniel Cremers. Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry. *European Conference on Computer Vision*, 2018.

[38] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.

[39] Zichao Zhang. rpg_trajectory_evaluation: Toolbox for quantitative trajectory evaluation of VO/VIO. https://github.com/uzh-rpg/rpg_trajectory_evaluation, 2018.

[40] Zichao Zhang and Davide Scaramuzza. A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. In *IEEE International Conference on Intelligent Robots and Systems*, pages 7244–7251, 2018.