Fusionierung von Dense Visual Odometry für RGB-D Kameras mit dem Ackermann Motion Model

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Visual Computing

eingereicht von

Marc Haubenstock
Matrikelnummer 1525175

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr.techn. Johann Blieberger
Mitwirkung: Univ.Ass.Dipl.-Ing.Dr.techn. Markus Bader

Wien, 31. Oktober 2018

Marc Haubenstock	Johann Blieberger

Fusionierung von Dense Visual Odometry für RGB-D Kameras mit dem Ackermann Motion Model

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Visual Computing

by

Marc Haubenstock
Registration Number 1525175

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof.Dr.techn. Johann Blieberger
Assistance: Univ.Ass.Dipl.-Ing.Dr.techn. Markus Bader

Vienna, 31st October, 2018

Marc Haubenstock   Johann Blieberger

Acknowledgements

I would like to thank Dr. Markus Bader for supporting me throughout the course of my thesis. Without his prior work, this thesis would not have been possible.

Danksagung

Ich möchte hiermit Dr. Markus Bader für seine Unterstützung danken. Ohne seine vorgängig durchgeführte Arbeit wäre meine Diplomarbeit nicht möglich gewesen.

Abstract

The Automation Systems Group of the TU Wien has built multiple racecars in the scale of 1:10 for education purposes [KBB16]. These racecars have steering encoders for odometry estimation and a color and depth (RGB-D) camera system for perception. Currently the process of estimating the position and orientation (egomotion/odometry) is done via steering odometry only. However, such odometry is prone to be affected by wheel slippage and adverse conditions [Sca11]. To overcome these issues we propose to use *Visual Odometry* (VO) with motion priors, in order to produce a more robust egomotion estimate.

Motived by recent research and large commercial interest in self-driving cars [Lan17] and recent research in lie group based odometry [KSC13b, LeM$^+$17, FCDS17, EKC16], the aim of this thesis is twofold. First, to implement and examine a VO system, based on the works of Kerl et al. [KSC13b] without image pyramids. Second, to apply the VO to the racecar's RGB-D camera and steering systems. We will use the Ackermann motion model [FMB13, WM10] to further improve the VO estimate. This motion model is derived from the racecar's steering commands and is better suited for vehicles with tricycle like steering.

We tested our implementation on the dataset of the Technical University of Munich (TUM) [SEE$^+$12] and on a dataset generated for this thesis at the Technical University of Vienna(TUW). Our results for the TUM dataset show that the coarse trajectory is estimated correctly but due to the lack of an image pyramid, we are $5\times$ worse than the works of Kerl et al. [KSC13b]. Due to environmental conditions, the TUW dataset produced very sparse depth measurements. As such the VO's estimated trajectory had a drift of ≈ 0.17 m/s. However, we observed that using the Ackermann motion model we are able to improve the VO's trajectory estimation and reduce the drift to ≈ 0.12 m/s.

Kurzfassung

Die Automation Systems Group der TU Wien hat verschiedene ferngesteuerte Autos im Maßstab 1:10 für Lehrzwecke gebaut [KBB16]. Diese Autos haben Steuerungsencoder für Odometry-Schätzungen, Farbkameras und Tiefenkameras für räumliche Wahrnehmung. Das Problem mit steuerungsbasierter Odometry ist, dass diese anfälling für Fehler durch Rutschen der Räder und schlechte Umgebungsbedingungen ist [Sca11]. Um diese Nachteile zu überwinden, verwenden wir *Visual Odometry* (VO) mit "motion Priors", um eine robustere Laufbahn zu ermitteln.

Motiviert durch jüngste Forschung in Lie-Gruppen basierter Odometry [KSC13b, LeM$^+$17, FCDS17, EKC16] und kommerzielles Interesse an selbst fahrenden Autos [Lan17], haben wir in dieser Arbeit zwei Aufgabenstellungen. Erstens, ein Lie-Gruppen basiertes VO System, basierend auf der Arbeit von Kerl et al. [KSC13b] ohne Imagepyramiden, zu entwicklen. Zweitens, das VO System auf dem ferngesteuerte Auto der TU Wien anzuwenden. Wir werden das Ackermann-Bewegungsmodell [FMB13, WM10] benutzen, um die VO-Schätzung zu verbessern. Dieses Bewegungsmodell ist von den Steuerungskommandos des Autos abgeleitet und eignet sich besser für Fahrzeuge mit Dreirad-basierter Steuerung.

Wir haben unsere Implementierung auf Datensätzen der Technischen Universität München (TUM) [SEE$^+$12] und eigens erstellten Daten, der TU Wien, getestet. Unsere Resultate für die TUM Datensätze zeigen, dass wir eine grobe Laufbahn schätzen können. Aber durch die fehlende Imagepyramide ist unsere Laufbahn 5$\times$ schlechter als in den Arbeit von Kerl et al. [KSC13b]. Wegen schlechter Aufnahmebedigunen, hat die Tiefenkamera für den TUW Datensatz nur spärlich besetzte Aufnahmen erstellt. Dadurch hat die VO-Schätzung einen Drift von ≈ 0.17 m/s. Mit dem Ackermann-Bewegungsmodell ist es uns aber gelungen, die VO-Schätzung zu verbessern und den Drift auf ≈ 0.12 m/s zu reduzieren.

Contents

Abstract **ix**

Kurzfassung **xi**

1 Introduction **1**
1.1 Aim of the Thesis 1

2 State of the Art **3**
2.1 Visual Odometry 3
2.2 Simultaneous Localization and Mapping 4
2.3 Sensor Fusion Enabled Localisation for Image Based Approaches . . . 5

3 Background **9**
3.1 Camera Model 9
3.2 Special Euclidean 3 Group and its Lie Algebra 12
3.3 Least Squares Optimization 14
3.4 Image Alignment 17
3.5 Ackermann Motion Model 18
3.6 Velocity Motion Model 20
3.7 Image Filters 21

4 Environment & Setup **23**
4.1 Intel Realsense ZR300 RGB-D Camera 23
4.2 Optitrack & ROS 24
4.3 TU Wien Race Car 25
4.4 Programming Environment & Data Preprocessing 27

5 Approach **29**
5.1 Direct Motion Estimation 29
5.2 Fusing Visual Odometry with a Motion Model 35

6 Evaluation **39**
6.1 TUM Dataset 39
6.2 TUW Dataset 48

7 Conclusion and Future Work **61**

8 Appendix **63**
8.1 Exponential and Logarithmic Maps 63
8.2 Plots 65

List of Figures **69**

List of Tables **70**

Bibliography **73**

CHAPTER 1

Introduction

The Automation Systems Group of the TU Wien has built multiple racecars in the scale of 1:10 for education purposes [KBB16]. These racecars have steering encoders for odometry estimation and a color and depth (RGB-D) camera system for perception. Currently the process of estimating the position and orientation(egomotion/odometry) is done via steering odometry only. However, steering based odometry is prone to be affected by wheel slippage and adverse conditions [Sca11]. To overcome these issues we propose to use *Visual Odometry* (VO) with motion priors, in order to produce robust egomotion estimates.

VO is the process of estimating the egomotion of an agent, e.g. vehicle, robot, augmented reality-device, by only using camera images; usually color and depth. It has been demonstrated that VO provides more accurate trajectory estimates when compared to wheel odometry [Sca11]. VO is an important sensory system for navigation in environments where no external reference system e.g. GPS is available [KSC13b]. Such VO methods can scale to real-time performance, that is, processing the incoming images at at least 30fps [TAC08], but are numerically unstable [KSC13b, ALJI18] and thus result in noisy egomotion estimates without proper outlier detection. In the past VO has been successfully applied on planetary rovers such as the NASA Mars exploration program [MCM05a]. Recently it has also gained traction in areas of quadrocopters and handheld mobile devices [KSC13b, SEC14, ALJI18]. Furthermore, VO systems can be used as underpinnings of more advanced and robust *Simultaneous Localisation and Mapping* (SLAM) systems, which not only estimate an agent's position, but also generate a consistent map of the observed environment [KSC13a].

1.1 Aim of the Thesis

Motived by recent research in lie group based odometry [KSC13b, LeM$^+$17, FCDS17, EKC16] and large commercial interest in self-driving cars [Lan17], the aim of this thesis is twofold. First, to implement and examine a VO system based on Lie group manifolds, from the ground up and second, to apply the VO to the racecar's RGB-D camera and steering systems. We will use the Ackermann motion model [FMB13, WM10] to further improve the VO estimate. This motion model is derived from the racecar's steering commands and is better suited for vehicles with tricycle like steering.

For the first part, this thesis will present and explain the mathematical underpinnings of a VO system with respect to non-linear least squares methods, Lie groups and algebra and perspective camera models. Furthermore, we aims to establish the theoretical and practical understanding of how a state of the art visual odometry system works and how the Ackermann motion model impacts the egomotion estimate. This will be accompanied by an evaluation of a complete visual odometry system built from the ground up for this project. The most relevant work is presented in [KSC13b], that is, it will use a camera noise model and motion prior to stabilize the tracking. Since we aim to keep the VO process as simple as possible, it will not include an image pyramid scheme to increase the motion range.

Kerl et al. [KSC13b] stabilize the egomotion estimates by using the covariance of the linear velocity model. However, this motion model is very general and might not be optimal of Ackermann steering type vehicles, such as 4-wheeled cars. Instead this work proposes to use the racecar's steering commands to generate an Ackermann motion model and use that instead of the linear velocity model. The car used will be a remote controlled racecar built by the Automation System's Group at the TU Wien [KBB16]. We will focus on an offline environment, where recorded color, depth images and odometry data are used to compute the visual odometry required.

The VO system will be evaluated against a published dataset by the Technical University of Munich(TUM) [SEE+12], which contains intensity images, depth images and acceleration data for the xyz axis. It will also be evaluated against a dataset recorded at the TU Wien with an OptiTrack capture system. The data set recorded at the Technical University of Munich is split into several groups. For the sake of comparison this work will only focus on a subset that appeared in the paper [KSC13b]. The data recorded at the TU Wien is necessary since steering/wheel encoder values are needed to generate an Ackermann motion model; such values are absent from the dataset of the Technical University of Munich. The evaluation metric that will be used to compare the data is the root mean square error (RMSE) [SEE+12], when compared to ground truth motion, as used in the works of Kerl et al. [KSC13b].

CHAPTER 2

State of the Art

This chapter will present state of the art approaches in image based localization. It will also serve as a brief introduction into each of the presented fields. The fields being presented here are: *Visual Odometry* (VO), *Simultaneous Localization and Mapping* (SLAM) and sensor fusion enabled localisation for image based approaches that is, image based approaches that are augmented with further sensors e.g. *Inertial Measurement Units* (IMUs).

2.1 Visual Odometry

The term *Visual Odometry* was coined by Nistér in his landmark paper published in 2004 [NNB04]. Although motion reconstruction from stereo image pairs has been studied since the late 80s [Mat89], his paper described the methodology for the two main camps which comprise the VO scene today: stereo VO and monocular VO. As the name implies stereo VO uses image pairs obtained from a stereo camera pair, while monocular VO only uses one camera. A current state-of-the-art stereo VO system can be found aboard the NASA's mars rover [MCM05b]. This stereo VO system uses a feature based, or indirect, approach, which was the standard until direct approaches were first published in 2007/2008 [Sca11].

One of the first dense stereo VO systems was developed by Comport et al. [CMR07] whereas Silveria et al. are credited for the dense monocular case [SMR08]. Direct approaches avoid costly feature extraction by basing the objective function on raw image data directly. The main advantage of these approaches are that they minimize true error based on actual measurements [TAC08]. The major disadvantage of *Direct* approaches is that they are more susceptible to changes in illumination. That may either be changes in the scene's illumination between frames or camera internal post-processing such as gamma-correction, or artefacts such as vignetting [EKC16].

It is also worth noting the difference between the terms *Direct* and *Dense* as these imply subtly different things but are sometimes use interchangeably. *Direct* refers to the fact that, as previously mentioned, the objective function is defined in image space. *Dense* on the other hand refers to the amount of pixels, implying that most of the image pixels are used in the trajectory estimation process. More recent methods that only use pixel information along image gradients are termed *Semi-Dense* i.e. Semi-Dense Visual Odometry [ESC13]. Furthermore, a more in-depth topology of VO systems can be found in [Sca11, EKC16].

2.2 Simultaneous Localization and Mapping

A subject that is very closely related to VO is that of SLAM or rather, in the context of VO, Visual SLAM (V-SLAM). For the purposes of this thesis the terms SLAM and V-SLAM will be used interchangeably. Whereas VO is only concerned with a local consistent trajectory, SLAM methods generate a map of their environment. As such, they employ further steps that enforce global consistency. This means VO can be used as a building block for a complete SLAM algorithm [Sca11].

A general SLAM system can be decomposed into a front-end and back-end system, Figure 2.1 [EHE$^+$12, LeM$^+$17]. The front-end performs local trajectory estimation. This can either be from direct methods [SMR08, ESC14] or indirect methods e.g. using features [MAMT15, MTKW03, Dav03].

Since feature based approaches were the most common method of VO, it is only natural that feature based SLAM methods are the norm as well [SMR08]. The feature descriptors range from simple corner detectors such as the Shi-Thomasi detector in the works of Davison et el. [Dav03] to more robust feature detectors such as ORB features used in the ORB-SLAM project [MAMT15].

The back-end then enforces the global consistency of the trajectory. Early works used filtering techniques such as the Kalman-Filter[SC86]. However, convergence for the Kalman filter is quadratic in the order of landmarks/features and therefore, not applicable for large scale real-time applications [MTKW03]. FastSLAM 2.0 [MTKW03] offered improved performance compared to Kalman-filter based techniques, by using particle filters, most current state of the art SLAM systems e.g. ORB-SLAM [MAMT15], VINS-Mono [QLS18] use *Bundle Adjustment* (BA) [LeM$^+$17].

BA with respect to Computer Vision and Photogrammetry was first presented by Higgs et al. [TMHF00] in 2000. While the first real-time implementation was given by Mouragnon et al. [MLD$^+$06]. It is an iterative optimization technique which minimizes the error for a joint set of parameters. In the context of Computer Vision this usually means 3D feature points and camera parameters i.e. extrinsic and intrinsic. A global BA step optimizes all camera poses and 3D features at once. This is computationally very costly i.e. cubic in the number of parameters, thus limiting its use to small spaces even for feature based

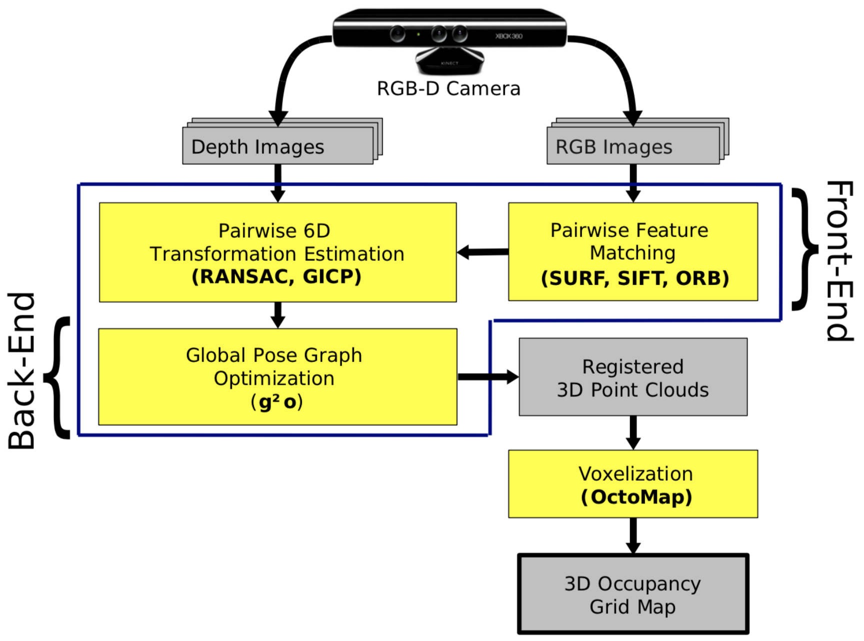

RGB-D Camera
Depth Images
RGB Images
Pairwise 6D Transformation Estimation (RANSAC, GICP)
Pairwise Feature Matching (SURF, SIFT, ORB)
Front-End
Back-End
Global Pose Graph Optimization (g²o)
Registered 3D Point Clouds
Voxelization (OctoMap)
3D Occupancy Grid Map

Figure 2.1: SLAM System Overview [EHE+12].

solutions [KM07]. State of the art BA solutions only use a subset of parameters that still enforce a high enough accuracy [SDMK11].

However, state of the art filtering approaches still exist, such as the Multi-State-Constrained Kalman Filter (MSCKF), which exhibits linear time complexity with the number of features [LM13] or the Discrete Extended Kalman Filter on Lie Groups (DLG-EKF) [BMGB13]. Furthermore, recently a filtering technique has been proposed named Exactly Sparse Delayed Filter on Lie Groups [LeM+17] which is able to improve on the findings of the DLG-EKF by using a sparse information matrix which makes performance comparable to graph based BA.

2.3 Sensor Fusion Enabled Localisation for Image Based Approaches

In this section we will present state of the art techniques which use additional sensors in conjunction with cameras for image acquisition. One segment of state of the art methods favour *Intertial Measurement Units* (IMUs) which are used in smartphones, or consumer grade vision systems. These IMUs are cheap and compliment camera sensors very well, as they are able to measure the metric scale of motion [FCDS17]. Another active segment of sensor fusion uses Light Detection and Ranging (LiDAR) sensors [ZS15]. However, depending on the robot being localized other sensors such as wheel encoders [MCM05b] or pressure sensors [Cre18] may be used as well.

Traditionally the integration of IMU measurements has been solved via filtering; specifically Extended Kalman Filters [LLB$^+$15]. However, similar to landmark based SLAM, maintaining an EKF over large measurements scales poorly as well as, exhibiting deteriorating accuracy due to continuing linearisation [SMD10, FCDS17]. As an alternative, state of the art techniques use graph-based smoothing [QLS18, FCDS17]. This is a similar concept to BA where a non-linear optimisation step is performed. In order to keep the computation time acceptable for real time applications, the smoothing is performed on a subset/bundle of the measurements termed *keyframes* and *preintegrated IMU factors*, for camera and IMU respectively, as seen in Figure 2.2; where the *structureless projection factor* encodes constraints on projected 3D landmarks. Although only a subset is used, it still achieves better accuracy than filtering techniques [SMD10, LLB$^+$15, FCDS17].

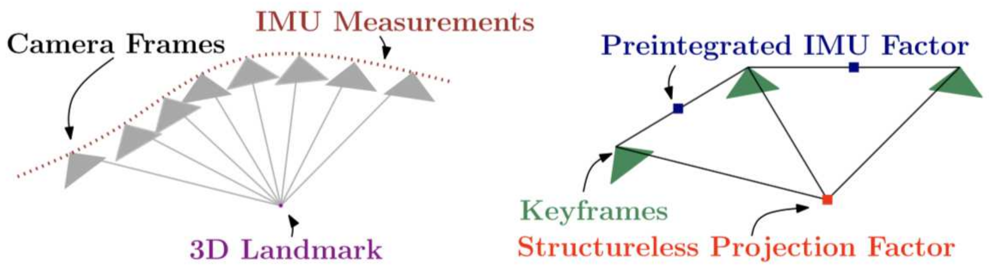


Figure 2.2: IMU VO Integration [FCDS17].

Lidar based VO named Visual-Liadar is also a very promising research field. A state of the art method proposed by Zhang et al. [ZS15] is currently (as of April 2019) leading the Kitti odometry bechmark [GLU12], an open source dataset used to evaluate the performance of SLAM and VO systems. Similar to the SLAM structure displayed by Figure 2.1, the Visual-Lidar system of Zhang et al. also works by splitting the general structure into two parts: Visual Odometry and Lidar Odometry. The VO is feature based and uses a weighted non-linear least squares optimization approach to estimate local trajectories; similar to what we will present in Section 3. The Lidar measurements are then used to remove the VO drift from the estimated trajectories. This is achieved via a multi-stage process, where the Lidar point cloud itself is denoised and then used to produce a global map of the environment and global consistent trajectories at 1Hz and 60Hz respectively.

Research concerned with using the Ackermann motion model in conjunction with VO approaches do not seem to be state of the art. Experiments have been done using EKF SLAM with Ackermann models [KYD15]. However, due to the aforementioned drawback on EKF based SLAM, this is no longer considered state of the art. Gräter et al. [GSL17] proposed to estimate Ackermann motion through the observed image sequences and use this to perform better outlier detection for feature based SLAM. Methods using

Dense VO with a motion model captured through external controls have not been found throughout this thesis' research.

CHAPTER 3

Background

Throughout this paper we will use the following notation concerning mathematical operations. Bold lowercase letters ($\mathbf{x}$) will represent vectors and bold uppercase letters ($\mathbf{H}$) will represent matrices. Scalars will be represented by lowercase letters (s), while functions can be represented as both cursive uppercase (F) and cursive lowercase letters (f). Images will be treated as functions and written as (I). An exception to this will be the image projection function which will be denoted as pi (π). As well as reserved letters **X**, X,Y,Z. Uppercase **X** will always represent a 3D point is space, where its coordiantes are given by X,Y,Z for the x,y and z axis respectively.

Group or type definitions such as the special Euclidean group SE(3) will be written in upper case letters. The Lie algebra of SE(3) will be written as $\mathfrak{se}(3)$. Group symbols will only occur for function definitions, or when explaining membership; as such there should be no ambiguity.

Sub and superscripts used throughout are as follows:

- $\mathbf{X}^{frame}$ - a 3D point in space with respect to a frame of reference "frame"
- ξ^k - A twist at iteration k
- f_i - The residual for an observation i.

3.1 Camera Model

In order to represent the workings of a physical camera in a computer program we need a formal abstract representation. The camera model is a set of mathematical equations which describes how 3D coordinates are mapped onto a 2D plane; the so-called *image plane*. In literature this function and the image plane itself are denoted with the symbol π [TV98, KSC13b].

Various camera models exist [HZ04, FW16] which accomplish this process of *projection*. We will use the *Affine sensor perspective projection* model [FW16, HZ04]. Since this is the only camera model we will consider, we will refer to it simply as the *perspective projection* model. We use this model, since the paper [KSC13b] we are comparing our work against, uses this model as well.

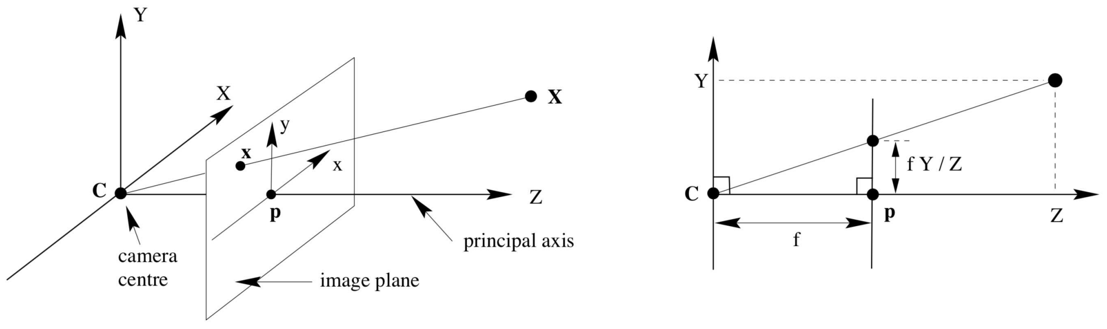


Figure 3.1: Perspective/Pinhole camera model[HZ04].

Figure 3.1 illustrates the *perspective camera model*. Often referred to as *pinhole camera model*, the names *pinhole* and *perspective* in this context are interchangeable [TV98]; we will call this the *perspective camera model*.

This model receives a 3D point in space $\mathbf{X}^{cam}$ = (X,Y,Z), with respect to the camera reference frame, and projects the point onto an image plane π. Again, multiple conventions exist on how to denote image coordinates. While [HZ04] uses the (x,y) convention in its diagrams, the paper we will be comparing our work against uses the (u,v) convention. We will be using the (u,v) convention as well. Therefore, the image coordinates are $\mathbf{x}^{img}$ = (x,y) = (u,v). For this section the 3D point $\mathbf{X}_{cam}$ will be abbreviated as $\mathbf{X}$.

$\mathbf{C}^{cam}$ is termed the *camera center* or the *optical center* [HZ04]. For this section $\mathbf{C}^{cam}$ will be abbreviated to $\mathbf{C}$. In the *perspective camera model*, Figure 3.1, the point $\mathbf{X}$ is projected along the line $\mathbf{CX}$ onto the image plane π. The line from the camera center perpendicular to the image plane is called the *principal axis*. The point where the *principal axis* intersects the image plane is called the *principal point* $\mathbf{p}$. The distance between the optical center and the principal point is called the focal length and is denoted with the symbol f.

In practice the sensor chips on the camera plane are not necessarily square and have to be scaled in the x and y direction independently. As such, the focal lengths are $f_x = s_x f$ and $f_y = s_y f$. Where s_x, s_y are the number of pixels per metric unit, usually taken as image width and image height respectively. Furthermore, the principal point of the camera model usually does not coincide with the origin of the image plane. This offset is modelled by the offset parameters o_x and o_y for x and y respectively [MKSS03].

Formalizing this process into a system of equations we have:

$$\pi(\mathbf{X}) = \Big(\frac{\mathrm{f}_x\mathrm{X}}{Z} + \mathrm{o}_x, \frac{\mathrm{f}_y\mathrm{Y}}{Z} + \mathrm{o}_y\Big) = (\mathrm{u}, \mathrm{v}) \tag{3.1}$$

The inverse being:

$$\pi^{-1}(\mathbf{x}) = \Big(\mathrm{X}, \mathrm{Y}\Big) = \mathrm{Z}\Big(\frac{\mathrm{u} - \mathrm{o}_x}{\mathrm{f}_x}, \frac{\mathrm{v} - \mathrm{o}_y}{\mathrm{f}_y}\Big) \tag{3.2}$$

The Jacobian being:

$$\frac{\delta\pi}{\delta\mathbf{X}} = \begin{bmatrix} \frac{\mathrm{f}_x}{Z} & 0 & \frac{-\mathrm{f}_x X}{\mathrm{Z}^2} \\ 0 & \frac{\mathrm{f}_y}{Z} & \frac{-\mathrm{f}_y Y}{\mathrm{Z}^2} \end{bmatrix} \tag{3.3}$$

In (3.1), the division by the depth Z is called the *perspective divide* and is non-linear in **X**; Distances and angles between points are not preserved [TV98].

We can also denote this process by factoring out the linear components into a matrix **K** called the *intrinsic matrix*.

$$\mathbf{X}^{'} = \begin{bmatrix} \mathrm{f}_x & 0 & \mathrm{o}_x \\ 0 & \mathrm{f}_y & \mathrm{o}_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{X} = \mathbf{KX} \tag{3.4}$$

$$\lambda \begin{bmatrix} \mathrm{u} \\ \mathrm{v} \\ 1 \end{bmatrix} = \mathbf{KX} \tag{3.5}$$

After (3.4) has been applied, it is then followed by an explicit division by Z in order to map $\mathbf{X}^{'}$ to the image plane. As such the image plane coordinates can also be interpreted as 3D coordinates with their depth fixed at 1 i.e. $\mathbf{X}_{image} = (\mathrm{u,v,1})$. This convention is termed *normalized image coordinates* [Sca11] as the projection of point **X** has been normalized to an image plane of focal length 1.

Since any depth information is lost by the *perspective division*, equation (3.4) is sometimes given with an unknown scale parameter λ [MKSS03, Sca11], as shown in equation (3.5).

The parameters $(\mathrm{f}_x, \mathrm{f}_y, \mathrm{o}_x, \mathrm{o}_y)$ in (3.4) are termed the *intrinsic parameters* and can be estimated via a process called *camera calibration* [Zha00]. This process wont be discussed at great length here since it is implemented in software packages such as *OpenCV* [Bra00]. In short, this process uses features of a known geometry in 3D space to estimate the intrinsic parameters of a camera.

This process also estimates non-linear distortion parameters associated with the camera's lens. These parameters can be used to remove these non-linear effects. All images, before being submitted to the presented software pipeline, will have undergone this procedure such that the perspective model holds.

3.2 Special Euclidean 3 Group and its Lie Algebra

The special Euclidean 3 group SE(3) is a subgroup of affine maps called *rigid motions*. *Rigid motions* are maps that preserve distances between points. However, the SE(3) has the additional property that it is orientation preserving as well [Gal11].

More formally the SE(3) group is a map transforming points from one Euclidean 3D space into another.

$$f : E \to F \tag{3.6}$$

where E and F are Euclidean affine spaces of dimension 3. The properties that a SE(3) group has to satisfy are, distance (3.7) and orientation (3.8) preserving:

$$\|f(\mathrm{a}) - f(\mathrm{b})\| = \mathrm{a} - \mathrm{b}, \forall \mathrm{a}, \mathrm{b} \in E \tag{3.7}$$

For any sequence $(\mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3)$ of 3 dimensional vectors and the orthonormal basis B1 of E and map f:

$$det_{B1}(\mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3) = det_{f(B1)}(\mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3) \tag{3.8}$$

A pose $\mathbf{P} \in \mathrm{SE}(3)$ is represented as a 4×4 matrix containing a 3×3 rotational matrix $\mathbf{R}$ and 3×1 translational vector $\mathbf{t}$; see (3.9).

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathrm{SE}(3) \tag{3.9}$$

The rotational matrix $\mathbf{R}$ is a member of the special orientation 3 group SO(3). As such, it has two important properties:

$$det(\mathbf{R}) = \pm 1 \tag{3.10}$$

$$\mathbf{R}^{-1} = \mathbf{R}^T \tag{3.11}$$

This gives rise to the definition of the inverse of the SE(3) matrix:

$$\mathbf{P}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{3.12}$$

Another very important characteristic of the SE(3) group is that it is a smooth manifold [Gal11]. As such, the SE(3) is also a Lie group with an associated Lie algebra $\mathfrak{se}(3)$. The $\mathfrak{se}(3)$ is a tangent space around the elements of the Lie group, Figure 3.2. This means differential quantities, such as Jacobians are well represented in the Lie algebra space and it is the optimal space, in which to perform optimizations. That is, it is not

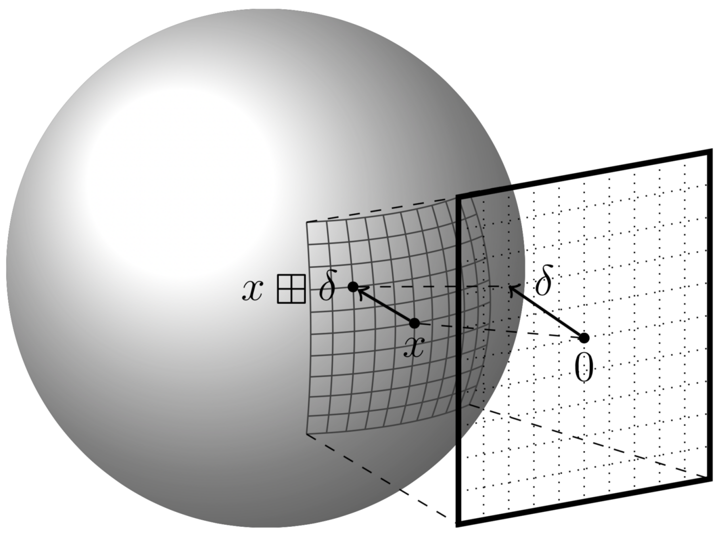


Figure 3.2: A representation of the group manifold (sphere) and its tangent space(plane)[HWFS11].

over-parametrized such as, Quaternions, nor does it suffer from singularities i.e. Gimbal lock in Tait-Bryan rotations [FMB13, HWFS11].

The Lie algebra, see (3.13), of the SE(3) matrix is a 6-dimensional vector, which can be split up into two 3-dimensional sub parts; one representing translation i.e. $\mathbf{u}$ and one representing rotations i.e. $\boldsymbol{\omega}$.

$$\boldsymbol{\xi} = \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} \mathrm{x}' \\ \mathrm{y}' \\ \mathrm{z}' \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \in \mathfrak{se}(3) \tag{3.13}$$

In the case of the SE(3) group, closed form solutions for the mapping between the Lie group, SE(3) and its Lie algebra, $\mathfrak{se}(3)$ exist in the form of the exponential and logarithmic map. A more detailed derivation of (3.14) and (3.15) can be found in Section 8.1.

$$exp : \mathfrak{se}(3) \rightarrow \mathrm{SE}(3) \tag{3.14}$$

$$log : \mathrm{SE}(3) \rightarrow \mathfrak{se}(3) \tag{3.15}$$

Furthermore we will use the box notation($\boxplus$, $\boxminus$) introduced in [HWFS11] to define operations between the Lie group and the Lie algebra space. The types of these functions is given in (3.16) and (3.17).

$$\boxplus : \mathrm{SE}(3) \times \mathfrak{se}(3) \rightarrow \mathfrak{se}(3) \tag{3.16}$$

$$\boxminus : \mathfrak{se}(3) \times \mathfrak{se}(3) \rightarrow \mathrm{SE}(3) \tag{3.17}$$

The boxplus ($\boxplus$) operator increments a pose $\mathbf{P}_t$ of the Lie group, by the Lie algebra $\boldsymbol{\xi}_{inc}$, see (3.18). The boxminus ($\boxminus$) operator gives the pose from $exp(\boldsymbol{\xi}_{t1})$ to $exp(\boldsymbol{\xi}_{t2})$, see (3.19). Both these operators will be used in Section 5.1.2.

$$\mathrm{P}_t \boxplus \boldsymbol{\xi}_{inc} = exp(\boldsymbol{\xi}_{inc})\mathbf{P}_t \tag{3.18}$$

$$\boldsymbol{\xi}_{t2} \boxminus \boldsymbol{\xi}_{t1} = exp(\boldsymbol{\xi}_{t1})^{-1} exp(\boldsymbol{\xi}_{t2}) \tag{3.19}$$

Finally, we define the Jacobian of the Lie group with respect to its transformation. Given a point $\mathbf{x}$ that is transformed by a pose $\mathbf{P}$ i.e. $\mathbf{Y} = \mathbf{PX}$. We can determine the change of $\mathbf{Y}$ with respect to its Lie algebra $\boldsymbol{\xi}$.

$$\frac{\delta \mathbf{Y}}{\delta \boldsymbol{\xi}} = \left.\frac{\delta exp(\boldsymbol{\xi})\mathbf{Y}}{\delta \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \sum_i exp(\mathbf{G}_i \mathbf{Y}) = \Big(\mathbf{I}_3| - \big[\mathbf{Y}\big]_\times\Big) \tag{3.20}$$

A more detailed derivation of (3.20) can be found in Section 8.1, where the values of $\mathbf{G}_i$ are the generators of the Lie algebra defined in Section 8.1.3. It is (3.20) that makes the lie algebra representation so attractive for least squares estimations and Kalman filters [HWFS11] since an objective function can directly be differentiated in terms of $\boldsymbol{\xi}$.

3.3 Least Squares Optimization

Given a function $f : \mathbb{R}^{\mathrm{n}} \rightarrow \mathbb{R}^{\mathrm{m}}$, $\mathrm{m} \geq \mathrm{n}$, we want to minimize $\|f(\mathbf{x})\|$ i.e.

$$\mathbf{x}^* = \mathrm{argmin}_{\mathbf{x}} F(\mathbf{x}) \tag{3.21}$$

where the residual function $F(\mathbf{x})$ is defined as:

$$F(\mathbf{x}) = \frac{1}{2}\sum_i^{\mathrm{m}} (f(\mathbf{x}_i))^2 = \|f(\mathbf{x})\|^2 = f(\mathbf{x})^T f(\mathbf{x}) \tag{3.22}$$

The factor of $\frac{1}{2}$ in (3.22) is present to eliminate the factor of 2 induced by differentiating a squared function. The value of $f(\mathbf{x}_i)$ can be interpreted as an independent observation. As such, the residuals of independent observations can simply be accumulated [Bis06]. The gradient of (3.22) is:

$$F^{'}(\mathbf{x}) = \mathbf{J}^T f(\mathbf{x}) \tag{3.23}$$

where $\mathbf{J}$ in (3.23) is,

$$\mathbf{J} = \frac{\delta f}{\delta \mathbf{x}} = \sum_{i}^{\mathrm{m}} \mathbf{J}_i = \sum_{i}^{\mathrm{m}} \frac{\delta f}{\delta \mathbf{x}_i} \tag{3.24}$$

For the simplest case, i.e. linear least squares of the form $f(\mathbf{x}) = \mathbf{b} - \mathbf{A}\mathbf{x}$, where $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{A} \in \mathbb{R}^{m \times n}$, the Jacobian is defined as: $\mathbf{J} = -\mathbf{A}$. From (3.23) it follows that:

$$F^{'}(\mathbf{x}) = -\mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}) \tag{3.25}$$

By setting (3.25) to zero, we obtain:

$$\mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{3.26}$$

where $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is called the *pseudo-inverse* or *Moore-Penrose inverse*.

3.3.1 The Gauss-Newton Method

For non-linear functions, we have to assume the function f has a continuous second derivative. Then we can write the Taylor expansion of f around $\mathbf{x}$ as:

$$f(\mathbf{x} \oplus \mathbf{h}) = f(\mathbf{x}) + \mathbf{J}\mathbf{h} + \mathrm{HoT} \tag{3.27}$$

where HoT in (3.27) are higher order terms i.e. derivatives of second order or higher. The Gauss-Newton method linearly approximates f for small $\|\mathbf{h}\|$ around $\mathbf{x}$ [MNT04]. The $\oplus$ operator is an abstract operator meaning *increment by*. The concrete definition of this operator depends on the domain of $\mathbf{p}$.

Disregarding the HoT, we can write (3.27) as:

$$f(\mathbf{x} \oplus \mathbf{h}) \approxeq f(\mathbf{x}) + \mathbf{J}\mathbf{h} \tag{3.28}$$

The residual function now becomes:

$$F(\mathbf{x} \oplus \mathbf{h}) \approxeq L(\mathbf{h}) = F(\mathbf{x}) + \mathbf{h}^T\mathbf{J}^T f + \frac{1}{2}\mathbf{h}^T\mathbf{J}^T\mathbf{J}\mathbf{h} \tag{3.29}$$

In order to find the optimizing step $\mathbf{h}^*$ (3.30) we need to set the first derivative to 0

$$\mathbf{h}^* = \text{argmin}_{\mathbf{h}} L(\mathbf{h}) \tag{3.30}$$

The derivative of (3.29) is then defined as:

$$L^{'}(\mathbf{h}) = \mathbf{J}^T f + \mathbf{J}^T \mathbf{J}\mathbf{h} \tag{3.31}$$

Setting (3.31) to 0 and rearranging for **h** gives:

$$\mathbf{h}^* = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T f \tag{3.32}$$

Iteratively solving for $\mathbf{h}^*$ and updating $\mathbf{x}$ as $\mathbf{x} = \mathbf{x} \oplus \alpha\mathbf{h}$ is termed the Gauss-Newton step. The parameter α is a user-defined step parameter, usually 1 in the Gauss-Newton algorithm. For a more thorough derivation we refer to [MNT04].

3.3.2 Iteratively Re-weighted Least Squares

One aspect of least squares methods, is that they can be derived from maximum likelihood estimators using Baye's therom [KSC13b]. Assuming we have a residual function f, a prior **h** and identical and independent observations we can write:

$$p(f|\mathbf{h}) = \prod_i p(f_i|\mathbf{h}). \tag{3.33}$$

Applying Baye's Rule on (3.33) we obtain a conditional probability for **h**:

$$p(\mathbf{h}|\ f) = \frac{p(f|\mathbf{h})p(\mathbf{h})}{p(f)}. \tag{3.34}$$

Where $p(f)$ in (3.34) denotes the prior distribution.

To find the maximum a posteriori distribution:

$$\mathbf{h}_{MAP} = \text{argmax}_{\mathbf{h}} \prod_i p(f_i|\mathbf{h}) \tag{3.35}$$

Instead of maximising (3.35) we can also minimize the negative log likelihood of (3.35). Ignoring the $p(f)$ term in (3.34), since it is a constant, we obtain:

$$\mathbf{h}_{MAP} = \text{argmin}_{\mathbf{h}} - \sum_i log(p(f_i|\mathbf{h})) - log(p(\mathbf{h})) \tag{3.36}$$

If we assume a constant prior in (3.36), the derivative with respect to **h** is defined as:

$$\frac{\delta \mathbf{h}_{MAP}}{\delta \mathbf{h}} = \sum_i \frac{\delta log(p(f_i|\mathbf{h}))}{\delta \mathbf{h}} = \sum_i \frac{\delta log(p(f_i|\mathbf{h}))}{\delta f_i} \frac{\delta f_i}{\delta \mathbf{h}} \tag{3.37}$$

As shown in [KSC13b, Sze10], we can define a weighting factor:

$$w(f_i) = \frac{\delta log(p(f_i))}{\delta f_i} \frac{1}{f_i} \tag{3.38}$$

Using (3.38) in (3.37) we obtain:

$$\frac{\delta \mathbf{h}_{MAP}}{\delta \mathbf{h}} = \frac{\delta f_i}{\delta \mathbf{h}} w(f_i) f_i. \tag{3.39}$$

Equation (3.39), when set to 0, minimizes the weighted least squares equation [KSC13b, Sze10]:

$$F = (w(f_i)f_i))^2 \tag{3.40}$$

It must be noted that for (3.38) $w(f_i)$ is defined as a derivative with respect to f_i. Therefore, when minimizing with respect to $\boldsymbol{h}$, it is a constant, in the sense that the chain rule does not have to be applied. From (3.39) we can observe that $w(.)$ scales the residual. As such we can introduce it as a diagonal matrix into (3.32) [KSC13b]:

$$\mathbf{h}^*_{weighted} = -(\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} f. \tag{3.41}$$

If f_i is not normally distributed, then $w(f_i)$ is not constant and has to be recomputed every iteration. For this reason the method is called *Iteratively Re-weighted Least Squares* (IRLS).

3.4 Image Alignment

Image alignment algorithms aim to establish the motion between two images. Applications for this include tracking, medial image registration and motion estimation [BM04]. The first algorithm using image alignment was proposed by Lucas and Kanade in 1981 to estimate the optical flow i.e. apparent motion between to images [LK81].

$$\sum_{\mathbf{x}} (I^*(\mathbf{x}) - I(W(\mathbf{x};\mathbf{p}))^2 \tag{3.42}$$

The residual function to minimize of such a problem is given in (3.42) [KHK13]. Where I^* is the reference image and I is the current or target image. $\mathbf{x}$ is an image pixel, $\mathbf{p}$ is a pose, and the function W is termed the warp function. It transforms pixels of image I into the coordinate frame of I^*. That is, I is warped into the coordinate frame of I^*. A central constraint of this formulation is the *photo-consistency assumption*. This assumes that pixels depicting the same 3D point will have an identical intensity value. Since, (3.42) is a least squares problem, gradient descent is the defacto standard approach [BM04].

$$\mathbf{p} = \mathbf{p} \oplus \Delta\mathbf{p} \tag{3.43}$$

The Lucas-Kanade algorithm assumes the pose $\mathbf{p}$ to be known and solves for the increment parameter $\Delta\mathbf{p}$. The pose is then updated as shown in (3.43). This is also called the *forward additive* approach [BM04].

3.4.1 Forward Compositional Algorithm

However, since we are following the works of Engel et al. we will instead use the *forward compositional* approach [KSC13b]:

$$\sum_{\mathbf{x}} (I^*(\mathbf{x}) - I(W(W(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p}))^2. \tag{3.44}$$

In each iteration of the Gauss-Newton process an update $\Delta\mathbf{p}$ to the current pose $\mathbf{p}$ is computed, (3.44). Each update $\Delta\mathbf{p}$ is taken to be around the origin. After each iteration, the warp is updated, (3.45)

$$W(\mathbf{x}; \mathbf{p}) = W(W(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p}). \tag{3.45}$$

The *forward compositional* approach has the advantage of being able to precompute parts of the Jacobian used in the Gauss-Newton process [KSC13b, KHK13]; a complete derivation is given in Section 5.1.

3.5 Ackermann Motion Model

The motion model used for this thesis is the Ackermann motion model. Most 4-wheeled or 3-wheeled vehicles can be described using Ackermann steering [FMB13]. In Ackermann models, torque is transmitted form the motor to front, but more often, rear wheels. Steering is handled by a single front wheel. In the case of 4-wheeled vehicles, where two wheel induce a rotation of the vehicle, the model can be transformed into an equivalent single virtual wheel [HWFS11]. An illustration of Ackermann type steering is shown in Figure 3.3.

The parameters of such a model are the 3D point $\mathbf{X}$, an action vector $\mathbf{u}$ comprised of linear speed v and angular rotation δ and the wheel base L which is the distance between the front and rear axel Figure 3.3.

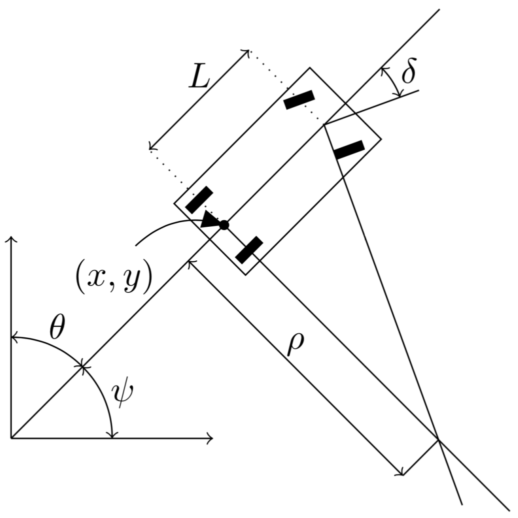


Figure 3.3: Ackermann Steering [WM10].

The pose increment in the robot's coordinate frame is given by (3.46). We can observe that in (3.46), the robot may not move along its y-axis explicitly. It may only move along its x-axis i.e. forward and re-orient itself.

$$\mathbf{X}_t^r(\mathbf{u}_t) = \begin{pmatrix} x_t^r \\ y_t^r \\ \theta_t^r \end{pmatrix} = \begin{pmatrix} v_t \\ 0 \\ \frac{v_t \tan(\delta_t)}{L} \end{pmatrix} \tag{3.46}$$

The current robot pose in world space at time t for a time interval Δt is given by (3.47) [FMB13, SC86]. Please note the distinction between the global robot pose $\mathbf{X}^w$ and the definition of a 3D point in space $\mathbf{X}$, Section 3.1. Both are 3-dimensional vectors, however, $\mathbf{X}^w$ is only defined on a 2D plane. That is movement along the y-axis of $\mathbf{X}$ is not defined. A transformation between the two spaces with be given in Section 5.2.

$$\mathbf{X}_t^w(\mathbf{X}_{t-1}^w, \mathbf{X}_t^r) = \begin{pmatrix} x_t^w \\ y_t^w \\ \theta_t^w \end{pmatrix} =$$
$$\begin{pmatrix} x_{t-1}^w + x_t^r \Delta t \cos(\theta_{t-1}^w) - y_t^r \Delta t \sin(\theta_{t-1}^w) \\ y_{t-1}^w + x_t^r \Delta t \sin(\theta_{t-1}^w) + y_t^r \Delta t \cos(\theta_{t-1}^w) \\ \theta_{t-1}^w + \theta_t^r \Delta t \end{pmatrix} = \begin{pmatrix} x_{t-1}^w + x_t^r \Delta t \cos(\theta_{t-1}^w) \\ y_{t-1}^w + x_t^r \Delta t \sin(\theta_{t-1}^w) \\ \theta_{t-1}^w + \theta_t^r \Delta t \end{pmatrix} \tag{3.47}$$

$$\mathbf{X}_t^w(\mathbf{X}_{t-1}^w, \mathbf{u}_t) = \begin{pmatrix} x_t^w \\ y_t^w \\ \theta_t^w \end{pmatrix} = \begin{pmatrix} x_{t-1}^w + v_t \Delta t \cos(\theta_{t-1}^w) \\ y_{t-1}^w + v_t \Delta t \sin(\theta_{t-1}^w) \\ \theta_{t-1}^w + \frac{v_t \tan(\delta_t)}{L} \Delta t \end{pmatrix} \tag{3.48}$$

However, it is also useful to define the world space transformation (3.47) in terms of the action vector $\mathbf{u}$. This definition (3.48) can be used to determine the Jacobian matrix with respect to the action vector $\mathbf{u}$. As such, we can define the Jacobian with respect to $\mathbf{X}^w_{t-1}$ as matrix G (3.49) and the Jacobian with respect to the action vector $\mathbf{u}_{t-1}$ as V (3.50).

$$G = \frac{\delta \mathbf{X}^w_t(\mathbf{X}^w_{t-1}, \mathbf{u}_t)}{\delta \mathbf{X}^w_{t-1}} = \begin{bmatrix} 1 & 0 & -v_t \sin(\theta_{t-1})\Delta t \\ 0 & 1 & v_t \cos(\theta_{t-1})\Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{3.49}$$

$$V = \frac{\delta \mathbf{X}^w_t(\mathbf{X}^w_{t-1}, \mathbf{u}_t)}{\delta \mathbf{u}_t} = \begin{bmatrix} \cos(\theta_{t-1})\Delta t & 0 \\ \sin(\theta_{t-1})\Delta t & 0 \\ \frac{\tan(\theta_{t-1})\Delta t}{L} & \frac{v_t \Delta t}{L\cos^2(\theta_{t-1})} \end{bmatrix} \tag{3.50}$$

Using G and V we are able to define the covariance of the motion model (3.51) [FMB13, SC86]. Where the matrix M is responsible for the noise of the action vector $\mathbf{u}$. For the purposes of this thesis, we did not model the noise of the action vector, thus it was set to the identity matrix i.e. $M = I_3$. (3.51) assumes multivariate Gaussian distributions of noise parameters. For small displacements, which are typical for motion models, this however, is an appropriate approximation [FMB13].

$$\Sigma_t = G^T \Sigma_{t-1} G + V^T M V \tag{3.51}$$

3.6 Velocity Motion Model

The velocity motion model is a simpler motion model when compared to the Ackermann motion model as it only describes straight line motions for a single point. We use this motion model to evaluate the motion prior of Kerl et al. [KSC13b]. The TUM dataset supplies acceleration values in the xyz axis. As such, we are able to construct a simple motion model which uses straight line kinematics (3.52).

$$\mathbf{X}_t = \mathbf{X}_{t-1} + \mathbf{v}_{t-1}\Delta t + \frac{1}{2}\mathbf{a}_t \Delta t^2 \tag{3.52}$$

In this case the action vector at time t $\mathbf{u}_t$ is comprised of the acceleration values along the xyz axis. The state vector $\mathbf{s}_t$ contains the position of the described point and the velocity values along each axis (3.53).

$$\mathbf{s}_t = \begin{pmatrix} x_t \\ y_t \\ z_t \\ v_{xt} \\ v_{yt} \\ v_{zt} \end{pmatrix} \tag{3.53}$$

The state update equation (3.54) with the action vector **u** now becomes:

$$\mathbf{s}_t(\mathbf{s}_{t-1}, \mathbf{u}_t) = \begin{pmatrix} x_t \\ y_t \\ z_t \\ v_{xt} \\ v_{yt} \\ v_{zt} \end{pmatrix} = \begin{bmatrix} x_{t-1} + v_{x,t-1}\Delta t + \frac{1}{2}a_{x,t}\Delta t^2 \\ y_{t-1} + v_{y,t-1}\Delta t + \frac{1}{2}a_{y,t}\Delta t^2 \\ z_{t-1} + v_{z,t-1}\Delta t + \frac{1}{2}a_{z,t}\Delta t^2 \\ v_{x,t-1} + a_{x,t}\Delta t \\ v_{y,t-1} + a_{y,t}\Delta t \\ v_{z,t-1} + a_{z,t}\Delta t \end{bmatrix} \tag{3.54}$$

The Jacobians **G** (3.55) and **V** (3.56) with respect to state and action respectively are defined as:

$$\mathbf{G} = \frac{\delta \mathbf{s}_t(\mathbf{s}_{t-1}, \mathbf{u}_t)}{\mathbf{s}_{t-1}} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.55}$$

$$\mathbf{V} = \frac{\delta \mathbf{s}_t(\mathbf{s}_{t-1}, \mathbf{u}_t)}{\mathbf{u}_t} = \begin{bmatrix} v_{x,t-1}\Delta t + a_{x,t}\Delta t & \frac{1}{2}\Delta t^2 & 0 & 0 \\ v_{y,t-1}\Delta t + a_{y,t}\Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ v_{z,t-1}\Delta t + a_{z,t}\Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 \\ a_{x,t} & \Delta t & 0 & 0 \\ a_{y,t} & 0 & \Delta t & 0 \\ a_{z,t} & 0 & 0 & \Delta t \end{bmatrix} \tag{3.56}$$

We note that in (3.56) the time t was made a part of the action vector **u**. If we approximate the distribution of **s** as a normal probability distribution [FMB13], similar to the Ackermann motion model, then we can use the same covariance update formulae (3.51) to derive a covariance matrix.

3.7 Image Filters

In order to compute the Jacobian for the non-linear least squares algorithm we will need to compute the image derivates in x and y. A more detailed explanation as to why we need these values is given in Section 5. Image gradients can be seen as approximations of the image derivates and can be used in its place [KHK13]. *OpenCV* has multiple implementations such as the Sobel and Scharr operator. Both are first order approximations which compute the gradient using filter convolutions.

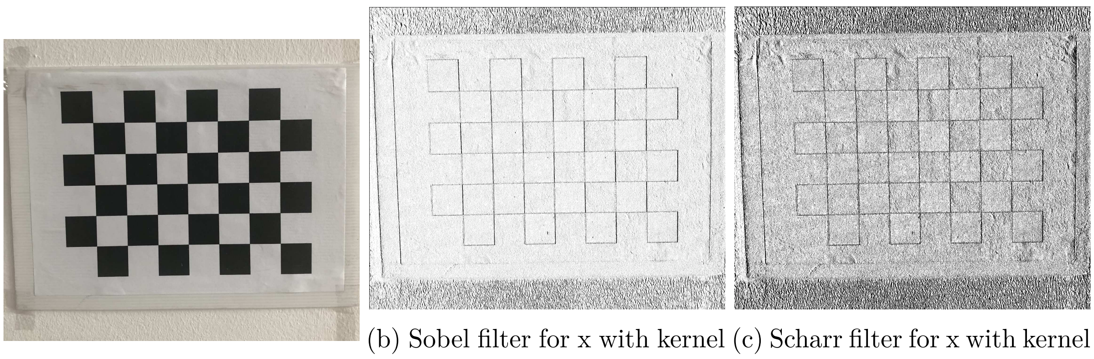
$$\text{sobel}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.57}$$

$$\text{scharr}_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \tag{3.58}$$

Furthermore, *OpenCV* also implements a 1D approximation of the sobel filter:

$$\text{sobel}_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{3.59}$$

The Scharr (3.58) and Sobel (3.57) filter kernels, that is kernel size of 3×3 and greater, combine smoothing and derivative approximation in the filter convolutions, while the 1D Sobel filter (3.59) only approximates the derivative, without a smoothing step. The given filter kernels (3.57), (3.58), (3.59) are approximation of the x derivative. For the y derivate the transpose is used. For this thesis we will use kernels (3.58),(3.59) to examine how they impact the motion estimation. During the application of our VO algorithm we will investigate what, if any, difference kernels, Figure 3.4 make.



(a) Base image.

(b) Sobel filter for x with kernel size 1.

(c) Scharr filter for x with kernel size 3.

Figure 3.4: Visualizations of image filters for changes in x mapped to 16 bit greyscale. Color space is inverted.

CHAPTER 4

Environment & Setup

This section will provide details on the hardware and software environments used to carry out the data acquisition. As mentioned in Section 1.1, we will also use the TUM dataset for evaluation purposes. This dataset has been used in the evaluation of state of the art VO and SLAM systems [ESC14, MAMT15, KSC13b]. Further details will be omitted as a publication for this dataset exists [SEE+12].

4.1 Intel Realsense ZR300 RGB-D Camera

Figure 4.1: Image of the Intel Realsense ZR300 [Int19].

For image and depth measurement, this thesis used the Intel Realsense ZR300 Camera 4.1. RGB images were captured at a rate of 30 frames per second (fps) with a resolution of 640×480. The depth images we captured at 30 fps with a resolution of 480×340. Since the size of color and depth images have to be equal for VO to work, the depth images have been upscaled via cubic interpolation using the *OpenCV* library [Bra00].

Depth is estimated via the left and right Infrared (IR) cameras as well as the IR laser projector. The depth image is aligned with the color image via the camera's driver. The range of the depth measurement is 0.55 m – 2.8 m [Int19].

Although the ZR300 driver has default camera intrinsic parameters i.e. focal length and optical center, we estimated them explicitly to increase the accuracy of the pixel backprojection. Camera calibration is a standard practice in state of the art computer

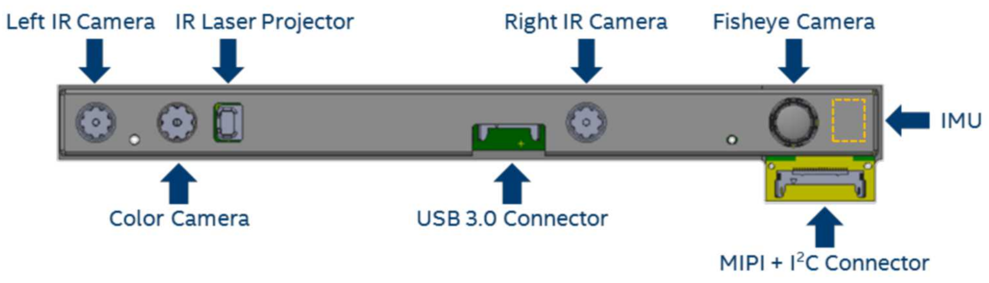


Figure 4.2: Diagram of the Intel Realsense ZR300's components [Int19].

vision systems [SEE+12, ESC14, MAMT15]. The backprojection was done offline on a checker board pattern 4.3 using *OpenCV*. More details on this topic are given in Section 3.1.

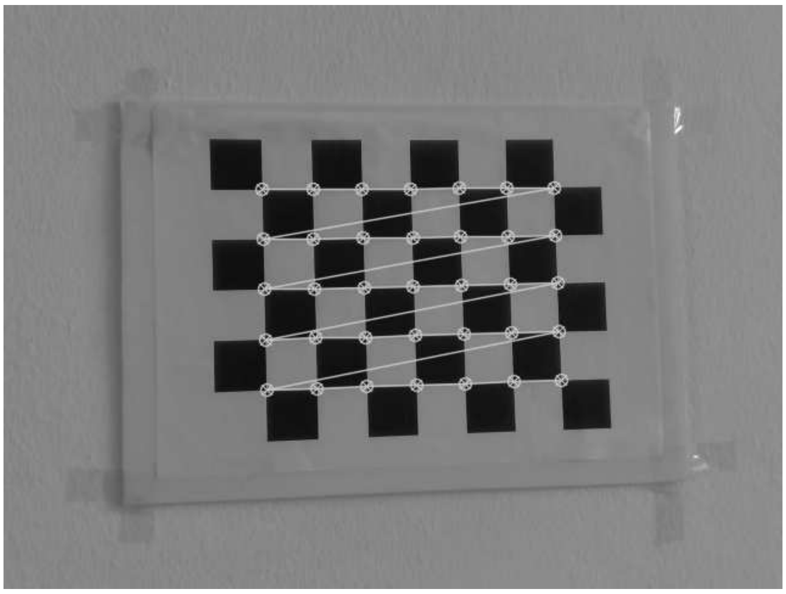


Figure 4.3: Checker board pattern used for camera calibration. Corners used for backprojection are highlighted.

4.2 Optitrack & ROS

In order to interface, manage and process control and sensor data we use the *Robot Operating System* (ROS). It is an opensource framework that is used widely in the robotics computer vision community. Many published VO or SLAM algorithms have ROS bindings [ESC14, MAMT15, LM13].

For this thesis we used ROS for its message passing and capturing capabilities. This allowed us to efficiently capture control and sensor data as well as to distribute this data to any attached devices, such as the racecar itself or a computer for monitoring purposes. The ROS architecture is comprised of a master server which acts as a registry to which ROS nodes can subscribe. These nodes can publish data to the ROS master server via topics. All other nodes registered to the same ROS master can listen on selected topics and receive the published data. All input data is converted to ROS messages, which is an interface standard for ROS. By conforming to this, any ROS powered device is able to receive and process our message. The ROS messages used are open source and freely available under the TUW-Robotics repository[1]. Specifically, these messages are processed by the Ardunio sub-system and converted into steering commands. The capture data is also written to disk in order to process it for the VO system.

For groundtruth data capture, The TU Wien uses an OptiTrack motion capture system using an array of OptiTrack's Prime 13 cameras, Figure 4.4. These cameras offer an image acquisition rate of 240 fps and precision in the range of $0.2 - 0.5$ mm [Opt19]. The cameras are connected via LAN to a host computer which can triangulate readings from OptiTrack markers, and display a pose in 3D space.

Figure 4.4: OptiTrack Prime 13 Camera [Opt19].

4.3 TU Wien Race Car

The TU Wien, under the supervision of Dr. Markus Bader, has produced a number of remote controlled race cars, Figure 4.5. These vehicles use off the shelf components and have 3D printed chassis. In order to acquire accurate pose data that can be used as ground truth values, the race car is able to mount OptiTrack markers at various locations around the vehicle.

[1]https://github.com/tuw-robotics (14.03.2019)

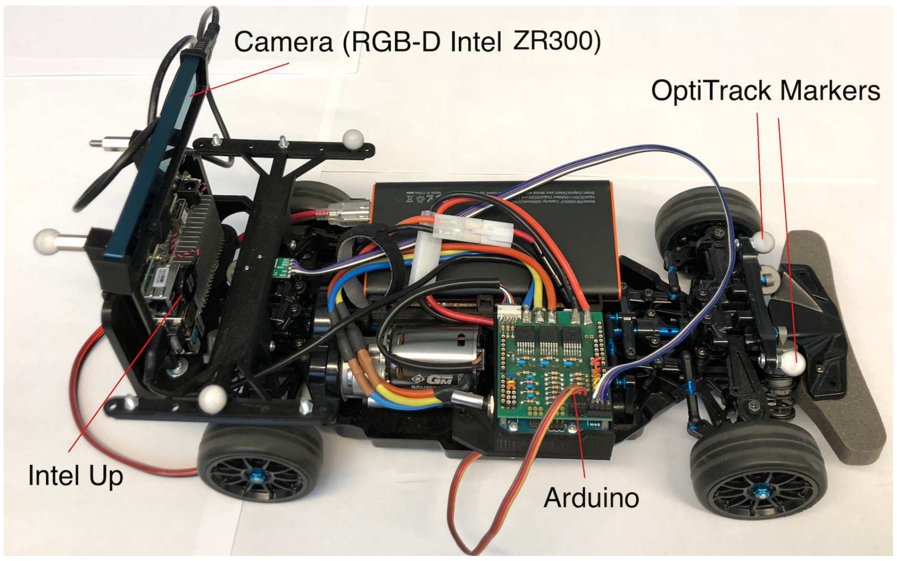


Figure 4.5: TU Wien Racecar.

Each vehicle is equipped with an Intel Up micro PC, for general purpose tasks such as running ROS, or receiving data from the attached camera, and an Arduino which is used for controlling the vehicle's BLDC engine and servos. Power to the Intel Up and engine is supplied via a AmazonBasics powerbank. A schematic view of the control hierarchy can be seen in, Figure 4.6.

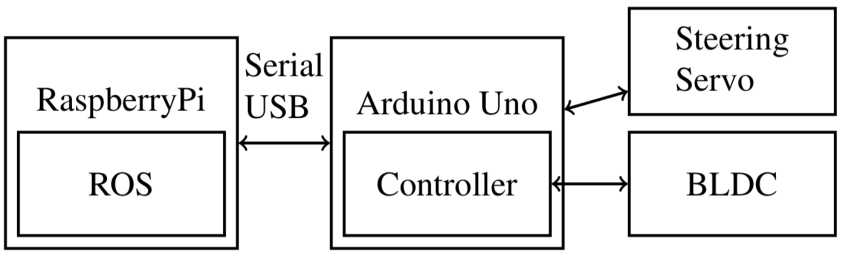


Figure 4.6: TU Wien Racecar Control Hierarchy [KBB16].

Steering currently works via a USB 2.0 gamepad controller, which connects to the Intel Up. The Intel Up receives gamepad commands and converts these to timestamped messages which can be passed via the ROS framework. These commands are passed to the arduino which converts them to steering commands for the engine and wheel servo. In this setup two different message types are sent between the racecar and the ROS master server: Odometry and Twist.

Odometry messages encode the racecar's pose via a 6-dimensional vector. Similar to the Lie algebra, Section 3.2, this vector can be broken up into a translational and rotational part. However, although both vectors have the same dimension, the interpretation is different. While the lie algebra vector represents the tangent space, the twist vector already encodes the vehicles pose via velocity and delta Euler angles, denoted $\theta_{x,y,z}$ (4.1).

$$\text{odometry\_message} = \begin{pmatrix} \mathrm{v}_x \\ \mathrm{v}_y \\ \mathrm{v}_z \\ \Delta\theta_x \\ \Delta\theta_y \\ \Delta\theta_z \end{pmatrix} \tag{4.1}$$

Twist messages encode the vehicles steering commands. In our case this will be the racecar's revolution(rev) and steering "angle"(δ) in the range of $[-1, 1]$, for left and right respectively. As such the twist message can be seen as a 2-dimensional vector 4.2. Both values are generated by the gamepad controller directly and therefore the "angle" command is not in radians but simply use as an input to the racecar's servos.

$$\text{steering\_message} = \begin{pmatrix} \text{rev} \\ \delta \end{pmatrix} \tag{4.2}$$

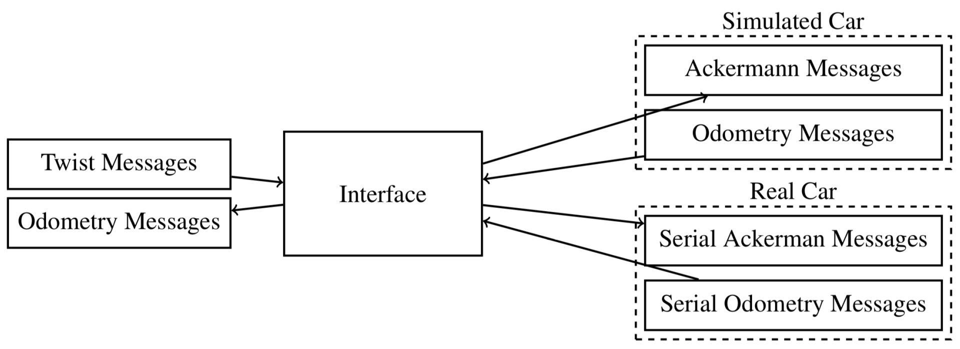


Figure 4.7: TU Wien Racecar Message Passing [KBB16].

4.4 Programming Environment & Data Preprocessing

All the captured data was written into ROS bag files, which is a data input output format for ROS applications. The relevant information, that is, images timestamps, ground truth and steering commands were extracted and written to files. As the depth images are of a lower resolution (320 × 480) than the intensity images (640 × 480) they were scaled up to the resolution of the intensity images; the scaling was done using cubic interpolation via *OpenCV*.

As a realtime application was beyond the scope of this thesis, the VO algorithm was implemented in Python 3.0. Python has very good support for parsing text files, and linear algebra via the NumPy library [vdWCV11]. Furthermore, NumPy supports *OpenCV* matrices out of the box, which made it a very good fit since we relied upon *OpenCV* for image processing.

All data was matched prior to VO execution. That is using the filenames of the intensity images as primary keys, the intensity images were matched with the corresponding depth images, ground truth, and steering commands. All intensity images were converted to greyscale using *OpenCV*. In order to make the greyscale images more robust to uniform changes in illumination between image frames, we applied a Z-standardisation. Z-standardisation is a linear re-scaling such that the data has zero value mean (μ) and a standard deviation (σ) of 1 [Bis06].

CHAPTER 5

Approach

The chapter will present the specifics of the lie algebra based motion estimation, given in Section 3. This includes specific definition of the functions used, explanation of parameters and pseudo-code in order to give a general understanding of how the estimation process was implemented.

5.1 Direct Motion Estimation

As stated in Section 3.4, we treat the pose estimation as an image alignment problem. The specific formulation we will use is the *forward compositional* approach (3.44). This means that for a pixel i of an image with total dimension D and iteration k the residual is defined as given in [KSC13b]:

$$F(\boldsymbol{\xi}^k) = \sum_i^{\mathrm{D}} F_i(\boldsymbol{\xi}^k) = \sum_i^{\mathrm{D}} (f_i(\boldsymbol{\xi}^k))^2 = \sum_i^{\mathrm{D}} (I^*(\mathbf{x}_i) - I(W(W(\mathbf{x}_i; exp(\boldsymbol{\xi}^k)); \mathbf{P}))^2 \tag{5.1}$$

The warp function W for pose $\mathbf{P} \in \mathrm{SE}(3)$ in 5.1 is defined as:

$$W(\mathbf{x}_i; \mathbf{P}) = \pi(\mathbf{P}(\pi^{-1}(\mathbf{x}_i), I^*_{depth}(\mathbf{x}_i), 1)^T) \tag{5.2}$$

Where I^*_{depth} is the depth image of image I^* with the same dimensions as I^*. Also note, that in the final projection π we implicitly drop the homogeneous coordinate. Since the back-projection of pixels into 3D coordiantes, i.e. the application of π^{-1} can be precomputed, we can simplify the warp W:

$$W(\mathbf{x}_i; \mathbf{P}) = \pi(\mathbf{P}\mathbf{X}_i) \tag{5.3}$$

Where $\mathbf{X}_i$ in (5.3) is the back-projection of pixel $\mathbf{x}_i$. Since the image projection function π and the pose matrix $\mathbf{P}$ are non-linear, according to (3.29), we have to linearise the residual around $\mathbf{x}$:

$$F(\mathbf{x_i} \oplus \boldsymbol{\xi}) \approx L(\boldsymbol{\xi}) = \sum_i^{\mathrm{D}} F_i(\mathbf{0}) + \boldsymbol{\xi}^T \mathbf{J}_i^T f_i(\mathbf{0}) + \frac{1}{2}\boldsymbol{\xi}^T \mathbf{J}_i^T \mathbf{J}_i \boldsymbol{\xi} \tag{5.4}$$

Where $f_i(\mathbf{0})$ in (5.4) equals $I^*(\mathbf{x}_i) - I(W(\mathbf{x}_i; \mathbf{P}))$. Note that $W(\mathbf{x}_i; \mathbf{P})$ is an accumulative value according to (3.45). This means that $f_i(\mathbf{0}^k) = f_i(\boldsymbol{\xi}^{k-1})$. Derivating (5.4) with respect to $\boldsymbol{\xi}$ we obtain:

$$\frac{\delta L(\boldsymbol{\xi})}{\delta \boldsymbol{\xi}} = \sum_i^{\mathrm{D}} \mathbf{J}_i^T f_i(\mathbf{0}) + \mathbf{J}_i^T \mathbf{J}_i \boldsymbol{\xi} \tag{5.5}$$

Setting 5.5 equal to 0 and solving for $\boldsymbol{\xi}$ we obtain the minimizing parameter update for iteration k :

$$\boldsymbol{\xi}^k = \sum_i^{\mathrm{D}} -(\mathbf{J}_i^T \mathbf{J}_i)^{-1} \mathbf{J}_i^T f_i(0). \tag{5.6}$$

As stated in (3.24), the Jacobian $\mathbf{J}_i$ of a residual function F with respect to the optimizing parameter, in our case $\boldsymbol{\xi}$, is defined as

$$\mathbf{J}_i = \frac{\delta f_i}{\delta \boldsymbol{\xi}} = \frac{\delta I}{\delta W_i} \cdot \frac{\delta W_i}{\delta \boldsymbol{\xi}} = \mathbf{J}_I \cdot \mathbf{J}_W. \tag{5.7}$$

From (5.3) we can deduce that $\delta W_i = \delta\pi \cdot \delta \mathbf{P} \mathbf{X}_i$. However, since we are interested in the Jacobian around $\boldsymbol{\xi} = 0$ we can assume the pose $\mathbf{P} = I$. This means that we can expand (5.7) as:

$$\mathbf{J}_i = \frac{\delta f_i}{\delta \boldsymbol{\xi}} = \frac{\delta I}{\delta \pi} \cdot \frac{\delta \pi}{\delta \mathbf{X}_i} \cdot \frac{\delta \mathbf{X}_i}{\delta \boldsymbol{\xi}} = \mathbf{J}_I \cdot \mathbf{J}_\pi \cdot \frac{\delta \mathbf{X}_i}{\delta \boldsymbol{\xi}} \tag{5.8}$$

The three derivatives which we have to define in 5.8 can be written as follows:

The Jacobian $\mathbf{J}_I$ (5.9) is the gradient of image I. This can be computed via image filters such as Sobel, or Scharr filters. These are standard image filters approximating the gradient and are implemented in *OpenCV*[Bra00]. We give a comparison between Sobel and Scharr filters in Section 6.1

$$\mathbf{J}_I = \begin{bmatrix} \nabla I_x & \nabla I_y \end{bmatrix}. \tag{5.9}$$

The Jacobian $\mathbf{J}_\pi$(5.10) is the Jacobian of the perspective projection function π (3.1). The Jacobian has been defined in (3.3). For the sake of completeness we will repeat it here.

$$\mathbf{J}_\pi = \begin{bmatrix} \frac{\mathrm{f}_x}{\mathrm{Z}_i} & 0 & \frac{\mathrm{f}_x}{\mathrm{Z}_i^2} \\ 0 & \frac{\mathrm{f}_y}{\mathrm{Z}_i} & \frac{\mathrm{f}_y}{\mathrm{Z}_i^2} \end{bmatrix} \tag{5.10}$$

Where the value of Z_i in (5.10) is the depth value for pixel $\mathbf{x}_i$. The values for (5.11) can be computed using (3.20). The homogeneous value of $\mathbf{X}$ is implicitly removed.

$$\frac{\delta \mathbf{X}_i}{\delta \boldsymbol{\xi}} = \mathbf{J}_{lie} = \begin{bmatrix} 1 & 0 & 0 & 0 & \mathrm{Z}_i & -\mathrm{Y}_i \\ 0 & 1 & 0 & -\mathrm{Z}_i & 0 & \mathrm{X}_i \\ 0 & 0 & 1 & \mathrm{Y}_i & -\mathrm{X}_i & 0 \end{bmatrix} \tag{5.11}$$

The term $\mathbf{J}_\pi \cdot \mathbf{J}_{lie}$ can be precomputed before the Gauss-Newton iteration starts. The image gradient $\mathbf{J}_I$ can also be precomputed, but has to be sampled at every iteration.

5.1.1 Weighted Gauss Newton Step

In order to increase robustness against image noise Kerl et al. use an IRLS scheme with an optional motion prior [KSC13b]. Since the derivation of the IRLS scheme has been given in 3.3.2, this section will explain the distribution used to weight the samples as well as the method used to compute it.

Kerl et al. observed the residual between image frames and found that the t-distribution was the best fit [KSC13b]. The parameters of the t-distribution functions are the mean μ the variance σ^2 and the degrees-of-freedom ν. μ is assumed to be 0 and ν is set to be 5 based on the empirical data of Kerl et al. [KSC13b].

$$w(\mathbf{x}_i) = \frac{\nu + 1}{\nu + (\frac{r_i}{\sigma})^2} = \frac{6}{5 + (\frac{r_i}{\sigma})^2} \tag{5.12}$$

The factor or σ^2 in (5.12) has to be computed iteratively (5.13) but the algorithm quickly converges [KSC13b].

$$\sigma^2 = \frac{1}{D} \sum_i^D r_i^2 \frac{\nu + 1}{\nu + (\frac{r_i}{\sigma})^2} \tag{5.13}$$

The final update step becomes:

$$\boldsymbol{\xi}^k = \sum_i^{\mathrm{D}} -(\mathbf{J}_i^T \mathbf{W} \mathbf{J}_i)^{-1} \mathbf{J}_i^T \mathbf{W} f_i(0) \tag{5.14}$$

where each diagonal element W_{ii} in (5.14), is equal to $w(\mathbf{x}_i)$.

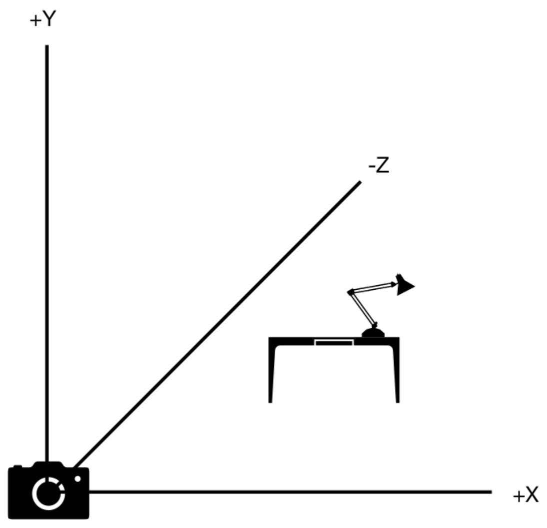


Figure 5.1: Coordiante System induced by our VO algorithm.

5.1.2 Visual Odometry Pipeline

The coordinate system used throughout the implementation with be a right handed with the camera viewing direction being along -Z, Figure 5.1. As stated in Section 3.2, the Lie algebra is a tangent space for the SE(3) group. This means that the sign of the generator affects the algorithms notion of what is "forward" and what is "backward" movement. To ensure that the estimation motion transforms conforms to this coordinate system two Lie algebra generators have to be negated. Specifically the generator responsible of movement along the Z-axis, as well as the generator responsible for rotation around the Y-axis i.e. pitch. With respect to the definition given in Section 8.1.3 this results in a negation of G_3 and G_5.

Steps 2 – 11 in Algorithm 5.1 allocate memory and precompute variables which are using the Gauss-Newton loop. Step 3 backprojects image coordinates into 3D space. Since the values of the depth image may be undefined, we supply a MAXDEPTH value. This assures that the whole pipeline works, even for undefined measurements. Furthermore, we generate a **valid_measurements** vector, which is an array of flags identifying if the corresponding pixels, of either the reference or target image I^*, I, is associated with a valid depth measurement or not. This allows us to decide if invalid measurements are permitted in the various functions of the Gauss-Newton loop. Specifically this concerns the functions GaussNewtonStep() and ComputeResidual(). Depending on the density of the depth image we may have to permit invalid measurements at certain points. The function ComputeValidPixels either returns the total image dimension D, or the sum of either **valid_measurements**$[:, :, \text{reference}]$ or **valid_measurements**$[:, :, \text{target}]$.

Steps 4-6 implement the functions which are defined n in (3.1), (3.3), (5.7) respectively. It must be noted that $\mathbf{x}_{new}$ is a 2D array of pixels $\mathbf{x}_{i\_new}$. The ComputeResidual()

Algorithm 5.1: Direct Motion Estimation

```
Data: $I^*, I, \nabla I_{xy}$, $\mathbf{K}$, $\alpha$, EPS, MAX, MAXDEPTH
Result: A pose $\mathbf{P} \in \mathrm{SE}(3)$.
begin
    $\mathrm{G}_{1:6} \longleftarrow \mathrm{ComputeGenerators}()$
    $(\mathbf{X}, \mathbf{valid\_measurements}) \longleftarrow \mathrm{BackprojectPixels}(\mathbf{x}, I^*, I^*_{depth}, I, I_{depth}, \mathbf{K}^{-1}, \mathrm{MAXDEPTH})$
    $\mathbf{x}_{new} \longleftarrow \mathrm{ProjectOntoImage}(\mathbf{X}, \mathbf{K})$
    $\mathbf{J}_\pi \longleftarrow \mathrm{ComputeJacobianPi}(\mathbf{X}, \mathbf{K})$
    $\mathbf{J}_W \longleftarrow \mathrm{PrecomputeJacobian}(\mathbf{J}_\pi, \mathrm{G}_{1:6}, \mathbf{X})$
    $\mathrm{D} \longleftarrow \mathrm{ComputeDimension}(I)$
    $\mathbf{f} \longleftarrow \mathrm{ComputeResidual}(I, \mathbf{x}_{new}, \mathrm{D}, \mathbf{valid\_measurements})$
    $\mathbf{P} \longleftarrow \mathbf{I}_4$
    $\mathrm{it} \longleftarrow 0$
    $\mathrm{eps} \longleftarrow \mathrm{System.MaxValue}$
    $\mathrm{F} \longleftarrow \mathrm{System.MaxValue}$
    while $eps > EPS$ and $it < MAX$ do
        $\Delta\xi \longleftarrow \mathrm{GaussNewtonStep}(\mathbf{J}_W, \mathbf{W}, \nabla I_{xy}, \mathbf{f}, \mathbf{valid\_measurements})$
        $\Delta\xi \longleftarrow \alpha \cdot \Delta\xi$
        $\mathbf{P} \longleftarrow \mathbf{P} \boxplus \Delta\xi$
        $\mathbf{Y}_{new} \longleftarrow \mathbf{PX}$
        $\mathbf{x}_{new} \longleftarrow \mathrm{ProjectOntoImage}(\mathbf{Y}_{new}, \mathbf{K})$
        $\mathbf{f} \longleftarrow \mathrm{ComputeResidual}(I, \mathbf{x}_{new})$
        $\mathbf{W} \longleftarrow \mathrm{ComputeWeighting}(\mathbf{v}, \mathrm{D}, \nu = 5, \mathrm{eps}_w = 0.00001, \mathrm{MAX\_IT}_w = 10000)$
        $\mathrm{F\_prev} \longleftarrow \mathrm{F}$
        $\mathrm{F} \longleftarrow \frac{\mathbf{f}^T\mathbf{W}\mathbf{f}}{\mathrm{D}}$
        $\mathrm{eps} \longleftarrow |\mathrm{F} - \mathrm{F\_prev}|$
        $\mathrm{it} \longleftarrow \mathrm{it} + 1$
    end
end
```

function at lines 8 and 18 is the application of the residual function f_i over all pixels $\mathbf{x}_{i\_new}$. Furthermore, the image coordinates are not normalized i.e. not scaled by the inverse of their respective dimension. We found that normalizing the pixel coordinates "shrinks" the manifold search space, resulting in the need for very small α. The depth values however, are scaled such that 1 meter corresponds to a value of 1.0 [SEE+12]. The scaling value is usually given by the camera manufacturer.

Algorithm 5.2: BackprojectPixels

```
   Data: $\mathbf{x}, I^*, I^*_{\text{depth}}, I, I_{\text{depth}}, \text{MAXDEPTH}, \mathbf{K}^{-1}$
   Result: 3D points $\mathbf{X}$, where $\mathbf{X}_i \in SE(3)$. valid_measurement
 1 begin
 2     for $(x, y) \in [0, Width], [0, Height]$ do
 3         depth $\longleftarrow I_{\text{depth}}(y, x)$
 4         depth* $\longleftarrow I^*_{\text{depth}}(y, x)$
 5         valid_measurement$[x, y, \text{target}] \longleftarrow$ True
 6         valid_measurement$[x, y, \text{reference}] \longleftarrow$ True
 7         if $depth = 0$ then
 8             valid_measurement$[x, y, \text{target}] \longleftarrow$ False
 9             depth $\longleftarrow$ MAXDEPTH
10         end
11         if $depth^* = 0$ then
12             valid_measurement$[x, y, \text{reference}] \longleftarrow$ False
13             depth* $\longleftarrow$ MAXDEPTH
14         end
15         (u, v) $\longleftarrow \mathbf{x}_i$
16         $\mathbf{X}_i \longleftarrow \text{depth} \cdot \mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$
17     end
18 end
```

The BackprojectPixels() algorithm Section 5.2 is very straight forward. **K** is the calibration matrix, discussed in Section 3.1. The parameter MAXDEPTH can be any acceptable maximum value. In our case it was set to the maximum value of I^*_{depth}.

Algorithm 5.3 is the implementation of the direct motion estimation given in the previous section; Section 5. A non-trivial implementation detail is the early exit condition in line 5. Where possible, we avoid invalid depth measurements as these create spurious data. This is not true for Algorithm 5.4: ComputeResidual, since the residual is defined over the whole image space (5.1). Algorithm 5.4 is very straight forward. It is a simple accumulation of the residuals. Lines 8 and 9 are worth mentioning. Since the image project π returns floating point values, in order to access discrete pixels, we have to floor the values.

Algorithm 5.3: GaussNewtonStep

```
   Data: $\mathbf{J}_W, \mathbf{W}, \nabla I_{xy}, \mathbf{f}$, valid_measurements
   Result: $\Delta\xi \in \mathfrak{se}(3)$
1  begin
2      $\mathbf{N} \longleftarrow \mathbf{I}_6$
3      $\mathbf{g} \longleftarrow \mathbf{0}$
4      for $(x, y) \in [0, Width], [0, Height]$ do
5          if not valid_measurements[x, y, reference] or not
            valid_measurements[x, y, target] then
6              continue
7          end
8          $f_i \longleftarrow \mathbf{f}(x, y)$
9          $\mathbf{J}_I \longleftarrow \nabla I_{xy}(x, y)$
10         $\mathbf{J}_i \longleftarrow \mathbf{J}_I \mathbf{J}_W(x, y)$
11         $w_i \longleftarrow \mathbf{W}(x, y)$
12         $\mathbf{g}_{inc} \longleftarrow -w_i \mathbf{J}_i^T f_i$
13         $\mathbf{g} \longleftarrow \mathbf{g} + \mathbf{g}_{inc}$
14         $\mathbf{N} \longleftarrow \mathbf{N} + w_i(\mathbf{J}_i^T \mathbf{J}_i)$
15     end
16     $\Delta\xi \longleftarrow \mathbf{N}^{-1}\mathbf{g}$
17 end
```

Algorithm 5.4: ComputeResidual

```
   Data: $I, \mathbf{x}_{new}$
   Result: $\mathbf{f}$
1  begin
2      for $(x, y) \in [0, Width], [0, Height]$ do
3          $\mathbf{f}(x, y) \longleftarrow 0$
4          (u, v) $\longleftarrow \mathbf{x}_{new}$
5          x_target $\longleftarrow floor(u)$
6          y_target $\longleftarrow floor(v)$
7          $\mathbf{f}(x, y) \longleftarrow I^*$(y, x) $- I$(y_target, x_target)
8      end
9  end
```

5.2 Fusing Visual Odometry with a Motion Model

Steering commands from the Racecar's arduino control system were captured at a rate of 10 measurements per second. How to establish optimal correspondences between the image frames and the measured control sequences is part of control theory and outside the scope of this thesis. Kerl et al. used interpolation to establish these correspondences in the TUM dataset [SEE+12]; we however chose to use simple quantization. That is, each image frame is associated with its closest steering measurement with respect to time.

5.2.1 Motion Prior

The motion prior is defined as the covariance $\mathbf{\Sigma}$ of a normally distributed motion model. This prior can be included in the motion estimation with the following adjustment to the IRLS formulation [KSC13b]:

$$\Delta\xi = -(\mathbf{J}^T\mathbf{W}\mathbf{J} + \mathbf{\Sigma}^{-1})^{-1}\mathbf{J}^T\mathbf{W}f + \mathbf{\Sigma}^{-1}(\xi_{t-1} - \xi_t^k) \tag{5.15}$$

The value of ξ_t in (5.15) is the motion estimation of the t^{th} image frame sequence. Furthermore, since the covariance (Σ) is a measure of the uncertainty, Kerl et al. use its inverse in order to scale the lie algebra according to certainty [KSC13b]. The Algorithm 5.5 is very similar to the standard motion estimation Algorithm 5.1. The difference is that it now takes two more parameters $\Sigma_{\text{prior}}^{-1}$ as well as the lie algebra from the previous frame pair (ξ_{t-1}). The new Gauss-Newton can simply be extended using the new input parameters, Algorithm 5.6.

Algorithm 5.5: Direct Motion Estimation with Motion Prior

```
   Data: $I^*, I, \nabla I_{xy}$, $\mathbf{K}$, $\alpha$, $\mathbf{\Sigma}_{\text{prior}}^{-1}$ , $\xi_{t-1}$, EPS, MAX, MAXDEPTH
   Result: A pose $\mathbf{P} \in$ SE(3).
1  begin
2      ...
3      $\xi \longleftarrow (0,0,0,0,0,0)$
4      while $eps > EPS$ and $it < MAX$ do
5          $\Delta\xi \longleftarrow$
             GaussNewtonStep_Prior($\mathbf{J}_W, \mathbf{W}, \nabla I_{xy}, \mathbf{f}, \Sigma_{\text{prior}}^{-1}, \xi_{t-1}, \xi,$ valid_measurements)
6          $\Delta\xi \longleftarrow \alpha \cdot \Delta\xi$
7          $\xi \longleftarrow ln(exp(\xi) \boxplus \Delta\xi)$
8          ...
9      end
10 end
```

Algorithm 5.6: GaussNewtonStep_Prior

```
   Data: $\mathbf{J}_W, \mathbf{W}, \nabla I_{xy}, \mathbf{f}, \Sigma^{-1}_{\text{prior}}, \xi_{t-1}, \xi^k,$ valid_measurements
   Result: $\Delta\xi \in \mathfrak{se}(3)$
 1 begin
 2     $\mathbf{N} \longleftarrow \mathbf{I}_6$
 3     $\xi_{\text{delta}} \longleftarrow \Sigma^{-1}_{\text{prior}}(\xi_{t-1} - \xi^k)$
 4     $\mathbf{g} \longleftarrow \mathbf{0}$
 5     for $(x, y) \in [0, Width], [0, Height]$ do
 6         $\ldots$
 7     end
 8     $\mathbf{N} \longleftarrow \mathbf{N} + \Sigma^{-1}_{\text{prior}}$
 9     $\mathbf{g} \longleftarrow \mathbf{g} + \xi_{\text{delta}}$
10     $\Delta\xi \longleftarrow \mathbf{N}^{-1}\mathbf{g}$
11 end
```

5.2.2 Pose Information

Inspired by Kerl et al. [KSC13b] and their usage of motion priors, we defined our own augmentation which uses the covariance as well as the pose estimation. Since the Lie algebra is a vector space [FMB13] we define a correction vector, based on the Ackermann motion model and scaled by the inverse of the motion covariance; line 5 in Algorithm 5.7. As opposed to Kerl et al. we do not need to adapt the GaussNewtonStep() function but simply "post process" the twist update $\Delta\xi$. With this, we conclude the overview of the VO algorithm and will provide the results of our experiments in the next section.

Algorithm 5.7: Direct Motion Estimation with Motion Prior

```
   Data: $I^*, I, \nabla I_{xy}$, $\mathbf{K}$, $\alpha$, $\mathbf{\Sigma}^{-1}_{\text{model}}$, $\xi_{\text{model}}$, EPS, MAX, MAXDEPTH
   Result: A pose $\mathbf{P} \in SE(3)$.
 1 begin
 2     $\ldots$
 3     while $eps > EPS$ and $it < MAX$ do
 4         $\Delta\xi \longleftarrow$ GaussNewtonStep($\mathbf{J}_W, \mathbf{W}, \nabla I_{xy}, \mathbf{f},$ valid_measurements)
 5         $\xi_{\text{inc}} \longleftarrow \alpha\mathbf{\Sigma}^{-1}_{model}(\Delta\xi \boxminus \xi_{\text{model}})$
 6         $\Delta\xi \longleftarrow \alpha \cdot \Delta\xi$
 7         $\mathbf{P} \longleftarrow \mathbf{P} \boxplus (\Delta\xi + \xi_{\text{inc}})$
 8         $\ldots$
 9     end
10 end
```

CHAPTER 6

Evaluation

Since we based this thesis on the works of Kerl et al. [KSC13b], we will use the same evaluation metric as presented by Sturm et al. [SEE$^+$12]. The metric used will be the *Root Mean Square Error* (RMSE) of the *Relative Pose Error* (RPE). Given a sequence of groundtruth trajectories $\mathbf{Q}_1 \dots \mathbf{Q}_n \in \mathrm{SE}(3)$ and a sequence of estimated pose trajectories $\mathbf{P}_1 \dots \mathbf{P}_n \in \mathrm{SE}(3)$ the RPE (6.1) is defined as:

$$\mathbf{E}_i(\Delta) = (\mathbf{Q}_i \mathbf{Q}_{i+\Delta})^{-1}(\mathbf{P}_i \mathbf{P}_{i+\Delta}) \tag{6.1}$$

Where i and Δ in (6.1) denote the starting frame and frame difference respectively. When $\Delta = 1$ this represents the error per frame, for cameras which capture images at 30 fps, $\Delta = 30$ represents the error per second. In order to calculate the RMSE for all n images with respect to a Δ we apply the RPE over all $m = n - \Delta$ images:

$$\mathrm{RMSE}_i(\mathbf{E} : i : n) = (\frac{1}{m} \sum_{i=1}^{m} \|trans(\mathbf{E}_i, \Delta)\|^2) \tag{6.2}$$

The function *trans* in (6.2) only selects the translational components from the pose $\mathbf{P}$. Sturm et al. argue that sufficient rotational errors will influence the translational error and because of this, translational errors are a sufficient metric; a more detailed discussion is given in [SEE$^+$12]. Throughout the Evaluation section we will use the RMSE in the scale of meters per second (m/s) i.e. $\Delta = 30$. This is the same scaling as used by Kerl et al. as it results for a more robust error description [KSC13b].

6.1 TUM Dataset

For the purposes of evaluation we examined image sequences with a length ranging from $180 - 300$ images, at 30 fps this translates to $6 - 10$ seconds. The reason for this, is that the Python implementation runs very slowly and may take up to 12 hours for a sequence of length 300, depending on the convergence of the Gauss-Newton algorithm. Each image is identified via its frame timestamp. The main sequence used for evaluation is the *fr2/desk1* dataset. It has a low average camera velocity of 0.19 m/s [SEE$^+$12] and performed best in the paper of Kerl et al. [KSC13b]. From this dataset we selected three sequences, two of length 300 and one of length 180 images. We also use a 120 image sequence from the dataset *fr1/desk2*. This dataset exhibits less texture and a much higher average camera motion with 0.43 m/s [SEE$^+$12]; as a result more motion blur as well. We used this dataset to highlight the limitations of our VO algorithm.

6.1.1 FR2/Desk1

The *fr2/desk1* scene captured a camera moving around a well textured table in a circular trajectory, Figure 6.1. The camera motion is to the right around the table with deviations in the xyz axes. An example image frame with its corresponding depth map can be seen in Figure 6.2. Please note the density of the depth image, as the depth images from the TU Wien dataset, in Section 6.2, will be a lot sparser. The grey values in the inverted depth image, Figure 6.2b encode image depth i.e. darker values correspond to pixels farther away from the camera. However, values of 1 i.e. white encode measurements with no reading. This can happen for example, when the imaged surface is too far for the sensor, or if the sensor data does not return to its source as can happen with specular reflections.

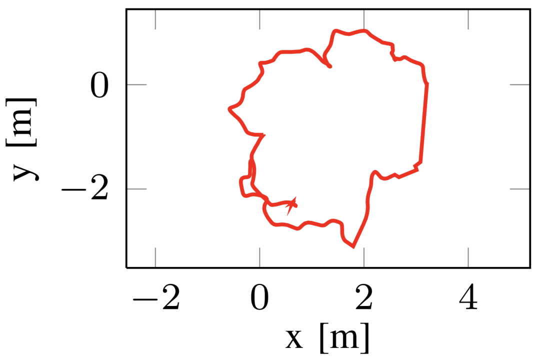
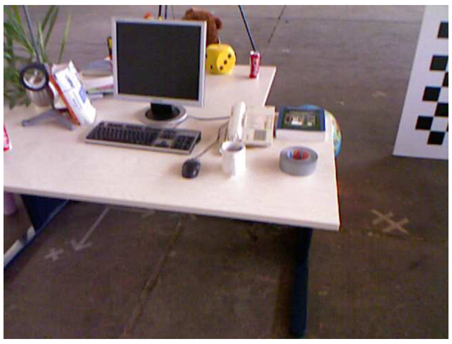


Figure 6.1: *fr2/desk1* trajectory in x and y [SEE$^+$12].

The image sequences, Figure 6.3 used for evaluation of the *f2d1* dataset are as follows:

- Sequence 1 with a length of 300 frames
- Sequence 2 with a length of 180 frames
- Sequence 3 with a length of 300 frames


(a) Intensity Image.

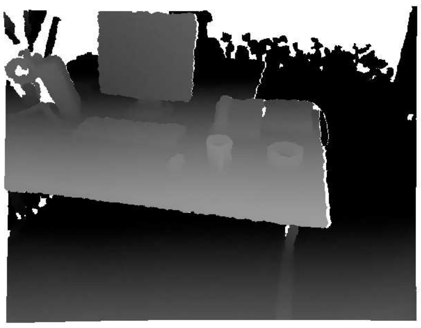

(b) Inverted Depth Image.

Figure 6.2: Intensity and depth frame pair.

(a) Sequence 1.

(b) Sequence 2.

(c) Sequence 3.

Figure 6.3: Starting intensity images of each sequence.

We begin by evaluating the overall trajectories produced by our VO algorithm for Figure 6.4. The first observation we can make is that the VO algorithm does produce a trajectory which is comparable to the (green) ground truth, labelled **gt**. However, the trajectory is not identical. We can observe that the axis with the greatest error is also the axis with the largest motion range. That is, for Figure 6.4, the x-axis, with a motion range of ≈ 1.4 meters. Therefore, we believe, the error is due to the lack of an image pyramid in our implementation, as stated in Section 1.1. Image pyramids are known to increase the estimated motion range [TAC08, NNB04] and due to our omission we estimate a coarser trajectory.

Next, we examine the two image filters presented in Section 3.7, i.e., Sobel and Scharr. As image gradients are a core component of the Gauss-Newton step, see Section 5.1, and the paper, we are comparing our work to [KSC13b], did not mention which filter they used, it is worth investigating which filter is the best performing one. We evaluated each filter on the three selected datasets, Figure 6.4.

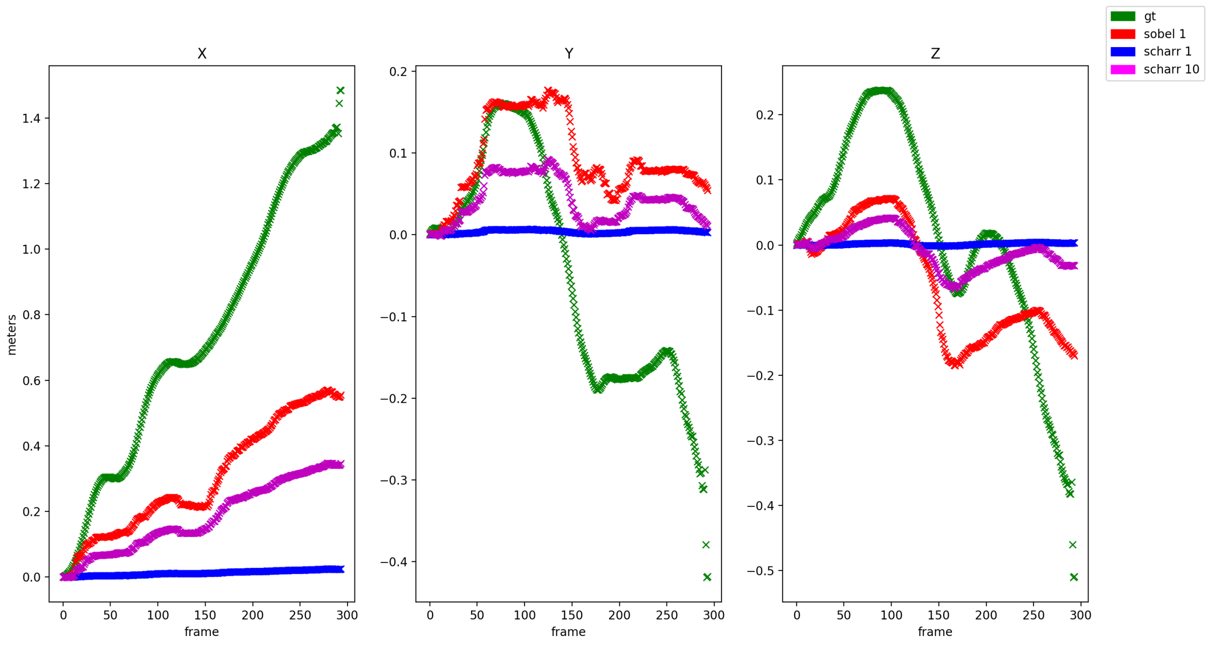
Focusing on the various filters used, In Figure 6.4a, we can see that the Scharr filter needs a gradient step size which is 10× larger than the step size of the Sobel three filter to be comparable. The reason for this could be that the values of the Scharr filter are up to 10× larger than the Sobel filter, see Section 3.7. Therefore, to induce the same motion the gradient step size has to be proportional to the filter size. Looking that all sequences we can see that the Sobel filter with a gradient step size of 1 performs consistently better than an equivalent Scharr filter of gradient step size 10. The specific RMSE values of the sequences is given in Table 6.1. The values show that the Sobel filter is ≈ 3 cm/s more accurate than the Scharr filter. However, both filters depend on the gradient step size parameter. As such, is is entirely possible that with a different selection of the gradient step size the Scharr filter will perform better. However, since the standard step size is 1 [MNT04] we chose the Sobel filter variant for the other TUM experiments.

	Sequence 1	Sequence 2	Sequence 3
Sobel	0.143 m/s	0.229 m/s	0.168 m/s
Scharr	0.172 m/s	0.251 m/s	0.196 m/s

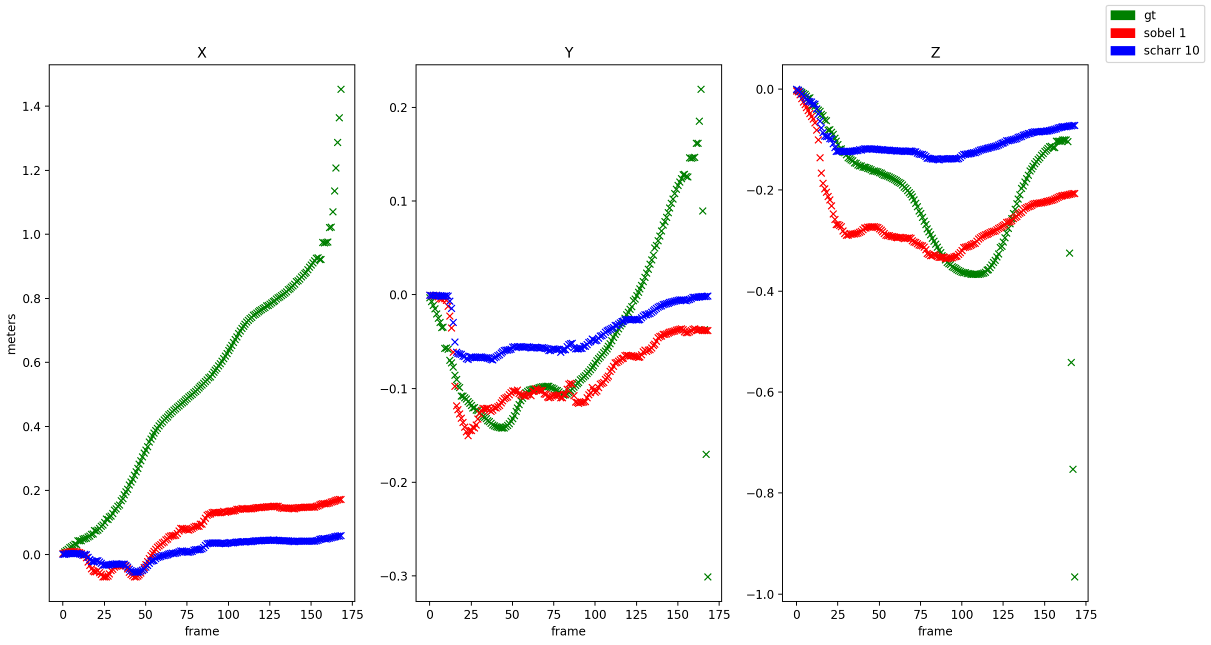
Table 6.1: RMSE(m/s) for filter experiments to 3 decimal places.

The next experiment we performed was investigating the effect of the epsilon/convergence bound for the weighting scheme on Sequence 2, Figure 6.5. Looking at Figure 6.5a, we can see that epsilon bounds ≥ 0.001 in general produce similar results. Lower bounds than this i.e. eps $= 0.1$, exhibit noisy behaviour as shown in the y and z axis, Figure 6.5a. We presume this noisy behaviour is due to the weighting scheme converging to soon and therefore, allowing noisy results to be used. Interestingly, the RMSE does not reflect this, as the error value for eps $= 0.1$ is the lowest. This can be explained by the x-axis compensating for the errors in y and z. Since we want our weighting scheme to produce a smooth trajectory, we chose eps $= 0.001$ for the remaining TUM experiments. Furthermore, examining Figure 6.5b we can see that without using a weighting scheme to detect outliers the trajectory estimated by the VO algorithm is very noisy, with an RMSE of 1.345 m/s, Table 6.2. This is expected behaviour [KSC13b] and indicates that our weighting scheme is working.

We also examined the influence of the motion prior as given by [KSC13b]. The formulation for this can be found in Section 5.2.1. As seen in the plots of the trajectory, Figure 6.6

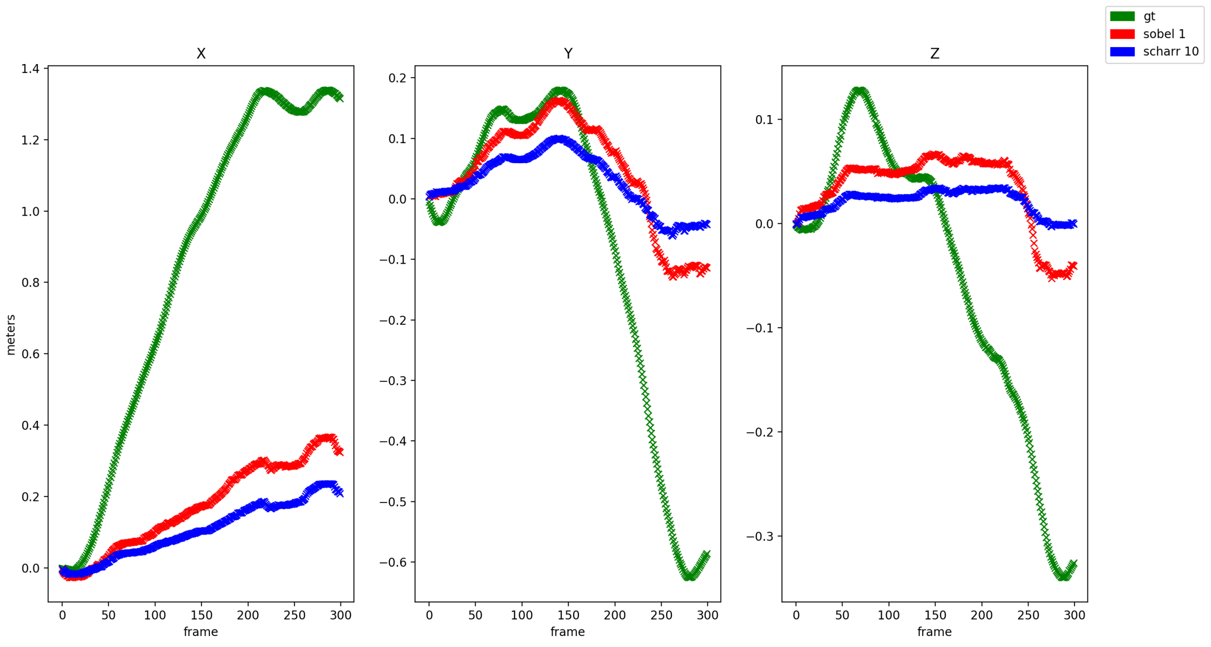


(a) Sequence 1.



(b) Sequence 2.

Figure 6.4: Starting intensity images of each sequence. The number after the image filter indicates the gradient step size. Ground truth (green), Sobel 1 (red), Scharr 1 (blue) and Scharr 10 (pink)



(c) Sequence 3.

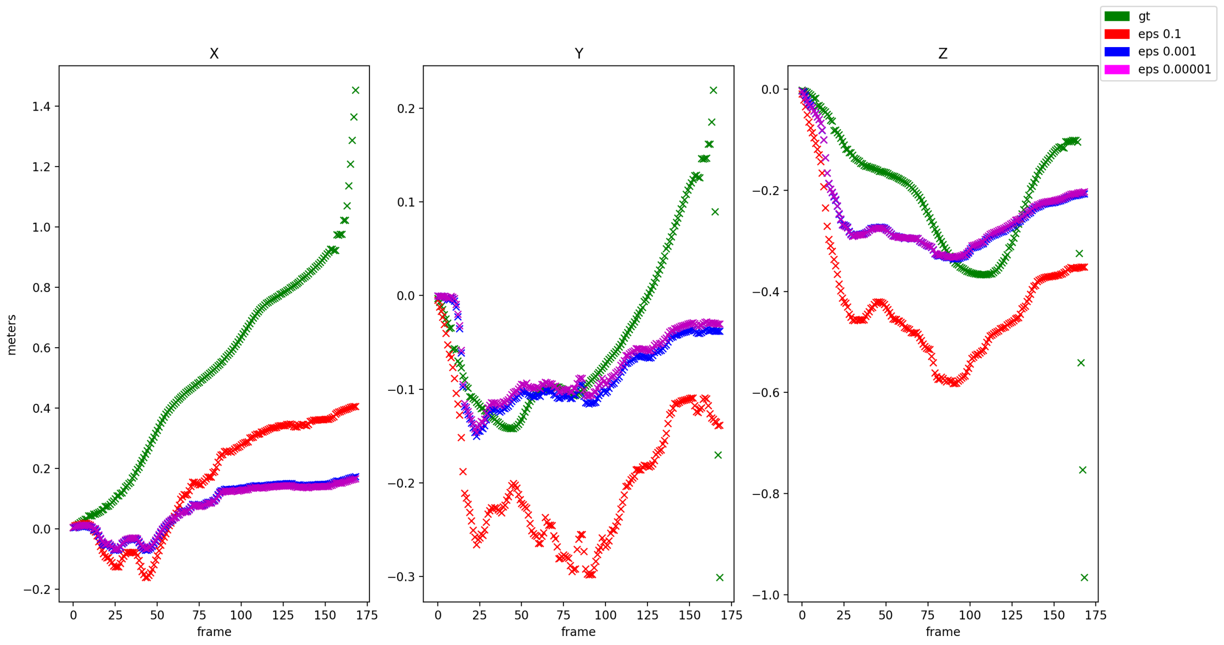
Figure 6.4: Starting intensity images of each sequence. The number after the image filter indicates the gradient step size. Ground truth (green), Sobel 1 (red), Scharr 1 (blue) and Scharr 10 (pink)

	Sequence 2
No Weighting	1.354 m/s
Eps = 0.1	0.223 m/s
Eps = 0.001	0.229 m/s
Eps = 0.00001	0.231 m/s

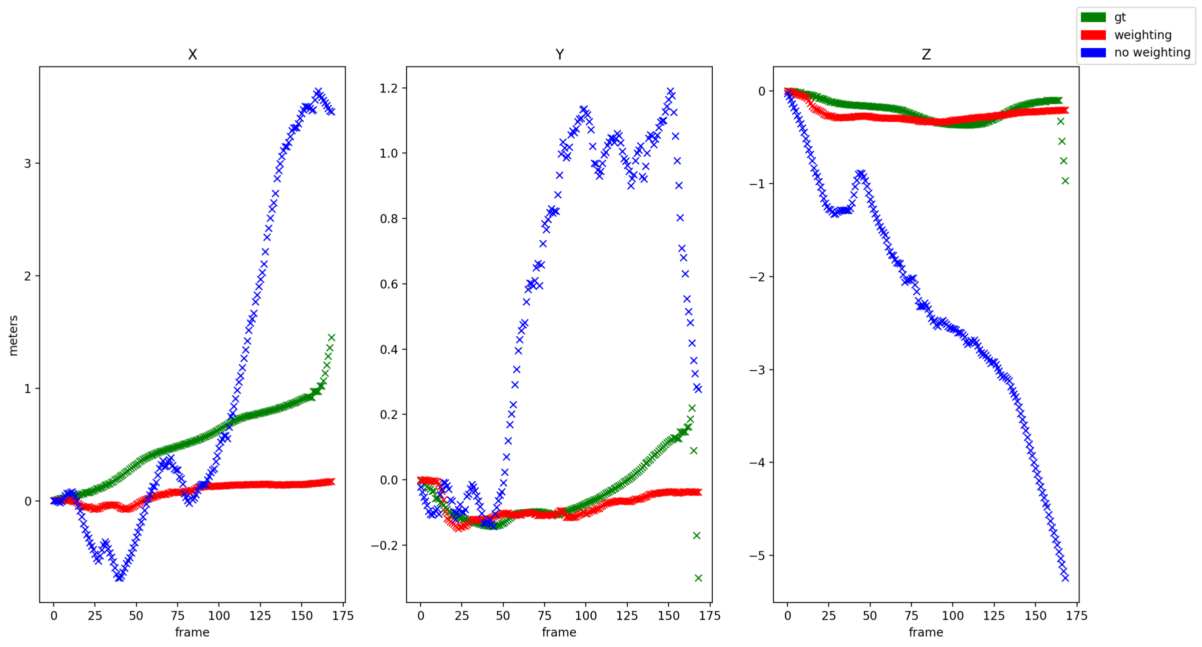
Table 6.2: RMSE(m/s) for weighting experiments to 3 decimal places.

and the RMSE values, Table 6.3, the improvement by the motion prior is the millimeter and sub millimeter level. This is similar to the results by Kerl et al. [KSC13b] and shows that the system is working as expected.

Finally, we want to compare the results of our VO algorithm to those of Kerl et al. [KSC13b]. First, we want to clarify that the results of Kerl et al. are for the whole image sequence which spans several minutes, while our VO results were restricted to 10 second sequences of the larger dataset due to runtime constraints. However, the results are consistent for all of our sequences. The VO of Kerl et al. is more accurate by

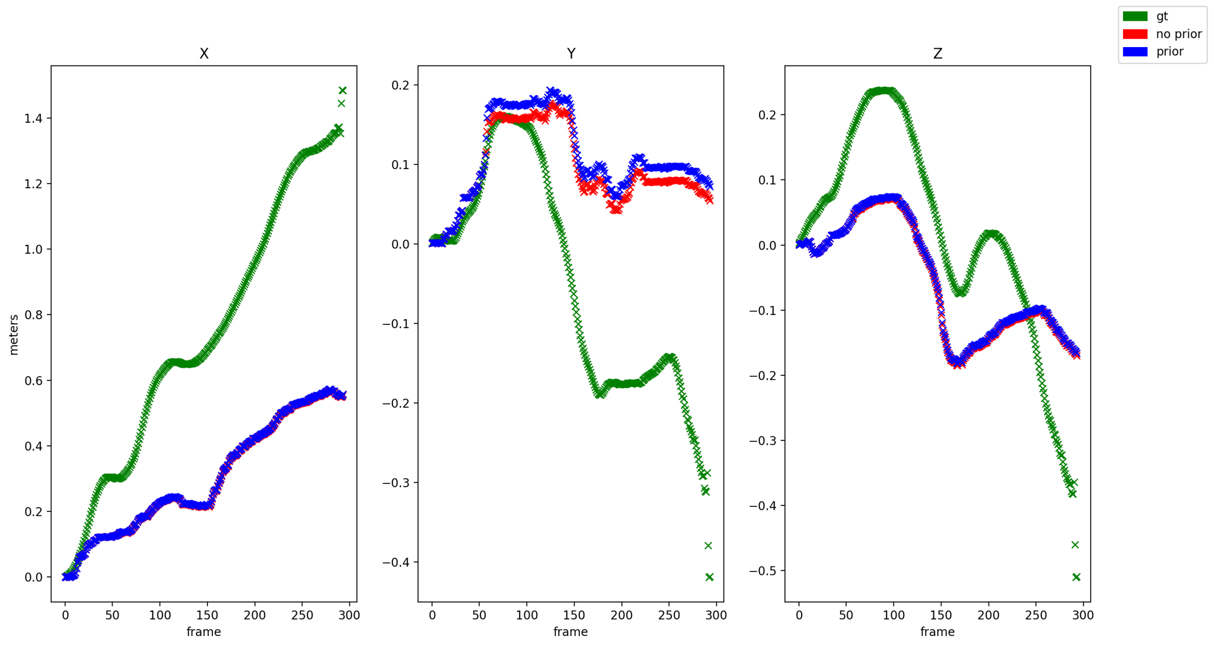


(a) Sequence 2 - Epsilon.

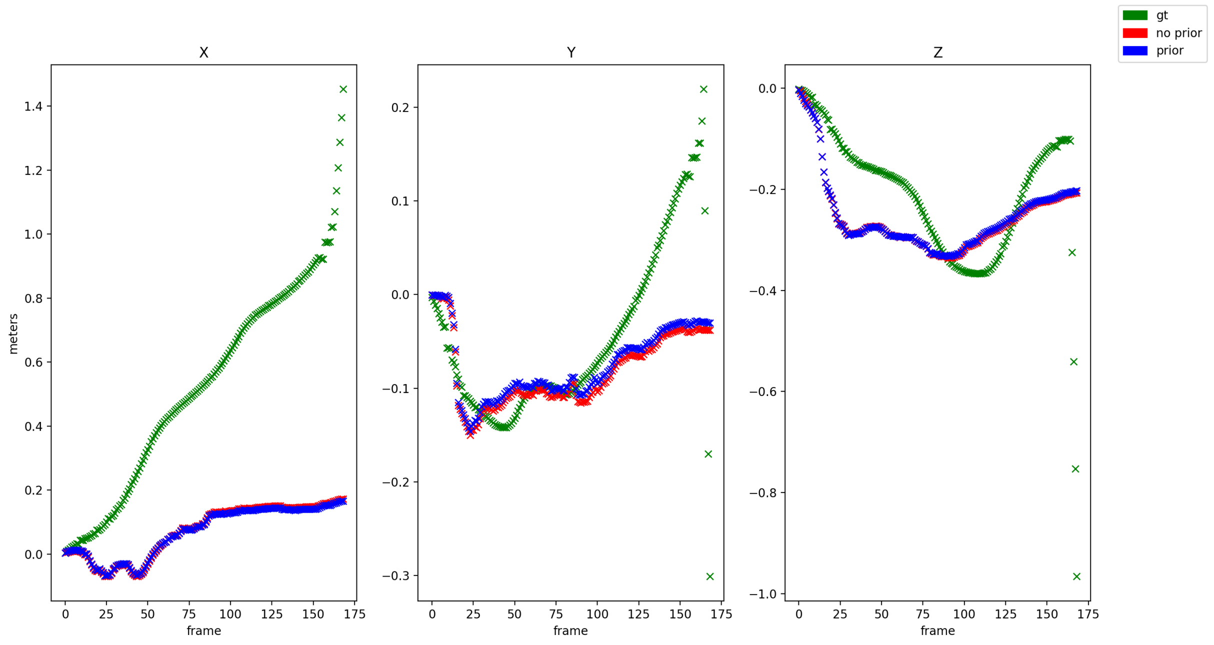


(b) Sequence 2 - Weighting.

Figure 6.5: Left, epsilon bounds. Right, general impact of the weighting scheme. Ground truth (green), epsilon 0.1 (red), epsilon 0.001 (blue) and epsilon 0.00001 (pink).

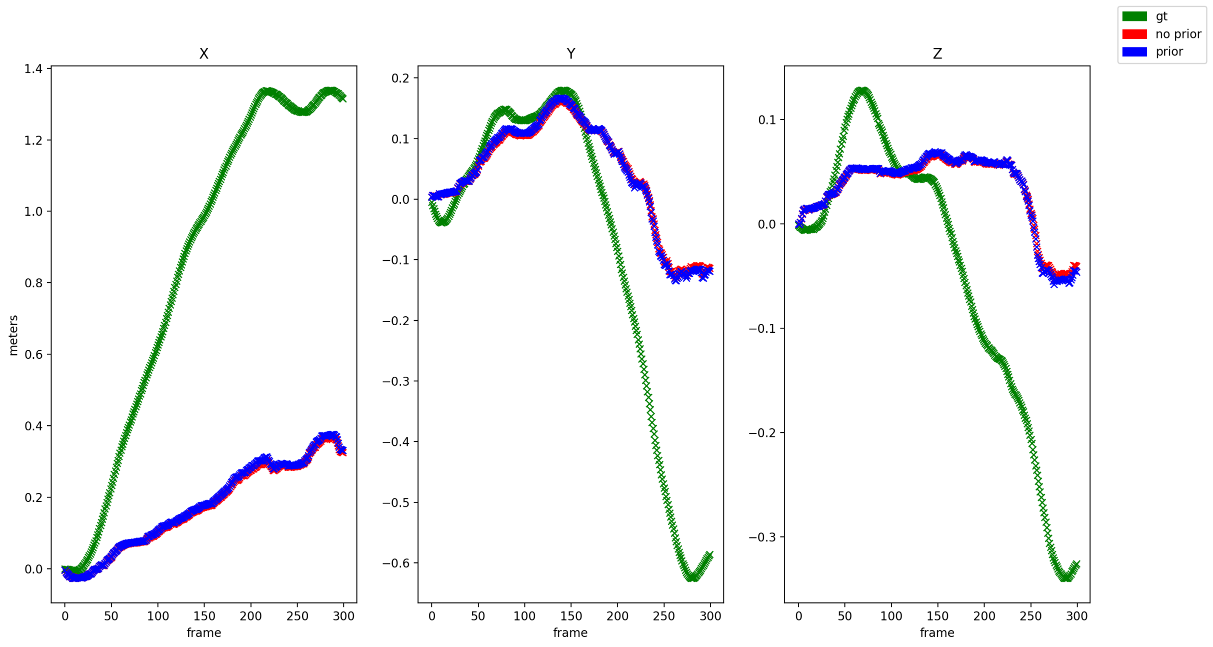


(a) Sequence 1.



(b) Sequence 2.

Figure 6.6: Influence of motion prior on VO. Ground truth (green), no prior (red) and with prior (blue).



(c) Sequence 3.

Figure 6.6: Influence of motion prior on VO. Ground truth (green), no prior (red) and with prior (blue).

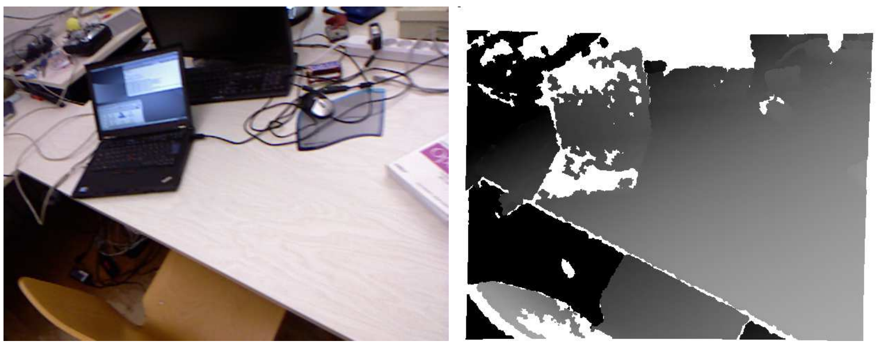
	Sequence 1	Sequence 2	Sequence 3	Kerl et al. [KSC13b]
No Prior	0.1434 m/s	0.2290 m/s	0.1684 m/s	0.020 m/s
Prior	0.1432 m/s	0.2308 m/s	0.1668 m/s	0.019 m/s

Table 6.3: RMSE(m/s) for motion prior experiments to 4 decimal places.

a factor of $5-10\times$. The main reason for this, we believe, is due to our implementation lacking an image pyramid scheme, as mentioned when examining Figure 6.4.

6.1.2 FR1/Desk2

We have also examined another sequence of the TUM dataset, Figure 6.7. Specifically this sequence has a much higher average camera velocity with 0.43 m/s, as opposite to with 0.19 m/s of the previous sequence. We examined this sequence in particular to highlight the limitations of our VO algorithm. Due to the lacking image pyramid and the higher camera velocity we fail to produce any consistent trajectory, Figure 6.8. Furthermore, we can see the limitations of the motion prior formulation as well. Due to the faulty trajectory estimate, the motion prior gets skewed towards the noise levels as



(a) Intensity Image. (b) Inverted Depth Image.

Figure 6.7: Intensity and inverted depth frame pair.

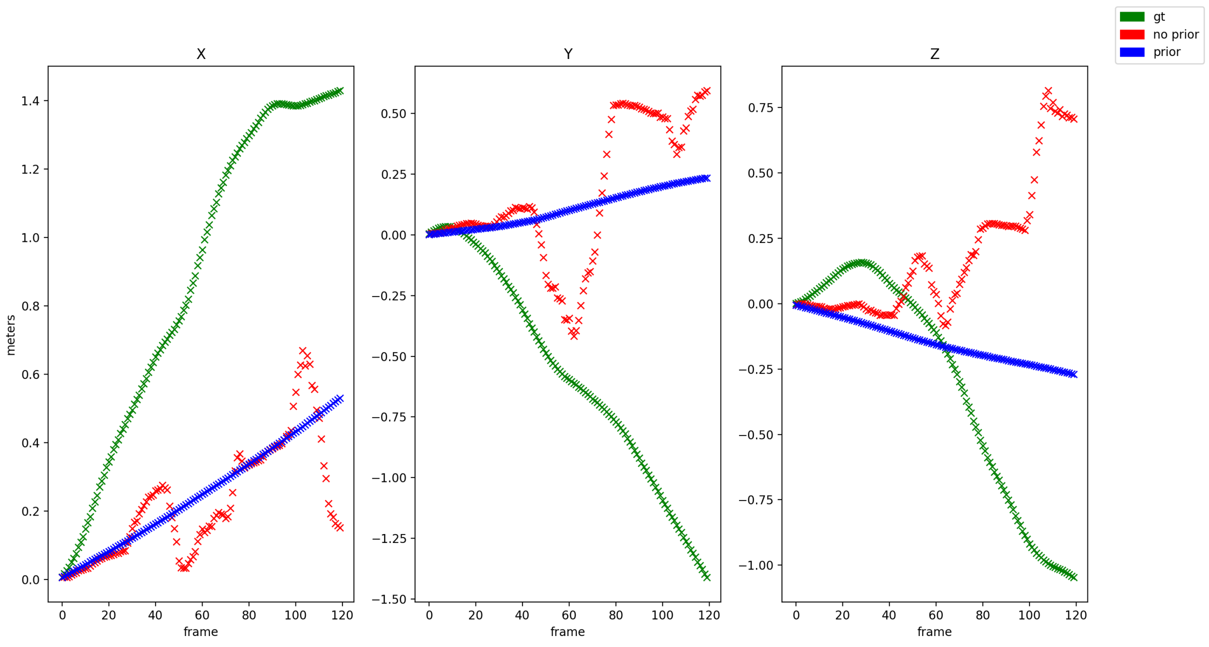


Figure 6.8: VO for a large average camera velocity of 0.43 m/s. Ground truth (green), no prior (red) and with prior (blue).

can be seen in the y-axis plot, Figure 6.8.

6.2 TUW Dataset

For the TUW dataset recorded at TU Wien we examined 3 image sequences with 300 frames at 30 fps each. The test environment was a large room with some furniture i.e.

	Sequence	Kerl et al. [KSC13b]
No Prior	0.865 m/s	0.046 m/s
Prior	0.606 m/s	0.049 m/s

Table 6.4: RMSE(m/s) for 6.8 to 3 decimal places.

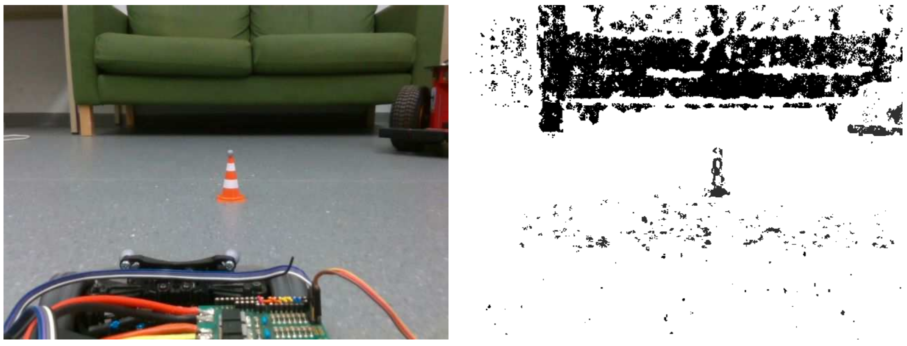

(a) Intensity Image. (b) Inverted Depth Image.

Figure 6.9: Intensity and depth frame pair.

chairs, tables cupboard and a sofa. The racecar's movement was restricted to driving along the room's floor at $\approx 0.01 - 0.1$ m/s and as such elevations of the y-axis were not considered; ideal conditions of the Ackermann motion model. Each image sequence exhibits a different motion i.e. forward-backward, forward-right, forward-left. Specifically we have the following sequences:

- Sequence 1 with a length of 300 frames and forwards-backwards motion.
- Sequence 2 with a length of 300 frames and forward-right motion.
- Sequence 3 with a length of 300 frames and forward-left motion.

An example output can be seen in Figure 6.9. We also want to draw attention to the depth image, Figure 6.9b. When compared to the TUM dataset, Figure 6.2, we can see that the racecar's depth image is a lot sparser. Even though the camera was mounted at an angle tilting downwards, the ZR300 depth camera seems to have had problems with the floor. One reason for this could be that the floor is made up of a glossy material. As such, specularities could have affected the depth measurements. Furthermore, the depth image does not seem to be aligned with its intensity image. This could be due to a timing difference between intensity and depth image acquisition, which was not compensated for. These are both important limitations of the acquired data and are reflected in the VO output.

	Sequence 1	Sequence 2	Sequence 3
Sobel 1	0.177 m/s	0.198 m/s	0.059 m/s
Scharr 10	0.173 m/s	0.167 m/s	0.0594 m/s

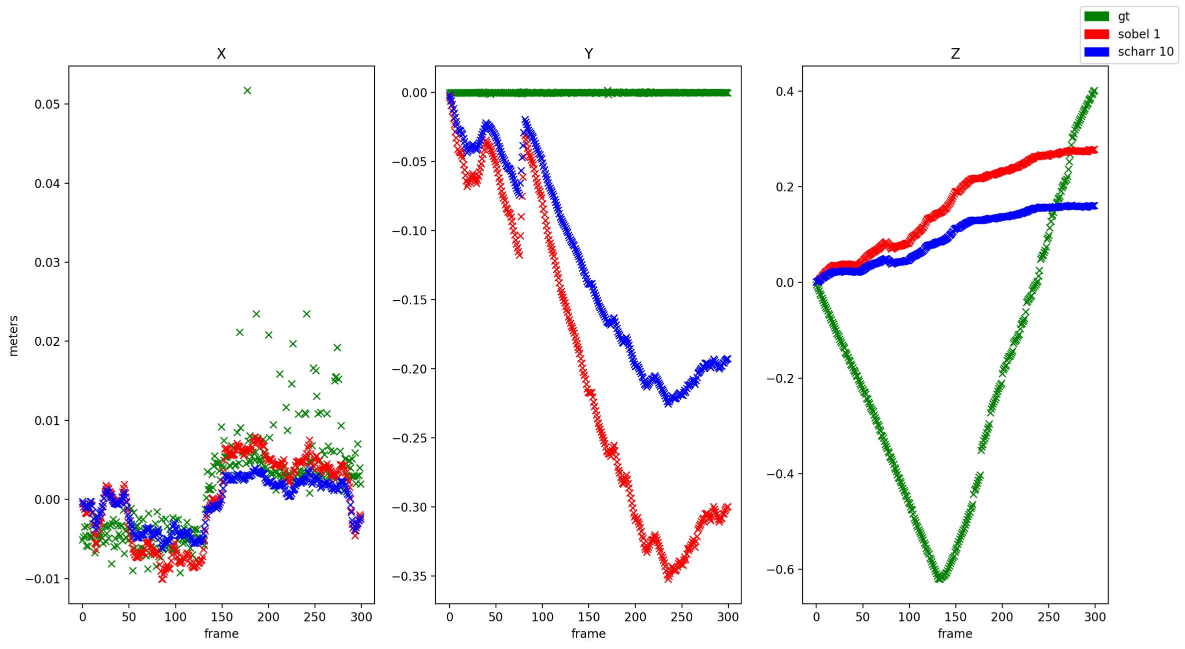
Table 6.5: RMSE(m/s) of 6.10 to 3 decimal places.

Similar to the TUM dataset we will evaluate the performance of the Scharr and Sobel filters with a gradient step size of 10 and 1 respectively. In contrast to the TUM datasets, when examining the RMSE values, Table 6.5, we observe that the Scharr filter seems to be better in this case. This might be due to the larger image kernel being able to, to some degree, counteract the sparse measurements. However, we want to re-iterate that the filter performance is tied to the gradient step size. As such, an optimal filter does not seem to exists, as both filters exhibits similar trajectories, Figure 6.10. Nevertheless, based on these measurements we chose to use the Scharr filter for the TUW dataset.

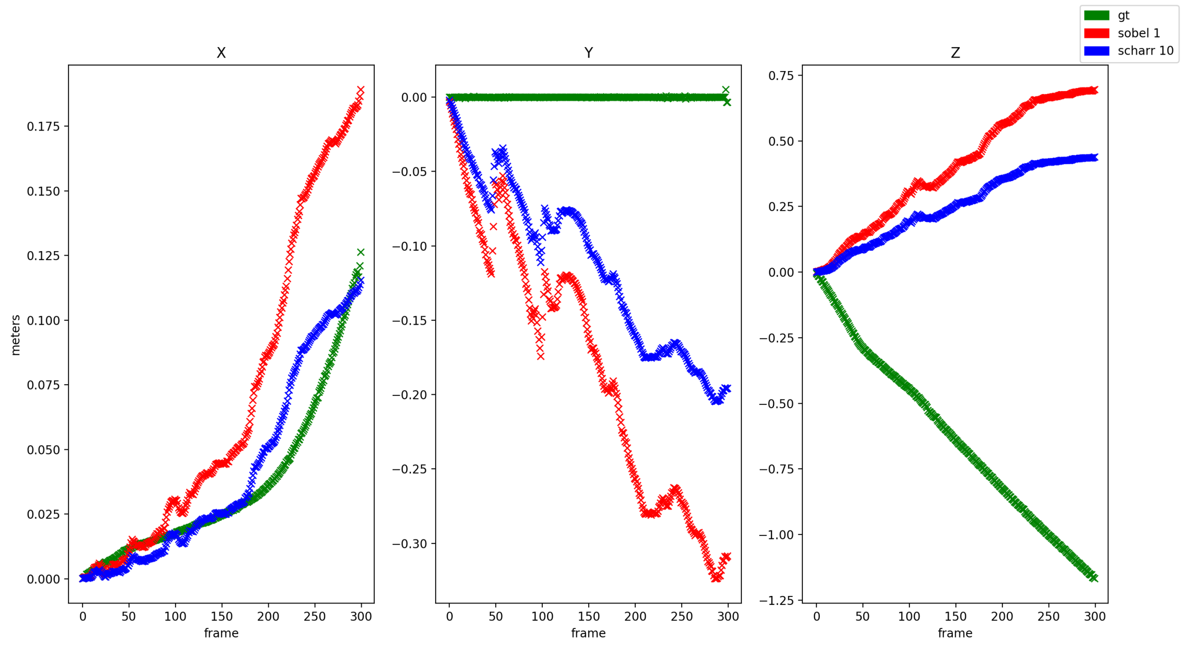
In general, the trajectories estimated for the TUW dataset are a lot worse than for the TUM dataset. The main reasons for this are the sparse measurements, Figure 6.9, and the constant presence of the racecar's front side, Figure 6.9a. This results in large parts of the image not to be used effectively; a more detailed discussion on this will be given below. Nevertheless, there are interesting observations we can make. First, we can observe that in each sequence at least one axis is tracked successfully, Figure 6.10. Furthermore, in sequences 1 and 2, Figure 6.10a, Figure 6.10b, the tracked x-axis is very close to the ground truth. We believe this is because valid data is available for a full cross section spanning the x-axis, Figure 6.9b. On the other hand, most of the bottom half of the depth images do not contain valid measurements. This could explain the discrepancies in the y-and z-axis, specifically in sequences 1 and 2 Figure 6.10a and Figure 6.10b respectively. We believe this might be due to the incomplete optical flow induced by the image sequence.

The VO trajectory estimation is based on the compositional image alignment, which induces a type of optical flow field [Sca11]. An example illustration of a simple 2D optical flow field is given in Figure 6.11. In that image the focus of expansion is located on the lower left. We can observe that a 2D image motion along the z-axis induces an optical flow in both x and y. Although the VO algorithm computes a full six degrees of freedom trajectory the image gradients encode this type of information. Removing half of the optical flow field below the focus of expansion yields full information in x, but only one way in y.

To verify this hypothesis we used the dense optical flow algorithm of Gunner [Gun03], which is implemented in *OpenCV*, as a comparison. This optical flow is only calculated between two images, i.e. no depth values are used. However, the limitations of Gunnar correspond to the limitations of our depth based VO, albeit for different reasons, i.e., less texture vs invalid depth measurements. Therefore, we believe it is a fair comparison. We can see the RGB image and the corresponding optical flow field in Figure 6.12a and

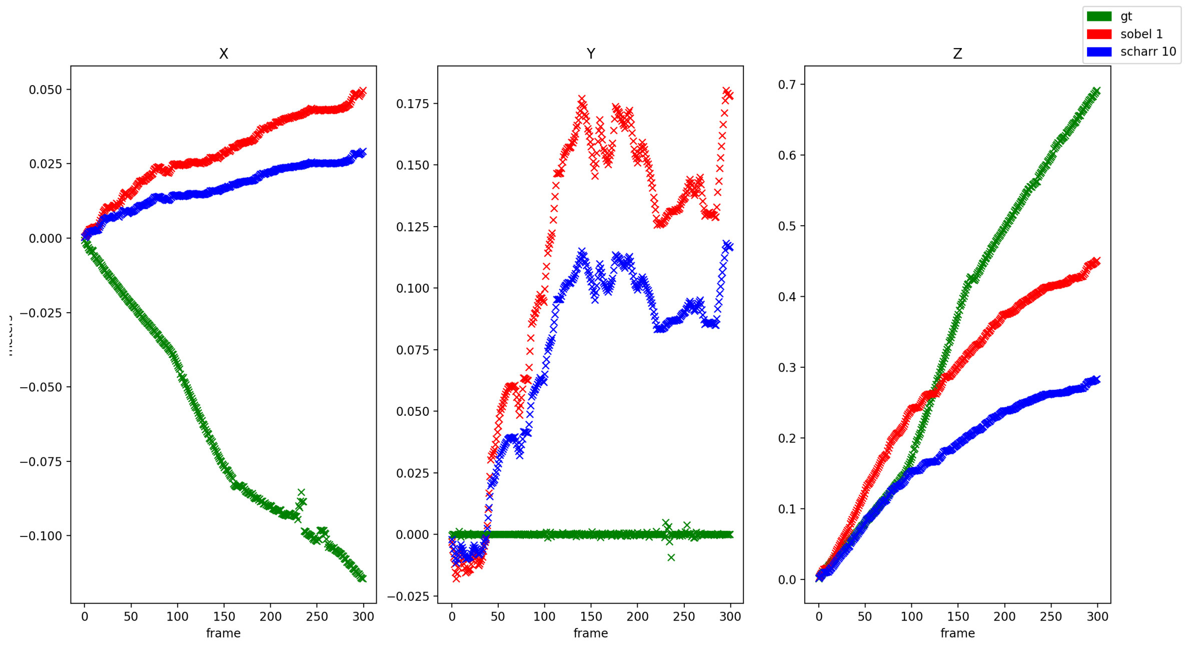


(a) Sequence 1.



(b) Sequence 2.

Figure 6.10: Sobel and Scharr filters for the TUW image sequences. Ground truth (green), Sobel 1 (red) and Scharr 10 (blue).



(c) Sequence 3.

Figure 6.10: Sobel and Scharr filters for the TUW image sequences. Ground truth (green), Sobel 1 (red) and Scharr 10 (blue).

Figure 6.12b, respectively. The color encoding of Figure 6.12b is based on the HSV color scheme. The color scheme is shown in Figure 6.12c. The indexing is equivalent to a unit circle moving counter-clockwise. That is, a direction along the x-axis corresponds to an angle of 0°. However, due to the "inverted" indexing of matrices/images along the y-axis, a green value represents an optical flow from the top to the bottom of an image.

Examining Figure 6.12b, we can observe that we have a full range of values along the left and right sides of the image, corresponding to movement along the x-axis. When looking at the top half of the image we see that the motion is correctly induces as "downwards" as well. The bottom of the image, however, is very sparse and noisy. Some patches, that correspond to "upwards" movement exist, namely the center of the image, which corresponds to the cone in Figure 6.12a. However, the majority of that image section seems to be invalid, due to the lack of texture. Moreover, since the racecar's front section is always present, see Figure 6.12a and Figure 6.13a, it obstructs the optical flow algorithm to induce apparent motion in that area as well.

Therefore, the lack of valid pixels at the bottom of the image is, we believe, why the VO estimates motion in the y axis in sequences 1 and 2, Figure 6.10a, Figure 6.10b respectively, are inducing a negative motion along the y-axis. A similar conclusion can be reached for Sequence 3, Figure 6.10c. Since the racecar is moving "backwards" i.e.

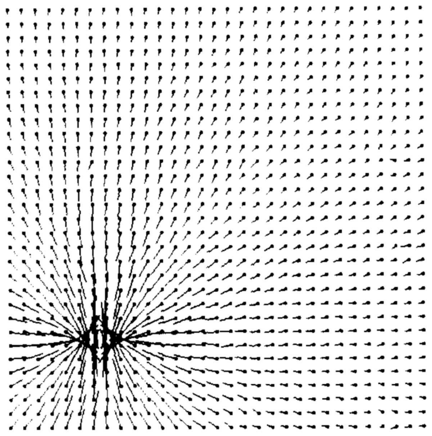


Figure 6.11: Optical flow field illustration for motion along the z axis [HS80].

	Sequence 1	Sequence 2	Sequence 3
No Prior	0.172483 m/s	0.166755 m/s	0.059444 m/s
Prior	0.172484 m/s	0.166736 m/s	0.059400 m/s

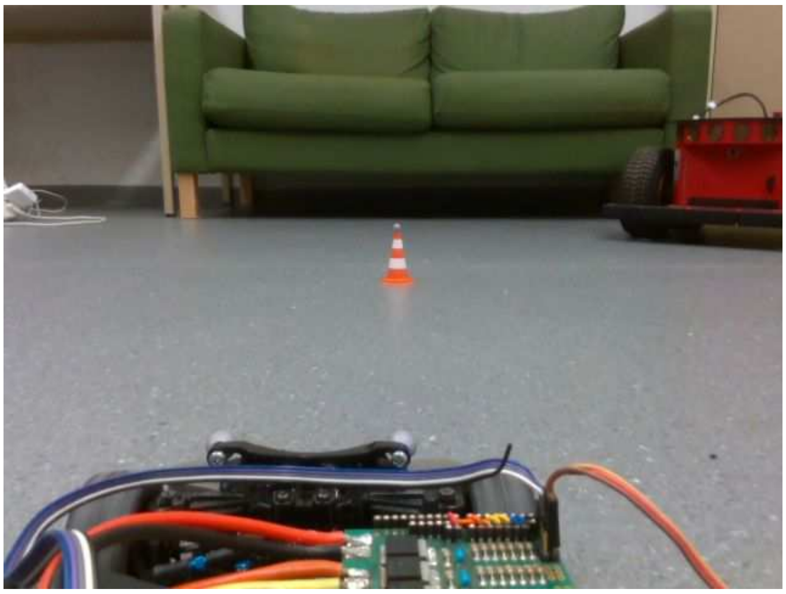
Table 6.6: RMSE(m/s) of 8.1 to 6 decimal places.

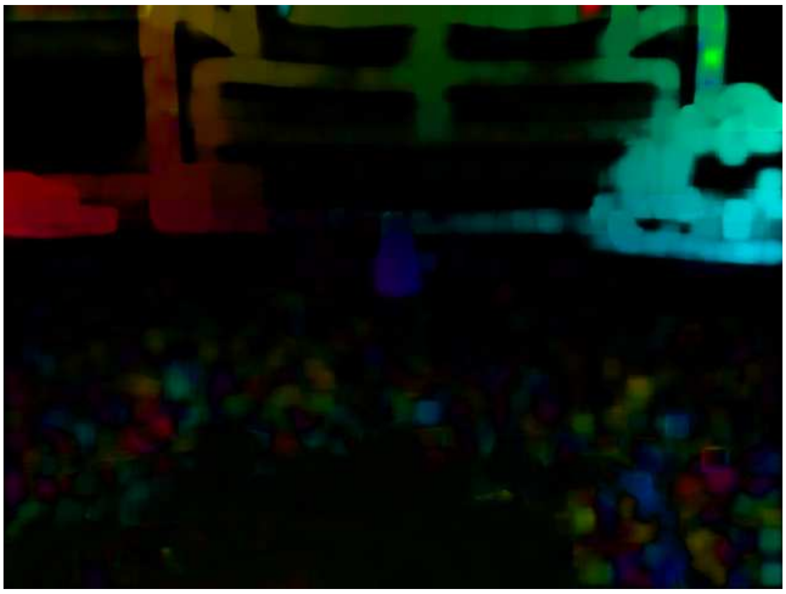
out of the image along the positive z-axis, the motion along the y-axis is "upwards". Furthermore, the motion along the x-axis is not recoverable, see Figure 6.13b, which is why we believe we are seeing the faulty VO estimate in the x-axis of Figure 6.10c.

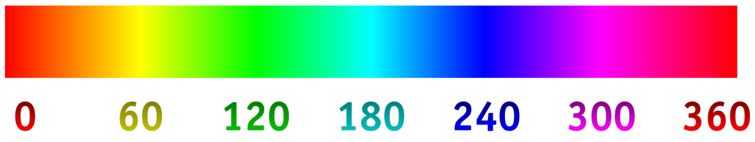
Concerning the motion prior formulation on this dataset, the effects on these sequences are very minor, Table 6.6. In sequences 1 and 2 the trajectory without the prior is only marginally better. However the differences in all three are in the sub millimeter category. The full trajectory plots are given in the Appendix Section 8.2. More interesting results, Figure 6.14, are obtained when looking at the VO data using the full Ackerman motion model as we defined in Section 5.2.2.

Regarding the Ackermann moiton model, when observing the z-axes, in Figure 6.14 that in general, a larger influence of the Ackermann motion model is able to "push" the VO trajectory into its direction. Examining the RSME, Table 6.7, we can see that the best results were achieved for the Ackermann motion model only. This might beg the question, why we are using VO in the first place. To this, we would like to reiterate that the camera input for the VO is very sparse. Furthermore, the conditions while recording the dataset were ideal for the Ackermann motion model. That is, a flat, even surface which the Ackermann motion model can approximate very well.

Nevertheless, we are still able to witness certain scenarios where the Ackermann motion model fails and the VO performs better, even when this is not the case overall. The first instance when this happens is in the first 100 frames of Figure 6.14b for the x-axis. The

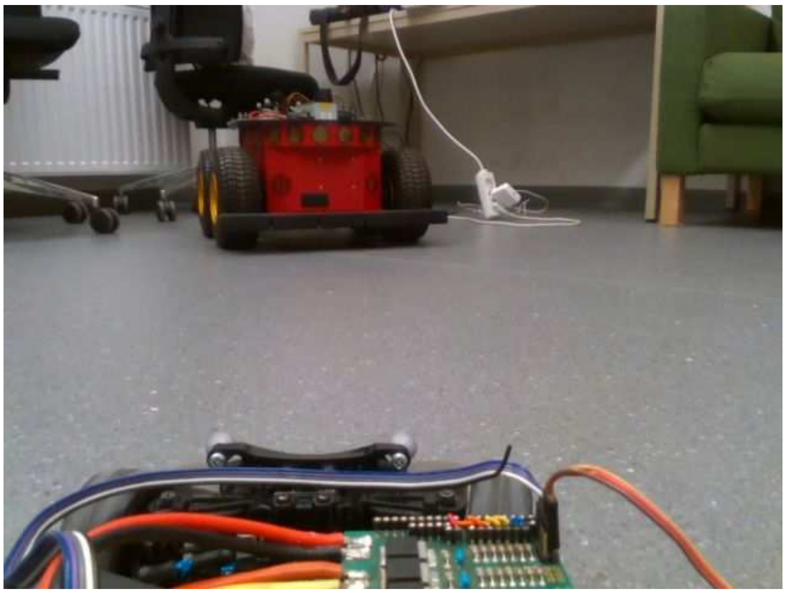

(a) Image used for optical flow calculation. Motion is along the negative z-axis.


(b) Corresponding optical flow field for motion along the z axis.

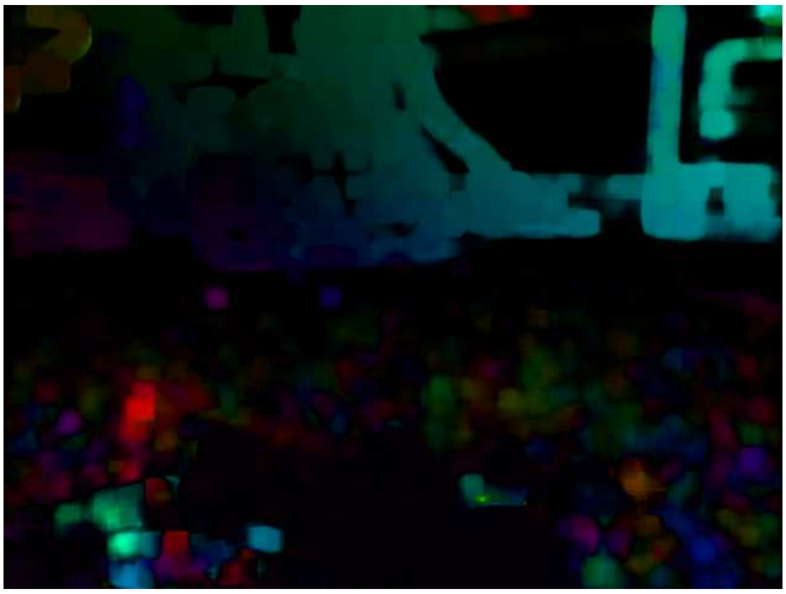


(c) Orientation encoding, in degrees, of the HSV color space.

Figure 6.12: Sequence 1.



(a) Image used for optical flow calculation. Motion is along the positive z-axis.



(b) Corresponding optical flow field for motion along the z axis.

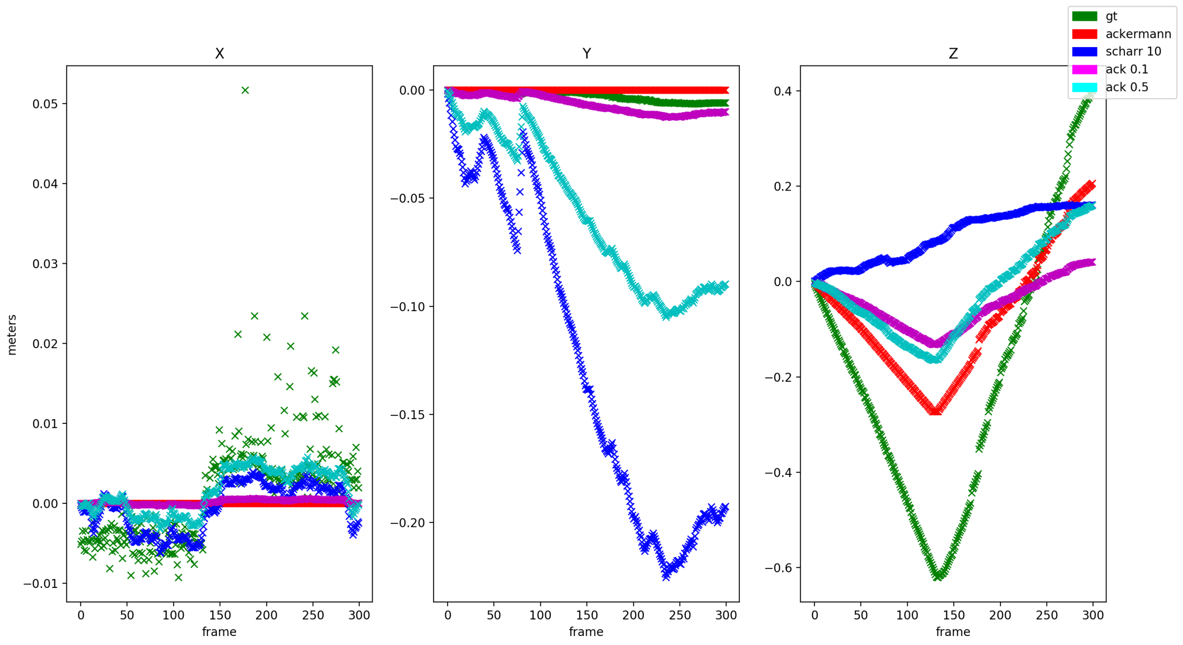
Figure 6.13: Sequence 3.

	Sequence 1	Sequence 2	Sequence 3
VO	0.172 m/s	0.167m/s	0.059m/s
Ackermann Only	0.094 m/s	0.071 m/s	0.042 m/s
Ackermann 0.1	0.140 m/s	0.122 m/s	0.054 m/s
Ackermann 0.5	0.122 m/s	0.119 m/s	0.044 m/s

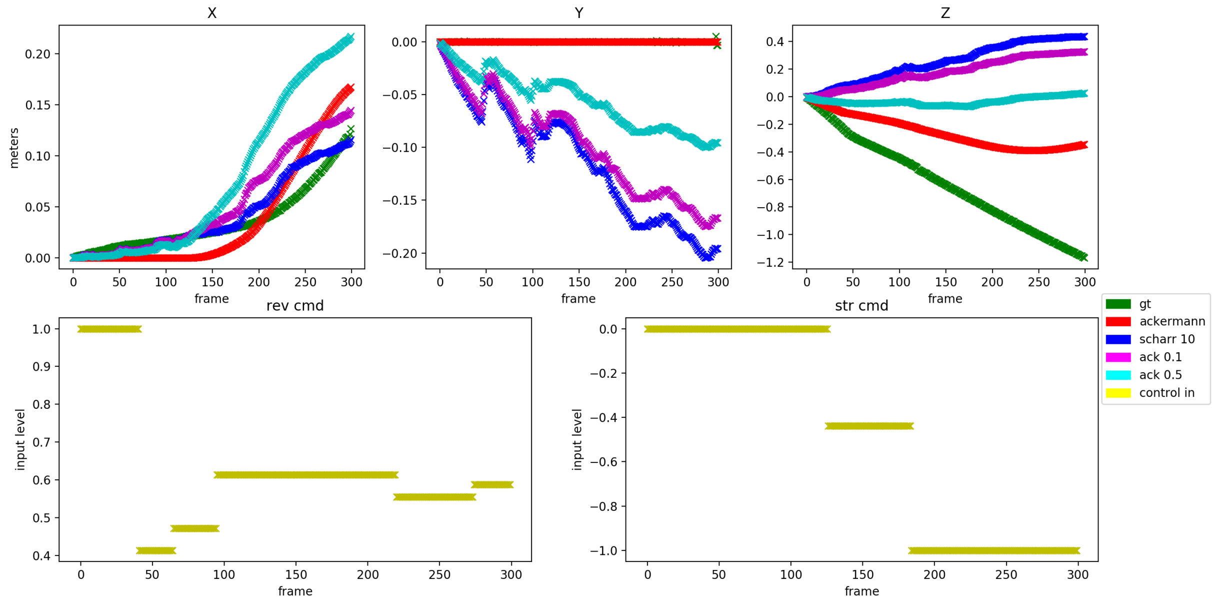
Table 6.7: RMSE(m/s) of Figure 6.14 to 3 decimal places.

selected interval is displayed in Figure 6.15. We can observe that although no steering input is present the racecar moves slightly to the right. This is not captured by the motion model, however the VO is able to correctly estimate the trajectory. We observe a similar situation in dataset 3, Figure 6.14c, where the racecar is changing its trajectory in the x-axis, but the motion model does not register this. However, the VO in this case also fails to produce a proper trajectory. Interestingly, it is able to correctly estimate the z-axis for that dataset. This shows that with an incomplete input the VO can at least estimate one trajectory correctly.

The reasons for these discrepancies is, we believe, due to gamepad delay, network congestion and/or imperfect servo mechanics. There seems to be a delay between the control signal being sent form the gamepad, and when it is executed by the racecar. This was not taken into account for, for this experiment, as it was only discovered after evaluating the data. The network congestion is an consequence of the capturing a lot of image data, with better network performance/lower data load this can be mitigated. The failure of the servos results in the wheels not returning to their exact original position i.e. 0°. As such, this is not a contrived failure of the experiment, but may also occur in other scenarios.

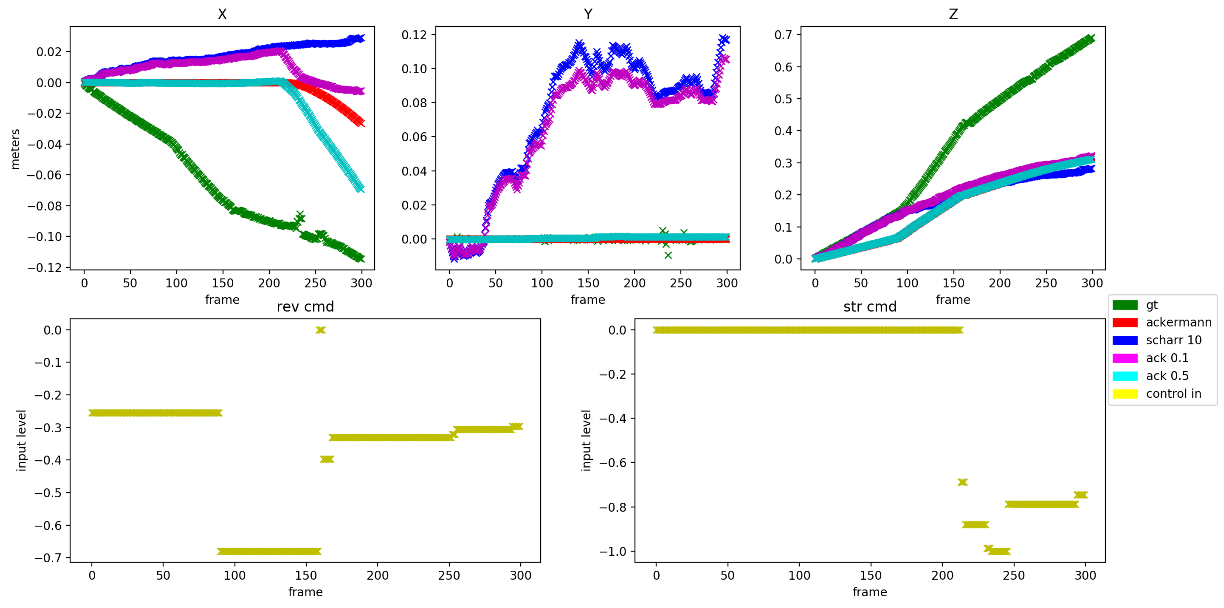


(a) Sequence 1.



(b) Sequence 2.

Figure 6.14: Using the full Ackermann motion model for the TUW image sequences. Ground truth (green), only Ackermann (red), schar 10 (blue), 0.1 Ackermann with VO, 0.5 Ackermann with VO, control inputs (yellow).



(c) Sequence 3.

Figure 6.14: Using the full Ackermann motion model for the TUW image sequences. Ground truth (green), only Ackermann (red), schar 10 (blue), 0.1 Ackermann with VO, 0.5 Ackermann with VO, control inputs (yellow).

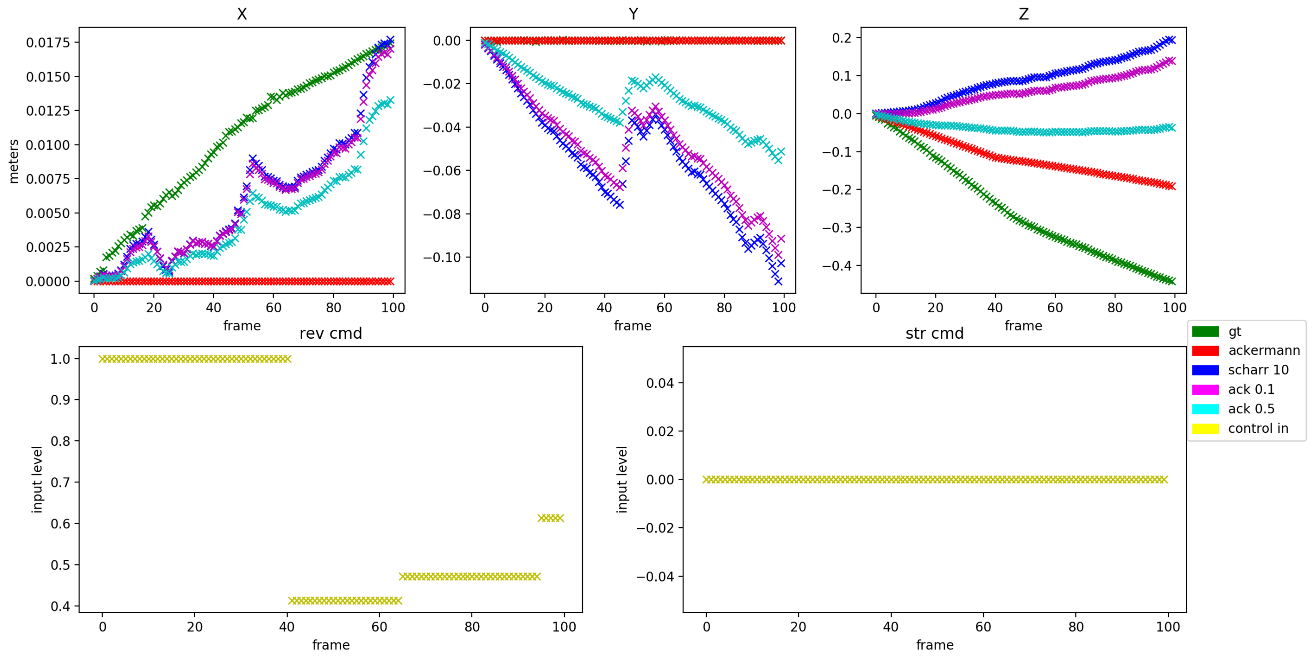


Figure 6.15: Sequence 2 first 100 frames. Ground truth (green), only Ackermann (red), schar 10 (blue), 0.1 Ackermann with VO, 0.5 Ackermann with VO, control inputs (yellow).

CHAPTER 7

Conclusion and Future Work

This thesis has built a VO system from the ground up, by deriving the lie algebra based Gauss-Newton IRLS from first principles, see Section 3. Furthermore, we have given pseudo-code which not only puts the theoretical IRLS into practical terms, but also examined pitfalls, which we have faced during the course of this thesis. These pitfalls include how to treat invalid measurements, how the generators of the lie algebra influence the VO's coordiante system and also what parameter spaces to normalize and how to do so. Although our RMSE was larger than the works of Kerl et al. [KSC13b] by a factor of $\times 5$, we have successfully estimated coarse trajectories of a camera input sequence. We believe the overall difference between the two VO trajectories is due to our lack of an image pyramid, which is known to increase the trajectory convergence range [NNB04, CMR07].

Furthermore, this thesis has also investigated a rudimentary fusion of the Ackermann motion model and the VO estimation algorithm. Currently we are not aware of any research examining the fusion of an independently generated motion model with an estimated VO trajectory. We have shown that the application of a motion model is able to compensate for a faulty VO trajectory if the motion model is closer to the ground truth than the VO. In our case we reduced RMSE from 0.17 m/s to around 0.12 m/s on the relevant axis.

Concerning future work, there are a lot of topics which can lead to an improvement of the VO estimation and topics which can use the VO algorithm as a supporting concept. First and foremost any further work should strive for a realtime implementation either using CPU based SIMD instructions [KSC13b], or GPU kernels. The Python implementation we developed is not able to support larger datasets or very tight error bounds i.e. $\leq 10^{-7}$ without unfeasible runtime of over 12 hours.

Once this is achieved further optimizations can be made, for example, an image pyramid or more advanced estimation techniques i.e. Leuvenberg-Marquart. The Leuvenberg-

Marquart technique is a non-linear least squares algorithm which is able to re-evaluate the gradient descent parameter at runtime, and therefore also achieves a better convergence rate. It has been successfully implemented in various SLAM systems [ESC14, MAMT15]. Furthermore, a more accurate modelling of the sensor noise and better sensors, in general e.g. wheel odometry can be used to further improve the accuracy of the motion model.

Finally, there are also topics which go beyond optimizing the algorithm established in this thesis. VO can be used as the local estimation component of larger localization systems such as SLAM [EHE^{+}12, LeM^{+}17]. The VO presented here is already used in state of the art direct SLAM systems [ESC14]. Furthermore, instead of depth images, the presented VO can be adapted to use other forms of sensor fusion. Current state of the art research uses IMU measurements instead of depth images. As we have seen in this work, depth images might be very sparse, even in controlled environments and IMUs have proved to be a very effective substitute [FCDS17]. Finally, further research can be done on the control aspect of the work presented here. The motion model is combined with the VO using a fixed ratio. Presumably better results could be achieved if the motion model is used when the VO produces faulty estimates and vice-versa.

CHAPTER 8

Appendix

8.1 Exponential and Logarithmic Maps

This section gives the closed form solutions for the exponential and logarithmic maps.

8.1.1 Exponential Map

$$\mathrm{P} = exp(\mathbf{v}) = \begin{bmatrix} \mathbf{R} & \mathbf{Vu} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{8.1}$$

$$\Omega = \left[\boldsymbol{\omega}\right]_\times = \left[\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}\right]_\times = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \tag{8.2}$$

$$|\boldsymbol{\omega}|^2 = \boldsymbol{\omega}^T\boldsymbol{\omega} = \theta^2 \tag{8.3}$$

$$A = \frac{\sin(\theta)}{\theta} \tag{8.4}$$

$$B = \frac{1-\cos(\theta)}{\theta^2} \tag{8.5}$$

$$C = \frac{1-A}{\theta^2} \tag{8.6}$$

$$\mathbf{R} = exp(\Omega) \equiv e^{\Omega} = \begin{cases} \mathbf{I}_3, & \text{if } |\boldsymbol{\omega}| = 0 \\ \mathbf{I}_3 + A\Omega + B\Omega^2, & \text{otherwise} \end{cases} \tag{8.7}$$

$$\mathbf{V} = \begin{cases} \mathbf{I}_3, & \text{if } |\boldsymbol{\omega}| = 0 \\ \mathbf{I}_3 + B\Omega + C\Omega^2, & \text{otherwise} \end{cases} \tag{8.8}$$

8.1.2 Logarithmic Map

$$\theta = \cos^{-1}\left(\frac{tr(\mathbf{R}) - 1}{2}\right) \tag{8.9}$$

$$\boldsymbol{\omega} = log(\mathbf{R}) = \frac{\theta}{2\sin(\theta}\left(\mathbf{R} - \mathbf{R^T}\right) \tag{8.10}$$

$$\mathbf{u} = log(\mathbf{Vt}) = \mathbf{V}^{-1}\mathbf{t} \tag{8.11}$$

$$\mathbf{V}^{-1} = \mathbf{I}_3 - \frac{1}{2}\Omega + \frac{1}{\theta^2}\left(1 - \frac{A}{2B}\right)\Omega^2 \tag{8.12}$$

8.1.3 Generators and Jacobians

The lie algebra of the group manifold is a vector base. As such, any element of the lie algebra can be decomposed into a linear combination of independent elements [FMB13]. These independent elements are termed *Generators*. The number of Generators directly correspond to the number of degrees of freedom, i.e., six in this case.

Each Generator can be derived by differentiating the group i.e. P ∈ SE(3) around the origin.

$$\mathrm{G}_1 = \frac{\delta \mathrm{P}}{\delta x'} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.13}$$

$$\mathrm{G}_2 = \frac{\delta \mathrm{P}}{\delta y'} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.14}$$

$$\mathrm{G}_3 = \frac{\delta \mathrm{P}}{\delta z'} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.15}$$

$$G_4 = \frac{\delta P}{\delta \omega_1}\bigg|_{\omega_1=0} = \frac{\mathbf{R}_x}{\delta\alpha}\bigg|_{\alpha=0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.16}$$

$$G_5 = \frac{\delta P}{\delta \omega_2}\bigg|_{\omega_2=0} = \frac{\mathbf{R}_y}{\delta\alpha}\bigg|_{\alpha=0} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.17}$$

$$G_6 = \frac{\delta P}{\delta \omega_3}\bigg|_{\omega_3=0} = \frac{\mathbf{R}_z}{\delta\alpha}\bigg|_{\alpha=0} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.18}$$

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.19}$$

$$\mathbf{R}_y = \begin{bmatrix} cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.20}$$

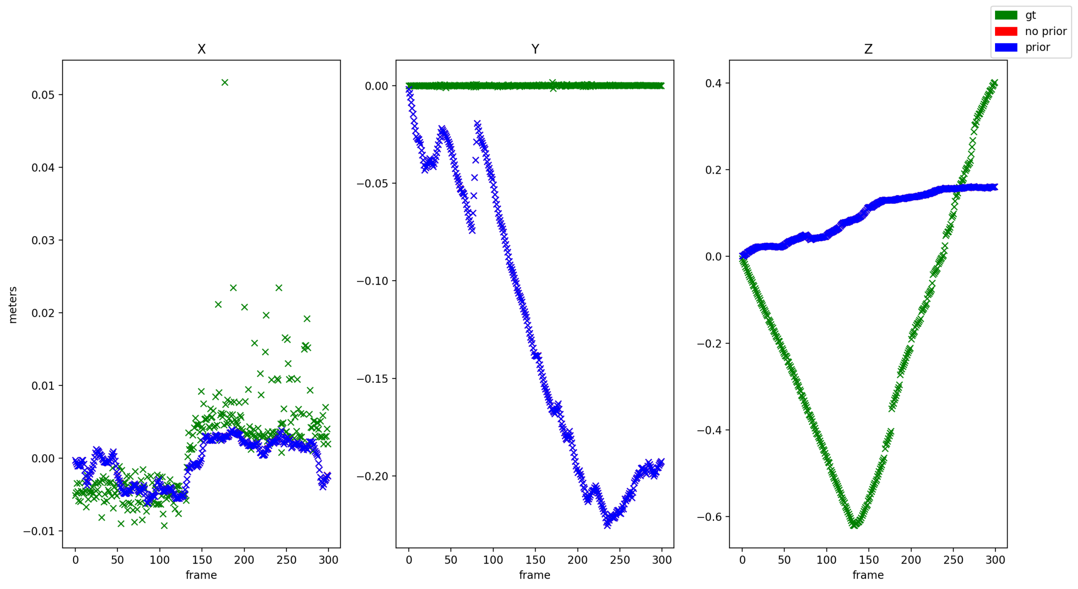
$$\mathbf{R}_z = \begin{bmatrix} cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.21}$$

$\mathbf{v} \in \mathfrak{se}(3) \equiv \mathbf{v}_1 G_1 + \mathbf{v}_2 G_2 + \mathbf{v}_3 G_3 + \mathbf{v}_4 G_4 + \mathbf{v}_5 G_5 + \mathbf{v}_6 G_6$

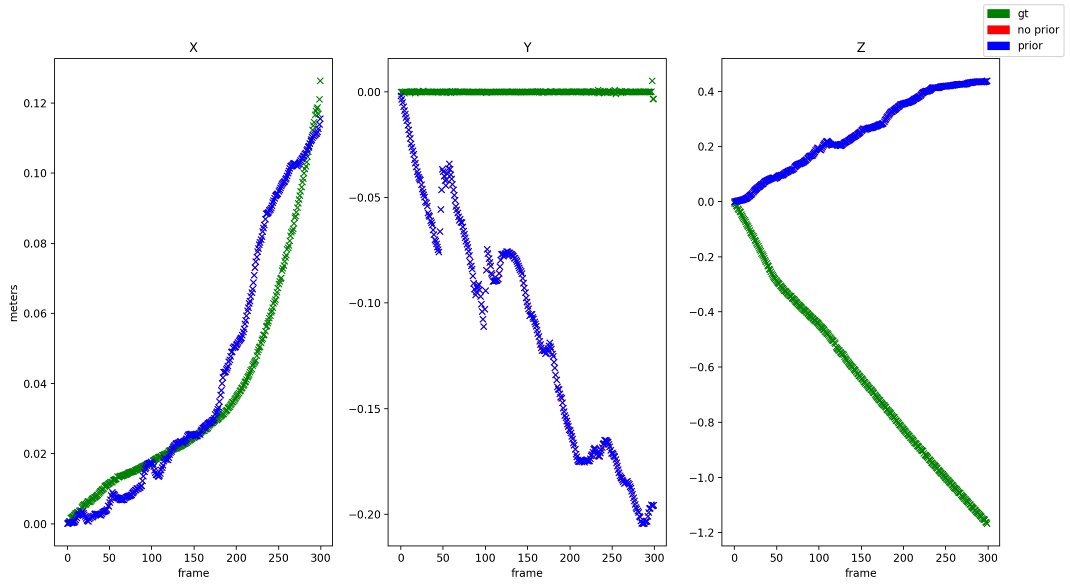
Given a point $\mathbf{x}$ that is transformed by a pose P i.e. $\mathbf{y} = P\mathbf{x}$, according to Blanco et al. [FMB13] we can define the Jacobian of $\mathbf{y}$ with respect to the transformation P as a Jacobian with respect to the lie algebra $\mathbf{v}$ around the origin.

$$\frac{\delta \mathbf{y}}{\delta \mathbf{v}} = \frac{\delta exp(\mathbf{v})\mathbf{y}}{\delta \mathbf{v}}\bigg|_{\mathbf{v}=0} = \Big(G_1\mathbf{y}|G_2\mathbf{y}|G_3\mathbf{y}|G_4\mathbf{y}|G_5\mathbf{y}|G_6\mathbf{y}\Big) = \Big(\mathbf{I}_3| - \big[\mathbf{y}\big]_\times\Big) \tag{8.22}$$

8.2 Plots

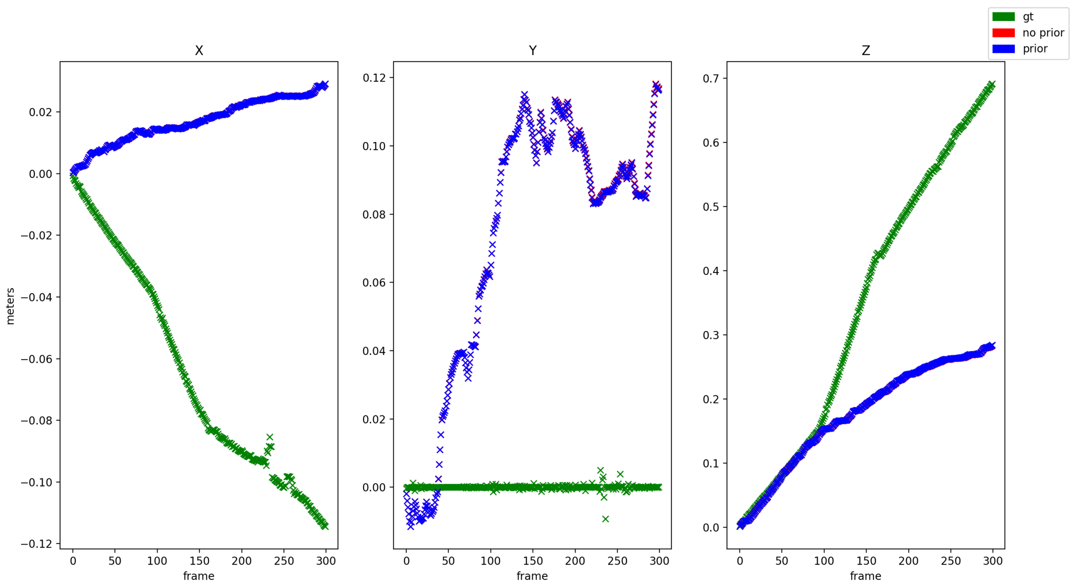


(a) Sequence 1



(b) Sequence 2

Figure 8.1: Motion prior for the TUW image sequences. Ground truth (green), no prior (red), prior (blue).



(c) Sequence 3

Figure 8.1: Motion prior for the TUW image sequences. Ground truth (green), no prior (red), prior (blue).

List of Figures

2.1 SLAM System Overview [EHE+12]. 5
2.2 IMU VO Integration [FCDS17]. 6

3.1 Perspective/Pinhole camera model[HZ04]. 10
3.2 A representation of the group manifold (sphere) and its tangent space(plane)[HWFS11]. 13
3.3 Ackermann Steering [WM10]. 19
3.4 Visualizations of image filters for changes in x mapped to 16 bit greyscale. Color space is inverted. 22

4.1 Image of the Intel Realsense ZR300 [Int19]. 23
4.2 Diagram of the Intel Realsense ZR300's components [Int19]. 24
4.3 Checker board pattern used for camera calibration. Corners used for backprojection are highlighted. 24
4.4 OptiTrack Prime 13 Camera [Opt19]. 25
4.5 TU Wien Racecar. 26
4.6 TU Wien Racecar Control Hierarchy [KBB16]. 26
4.7 TU Wien Racecar Message Passing [KBB16]. 27

5.1 Coordiante System induced by our VO algorithm. 32

6.1 *fr2/desk1* trajectory in x and y [SEE+12]. 40
6.2 Intensity and depth frame pair. 41
6.3 Starting intensity images of each sequence. 41
6.4 Starting intensity images of each sequence. The number after the image filter indicates the gradient step size. Ground truth (green), Sobel 1 (red), Scharr 1 (blue) and Scharr 10 (pink) 43
6.4 Starting intensity images of each sequence. The number after the image filter indicates the gradient step size. Ground truth (green), Sobel 1 (red), Scharr 1 (blue) and Scharr 10 (pink) 44
6.5 Left, epsilon bounds. Right, general impact of the weighting scheme. Ground truth (green), epsilon 0.1 (red), epsilon 0.001 (blue) and epsilon 0.00001 (pink). 45
6.6 Influence of motion prior on VO. Ground truth (green), no prior (red) and with prior (blue). 46

6.6 Influence of motion prior on VO. Ground truth (green), no prior (red) and with prior (blue). 47
6.7 Intensity and inverted depth frame pair. 48
6.8 VO for a large average camera velocity of 0.43 m/s. Ground truth (green), no prior (red) and with prior (blue). 48
6.9 Intensity and depth frame pair. 49
6.10 Sobel and Scharr filters for the TUW image sequences. Ground truth (green), Sobel 1 (red) and Scharr 10 (blue). 51
6.10 Sobel and Scharr filters for the TUW image sequences. Ground truth (green), Sobel 1 (red) and Scharr 10 (blue). 52
6.11 Optical flow field illustration for motion along the z axis [HS80]. 53
6.12 Sequence 1. 54
6.13 Sequence 3. 55
6.14 Using the full Ackermann motion model for the TUW image sequences. Ground truth (green), only Ackermann (red), schar 10 (blue), 0.1 Ackermann with VO, 0.5 Ackermann with VO, control inputs (yellow). 57
6.14 Using the full Ackermann motion model for the TUW image sequences. Ground truth (green), only Ackermann (red), schar 10 (blue), 0.1 Ackermann with VO, 0.5 Ackermann with VO, control inputs (yellow). 58
6.15 Sequence 2 first 100 frames. Ground truth (green), only Ackermann (red), schar 10 (blue), 0.1 Ackermann with VO, 0.5 Ackermann with VO, control inputs (yellow). 59

8.1 Motion prior for the TUW image sequences. Ground truth (green), no prior (red), prior (blue). 66
8.1 Motion prior for the TUW image sequences. Ground truth (green), no prior (red), prior (blue). 67

List of Tables

6.1 RMSE(m/s) for filter experiments to 3 decimal places. 42
6.2 RMSE(m/s) for weighting experiments to 3 decimal places. 44
6.3 RMSE(m/s) for motion prior experiments to 4 decimal places. 47
6.4 RMSE(m/s) for 6.8 to 3 decimal places. 49
6.5 RMSE(m/s) of 6.10 to 3 decimal places. 50
6.6 RMSE(m/s) of 8.1 to 6 decimal places. 53
6.7 RMSE(m/s) of Figure 6.14 to 3 decimal places. 56

List of Algorithms

5.1 Direct Motion Estimation 33
5.2 BackprojectPixels 34
5.3 GaussNewtonStep 35
5.4 ComputeResidual 35
5.5 Direct Motion Estimation with Motion Prior 36
5.6 GaussNewtonStep_Prior 37
5.7 Direct Motion Estimation with Motion Prior 37

Bibliography

[ALJI18] J. An, J. Lee, G. Jeong, and I. Ihm. Tracking and RGB–D camera on mobile devices using an improved frame-to-frame pose estimation method. *IEEE Winter Conference on Applications of Computer Vision*, 2018.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[BM04] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, Feb 2004.

[BMGB13] G. Bourmaud, R. Mégret, A. Giremus, and Y. Berthoumieu. Discrete extended kalman filter on lie groups. In *21st European Signal Processing Conference (EUSIPCO 2013)*, pages 1–5, Sep. 2013.

[Bra00] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[CMR07] A.I. Comport, E. Malis, and P. Rives. Accurate quadrifocal tracking for robust 3D visual odometry. *IEEE International Conference on Robotics and Automation*, 2007.

[Cre18] V. Creuze. Monocular odometry for underwater vehicles with online estimation of the scale factor. 2018.

[Dav03] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1403–, Washington, DC, USA, 2003. IEEE Computer Society.

[EHE$^+$12] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB–D SLAM system. In *2012 IEEE International Conference on Robotics and Automation*, pages 1691–1696, May 2012.

[EKC16] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *CoRR*, abs/1607.02565, 2016.

[ESC13] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for monocular cameras. *IEEE International Conference on Computer Vision*, 2013.

[ESC14] J. Engel, T. Schöps, and D. Cremers. LSD–SLAM: Large-scale direct monocular SLAM. *European Conference on Computer Vision*, 2014.

[FCDS17] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *Trans. Rob.*, 33(1):1–21, February 2017.

[FMB13] J. Ferdándenz-Madrigal and J. Blanco. *Simultaneous Localization and Mapping for Mobile Robotics.* IGI Global, 2013.

[FW16] W. Förstner and B. Wröbel. *Photogrammetric Computer Vison.* Springer, ISBN: 9783319115504, 2016.

[Gal11] J. Gallier. *Geometric Methods and Applications For Computer Science and Engineering.* Springer, 2011.

[GLU12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[GSL17] J. Gräter, T. Strauss, and M. Lauer. Momo: Monocular motion estimation on manifolds. *CoRR*, abs/1708.00397, 2017.

[Gun03] F. Gunnar. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[HS80] B. K.P. Horn and B. G. Schunck. Determining optical flow. Technical report, Cambridge, MA, USA, 1980.

[HWFS11] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *CoRR*, abs/1107.1119, 2011.

[HZ04] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Int19] Intel. Intel realsense ZR300 specs, 2019.

[KBB16] E. Kaltenegger, B. Binder, and M. Bader. Controlling and Tracking an Unmanned Ground Vehicle with Ackermanndrive. In *Proceedings of the Austrian Robotics Workshop (ARW-16)*, Wels, Austria, May 2016.

[KHK13] S. Klose, P. Heise, and A. Knoll. Efficient compositional approaches for real-time robust direct visual odometry from RGB–D data. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1100–1106, Nov 2013.

[KM07] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society.

[KSC13a] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for RGB–D cameras. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[KSC13b] C. Kerl, J. Sturm, and D. Cremers. Robust odometry for RGB–D cameras. *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[KYD15] M. Korkmaz, N. Yılmaz, and A. Durdu. Comparison of the SLAM algorithms: Hangar experiments. volume 42, 12 2015.

[Lan17] R. Lanctot. Accelerating the future: The economic impact of the emerging passenger economy. Technical report, Strategy Analytics, 2017.

[LeM+17] K. Lenac, J. Ćesić, I. Marković, I. Cvišić, and I. Petrović. Revival of filtering based slam? exactly sparse delayed state filter on lie groups. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1012–1018, Sep. 2017.

[LK81] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[LLB+15] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[LM13] M. Li and A. I. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.

[MAMT15] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.

[Mat89] L. Matthies. *Dynamic Stereo Vision*. PhD thesis, Carnigie Mellon University, 1989.

[MCM05a] M. Maimone, Y. Cheng, and L. Matthies. Two years of visual odometry on the Mars exploration rover. *IEEE International Conference on Systems, Man and Cybernetics*, 2005.

[MCM05b] M. Maimone, Y. Cheng, and L. Matthies. Visual odometry on the Mars exploration rovers. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 903–910 Vol. 1, Oct 2005.

[MKSS03] Y. Ma, J. Košecká, S. Soatto, and S. Sastry. *An Invitation to 3D Vision*. Springer, 2003.

[MLD$^+$06] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real time localization and 3D reconstruction. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 363–370, June 2006.

[MNT04] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems (2nd ed.), 2004.

[MTKW03] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. *Proc. IJCAI Int. Joint Conf. Artif. Intell.*, 06 2003.

[NNB04] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[Opt19] OptiTrack. Optitrack prime 13, 2019.

[QLS18] T. Qin, P. Li, and S. Shen. VINS-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, Aug 2018.

[SC86] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainly. *Int. J. Rob. Res.*, 5(4):56–68, December 1986.

[Sca11] F. Scaramuzza, D. Fraundorfer. Visual odometry part 1: The first 30 years. *IEEE Robotics and Automotive Magazine*, 2011.

[SDMK11] H. Strasdat, A. J. Davison, J. M. M. Montiel, and K. Konolige. Double window optimisation for constant time visual SLAM. In *2011 International Conference on Computer Vision*, pages 2352–2359, Nov 2011.

[SEC14] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. *IEEE International Symposium on Mixed and Augmented Reality*, 2014.

[SEE^{+}12] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB–D slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[SMD10] H. Strasdat, J. M. M. Montiel, and A. J. Davison. Real-time monocular SLAM: Why filter? In *2010 IEEE International Conference on Robotics and Automation*, pages 2657–2664, May 2010.

[SMR08] G. Silveira, E. Malis, and P. Rives. An efficient direct approach to visual SLAM. *IEEE Transactions on Robotics*, 2008.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[TAC08] T. Tykkälä, C. Audras, and A. Comport. Direct iterative closest point for real-time visual odometry. *IEEE International Conference on Computer Vision Workshops*, 2008.

[TMHF00] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[TV98] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall Inc, ISBN: 0132611082, 1998.

[vdWCV11] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *CoRR*, abs/1102.1523, 2011.

[WM10] A. J. Weinstein and K. L. Moore. Pose estimation of ackermann steering vehicles for outdoors autonomous navigation. In *2010 IEEE International Conference on Industrial Technology*, pages 579–584, March 2010.

[Zha00] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[ZS15] J. Zhang and S. Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2015.