Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Institute for Computer Science XVII
Robotics

Bachelor's thesis

Kalman filter for pose estimation on spherical mobile mapping systems

Tim Schubert

April 2024

First reviewer:	Prof. Dr. Andreas Nüchter
Advisor:	Fabian Arzberger

.

Abstract

To obtain exact mapping results with mobile mapping systems, the underlying pose estimation must be as accurate as possible. However, single sensors often do not meet these requirements to an adequate extent. One solution for this problem is to Kalman filter the pose estimates of multiple sensors via sensor fusion. This approach is widely studied for robotic systems, like unmanned aerial vehicles or wheel-based systems. The same cannot be said for spherical mobile mapping systems, which come with a motion model, characterized by heavy rotational movement. This thesis proposes a real-time Kalman filter implementation, providing 6-degree-of-freedom pose estimation. Therefore, inertial pose estimation, based on filtered inertial measurement unit data, and visual-inertial pose estimation, based on an onboard RealSense tracking camera's data, are used as measurement input for the Kalman filter. As an extension to the conventional Kalman filter, the measurement noise covariance matrices are able to adapt to the current angular velocities of the system and tracking conditions of the camera. Based on the calculated trajectory estimation of the Kalman filter and data from an onboard light detection and ranging sensor, point clouds of the environment are generated. These are evaluated by comparing them to high-accuracy ground truth point clouds, which are computed by a semi-rigid SLAM algorithm, proposed in [14].

Zusammenfassung

Um exakte Kartierung mit einem mobilen Mapping-System zu ermöglichen, ist es zwingend notwendig, die Pose des Systems so genau wie möglich zu bestimmen. Das Nutzen von einfachen Sensoren erfüllt diese Anforderung in der Regel jedoch nicht. Ein Lösungsansatz für dieses Problem ist die Anwendung eines Kalman Fiters, das die Pose anhand mehrerer Messquellen durch Sensorfusion schätzt. Dieser Ansatz ist für robotische Systeme, wie unbemannte Fluggeräte oder radbasierte Roboter, bereits umfassend untersucht. Das Gleiche ist allerdings nicht für sphärische mobile Mapping-Systeme der Fall, welche ein rotationslastiges Bewegungsmodell mit sich bringen. In dieser Bachelorarbeit wird ein Echtzeit-Kalman-Filter vorgestellt, welches eine Schätzung der aktuellen Pose in 6 Freiheitsgraden ermöglicht. Hierfür werden inertiale Pose-Schätzungen, basierend auf gefilterten Daten von inertialen Messeinheiten, sowie visuell-inertiale Pose-Schätzungen einer integrierten RealSense Tracking-Kamera als Messeingang für das Kalman Filter genutzt. Als Erweiterung des konventionellen Kalman Filters sind die Messrauschkovarianzmatrizen in der Lage, sich an die aktuellen Winkelgeschwindigkeiten des Systems sowie den Tracking-Bedingungen der Kamera anzupassen. Basierend auf der berechneten Trajektorien-Schätzung des Kalman Filters, sowie den gesammelten Daten eines integrierten Laserscanners, werden Punktwolken der Umgebung generiert. Diese werden anschließend ausgewertet, indem sie mit hochgenauen Referenzpunktwolken verglichen werden, welche mithilfe eines halb-rigiden SLAM Algorithmus erstellt wurden, der in [14] vorgestellt wird.

Danksagung

Hiermit danke ich allen Personen, die mich sowohl bei dieser Bachelorarbeit als auch während meines gesamten bisherigen Studiums unterstützt haben. In erster Linie danke ich meinem Betreuer Fabian Arzberger, der jederzeit bei fachlichen, als auch technischen Fragen zur Stelle war und mir stets das Gefühl gegeben hat, einen Ansprechpartner zu haben. Deine Betreuung gab mir viel Motivation, das bestmögliche Ergebnis zu erzielen und ich sage zudem vielen Dank für dein Vertrauen und deine Geduld. Außerdem gilt ein großer Dank meiner Freundin Kristin Edling, die dauerhaft zu 100 Prozent hinter mir stand und mich in den letzten Jahren in jeglicher Form, innerhalb und außerhalb des universitären Alltags, bekräftigt hat. Ohne deinen Beistand wäre ich heute nicht da, wo ich jetzt bin. Ebenso danke ich meinen Eltern, Großeltern und Geschwistern, die mir viel Kraft spendeten durch den bedingungslosen Beistand in allen Hochs und Tiefs. Ihr gabt mir immer das Gefühl, auf dem richtigen Weg zu sein.

Contents

List of Figures

List of Tables

List of Symbols

General variables:

ϕ	Roll angle, i.e., rotation around x axis.
θ	Pitch angle, i.e., rotation around y axis.
ψ	Yaw angle, i.e., rotation around z axis.

Kalman Filter variables:

$\boldsymbol{x}_k \in \mathbb{R}^n$	Process state vector at index k.
$\boldsymbol{u}_k \in \mathbb{R}^l$	Process control vector at index k.
$\boldsymbol{v}_k$	Process noise vector at index k.
$\boldsymbol{w}_k$	Measurement noise vector at index k.
$\boldsymbol{y}_k \in \mathbb{R}^m$	Measurement vector at index k.
$\boldsymbol{z}_{k+1} \in \mathbb{R}^m$	Measurement prediction vector.
$\boldsymbol{\nu}_k$	Innovation vector at index k.
$\boldsymbol{F}^{n\times n}$	State transition matrix.
$\boldsymbol{G}^{n\times l}$	Control input Matrix.
$\boldsymbol{P}^{n\times n}$	State error covariance matrix.
$\boldsymbol{H}^{m\times n}$	Measurement prediction matrix.
$\boldsymbol{Q}^{n\times n}$	Process noise covariance matrix.
$\boldsymbol{S}^{m\times m}$	Innovation covariance matrix.
$\boldsymbol{K}^{n\times m}$	Kalman gain.
$\boldsymbol{R}_{sensor}^{m\times m}$	Measurement noise covariance matrix for a sensor's measurement vector.

Chapter 1

Introduction

One of the fundamental tasks in the field of robotics is the determination of a robot's pose. Knowledge about the current position and orientation relative to its surroundings has to be as accurate as possible, no matter the size or application of the robot. This applies for robotic systems in all environments, including a drone's aerial navigation, ground-based robots moving in different types of terrain, marine vehicles steering underwater, spacecrafts exploring the vacuum of space or rovers facing the challenges of extraterrestrial landscapes. Especially with the uprise of autonomous systems, detailed mapping and human-robot interactions, the urgency of accurate and precise pose estimation increases to ensure security and reliability for the systems itself and its users. Some emphasizing examples are the pose estimation of NASA's Perseverance rover using visual odometry [3] for localization and path determination purposes, the real-time pose estimation of an unmanned aerial vehicle, capable of autonomous takeoff and landing [6] and the estimation of a robotic arm's position and orientation via image analysis [7]. In the scope of this thesis the focus will lie on the pose estimation for a robotic system of spherical shape, traversing on the ground by a rolling motion.

1.1 Pose Estimation on Spherical Mobile Mapping Systems

While the problem of pose estimation in robotic systems is almost as old as robotics itself, the projection of the problem to spherical-shaped robots is rather novel. Compared to conventional wheeled robotic systems, spherical robots have the advantage of omnidirectional movement and robustness in challenging environments. Thus, Zevering et al. propose an autonomous 3D mapping concept for a spherical robot system in their work "L.U.N.A. - A Laser-Mapping Unidirectional Navigation Actuator" [45]. Here, a robot prototype is introduced, that uses the advantages of robustness to challenging environments to enable possible future exploration of lunar cave systems. This use case prevents the application of a Global Navigation Satellite System (GNSS) and magnetometer data to gather information about the sphere's position and orientation. Thus, their follow-up studies [5] and [46] research pose estimation of such a sperical mobile mapping system (SMMS) using visual and inertial data to estimate the pose, which on top of that serves as the foundation this thesis is built upon. However, the pose estimation for a spherical system brings additional challenges, which are described in section 1.2.

1.2 Problem Description

To estimate the pose of a spherical robotic system, conventional pose estimation methods must be adjusted to be applicable to spherical systems. Many robotic systems use wheel odometry or GNSS data to get positional and orientational information. The spherical mobile mapping system, used in this work, uses neither of these and is limited to visual and inertial odometry, provided by two Inertial Measurement Units (IMU) and a tracking camera. Additionally, these sensors experience constant and possibly fast angular velocities due to the sphere's rolling motion. Hence, the pose estimates based on the IMU's and the tracking camera's data on their own are often not sufficiently accurate to reliably use them for further applications like Simultaneous Localization and Mapping (SLAM). A solution is the usage of sensor fusion, which provides an estimated pose based on the sensors' data, as well as a kinematic model predicting the next pose. Spherical systems also need a different kinematic model than conventional ground-based robotic systems. This brings further challenges, like pose estimation inaccuracy and computational complexity, particularly when accounting for rotations of sensors, which are not centered on the sphere. At last, the estimated pose needs to be evaluated for its accuracy, requiring additional ground truth data or pose estimates, provided by other existing approaches with high precision and accuracy.

1.3 Scientific Contribution

A method to estimate the 6-degree-of-freedom (DoF) pose of a spherical mobile mapping system, moving on a plane surface is proposed. Therefore, the filtered data of two IMUs and a tracking camera is fused via a Kalman filter (KF). The KF implementation is further enhanced by dynamically adjusting measurement noise covariance matrices, based on the angular velocities of the sphere and the camera's tracking confidence. This extension serves the purpose of increasing the accuracy and robustness of the estimated position and orientation, which are impaired by the sphere's rotational movement. Furthermore, the suggested methods are evaluated by comparing point clouds, based on the KF's pose estimation and captured Light Detection and Ranging (LiDAR) data to a ground truth point cloud of the experimental environment. This serves the purpose of evaluating the addition of dynamically adjusting measurement noise covariance matrices against the conventional static approach. Additionally, the results are compared to the previously mentioned Delta filter's pose estimation, to draw conclusions about the performance of the Kalman filter algorithm. This enables additional understanding about the use of the proposed KF pose estimates for further applications (e.g., SLAM) on spherical mobile mapping systems. Note, that the system is simplified to linear kinematics and the assumption that the sensors are centered on the sphere.

1.4 Thesis Structure

This section serves as an overview of the thesis outline, which consists of seven more chapters. Chapter 2 presents the current state-of-the-art in terms of spherical robots (2.1) and pose estimation methods (2.2). To underline the discussed concepts, representing studies are briefly outlined. In section 3 the necessary mathematical fundamentals are addressed and the Kalman filter algorithm is introduced. The robotic system, used for recording data and evaluating the proposed methods, is defined in chapter 4. Section 5 informs about the implementation of the Kalman filter algorithm and its addition of adaptive measurement noise covariance matrices, as well as the surrounding software methods. Before evaluating the algorithm's results and presenting the experimental setup to acquire the data in section 7, the methodology of evaluation and configuration is described in chapter 6. At last, conclusions of the thesis are drawn in chapter 8. Furthermore, any future work to extend the research in this field is suggested here.

Chapter 2

Related Work & State of the Art

2.1 Spherical Robots

Compared to conventional robotic systems, spherical robots are a more narrow area of research. Many studies focus on the general design principles of such spherical systems, some of which are presented by Crossley and Vincent's literature review [11], e.g., the Bicchi design, the pendulum design and a two-hemispheric approach. In contrast to conventional rigid spheres, Sugiyama et al. [40] suggest a deforming sphere that moves by crawling and jumping, eliminating drawbacks like its large weight and complexity of locomotion.

Apart from design studies, there are several papers that deal with the localization and pose estimation of spheres. Zevering et al. [46] introduced a method to estimate the pose of a spherical robot; based on inertial measurement unit data, using existing filtering methods, while mitigating orientational jitter and slip effects, caused by motion. Another study from this lab by Arzberger et al. [5] introduce a Delta filter to estimate the pose of the sphere. The system represents the same system as the one for this thesis and, thus, gets described in more detail in section 4.2. Using the visual-interitial-odometry provided by a tracking camera and IMUs, a 6-DoF pose estimate is computed in real-time by calculating a weighted geometric mean for each measurement. Hence, the filter operates without knwoledge of the sensor's covariances, by determining how similar the measurement is to the geometric mean of the previous measurements. These later two studies serve as a fundamental approach for further applications of this system (e.g., SLAM). Additionally, He et al. present a method for localizing spherical underwater robots via a multi-robot cooperative localization system using a Kalman filter algorithm as an alternative to visual position and attitude determination [19]. Another branch of research is the work by Chen et al. in [9], who developed a control scheme for a pendulum driven, non-holomonic spherical robot, i.e., a spherical robot with constraints in terms of velocity and acceleration. They use a neural network, able to handle unknown uncertainties and external disturbances while outperforming a conventional Fuzzy-PID controller in terms of system response smoothness. In contrast to many spherical robot studies focusing on motion on even surfaces, Sabet et al. [35] demonstrate a path planning algorithm for spherical robots on uneven, three-dimensional terrains. Therefore, they introduce a set of dynamic equations for the system as well as an analysis of the robot's energy and power consumption. This enables path planning for scenarios with no knowledge of the surrounding environment.

2.2 Pose Estimation Methods

While pose estimation on spherical robots might still be a rather new research field, various approaches using onboard sensors; apart from spherical systems have been explored. In general, multi-sensor pose estimation can be categorized into two integration architectures.

The first integration architecture processes the sensors' data independently, without directly influencing the initial estimators, e.g., a complementary filter providing a pose estimate from raw IMU data. The results are then combined to a single pose estimate using filtering methods. This architecture is called *loosely coupled* and brings advantages, like more flexibility due to the independent development of low-level data processing. However, it brings the risk of complexity for large applications with many measurement sources. The loosely coupled architecture also represents the multi-sensor integration architecture used for pose estimation in this thesis.

The second approach is the *tightly coupled* architecture. Here, the data from the sensors is also used to get one resulting state estimate, but the individual initial estimators are affected by this process [29]. Consequently, the measurements are strongly interdependent, which, on the one hand, results in less complexity for large applications, but, on the other hand, increases the chance for pose estimation failure when one sensor provides no or false data. Additionally the effects of the individual systems on other systems must be carefully considered and well understood to ensure accurate results. Many tightly coupled approaches are based on factor graph optimization, as seen in the presented studies below.

Some examples of loosely coupled pose estimation are the following. A study by Wael Farag [15] presents a real-time localization method for autonomous vehicles using a combination of the Monte Carlo localizaion algorithm, a PF and an unscented Kalman filter (UKF). Especially when dealing with non-linear motion systems, the PF proves to be an effective method for multi-sensor fusion. In this study, the raw data of a LiDAR and radar sensor is fused by the UKF to generate "pole-like static-object pose estimation", serving as landmarks for the vehicle's localization. Later, a PF estimates the pose of the vehicle in reference to an offline-generated map. The Monte Carlo algorithm then uses the output of the filters to achieve the best possible vehicle pose estimate. With neural networks finding increasingly more applications, robotic pose estimation is one of them. Valente et al. [41] demonstrate how a convolutional neural network, with preproccesed 2D laser scanner data and camera images as input, fuses the data to provide a vehicle's odometry estimate. One of the main research fields in pose estimation is unmanned aerial vehicles (UAV). To estimate the 3D pose of a small UAV, Strohmeier et al. suggest a loosely coupled extended Kalman filter (EKF), based on data from an IMU, a barometric pressure sensor, and an Ultra-Wide Band transceiver network [39]. The resulting positional accuracy with a Root-Mean-Square Error is 0.20 cm, while the orientational accuracy with a Root-Mean-Square Error lies at 1.93 °. In [13], Hao Du et al. demonstrate the effectiveness of estimating the 3D state of a UAV via an EKF in real-time. They use a system consisting of several sensors, including an IMU, a barometer, a GNSS receiver, an RGB-D camera and more. Their study also shows that pose estimation is not limited to environments with GNSS access, making it suitable for a variety of applications. A similar approach is presented in [18] by estimating the pose of a quadrotor in an indoor environment using Kalman filter multisensory fusion in real-time. Although, loosely-coupled autonomous navigation is not only limited to UAVs, as shown by Manzanilla et al. [26], who use visual and inertial data, fused via an EKF to autonomously track an underwater robot's trajectory, while additionally generating a map of the environment using PTAM (Parallel Tracking and Mapping). Another study, done by Shmaliy et al. [38], indicates that the *Unbiased Finite Impulse Response Filtering* method is an additional valid alternative to the KF to estimate a system's state, e.g., robot self-localization using radio-frequency identification tags. Especially when having non-Gaussian noise, the KF estimates suffer from inaccuracy, thus, making this proposed filter a valuable alternative.

On the contrary, there are several studies using the tightly coupled architecture, some of which are the following. In [10] Cioffi et al. propose a pose estimation method based on global positional information, as well as visual and inertial measurements for a non-linear system. They make use of the measurements' correlations to achieve accurate pose estimation results without any significant computational drawbacks, which outperforms state-of-the-art approaches of loosely coupled architecture. Furthermore, Zhao et al. show in [47], how tightly coupled graph optimization and dense visual-inertial SLAM with deep neural networks achieve enhanced state estimation for poses and sparse maps. Their results show better performance in terms of dense depth estimation and trajectory estimation compared to existing approaches in this research field. Another study making use of the tightly coupled architecture is the work of Shan et al. in [37]. Here, a framework for real-time state estimation and mapping is presented, based on lidar-visual-inertial odometry. Therefore, their system integrates a 3D LiDAR sensor, a monocular camera and an IMU, and maintains a factor graph with different constraints. The system is divided into a lidar-inertial subsystem and a visual-inertial subsystem, which are interacting with each other, resulting in an improvement of robustness and accuracy of the framework. At last, the study of Chen et al. [8] proposes a method to reduce the computational complexity for real-time SLAM applications on mobile devices. Next to RGB, depth and IMU information, their SLAM system uses structured plane information to reduce the number of map points and increase the system's effectiveness. All the information is integrated into a single optimization framework, enabling tightly coupled system optimization.

Chapter 3

Fundamentals

3.1 Kinematic Model for Spherical Bodies

A kinematic model serves as a description of an object's motion in space. In order to estimate the pose of a spherical body, it is crucial to have a set of equations, since it builds the foundation of the prediction step of the KF algorithm. The following equations explain the linear motion of a sphere, derived from a given angular velocity $\boldsymbol{\omega}$ and a known sphere radius r_{sphere}. Furthermore, the normal vector is represented by $\boldsymbol{n} = \begin{bmatrix} n_x, & n_y, & n_z \end{bmatrix}^T$, with $\boldsymbol{n}^T$ denoting the transpose of $\boldsymbol{n}$. At first the tangential velocity $\tilde{\boldsymbol{v}}$ is given by:

$$\begin{bmatrix} \tilde{v}_x \\ \tilde{v}_y \\ \tilde{v}_z \end{bmatrix} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} r_{\text{sphere}}. \tag{3.1}$$

Then, the distance moved in each axis in a timespan ΔT is calculated:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \tilde{v}_x \\ \tilde{v}_y \\ \tilde{v}_z \end{bmatrix} \Delta T. \tag{3.2}$$

Finally, the distance is projected onto the plane using $\boldsymbol{n}$:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \times \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} -\Delta z n_y + \Delta y n_z \\ \Delta z n_x - \Delta x n_z \\ -\Delta y n_x + \Delta x n_y \end{bmatrix} = \begin{bmatrix} -\omega_z n_y + \omega_y n_z \\ \omega_z n_x - \omega_x n_z \\ -\omega_y n_x + \omega_x n_y \end{bmatrix} \Delta T \cdot r_{\text{sphere}}. \tag{3.3}$$

Applied for the assumption of moving on a plane surface, the normal vector for the kinematic model of the SMMS is given by $\boldsymbol{n} = \begin{bmatrix} 0, & 0, & -1 \end{bmatrix}^T$, which results in

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -\Delta y \\ \Delta x \\ 0 \end{bmatrix} = \begin{bmatrix} -\omega_y \\ \omega_x \\ 0 \end{bmatrix} \Delta T \cdot r_{\text{sphere}}. \tag{3.4}$$

3.2 Coordinate System Transformation

Due to each sensor on the SMMS providing data in its own local coordinate system, it is necessary to transform the local measurements into a global coordinate system. Without this transformation, the concepts presented in section 3.6 are not possible. To perform such a transformation, the position and attitude of a body need to be transformed. The positional transformation is called translation, while the attitude's transformation is called rotation. A transformation in the context of this thesis is usually done in three-dimensional space.

3.2.1 Translation

The three-dimensional translation of a body can be broken down to the addition of vectors [23]. Let the current position of a rigid body be $\boldsymbol{p} = \begin{bmatrix} x, & y, & z \end{bmatrix}^T$. A change in position by the vector $\boldsymbol{\Delta p} = \begin{bmatrix} \Delta x, & \Delta y, & \Delta z \end{bmatrix}^T$, is referred to as a translation an can be written as

$$\boldsymbol{p} + \boldsymbol{\Delta p} = \boldsymbol{p'} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \end{bmatrix}. \tag{3.5}$$

3.2.2 Rotation

Rotating a body in three-dimensional space enables a point to be rotated around the x-, y- and z-axis, as well as an arbitrary axis, by a certain angle [23]. These rotations are represented in different ways, as presented in section 3.3.

3.3 Attitude Representation

There are mainly three ways to represent the attitude of a body: being unit quaternions, Euler angles and rotation matrices. Each of them can be converted into another using trigonometric equations, a shown in detail in [12]. These are also the representations, used throughout this thesis. Each of them has advantages as well as disadvantages, depending on the use case. The content of this section is mainly based on the work of James Diebel in "Representing attitude: Euler angles, unit quaternions, and rotation vectors" [12].

3.3.1 Euler Angles

Leonhard Euler introduced a set of three Euler angles, that became a common way to represent a body's orientation relative to a fixed coordinate frame, since they are the most readable construct of the three stated above. They are widely referred to as roll (ϕ), pitch (θ) and yaw (ψ) in the fields of robotics and aerospace engineering. Each angle represents one rotation about one of the three distinct axes and, consequently, brings one degree of freedom of the body. Note, that the rotations must be performed sequentially, the order of which is determined by one of the 12 possible conventions, e.g., $[\phi \to \theta \to \psi]$, $[\theta \to \phi \to \psi]$, or $[\psi \to \theta \to \phi]$. Depending on the sequence of rotations, the resulting attitude varies. The convention used in this thesis is $[\psi \rightarrow \theta \rightarrow \phi]$. Despite their intuitiveness, Euler angles have two main flaws. These are the so-called gimbal lock, caused by mathematical singularities, and less accuracy when used to integrate incremental changes in attitude over time, compared to quaternions [12].

3.3.2 Quaternions

An alternative to Euler angles, the quaternions, were presented by William Rowan Hamilton in the 19th century. Quaternions do not suffer from the same flaws as Euler angles. This makes it attractive for calculations, which rely on high attitude accuracy. A quaternion $\boldsymbol{q} \in \mathcal{H}$ can be represented as a four-dimensional vector in the following way:

$$\boldsymbol{q} = \begin{bmatrix} q_0, & q_1, & q_2, & q_3 \end{bmatrix}^T . \tag{3.6}$$

However, this representation already causes the first disadvantage. It is not as readable and intuitive as Euler angles because the four parameters do not have a simple physical meaning. In addition to the four dimensions, the quaternion only represents a pure rotation when in unity form, meaning

$$|\boldsymbol{q}| = 1. \tag{3.7}$$

Finally, quaternions come with some non-trivial operations, distinguishing them further from Euler angles and rotation matrices [12], e.g. the non-commutative product of two quaternions, called *Hamilton product*.

3.3.3 Rotation Matrices

A rotation matrix $\mathfrak{R}$ in context of three-dimensional attitude representation is a 3×3 construct, which rotates a vector multiplied to it by a given Euler angle γ, while the vector's length remains the same [12]. The representation of a rotation by a sequence of rotations around three axes, as introduced in section 3.3.1, is then embodied by $\mathfrak{R}$, also referred to as *direction cosine matrix*. $\mathfrak{R}$ must be orthogonal, guaranteeing both

$$\mathfrak{R}^T = \mathfrak{R}^{-1} \tag{3.8}$$

$$\mathfrak{R}^T \times \mathfrak{R} = \boldsymbol{I}, \tag{3.9}$$

with $\boldsymbol{I}$ representing the identity matrix and $\mathfrak{R}^{-1}$ denoting the inverse of $\mathfrak{R}$. There exists exactly one rotation matrix for a rotation around each of the three axes in counterclockwise direction by γ. Such a rotation around the x-, y- or z-axis requires the corresponding rotation matrix, as shown in [44], which is written as:

$$\mathfrak{R}_{\boldsymbol{x}}(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix} \tag{3.10}$$

$$\mathfrak{R}_{\boldsymbol{y}}(\gamma) = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \tag{3.11}$$

$$\mathfrak{R}_{\boldsymbol{z}}(\gamma) = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.12}$$

The colum and row vectors of the matrix provide information about how the body's current local orientation is located in relation to a fixed global coordinate system. A rotation matrix

$$\mathfrak{R} = \begin{bmatrix} \boldsymbol{r}_1, & \boldsymbol{r}_2, & \boldsymbol{r}_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.13}$$

encodes the basis vectors of the global coordinate system expressed in the local coordinate system in its column vectors. The rows of $\mathfrak{R}$ represent the basis vector's local coordinate system expressed in the global coordinate system. Thus, when an element in $\mathfrak{R}$ changes its value, the corresponding column- or row-vector entry changes, which provides information about the object's orientation [12].

3.4 Pose Definition

An object's 6-DoF pose $\boldsymbol{\rho}$ consists of the object's position $\boldsymbol{p}$ and orientation $\boldsymbol{\beta}$. Let $\boldsymbol{\rho} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\beta} \end{bmatrix}$ with $\boldsymbol{p} \in \mathbb{R}^3$ and $\boldsymbol{\beta} \in \mathbb{R}^3$, assuming $\boldsymbol{\beta}$ is represented as Euler angles, i.e., $\boldsymbol{\rho} \in \mathbb{R}^6$. The role of a 6-DoF pose in combination with rotation matrices, in the scope of this thesis, is relevant regarding the following scenario. Let an orientation at timestep k be encoded in a rotation matrix $\mathfrak{R}_k$, and the position of that timestep be given as $\boldsymbol{p}_k$, summarized as $\boldsymbol{\rho}_k = \begin{bmatrix} \boldsymbol{p}_k \\ \mathfrak{R}_k \end{bmatrix}$. Then the change between the timesteps k and $k+1$ is determined by

$$\Delta\boldsymbol{\rho} = \begin{bmatrix} \boldsymbol{p}_{k+1} - \boldsymbol{p}_k \\ \mathfrak{R}_{k+1}^{-1} \cdot \mathfrak{R}_k \end{bmatrix}. \tag{3.14}$$

This becomes relevant for the configuration and calculation of the Kalman filter parameters, described in chapter 5. Additionally this calculated delta pose between the timesteps k and $k+1$ is later used to iteratively update the total pose estimate $\boldsymbol{\rho}_{\text{total}}$:

$$\boldsymbol{\rho}_{k+1,\text{total}} = \Delta\boldsymbol{\rho}_{k+1} \cdot \boldsymbol{\rho}_{k,\text{total}} = \begin{bmatrix} \Delta\boldsymbol{p}_{k+1} + \boldsymbol{p}_k \\ \Delta\mathfrak{R}_{k+1} \cdot \mathfrak{R}_k \end{bmatrix}. \tag{3.15}$$

Another pose representation is a 4×4 spatial transformation matrix $\boldsymbol{\mathcal{T}}$, which consists of a rotation matrix $\boldsymbol{\mathfrak{R}}$, a translation vector $\boldsymbol{t}$ and a scaling factor s, corresponding to

$$\boldsymbol{\mathcal{T}} = \begin{bmatrix} \boldsymbol{\mathfrak{R}} & \boldsymbol{t} \\ \boldsymbol{0} & s \end{bmatrix}. \tag{3.16}$$

Here, $\boldsymbol{\mathfrak{R}}$ encodes to the orientation in space, while $\boldsymbol{t}$ corresponds to the object's position. Usually $s = 1$, ensuring the length of a vector multiplied with $\boldsymbol{\mathcal{T}}$ stays the same. Thus, $\boldsymbol{\mathcal{T}}$ represents the rotation (attitude) and translation (position) of an object, making it an alternative to the pose representation above. $\boldsymbol{\mathcal{T}}$ is additionally used to transform a point $\boldsymbol{x}$ to a point $\boldsymbol{y}$ by matrix-vector multiplication as follows [42]:

$$\boldsymbol{\mathcal{T}} \times \boldsymbol{x} = \boldsymbol{y}. \tag{3.17}$$

3.5 Covariance Matrix

A covariance matrix, or variance-covariance matrix, $\boldsymbol{C}$ is a symmetrical, positive semi-definitve square matrix, containing all covariances between a random vector $\boldsymbol{a}$'s entries. $\boldsymbol{a}$ is referred to as a random vector, when all variables are random and jointly distributed. To show which covariance matrix $\boldsymbol{C}$ corresponds to which random vector, the notation $\boldsymbol{C_a}$ is used. $\boldsymbol{C_a}$ is given by

$$\boldsymbol{C_a} = E[(\boldsymbol{a} - E\boldsymbol{a})(\boldsymbol{a} - E\boldsymbol{a})^T] \tag{3.18}$$

with $E\boldsymbol{a}$ being the expectation of $\boldsymbol{a}$ [33]. Entries off the main diagonal of $\boldsymbol{C}$ represent covariances between a pair of entries of $\boldsymbol{a}$. $\boldsymbol{C}$'s main diagonal, on the other hand, consists of variances σ^2 of the random variables, i.e., describes how much a variable's value spreads around its dataset's mean. The variance of a set of data points is calculated using the following equations:

$$\bar{b} = \frac{1}{N} \sum_{i=1}^{N} b_i \tag{3.19}$$

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (b_i - \bar{b})^2. \tag{3.20}$$

In this context, $\bar{b}$ represents the mean of the dataset, b_i stands for the current data point, and N marks the overall number of points in the dataset.

3.6 Kalman Filter

The following sections regarding the Kalman filter are mostly based on the work of Youngjoo Kim and Hyochoong Bang in "Introduction to Kalman Filter and Its Applications" [24] as well as Gregory F. Welch in "Computer Vision: A Reference Guide" [43]. Rudolf E. Kalman introduced a recursive algorithm in the 1960s, that estimates the state of a linear process using discrete measurements with gaussian noise distribution. The measured data is obtained from multiple different sources, integrating them into one signal. This process is known as sensor fusion and is described in detail by J.Z. Sasiadek in “Sensor fusion” [36]. Thanks to its computational effectiveness and simplicity, the KF is utilized across a wide spectrum of applications, as presented in section 3.6.2. In addition to the KF, further improvements and variations have been implemented in recent time, which for example extend the KF to non-linear processes, like the EKF and the UKF. Thus, the regular Kalman filter is often also referred to as *linear Kalman filter* (LKF). In contrast to the LKF, the EKF linearizes the non-linear system dynamics and measurement equations around the current state-estimate using a first-order Taylor-approximation, resulting in 1st order accuracy. However, this approach brings the risk of propagating errors through the linearization process, a flaw the UKF does not suffer from. Here, a minimal set of sample points, so-called *sigma-points*, is selected around the current state estimate, using a sampling technique, called *unscented Transformation*. These are then propagated through the non-linear functions of the system, which provides the 3rd order accurate mean and covariances of the system state [25].

3.6.1 Algorithm

The Kalman filter is separated into two stages: the prediction step and the update step. Propagation of the next state is handled by the prediction step, based on the updated state of the prior iteration. Afterwards the new prediction is corrected by the update step based on the latest measurements. Furthermore, the update step takes the covariances of the measurement sources into account to gain more accuracy. The filter then iteratively estimates $\boldsymbol{x}_k$ and $\boldsymbol{P}_k$, with the ultimate goal of minimizing $\boldsymbol{P}_k$, i.e., minimizing the estimation error uncertainty. Thus, these two parameters need to be initialized before the first iteration with the processes known initial state $\boldsymbol{x}_0$ and initial covariance $\boldsymbol{P}_0$. It is common practice to choose a large $\boldsymbol{P}_0$, since $\boldsymbol{P}$ will converge quickly to appropriate values [24].
The transition of the state from timestep k to $k+1$ is described by the following equation:

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{G}\boldsymbol{u}_k + \boldsymbol{v}_k \tag{3.21}$$

To bring the measurement and state into relation, the measurement at timestep k is given by:

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{w}_k \tag{3.22}$$

Onwards, the usage of “$\hat{}$” marks a variable as an estimate. The symbol “$^-$” tags a variable as “a priori”, meaning it is a prediction without measurement correction, calculated in the prediction step. “A posteriori” is represented by “$^+$”, speaking of a prediction, corrected by measurements and calculated in the update step. An estimated state before correction is consequently written as $\hat{x}^-$ and after correction as $\hat{x}^+$.

Prediction Step:

$$\hat{\boldsymbol{x}}_{k+1}^- = \boldsymbol{F}\hat{\boldsymbol{x}}_k^+ + \boldsymbol{G}\boldsymbol{u}_k \tag{3.23}$$

$$\boldsymbol{z}_{k+1} = \boldsymbol{H}\hat{\boldsymbol{x}}_{k+1}^- \tag{3.24}$$

$$\boldsymbol{P}_{k+1}^{-} = \boldsymbol{F}\boldsymbol{P}_{k}^{+}\boldsymbol{F}^{T} + \boldsymbol{Q} \tag{3.25}$$

Equation (3.23) describes how the state prediction is calculated. $\boldsymbol{F}$ transitions the KF output state of the previous iteration to the predicted state. Furthermore, $\boldsymbol{G}$ multiplied with $\boldsymbol{u}_k$ represents additional input to the system. The sum of these two factors result in the new state prediction. The prediction step also aims to estimate the next measurements. Therefore, a matrix $\boldsymbol{H}$ is needed, to extract the measurement prediction out of the predicted state $\hat{\boldsymbol{x}}_{k+1}^{-}$, as shown in equation (3.24). Marking the last calculation of the first KF step, the prediction of the state error covariance matrix is handled via equation (3.25). Depending on $\boldsymbol{Q}$'s process noise uncertainty, the state estimate can become less certain after prediciton steps, due to $\boldsymbol{Q}$ being positive definitive as shown in [16].

Update Step:

$$\boldsymbol{\nu}_{k+1} = \boldsymbol{y}_{k+1} - \boldsymbol{z}_{k+1} \tag{3.26}$$

$$\boldsymbol{S}_{k+1} = \boldsymbol{H}\boldsymbol{P}_{k+1}^{-}\boldsymbol{H}^{T} + \boldsymbol{R} \tag{3.27}$$

$$\boldsymbol{K}_{k+1} = \boldsymbol{P}_{k+1}^{-}\boldsymbol{H}^{T}\boldsymbol{S}^{-1} \tag{3.28}$$

$$\hat{\boldsymbol{x}}_{k+1}^{+} = \hat{\boldsymbol{x}}_{k+1}^{-} + \boldsymbol{K}\boldsymbol{\nu}_{k+1} \tag{3.29}$$

$$\boldsymbol{P}_{k+1}^{+} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\boldsymbol{P}_{k+1}^{-} \tag{3.30}$$

The update step now aims to improve the state's estimation, using the measurements provided by information sources such as sensors. At first, the innovation is calculated by equation (3.26). $\boldsymbol{\nu}_{k+1}$ tells, how accurate the measurement prediction $\boldsymbol{z}_{k+1}$ of the prediction step was, by subtracting it from the actual measurement $\boldsymbol{y}_{k+1}$. Next, the innovation covariance matrix is calculated by equation (3.27). This matrix encodes the uncertainty of $\boldsymbol{\nu}_{k+1}$. Since $\boldsymbol{\nu}_{k+1}$ depends on the measurement, $\boldsymbol{S}$ depends on $\boldsymbol{R}$, which holds information about the measurement's certainty. When the uncertainty about the measurements is high, the uncertainty of the innovation calculation is high as well. In equation (3.28), the Kalman gain is calculated. This matrix, as well as the innovation $\boldsymbol{\nu}_{k+1}$ and the state prediction $\hat{\boldsymbol{x}}_{k+1}^{-}$ is used to correct the state estimate of the prediction step. Thus, equation (3.29) provides the first of the two Kalman filter outputs. The second output is generated in equation (3.30). Here, the predicted state error covariance matrix $\boldsymbol{P}_{k+1}^{-}$ is corrected. Both $\hat{\boldsymbol{x}}_{k+1}^{+}$ and $\boldsymbol{P}_{k+1}^{+}$ will then be used in the next iteration of the algorithm.

3.6.2 Applications

Chapter 2 presents several use-cases for pose estimation via KF and its extensions, while chapter 5 shows, how the LKF is be used to estimate the pose of a spherical mobile mapping system. This, however, only shows how the algorithm is used in the field of robotics, but the KF and

its extensions are applied in various more scientific and industrial fields. Thus, this section briefly presents three additional examples to give an idea of the vast possibilities in terms of KF applications.
In [27], a way to efficiently identify the fundamental component from a distorted signal in power systems is presented, using a Kalman filter based algorithm. They compare their algorithm to other non-KF based approaches and find it to be computationally simpler, as well as about equally fast and precise.
Next to the signal processing research field, the medical field also makes use of the KF. Burak Alacam and Yazici show in [2], how an EKF is utilized in the research of cancerous tumors using near-infrared measurements.
Additionally, Jiang et al. propose in [22], how a KF is utilized for aviation surveillance. They combine the KF with a neural network, which predicts certain parameters of the KF and, therefore, lets the KF be trainable and have dynamic parameter estimation. This approach improves the adaptability in dynamic environments, as well as the accuracy and stability of the traditional KF.

Chapter 4

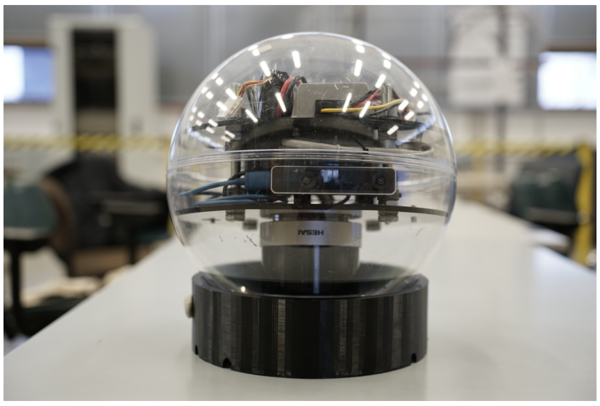
System Definition

4.1 Robot Operating System

The Robot Operating System (ROS) is an open-source system for robotic applications, providing the necessary tools, libraries and functionalities to develop and run software across multiple systems. Some of the services coming with ROS are package management, communication between processes and hardware abstraction. Even though it is not a downright real-time framework, data handling, like streams to dedicated topics, is possible through the use of a peer-to-peer process communication network. ROS supports Unix-based systems and has multiple distribution releases [34]. The work for this thesis was done using ROS-melodic and Ubuntu 18.04.6 LTS.

4.2 Spherical Mobile Mapping System

The foundation for obtaining data, as well as testing and evaluating the Kalman filter algorithm, is the spherical mobile mapping system. For this, several components are needed, as described in the following sections. To ensure the spherical shape and movement along with protection of the electronic components, the SMMS's outside consists of a demountable transparent plastic shell with a radius of 14.5 cm, as depicted in figure 4.1a. In the same figure the RealSense T265 tracking camera (4.3.2) is depicted in the center of the SMMS, as well as the Hesai Pandar XT32 LiDAR sensor (4.3.3) underneath the camera. Figure 4.1b shows the interior, which is mounted onto a 3D-printed frame, that lays inside the plastic shell without freedom of movement to ensure a solid system. It also depicts one of the IMUs, mounted on the lower one of the two 3D-printed black plates. The second IMU sits on the opposite side of the same plate. In addition to the sensors, a “BMAX Mini” is used as the processing and data storage unit, which is located in between the plates in figure 4.1b. The voltage converter, that ensures a constant supply of 12 V, is depicted the same figure on the left side of the upper plate. Behind the voltage converter sits the 9000 mAh lithium polymer battery, with 11.1 V providing the system's power and enabling an approximate runtime of 2 h. Connection establishment with the robot is made possible via a Wi-Fi module. The SMMS used for this thesis does not use any actuators for self-propulsion, therefore requiring outside manual movement.


(a) Front view of the spherical mobile mapping system inside the protecting plastic shell

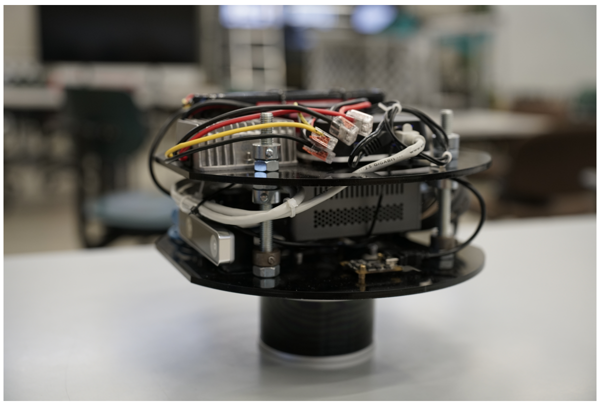

(b) Side view of the spherical mobile mapping system's interior

Figure 4.1: Spherical Mobile Mapping System

(a) Phidget Spatial 3/3/3 No.1044_1b [31], used for retrieving pose data

(b) RealSense T265 tracking camera [17], used for retrieving pose data

(c) Hesai Pandar XT32 LiDAR [21], used to caputure 3D data

Figure 4.2: Sensors installed on the SMMS

4.3 Sensors

All data aquisition for this thesis was done using the sensors described in this section. The spherical mobile mapping system possesses two IMUs, a RealSense tracking camera, and a Hesai Pandar LiDAR sensor.

4.3.1 Inertial Measurement Units

To obtain information about the sphere's pose, two "Phidget Spatial 3/3/3 No.1044˙1b" are used, depicted in figure 4.2a. The sensor consists of a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer. The accelerometer measures the rate of change of velocity along each axis. The gyroscope provides information about the angular velocity along each axis. At last, the magnetometer detects the magnetic field acting upon each axis. Additionally, the current orientation of the IMU is directly available in the form of quaternions [31]. Using this data, a pose for each sensor is generated using a symbiosis of the Complementary and Madwick filter. This process is presented in further detail by Jasper Zevering et al. in "IMU-based pose-estimation for spherical robots with limited resources" [46]. The output pose is available at a rate of 125 Hz.

4.3.2 RealSense Tracking Camera

A second source for pose data of the SMMS is the "RealSense T265" Tracking Camera, which is shown in figure 4.2b. It consists of a set of wide Field-of-View (FOV) fisheye cameras with a FOV of 165°. The camera combines visual- and inertial-odometry (VIO) tracking, by making use of detected images and an internal IMU. RealSense uses a proprietary fusion algorithm to generate a pose from the image and IMU data, which is available at a rate of 200 Hz. Further data, like the raw IMU and camera data, as well as a tracking confidence factor, is provided as well [17].

4.3.3 Hesai Pandar XT32 LiDAR

To generate point clouds, which are used to evaluate the Kalman filter's accuracy, LiDAR data is required, so the SMMS is equipped with a "Hesai Pandar XT32" LiDAR sensor, depicted in figure 4.2c. Having a horizontal FOV of 360° and a detection range of 0.05 m to 120 m, it is well suited for mapping purposes close to the ground while experiencing rolling motion. The vertical FOV lies at 31° $(-16°\text{to} + 15°)$. Since all experiments are performed indoors, the sensor's maximum range is also adequate. With a wave length of 905 nm, the sensor operates in the near-infrared spectrum. It is providing data with a frame rate of 20 Hz, while scanning points at a rate of $640,000\,{}^{\text{pt}}/_{\text{s}}$ [20].

Chapter 5

Implementation

To implement the Kalman filter algorithm on the SMMS, a dedicated ROS node is used, that handles all instances of the KF and the sensors. It consists of two ROS packages with one C++ source file each. The first package is responsible for several tasks. For one, the KF algorithm itself, which is further described in section 5.2. After receiving the sensor's data via callback methods, the camera's data needs to be interpolated, as described in section 5.1. Since the IMUs and the RealSense tracking camera provide their data in different coordinate frames, the data is transformed into the same frame before being handed to the KF to get a correct pose estimate. Hence, the camera data is transformed into the IMUs' frame via rotation matrix multiplication. Due to connection issues, that occasionally occur with the RealSense T265, a second package is used. As soon as the camera suffers from a disconnect, the package ensures a continuing data stream by republishing the data, needed in the Kalman filter package. Note, that this package was partly existing beforehand and was provided by Fabian Arzberger. However, there have been made several additions to enable the Kalman filter's operation (e.g., additional data publishing). This package also contains a custom ROS message to publish the RealSense T265's confidence level for each pose estimate, the purpose of which is explained in section 5.2.2. The ROS frames of the running Kalman filter node are depicted in figure A.2. Instead of the conventional way of the *map* frame being the parent frame of the *odom* frame, in this implementation the odom frame is the parent frame of two map frames. This is due to the reason, that the SMMS runs the Kalman filter and Delta filter parallel with one dedicated map frame each.

5.1 Sensor Data Interpolation

It is stated in section 4.3, that the IMU-based pose estimation is published at a frequency of 125 Hz, whereas the RealSense T265 publishes its pose estimation at 200 Hz. With the KF fusing the sensors measurements, it is necessary to provide data from the same timestep to the filter. Fabian Arzberger et al. shared a method for solving this problem for their Delta filter in [5]. Due to their filter using the same sensors on the same system, the presented interpolation method is applied before the data is handed to the KF. The camera is publishing more frequently, thus its data is being accumulated in a queue and then interpolated, as shown in [5]. This implies that the KF's pose estimation output rate is limited by the IMU-based pose estimation's publishing rate of 125 Hz.

5.2 Kalman Filter Implementation Method

In section 3.6 the Kalman Filter is introduced in general. However, the particular design of the KF for the SMMS is presented in this chapter, addressing the configuration of the algorithms variables in section 5.2.1 as well as the software structure in section 5.2.3. Furthermore, the LKF is extended by making the measurement noise covariance matrices $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ adaptive, which is described further in section 5.2.2. To make working with matrices and vectors of up to 9 dimensions possible, the "Eigen" C++ package is used. Operations like coordinate transformations and handling poses are done using conventional ROS packages.

5.2.1 Parameter Configuration

The state vector $\boldsymbol{x}_k$ consists of the change of the pose $\boldsymbol{\rho}$ from iteration $k-1$ to k and of the current angular velocities of each axis $\boldsymbol{\omega}$. It is important to note that the KF works with $\Delta\boldsymbol{\rho}$, so that gimbal lock is avoided, due to this KF implementation estimating the sphere's attitude as Euler angles. The way $\Delta\boldsymbol{\rho}$ is calculated for each iteration is described in equation (3.14). This makes $\boldsymbol{x}_k$ a 9×1 vector, which is written as

$$\boldsymbol{x}_k = \begin{bmatrix}\Delta\boldsymbol{\rho}_k, & \boldsymbol{\omega}_k\end{bmatrix}^T = \begin{bmatrix}\Delta x_k, & \Delta y_k, & \Delta z_k, & \Delta\phi_k, & \Delta\theta_k, & \Delta\psi_k, & \omega_{x,k}, & \omega_{y,k}, & \omega_{z,k}\end{bmatrix}^T. \tag{5.1}$$

Since the KF needs an initial value for the state, $\boldsymbol{x}_0$ is initialized with

$$\boldsymbol{x}_0 = \mathbf{0}^{9\times1}. \tag{5.2}$$

The angular velocity of the x- and y-axis from iteration $k-1$ is used to calculate the positional prediction for the k-th iteration by executing equation (3.23). To apply this equation, and to stay true to the kinematic model, described in equation (3.4), $\boldsymbol{F}$ must be a 9×9 matrix of the following form:

$$\boldsymbol{F}' = \begin{bmatrix} 0 & -\Delta T \cdot r_{\text{sphere}} & 0 \\ \Delta T \cdot r_{\text{sphere}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.3}$$

$$\boldsymbol{F} = \begin{bmatrix} \mathbf{0}^{3\times6} & F' \\ \mathbf{0}^{6\times6} & \mathbf{0}^{6\times3} \end{bmatrix}. \tag{5.4}$$

ΔT represents the time passed between two iterations. Additionally, the process control vector $\boldsymbol{u}_k$ holds the orientation and angular velocity, which are not calculated via the kinematic model. Thus, $\boldsymbol{u}_k$ and $\boldsymbol{G}$ are defined as:

$$\boldsymbol{u}_k = \begin{bmatrix}0, & 0, & 0, & \Delta\phi_{k,\text{cam}}, & \Delta\theta_{k,\text{cam}}, & \Delta\psi_{k,\text{cam}}, & \omega_{x,k,\text{cam}}, & \omega_{y,k,\text{cam}}, & \omega_{z,k,\text{cam}}\end{bmatrix}^T \tag{5.5}$$

$$\boldsymbol{G} = \begin{bmatrix} \mathbf{0}^{3\times3} & \mathbf{0}^{3\times6} \\ \mathbf{0}^{6\times3} & \boldsymbol{I}^{6\times6} \end{bmatrix}. \tag{5.6}$$

Note, that the camera measurements are used for $\boldsymbol{u}_k$. The reason for this is, that the calculated variances of the camera conclude, that the camera provides more accurate measurements overall, as seen in equations (5.13) - (5.14). The state error covariance matrix $\boldsymbol{P}_k$ converges with increasing iterations; hence, its initial value $\boldsymbol{P}_0$ is set to

$$\boldsymbol{P}_0 = 10 \cdot I^{9\times 9}, \tag{5.7}$$

containing sufficiently large variances on the main diagonal. With the RealSense T265 tracking camera and the two filtered IMU data providing direct pose estimates and raw inertial data, as stated in 4.3, the measurement vectors $\boldsymbol{y}_{k,\text{imu}}$ and $\boldsymbol{y}_{k,\text{cam}}$ are, like $\boldsymbol{x}_k$, made up of 6 elements for $\Delta\boldsymbol{\rho}_k$ and 3 elements for $\boldsymbol{\omega}_k$, resulting in the following two vectors:

$$\boldsymbol{y}_{k,\text{cam}} = \begin{bmatrix}\Delta x_k, & \Delta y_k, & \Delta z_k, & \Delta\phi_k, & \Delta\theta_k, & \Delta\psi_k, & \omega_{x,k}, & \omega_{y,k}, & \omega_{z,k}\end{bmatrix}^T_{\text{cam}} \tag{5.8}$$

$$\boldsymbol{y}_{k,\text{imu}} = \begin{bmatrix}\Delta x_k, & \Delta y_k, & \Delta z_k, & \Delta\phi_k, & \Delta\theta_k, & \Delta\psi_k, & \omega_{x,k}, & \omega_{y,k}, & \omega_{z,k}\end{bmatrix}^T_{\text{imu}}. \tag{5.9}$$

Note, that Δz_k of $\boldsymbol{y}_{k,\text{imu}}$ is always equal to 0, since the IMU filter does not provide data for this positional parameter. Hence, the measurement prediction vector $\boldsymbol{z}_{k+1}$ has the corresponding entries:

$$\boldsymbol{z}_{k+1} = \begin{bmatrix}\Delta x_{k+1}, & \Delta y_{k+1}, & \Delta z_{k+1}, & \Delta\phi_{k+1}, & \Delta\theta_{k+1}, & \Delta\psi_{k+1}, & \omega_{x,k+1}, & \omega_{y,k+1}, & \omega_{z,k+1}\end{bmatrix}^T. \tag{5.10}$$

Consequently, $\boldsymbol{H}$ must be a 9×9 identity matrix to translate the predicted state to the measurement prediction. For dimensioning the measurement noise covariance matrices $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, two approaches are applied. The first approach is to record the sensor's data while the SMMS is stationary in its 3D-printed mount. Then the variance for each state variable is calculated, according to section 6.1, using the recorded data set and formula (3.20), resulting in the following matrix entries:

$$\boldsymbol{R}_{\text{cam}} = \begin{bmatrix}
1.229\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 2.974\times10^{-10} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 8.383\times10^{-6} & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 3.712\times10^{-5} & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 4.393\times10^{-5} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 3.989\times10^{-5} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 4.191\times10^{-6} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4.838\times10^{-6} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.326\times10^{-6}
\end{bmatrix} \tag{5.11}$$

$$\boldsymbol{R}_{\text{imu}} = \begin{bmatrix}
2.876\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1.892\times10^{-8} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0.001 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 2.958\times10^{-3} & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1.947\times10^{-5} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 4.273\times10^{-4} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 6.131\times10^{-6} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.031\times10^{-6} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6.382\times10^{-6}
\end{bmatrix}. \tag{5.12}$$

The variance for the z-position of $\boldsymbol{R}_{\text{imu}}$ is set by hand, since the IMU filter does not provide z-axis data. The mean of the variances on the main diagonal of $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ gives an idea about the overall accuracy of the data provided by each sensor:

$$\overline{\sigma^2_{\text{cam}}} = 1.6189 \times 10^{-5} \tag{5.13}$$

$$\overline{\sigma^2_{\text{imu}}} = 4.278 \times 10^{-4}. \tag{5.14}$$

The results for the mean variances conclude, that the data, provided by the camera is more accurate, because $\overline{\sigma^2_{\text{cam}}} < \overline{\sigma^2_{\text{imu}}}$. However note, that for the calculation of $\overline{\sigma^2}$ for each sensor, more decimal places of the variances are used than given in equations (5.11) - (5.12) and that $\overline{\sigma^2_{\text{imu}}}$ is calculated using one less variance, due to the manually set z-axis variance.

Alternatively, the matrices entire main diagonals get tuned by hand. Therefore, the variance values are determined by using the knowledge, acquired while working with the system. These values are then tuned further and evaluated, until the KF output is sufficiently accurate. Pérez et al. show in [32], that even when the theoretical values of $\boldsymbol{Q}$ and $\boldsymbol{R}$ are calculated, tuning them manually often provides more accurate results. This method leads to these matrices:

$$\boldsymbol{R}_{\text{cam}} = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{5.15}$$

$$\boldsymbol{R}_{\text{imu}} = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.001 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}. \tag{5.16}$$

Note, that the variance for the position on the z-axis, which is the third entry of the main diagonal of $\boldsymbol{R}_{\text{imu}}$ and $\boldsymbol{R}_{\text{cam}}$, is different for each measurement source. Since it is assumed, that the SMMS moves on a plane surface, and the z-position, provided by the IMU-based pose estimate, is always zero, the IMU-based z-position is assumed to be more accurate, due to the cam providing deviating values from zero. Additionally, it is known, that the z-position and ψ-angle estimates by the camera suffer from inaccuracy on this system, compared to the corresponding values, based on the IMUs, due to the rolling motion. Therefore, the variance for the z-position is reduced in $\boldsymbol{R}_{\text{imu}}$ while the variances of the z-position and ψ-angle are increased in $\boldsymbol{R}_{\text{cam}}$, so the KF relies more on the IMU-based z-position and ψ-angle. This also implies, that the KF relies less on $\omega_{z,\text{imu}}$, because it has no direct influence on the IMU-based z-position and, thus, not on the KF estimate for the z-position, resulting in an increased variance value on the last entry of the main diagonal of $\boldsymbol{R}_{\text{imu}}$.

The next manually set matrix, is the process noise covariance matrix $\boldsymbol{Q}$, leading to the following entries:

$$\boldsymbol{Q} = 0.1 \cdot I^{9\times 9}. \tag{5.17}$$

The measurement noise vector $\boldsymbol{w}_k$ from equation (3.22) is neglected, since the camera already provides direct pose estimates, internally computed by the raw data, and the IMU's raw data is already filtered as stated in section 4.3.1. Additionally, the process noise vector $\boldsymbol{v}_k$ is neglected too, due to a simplified assumption about the environment's model with no noise caused by wind or other external influences.

5.2.2 Dynamic Variances

To ensure accurate pose estimation, even when the SMMS experiences fast rotations or poor visibility conditions, the measurement noise covariance matrices dynamically adjust to high angular velocities and low camera tracking confidence. The tracking confidence of the camera is computed internally by RealSense and passed to the user as an unsigned integer from 0 to 3, the representation of which is shown in table 5.1. In terms of adaptive noise covariance matrices, the research of Akhlagi et al. [1] show, that mainly the ratio of $\boldsymbol{Q}_k/\boldsymbol{R}_k$ contributes to the performance of their EKF implementation, rather than the individual values of $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$. Hence, their finding is transferred to this implementation of the KF, so the main focus lies on this ratio. The difference from the named study is, that their noise covariance matrix ratio remains the same for all iterations, only changing before another full set of iterations. In this thesis, however, the ratio changes each iteration, since the angular velocities in each iteration are different from those in the previous iteration (except for still standing). This results in changing entries on the main diagonal of the measurement noise covariance matrices $\boldsymbol{R}_{\text{k,imu}}$ and $\boldsymbol{R}_{\text{k,cam}}$, while keeping the process noise covariance matrix $\boldsymbol{Q}_k$ static, resulting in changing ratios of $\boldsymbol{Q}_k/\boldsymbol{R}_{\text{k,imu}}$ and $\boldsymbol{Q}_k/\boldsymbol{R}_{\text{k,cam}}$. While making $\boldsymbol{Q}_k$ adaptive, brings more robustness, especially against initialization errors, keeping $\boldsymbol{Q}_k$ static further decreases the complexity and thus increases the computational efficiency of the KF. Additionally, the systems dynamics are assumed to be constant, further supporting the decision to keep the process noise covariance matrix constant in this case.

For scaling, the current angular velocities $\boldsymbol{\omega}_k$ of each axis of the IMUs and the camera are taken into account. Therefore, two approaches are implemented to calculate a scaling factor $\mu_{\boldsymbol{\omega},\boldsymbol{k}}$ for each sensor in each iteration. The first approach makes use of the exponential function, denoted as e:

$$||\boldsymbol{\omega}_k|| = \sqrt{\omega_{x,k}^2 + \omega_{y,k}^2 + \omega_{z,k}^2} \tag{5.18}$$

$$\mu_{\boldsymbol{\omega},\text{k,imu}} = e(||\boldsymbol{\omega}_{\text{k,imu}}||) \tag{5.19}$$

$$\mu_{\boldsymbol{\omega},\mathrm{k,cam}} = e(||\boldsymbol{\omega}_{\mathrm{k,cam}}||). \tag{5.20}$$

Thus, the length of $\boldsymbol{\omega}_k$ (5.18) determines the scaling factor. The exponential function is used to ensure $\mu_{\omega,k} \geq 1.0$, since $||\boldsymbol{\omega}_k|| \geq 0.0$, which becomes clear when looking at the graph of the exponential function in figure A.1a. The graph also depicts, that the values, returned by the exponential function, grow very big, very fast. Therefore, high angular velocities cause the entries of $\boldsymbol{R}_k$ of the corresponding sensor to be scaled quite aggresively. That is why the scaling factor for each measurement noise covariance matrix gets limited by a lower bound of 1.0 and an upper bound of 1000.0. Very large variance entries in the measurement noise covariance matrices result in the KF relying heavily on the state prediction. Since the model, the prediction is based on, is kept rather simple, limiting the scaling factors makes the KF more subsceptible to unpredictable motion.

Approach two does not use the exponential function, but rather the natural logarithm denoted as ln. To make the natural logarithm a suitable function for returning a scaling factor, some adjustments are required:

$$\mu_{\boldsymbol{\omega},\mathrm{k,imu}} = ln(||\boldsymbol{\omega}_{\mathrm{k,imu}}|| + 1) + 1. \tag{5.21}$$

$$\mu_{\boldsymbol{\omega},\mathrm{k,cam}} = ln(||\boldsymbol{\omega}_{\mathrm{k,cam}}|| + 1) + 1. \tag{5.22}$$

The adjustments are necessary, to avoid a scaling factor of 0.0 and to ensure a scaling factor of 1.0 for $||\boldsymbol{\omega}_k|| = 0.0$. This becomes clear, when looking at the graph for the adjusted ln plot in figure A.1b. In contrast to the exponential method, this second approach serves as a less aggressive variant and, thus, as a compromise between static measurement noise covariance matrices and approach using e.

At last, the resulting scaling factor $\mu_{\mathrm{k,cam}}$ of either approach is adjusted by the camera's tracking confidence value, which is translated to a confidence factor $\mu_{\mathrm{k,conf}}$ as shown in table 5.1. The confidence factor depends on multiple elements, that influence pose estimation. For one the lighting conditions of the environment. On the one hand: if there is not enough light, the camera is unable to track enough visual features. Low light additionally causes higher sensor noise. On the other hand: if there is too much light, pointing directly to the camera, visual features are also hard to detect. Another factor is high acceleration of the camera, causing inaccurate measurements of the internal IMU and break assumptions of RealSense's fusion algorithm. Furthermore, an environment with few visual features, e.g., a blank, textureless wall, or too many features cause lower tracking confidence [17]. This results in the following final scaling factors:

$$\mu_{\mathrm{k,imu}} = \mu_{\boldsymbol{\omega},\mathrm{k,imu}} \tag{5.23}$$

$$\mu_{\mathrm{k,cam}} = \mu_{\boldsymbol{\omega},\mathrm{k,cam}} \cdot \mu_{\mathrm{k,conf}} \tag{5.24}$$

and, therefore, to the following scaled measurement noise covariance matrices:

$$\boldsymbol{R}_{\mathrm{k,imu,scaled}} = \mu_{\mathrm{k,imu}} \cdot \boldsymbol{R}_{\mathrm{k,imu}} \tag{5.25}$$

$$\boldsymbol{R}_{\mathrm{k,cam,scaled}} = \mu_{\mathrm{k,cam}} \cdot \boldsymbol{R}_{\mathrm{k,cam}}. \tag{5.26}$$

Table 5.1: RealSense T265 Tracking Confidence

Value	**Tracking Confidence**	**Translated Factor $\mu_{\mathbf{conf}}$**
0	Failed	1000
1	Low	100
2	Medium	10
3	High	1

5.2.3 Software Structure

In section 3.6 it is stated, that the KF is composed of two steps: a prediction step followed by an update step. Each step of the algorithm is implemented as a function, with the necessary parameters passed to it. These functions then update a global nine-dimensional vector, which represents $\boldsymbol{x}_k$. The prediction step gets ΔT and $\boldsymbol{u}_k$ passed as arguments and calculates the prediction of the current state. However, the system brings two sources, providing a measurement vector $\boldsymbol{y}_k$. One by the camera, and one by the filtered IMU data. Therefore, the update step is split into two sequential steps, where each step corrects the current state vector estimate. Petovello et al. show in [30] that this approach does not suffer from less reliability in the algorithm's output than a simultaneous update step, while simplifying the software and improving its efficiency. The first update step uses $\boldsymbol{y}_{k,\text{imu}}$ to correct the state prediction. Since the KF works with the delta values of the $\boldsymbol{\rho}$ components, these values must be computed first, by using equation (3.14). To ensure equation (3.27), the corresponding measurement noise covariance matrix $\boldsymbol{R}_{\text{imu}}$ is passed to the update function as an argument, next to $\boldsymbol{y}_{k,\text{imu}}$. Then the second update step is performed by correcting the updated state from the first update step, using $\boldsymbol{y}_{\text{k,cam}}$ and $\boldsymbol{R}_{\text{cam}}$. The measurement noise covariance matrices need to be passed to the functions, due to the possibility of scaling the matrices beforehand, according to section 5.2.2. At last, it is necessary to apply the delta output of the KF pose estimate to the total pose estimate. This is done using equation (3.15), after which the total pose estimate is published. To get an overview of the software structure, the code hast been simplified as pseudocode in 1. For further details, consider accessing the project on GitHub [4].

Algorithm 1: Simplified Kalman Filter Software Structure

```
1  $\boldsymbol{x}_k \leftarrow$ initialize global state, which holds parameters according to equation (5.1);
2  $\boldsymbol{\rho}_k \leftarrow$ total pose estimate, which is updated iteratively and published;
3  Function Main():
4      initialize static matrices and vectors according to 5.2.1 and call ApplyAndPublishKF
         when $\boldsymbol{data}_{\text{imu}}$ arrives via callback method
5  Function ApplyAndPublishKF($\boldsymbol{data}_{imu}, \Delta T$):
6      $\boldsymbol{data}_{\text{cam}} \leftarrow$ interpolated according to 5.1;
7      $\boldsymbol{y}_{k,\text{cam}} \leftarrow$ holds calculated and transformed $\Delta\boldsymbol{\rho}_{k,\text{cam}}$ values and $\boldsymbol{\omega}_{k,\text{cam}}$;
8      $\boldsymbol{y}_{k,\text{imu}} \leftarrow$ holds calculated and transformed $\Delta\boldsymbol{\rho}_{k,\text{imu}}$ values and $\boldsymbol{\omega}_{k,\text{imu}}$;
9      $\boldsymbol{u}_k \leftarrow$ holds parameters according to (5.5);
10     $\boldsymbol{R}_{\text{k,imu}} \leftarrow$ adjusted according to (5.25), if desired;
11     $\boldsymbol{R}_{\text{k,cam}} \leftarrow$ adjusted according to (5.26), if desired;
12     $\boldsymbol{x}_k =$ PredictionStep($\Delta T, \boldsymbol{u}_k$);
13     $\boldsymbol{x}_k =$ UpdateStep($\boldsymbol{y}_{k,imu}, \boldsymbol{R}_{k,imu}$);
14     $\boldsymbol{x}_k =$ UpdateStep($\boldsymbol{y}_{k,cam}, \boldsymbol{R}_{k,cam}$);
15     $\boldsymbol{\rho}_k \leftarrow$ apply KF delta estimate to total pose estimate according to (3.15) and
         publish;
16 Function PredictionStep($\Delta T, \boldsymbol{u}_k$):
17     equations (3.23)-(3.25)
18     return $x_k$;
19 Function UpdateStep($\boldsymbol{y}_k, \boldsymbol{R}_k$):
20     equations (3.26)-(3.30)
21     return $x_k$;
```

Chapter 6

Methodology

6.1 Covariance Matrix Calculation

Next to manually tuning the entries of the main diagonal of $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, the variance values for each state parameter are calculated. To acquire the data for this calculation and ensure as accurate values as possible, the SMMS is put into its mount, and a bagfile is recorded for 12.8 sec. While recording, the movement around the SMMS is minimized to prevent falsification of the measurements. Now the recorded values for $\boldsymbol{\rho}_k$ and $\boldsymbol{\omega}_k$ of each sensor are used to determine the variance of each entry of $\boldsymbol{x}_k$. Therefore, the *matlab* script A is used, implemented in the open source software *GNU Octave*. This script uses a .csv file as input and then calculates the variance for the specified column entries, according to equation (3.20). Note, that the script assumes the pose and angular velocity parameters to be stored in the same .csv file. Other than that, only the number of entries and the input file must be changed, depending on the sensor.

6.2 Ground Truth

For a conclusive evaluation of the pose estimation provided by the KF, the point cloud, based on the calculated trajectory, is compared to a point cloud, based on a ground truth trajectory. This ground truth trajectory must have a higher accuracy than the trajectories it is compared with. Therefore, the recorded trajectories are enhanced by applying *6-DOF Semi-Rigid SLAM for Mobile Scanning*, proposed by Elsberg et al. [14]. Here, the recorded pose estimates of the tracking camera and the pose estimates based on the filtered IMU data are combined into a sequentiel pose estimate. Furthermore, the estimate difference of each timestep is calculated. Then, a single octree, representing the entire point cloud is used to do fast nearest neighbor search. As a result, the closest measurement is found for each measured point. Using this and a set of further equations, as well as the covariances of the measurement sources, estimates of pose differences are computed. These estimates are optimized, by maximizing the likelihood of the pose estimates with corresponding covariances, until the trajectory change reaches the desired precision. For a more detailed description of the algorithm, refer to the study of Elsberg et al. [14]. Consequently, this algorithm provides a highly accurate trajectory estimation and, thus, a closer representation to the actual environment.

6.3 3D Toolkit

The 3D Toolkit (3DTK) is a program developed by a collaboration of institutions, some of which are the Julius Maximilians University Würzburg, the University of Osnabrück, and the Jacobs University. It is described as follows: "The 3D Toolkit provides algorithms and methods to process 3D point clouds. It includes automatic high-accurate registration (6D simultaneous localization and mapping, 6D SLAM) and other tools, e.g., a fast 3D viewer, plane extraction software, etc. Several file formats for the point clouds are natively supported, new formats can be implemented easily" [28]. 3DTK is used for multiple tasks in the scope of this thesis in cooperation with ROS tools.

For one, the 3DTK tool *ros_listener* creates a set of .3d files with the corresponding .pose files, which are needed to create a point cloud. The Kalman filter algorithm is able to calculate and publish pose estimates, when playing a .bag file, which stores the data provided by the sensors of a recorded experiment with the SMMS. These published KF pose estimates and the published Hesai Pandar laser scan data from the .bag file are then used by the ros_listener to create the .3d and .pose files. To achieve this, the user specifies a ROS topic for the pose stream and a ROS topic for the laser scan stream, which is subscribed to by the ros_listener. The output is then stored in the specified folder. To ensure flawless performance of the next tools, the point clouds are reduced by *scan_red*, which splits the point clouds into so-called octrees up to a specified limit. Every of these octrees then stores a specified number of points. Depending on the parameters used, this tool is able to reduce the computational efforts for visualization and further processing of a point cloud significantly. Next, the tool *imshow* allows the portrayal of 3D point clouds, which makes a first visual analysis and the export of the point cloud's visualization possible. Since the point clouds are created using .pose and .3d files, the resulting point cloud's accuracy largely depends on the provided poses' accuracy. Using the ground truth point cloud, generated as stated in section 6.2, the pose estimates and their resulting point cloud are quantified with another tool from 3DTK. This tool is *scan2scan_distance* and its purpose is to determine the point-to-point error between a ground truth point and the corresponding point of another point cloud. Note, that scan2scan_distance takes two .3d files as input: one for the ground truth point cloud and one for the point cloud, that is compared with ground truth. To generate a single .3d file out of the many .3d files the tool *condense* is used. Furthermore it is necessary to specify the radius, which scan2scan_distance searches around a ground truth point for the corresponding point of the second point cloud. The output of scan2scan_distance is a single point cloud, that now encodes the point-to-point errors of the two compared point clouds. The differences can be visualized in imshow by colouring the points, based on their reflectance, so that points with high reflectance represent a large point-to-point error and points with low reflectance denote small point-to-point error. Therefore, the generated point clouds, based on the KF pose estimates, are compared to the ground truth point clouds to determine the pose estimation accuracy. Finally, the output point cloud of scan2scan_distance is used as input for a tool called *histogram.cc*, which analyses the point cloud to determine characteristics, e.g., the mean point-to-point error of all compared points. This analysis is then used as input for *histogramcolor.plot*, which generates a histogram, presenting which differences occurs how many times. These last two tools are not directly part of the 3D Toolkit but add-on tools, provided by the chair of robotics of the Julius Maximilians University Würzburg.

Chapter 7

Evaluation & Experiments

The first section of this chapter describes the experiments used to evaluate the pose estimation of the KF in section 7.1. Additionally, the results of the different KF implementations are compared with each other and with the previously developed Delta filter for the same system in section 7.2. Since the main purpose of the SMMS is the creation of a 3D map of its environment, the KF pose estimation evaluation is done via point cloud analysis, using the tools described in section 6.3. To understand the presented results, possible causes for errors within the system and the environment are described in section 7.2.3.

7.1 Experiment Setup

The SMMS is moved along different trajectories within the robotics hall of the University of Würzburg to record the sensors' data and to later evaluate the pose estimate outputs of the KF. Before each recording, the IMUs are calibrated, hence, the SMMS sits stationary in its mount for a few seconds. Then, the SMMS is placed on the ground and is set in motion. The sphere is moved manually with a rolling motion, while the IMUs, the tracking camera and the LiDAR collect data, which gets stored in a .bag file. Note that while moving the system on the ground, the camera's and LiDAR's visibility must remain as unhindered as possible to ensure representing results. The recordings are done in good lighting conditions to ensure high tracking confidence of the RealSense T265. Figure 7.1 depicts the environment of the experiments. Next to many rather thin structures like chair- and table-legs, there are several glass elements that get penetrated by the LiDAR's emitted and detected laser, e.g., windows, doors and shelves. These need to be taken into account when evaluating the generated point clouds. Since the SMMS is moved manually, there is motion around it, which gets excluded from the generated point clouds, by ignoring all measured points with a distance less than 1.0 m in the generation process. While recording, no further movement of the environment is present to prevent falsification of the results.

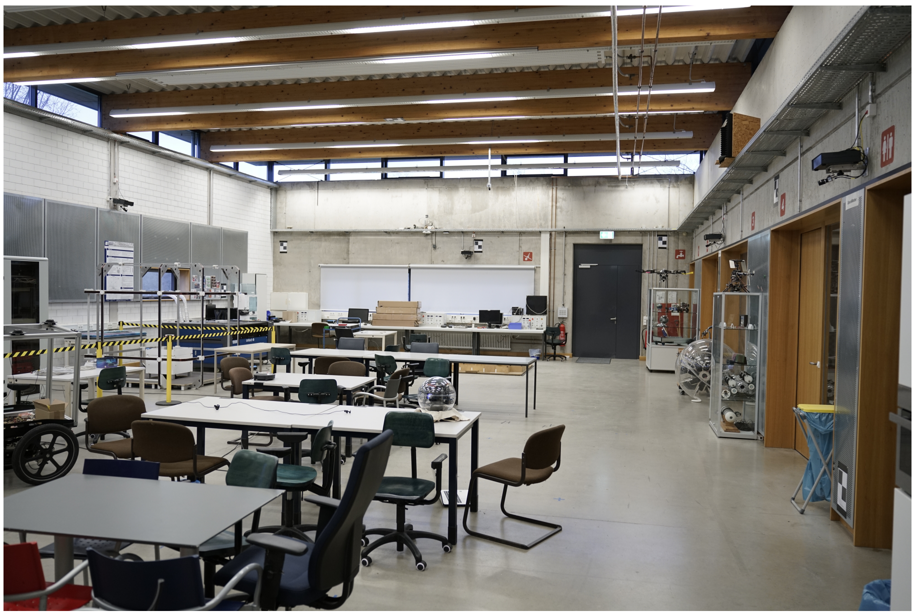
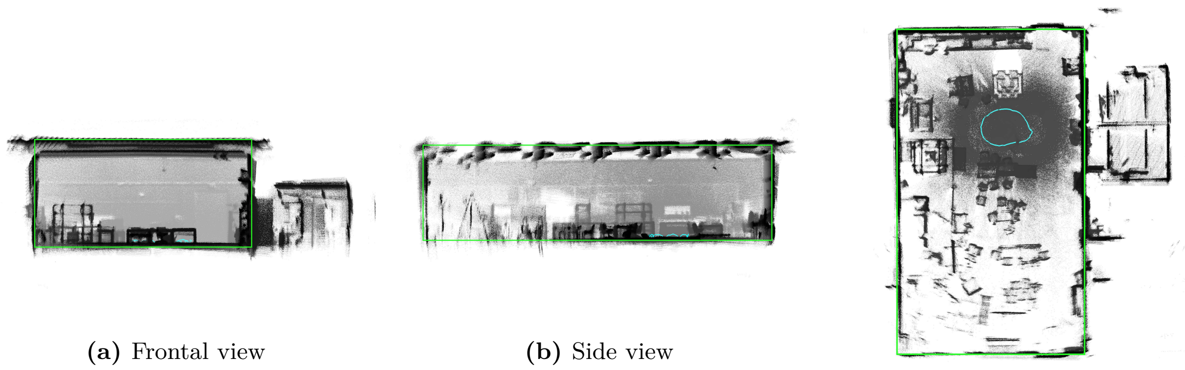


Figure 7.1: The experiment environment, where all trajectories for the evaluation were recorded.

7.2 Evaluation

In this section, the pose estimation of different implementations of the KF are evaluated by comparing point clouds, based on their estimated trajectory, with the corresponding ground truth point cloud. Especially the performance of the KF with adaptive measurement noise covariance matrices, compared to the conventional KF, is of special focus. The comparison of the two KF implementations with static measurement noise covariance matrices is done using a circular trajectory. This circular trajectory is also evaluated for the measurement source estimations of the camera and IMU by themselves, as well as the Delta filter, to get a better picture of the KF's performance. Then, another larger trajectory is used to compare the different KF implementations against each other in a more challenging scenario. For the sake of the evaluations extend, the second, larger trajectory is not evaluated against the Delta filter and the measurement source estimations. Note, that in general, when speaking of drift around the ϕ-axis, it corresponds to drift depicted in the frontal view of a point cloud, i.e., the view parallel to the x-axis. Drift around the θ-axis corresponds to drift, depicted in the side view (parallel to the y-axis), and drift around the ψ-axis corresponds to drift, seen in the top view (parallel to z-axis). To draw further conclusions about the performance of the KF, the mean point-to-point error (MPTPE) is presented for the different estimators. Additionally, multiple point clouds are presented: point clouds, depicting the environment based on the pose estimates, as well as point clouds, depicting the point-to-point errors between the ground truth- and estimation-based point cloud.



(a) Frontal view **(b)** Side view **(c)** Top view

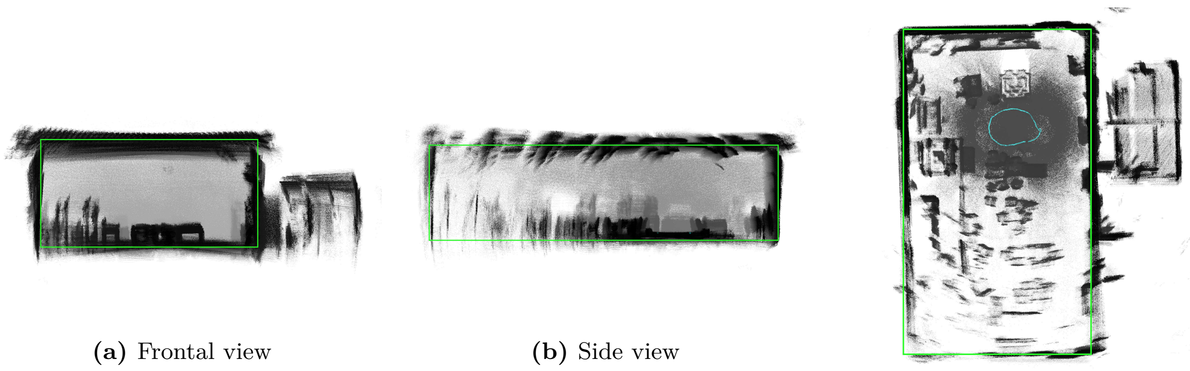
Figure 7.2: Point cloud, based on the ground truth estimation of a circular trajectory, calculated as described in 6.2. To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

7.2.1 Linear Kalman Filter with Static Measurement Noise Covariance Matrices

At first, the KF implementations with static measurement noise covariance matrices $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ are evaluated. In section 5.2.1 the two approaches for determining $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ are presented. Now, the sections 7.2.1.1 and 7.2.1.2 evaluate these two approaches. The approach with the more accurate result is then evaluated more thoroughly and also determines, how $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ are initialized in implementations with adaptive measurement noise covariance matrices. The comparison of the measurement noise covariance matrices initialization is done visually, without point-to-point errors and with a single, circular trajectory, since the results are rather clear. This circular trajectory has a radius of about 2.5 m and was recorded over a timespan of 1.04 minutes. The recording started and ended on the same spot, marked by a cross on the floor. In the point clouds, the trajectory is seen as a light-blue path. Since the trajectory was recorded in the upper half of the robotics hall, from the top views' perspective 7.2c, the lower half has a lower resolution, i.e., a lower point density. The green rectangles serve as a reference, making the detection of drift or general pose estimation errors more perceivable and were edited onto the point cloud after its creation. The point clouds, based on the IMU-only pose estimation, camera-only pose estimation and the ground truth poses, are depicted in figures 7.2 - 7.4. Here, it is visible, that the IMU-only-based pose estimation, seen in figure 7.3 provides point clouds with less drift in all three axes, compared to the point clouds, based on the camera-only pose estimation, seen in figure 7.4. Furthermore, the IMU-based pose estimation provides a circular trajectory, closer in shape and size to the ground truth trajectory in figure 7.2.

7.2.1.1 Calculated Measurement Noise Covariance Matrices

Using the approach of calculating the variances, as described in 6.1, results in the point cloud depicted in figure 7.5. However, using calculated variances, there already exists a significant drift, present in all axes, and especially in the ϕ- and ψ-axes. Here the walls and roof spread



(a) Frontal view **(b)** Side view **(c)** Top view

Figure 7.3: Point cloud, based on the IMU-only pose estimation of a circular trajectory, computed as proposed by Zevering et al. in [46]. To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

rather widely around the reference rectangle. It is also recognizable, that the trajectory estimate by the KF is not a closed circle but has an offset between the start and end point. Overall, the resulting point cloud visually does not exceed the accuracy of the trajectory, provided by the filtered IMU data in figure 7.3.

7.2.1.2 Manually Tuned Measurement Noise Covariance Matrices

The second approach for static measurement noise covariance matrices is manually tuning $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, as shown in equations (5.11) - (5.12). Using this approach, the KF pose estimation for the same recording, results in the point cloud, as seen in figure 7.6. Here, the trajectory is an almost closed circular path, which is an improvement compared to the result of section 7.2.1.1. Also, it is recognizable in figure 7.6a and 7.6c, that the drift of the ϕ- and ψ-axes is noticeably reduced. The width of the walls and roof stays a lot closer to the reference rectangle and ground truth. This is also backed by the fact, that details like chairs, tables and shelves are less distorted. However, there is no significant change in the side view when comparing figure 7.5b and 7.6b and also no real visual improvement to the IMU-only pose estimation-based point cloud in figure 7.3, but the results look rather similar. When comparing the resulting point cloud to the point cloud, based on camera-only pose estimation in figure 7.4, the KF with manually tuned $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ provides a point cloud, much closer to the ground truth point cloud, presented in figure 7.2. Therefore, the conclusion is drawn, that the manually tuned $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ cause the KF to provide more accurate pose estimates. Hence, this approach is used for the measurement noise covariance matrices' initialization in the upcoming KF implementation evaluations, and is now referred to as *static KF*. Longer trajectories suffer from even more drift and motion distortion; thus, choosing the more accurate approach is crucial to estimating poses as accurately as possible.

Now that the approach for the initialization of the measurement noise covariance matrices is determined, the performance of this static KF implementation is evaluated in more detail against the ground truth point cloud, as well as point clouds, based on the trajectory provided by the

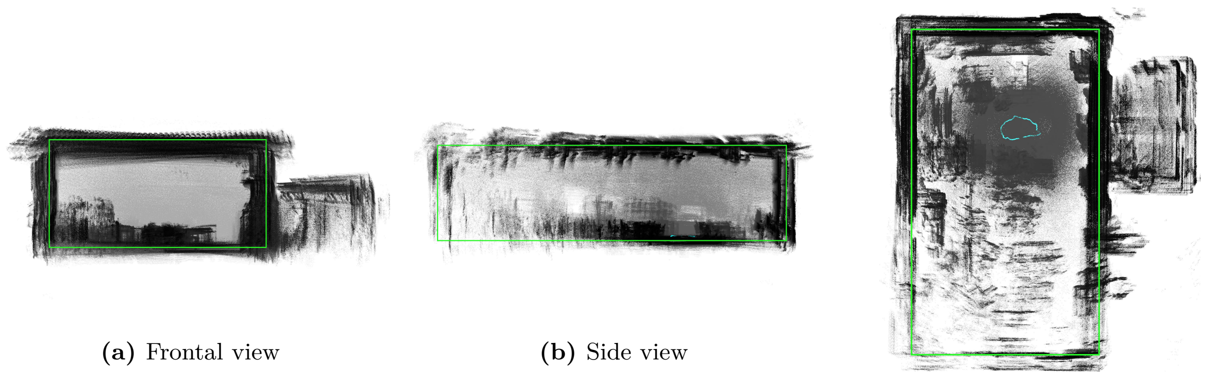

(a) Frontal view (b) Side view (c) Top view

Figure 7.4: Point cloud, based on the RealSense T265 tracking camera's pose estimation of a circular trajectory. To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

RealSense T265 camera and the filtered IMU data. The following MPTPE's are summarized in table 7.1, for the different estimators and the circular trajectory. The static KF's MPTPE lies at 6.579 cm, which is a large improvement to the camera-only pose estimation, which has a MPTPE of 22.306 cm. That relatively speaking inaccurate camera pose estimation contradicts the assumption, drawn in section 5.2.1, that smaller calculated variances result in more accurate pose estimation. Since the estimation of the camera is much less accurate, the KF is not able to exceed the accuracy of the IMU-only pose estimation of Zevering et al. [46] with a MPTPE of 5.691 cm. Their MPTPE difference lies at 0.888 cm. However, the static KF provides a more accurate pose estimation than the Delta filter, which has a MPTPE of 9.021 cm and also suffers from the camera's inaccurate pose estimation. The static KF's point-to-point errors of the circular trajectory are visualized on the left side of figure 7.13. These and the MPTPE demonstrate, that the static KF suffers from a rather small global drift, causing the most point-to-point errors > 10 cm to be located around straight elements, like walls, tables and the roof. Though, in general, it provides an accurate point cloud of the environment.

In addition to evaluating the circular trajectory, the different Kalman filter implementations are evaluated against ground truth for a more challenging, larger trajectory, to determine the most accurate KF implementation. The recording for this trajectory took 4.31 minutes, more than four times longer than the recording of the circular trajectory. This trajectory also started and ended in the same spot, marked with a cross on the floor of the robotics hall. The light-blue trajectory of the ground truth point cloud in figure 7.7 reflects this, by being a closed trajectory. In figure 7.7, the point cloud, based on the ground truth trajectory is depicted, whereas figure 7.8 shows the point cloud, calculated by the static KF. The drift in all three axes is now much more significant, especially in the ψ-axis. Furthermore, details like chairs, tables and doors are also rather distorted. However, the trajectory itself is recognizable, with a slight offset between the start and end points. Table 7.2 summarizes the MPTPE's of the large trajectory's pose estimation by the static KF and two implementations, presented in section 7.2.2. The MPTPE of the static KF increased by 13.538 cm, compared to the circular trajectory, and now lies at

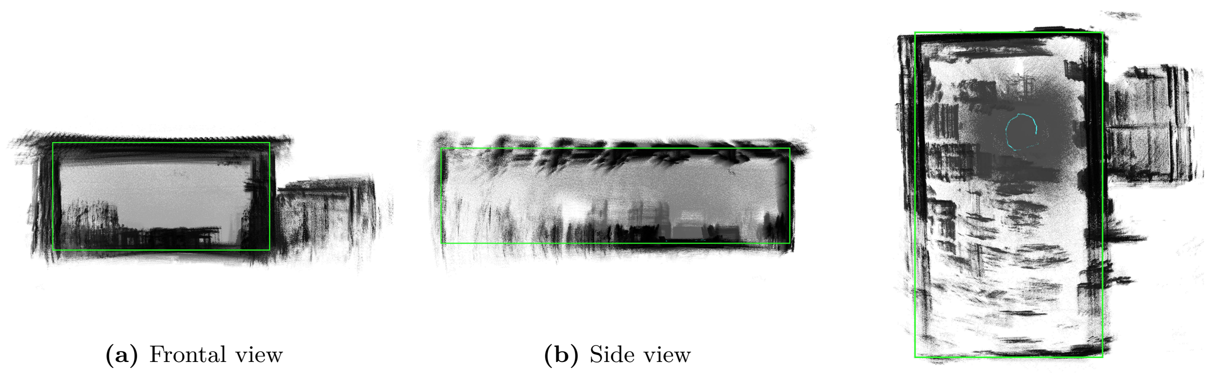

(a) Frontal view **(b)** Side view **(c)** Top view

Figure 7.5: Point cloud, based on the KF pose estimation of a circular trajectory, using static $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, containing calculated variances, according to equations (5.11) - (5.12). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

20.117 cm. On the left side of figure 7.14, the point-to-point errors are visualized for the large trajectory. The slight drift of the circular trajectory is now much more obvious. Again, the straight elements show the most drift, but now consist of much more points with a distance to ground truth > 20 cm, which is also visible in the histogram below. Thus, the conclusion is drawn, that the static KF suffers from global drift, which increases with increasing size and complexity of the trajectory.

7.2.2 Linear Kalman Filter with Dynamic Measurement Noise Covariance Matrices

Now that the static $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ are initialized and the static KF's pose estimation is evaluated, the next step is to evaluate the performance of the KF with adaptive measurement noise covariance matrices using the factors introduced in section 5.2.2. Like in section 7.2.1.2, the pose estimation accuracy is evaluated for the shorter, circular trajectory and the larger trajectory. The KF with dynamically adjusting measurement noise covariance matrices is referred to as *dynamic KF* from now on.

7.2.2.1 Exponential Function Factor

At first, the approach, using the exponential function is evaluated for the circular trajectory. The resulting point cloud is depicted in figure 7.9. When visually comparing the top view (7.9c) to the top view of the static KF (7.6c), the exponential factor reduces the drift of the ψ-axis. This becomes clear when looking at the width of the walls, which are more narrow and, thus, closer to the ground truth top view in figure 7.2. Additionally, the drift in the other two axes is slightly reduced, when looking at the frontal and side views. When taking the MPTPE in table 7.1 into account, it gets even clearer, that the dynamic $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$ matrices reduce the KF's global drift. The MPTPE for the dynamic KF, using the exponential approach, lies

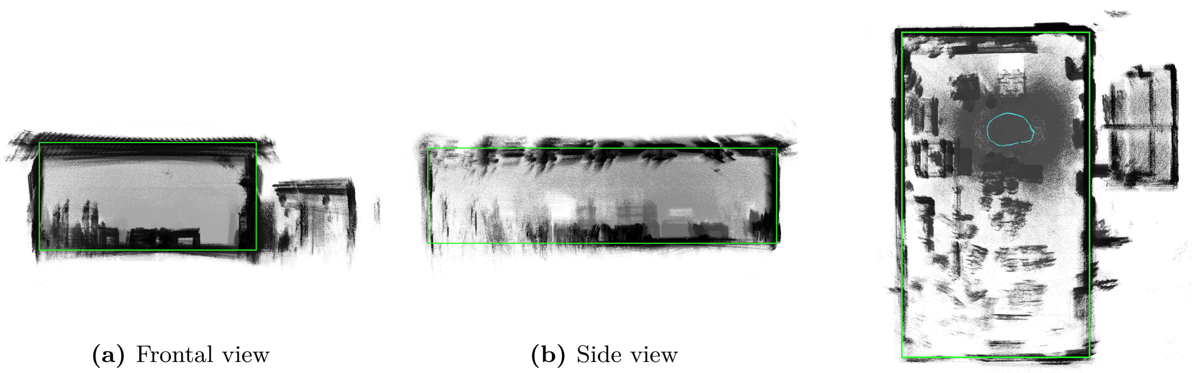
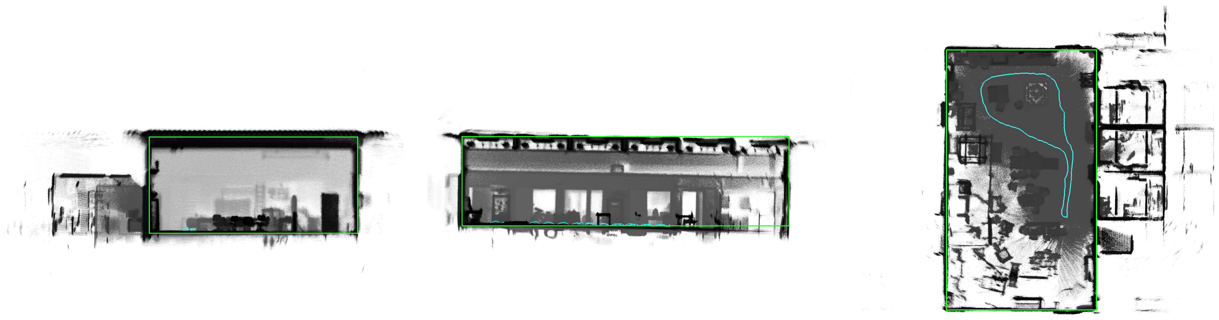

(a) Frontal view **(b)** Side view **(c)** Top view

Figure 7.6: Point cloud, based on the KF pose estimation of a circular trajectory, using static $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, containing manually tuned variances, according to equations (5.15) - (5.16). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

at 6.579 cm, reducing the static KF's MPTPE by 0.552 cm. However, the KF is still not able to outperform the IMU-only-based pose estimation of the circular trajectory, but it reduced the MPTPE difference to 0.336 cm, consequently outperforming all estimators, except for the IMU-only approach. Figure 7.13 directly compares the static KF with the KF implementation of this section. On the right side, the point-to-point errors of the dynamic KF with the exponential scaling factor are visualized. The dynamic KF's point clouds illustrate the decrease of point-to-point errors > 10 cm around straight elements. Additionally, the histogram shows the increase of the occurrences of distances to ground truth < 5 cm.

Next, the larger trajectory is evaluated. For this trajectory the static KF suffered from significant drift in all axes, as well as much distortion regarding the walls and detailed features of the robotics hall. The introduced exponential function scaling factor, based on angular velocities and the tracking confidence improves this behavior in some aspects. When comparing the static KF's resulting point cloud in figure 7.8 with the result in figure 7.10, it becomes clear, that the distiortion of walls is reduced in all three viewing angles. Furthermore, the robotic hall's furniture is also less distorted, when viewing them from the front and side in figures 7.10a and 7.10b. However, there is now a more significant drift in the θ-axis, visible in figure 7.10b. The MPTPE is reduced by 2.183 cm, compared to the static KF and now lies at 17.934 cm, thus, exceeding the other KF implementation's accuracy. In figure 7.14, the large trajectory's point-to-point errors are compared between the static KF and this dynamic KF approach. The scaling factor decreases the number of point-to-point errors > 40 cm and makes them spread less widely around the walls of the robotics hall. Again, the histogram indicates the increase of the occurrences of distances to ground truth < 5 cm, which now surpasses the static KF's maximum value of about 40000 to about 45000.



(a) Frontal view **(b)** Side view **(c)** Top view

Figure 7.7: Point cloud, based on the larger trajectory's ground truth estimation, generated as described in section 6.2. To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

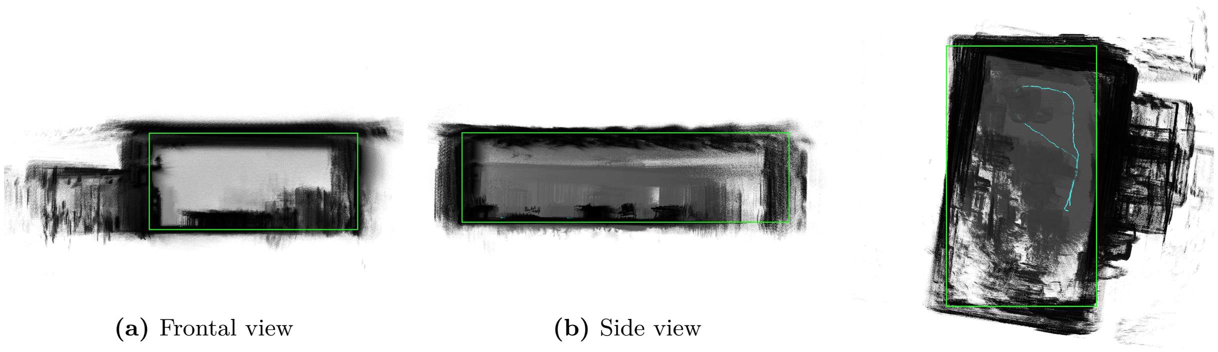
Table 7.1: Comparison of the MPTPE of the three different KF implementations with the Delta filter (DF) and their measurement source estimations for the circular trajectory.

Estimator	IMU	Camera	static KF	dynamic KF (e)	dynamic KF (ln)	DF
MPTPE [cm]	5.691	22.306	6.579	6.027	6.033	9.021

7.2.2.2 Logarithmic Function Factor

Next, the logarithmic approach for the dynamic KF is evaluated, which adjusts the measurement noise covariance matrices less radically, than the exponential function-based scaling factor. The application of this factor results in the point cloud, seen in figure 7.11 for the circular trajectory. Here, the results are very similar to the result of the previous approach in section 7.2.2.1. The distortion in all axes is slightly reduced, compared to the static KF. When comparing the MPTPE, the logarithmic scaling factor does not provide more accurate pose estimates than the exponential scaling factor, with a MPTPE of 6.033 cm. However, the difference only lies at 0.006 cm.

The point cloud for the larger trajectory in figure 7.12 also deviates from the previous dynamic approach. The walls are more distorted and wider from all three perspectives, but not as much as in the static KF's point cloud in figure 7.8. Furthermore, the drift in the θ-axis decreases, compared to the xponential scaling approach, as seen in figure 7.10b. This shows, that the scaling factor, based on the natural logarithmic function serves as a compromise between the static KF and the dynamic KF using a scaling factor, based on the exponential function. The MPTPE lies at 18.008 cm, thus, 0.074 cm larger than the exponential scaling approach. A possible reason for this is, that the logarithmic scaling factor reduces drift in one axis but increases drift in the other two axes. Thus, the overall best KF implementation is the dynamic KF, using the exponential scaling factor, which provides the best results for both trajectories, compared to the other KF implementations.

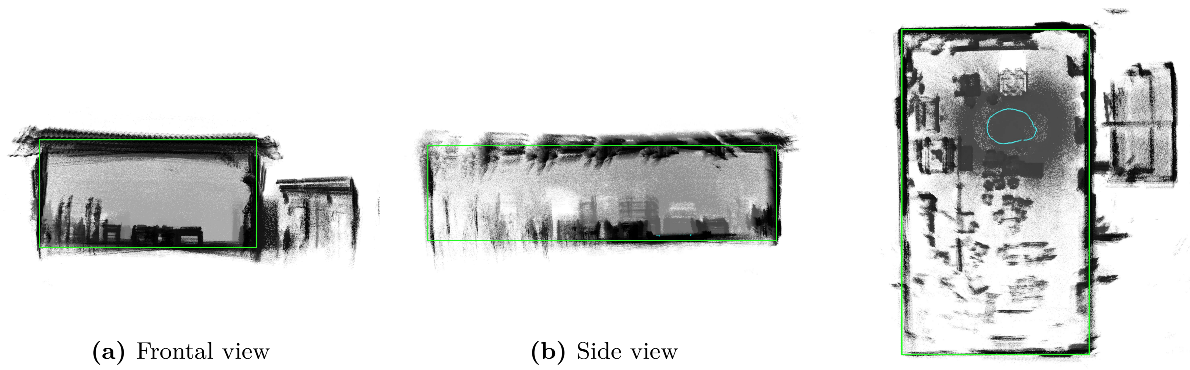


(a) Frontal view (b) Side view (c) Top view

Figure 7.8: Point cloud, based on the KF pose estimation of the larger trajectory, using static $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, containing manually tuned variances, according to equations (5.15) - (5.16). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

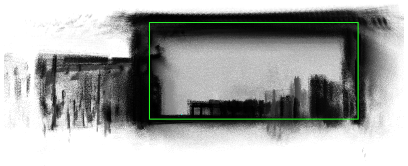
Table 7.2: Comparison of the MPTPE of the three different KF implementations for the large trajectory, to evaluate the introduced scaling factors for a longer, more challenging recording.

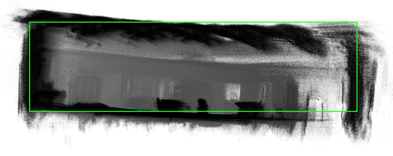
Estimator	static KF	dynamic KF (e)	dynamic KF (ln)
MPTPE [cm]	20.117	17.934	18.008



(a) Frontal view (b) Side view (c) Top view

Figure 7.9: Point cloud, based on the KF pose estimation of a circular trajectory, using dynamically adjusting $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, according to the exponential approach (5.19) - (5.20) and (5.23) - (5.26). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.


(a) Frontal view


(b) Side view

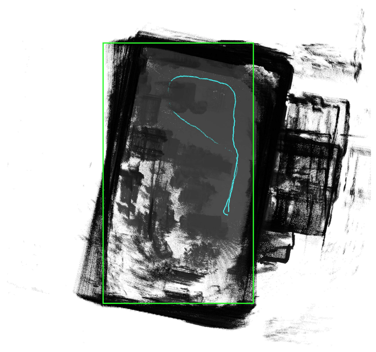
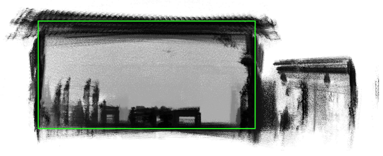
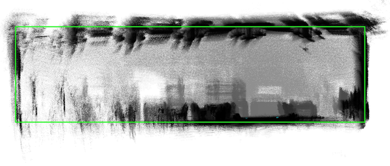

(c) Top view

Figure 7.10: Point cloud, based on the KF pose estimation of the larger trajectory, using dynamically adjusting $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, according to the exponential approach (5.19) - (5.20) and (5.23) - (5.26). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.


(a) Frontal view


(b) Side view

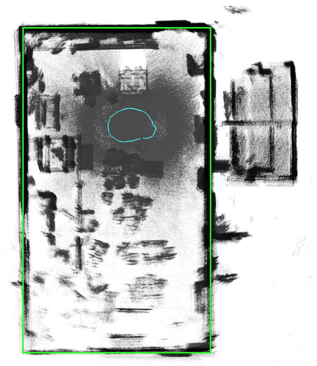

(c) Top view

Figure 7.11: Point cloud, based on the KF pose estimation of a circular trajectory, using dynamically adjusting $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, according to the logarithmic approach (5.21) - (5.26). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

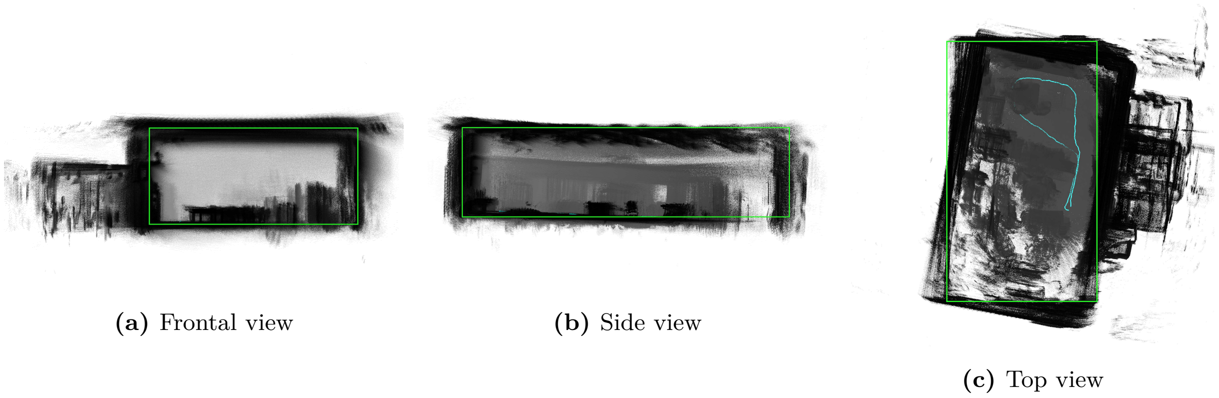
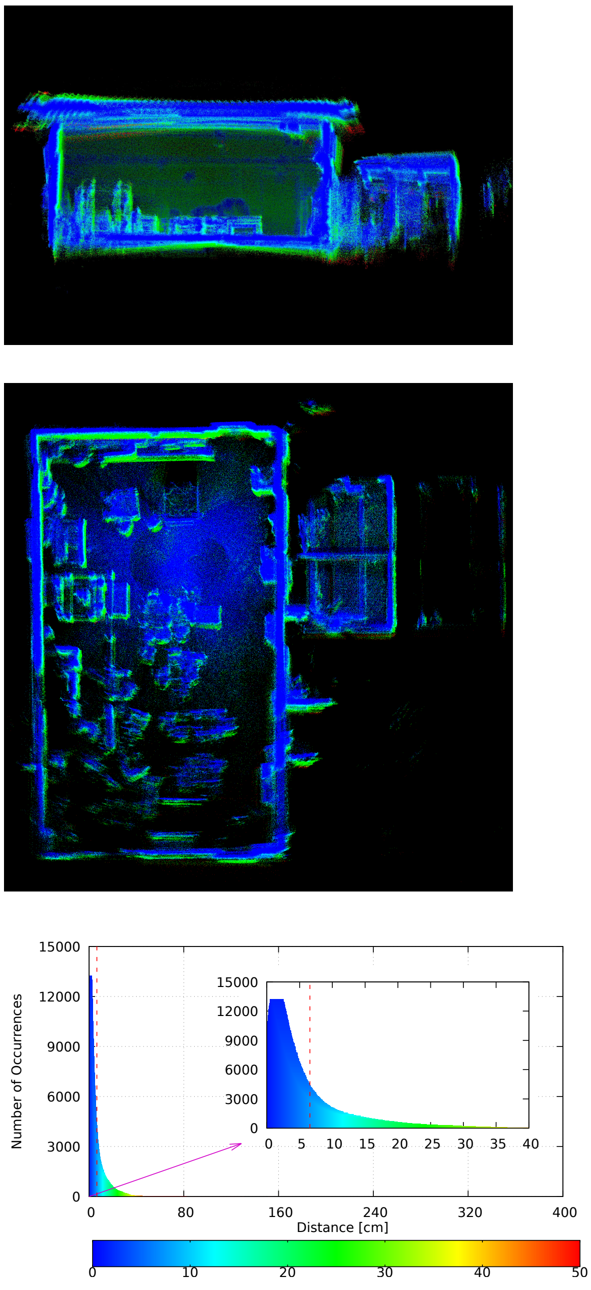
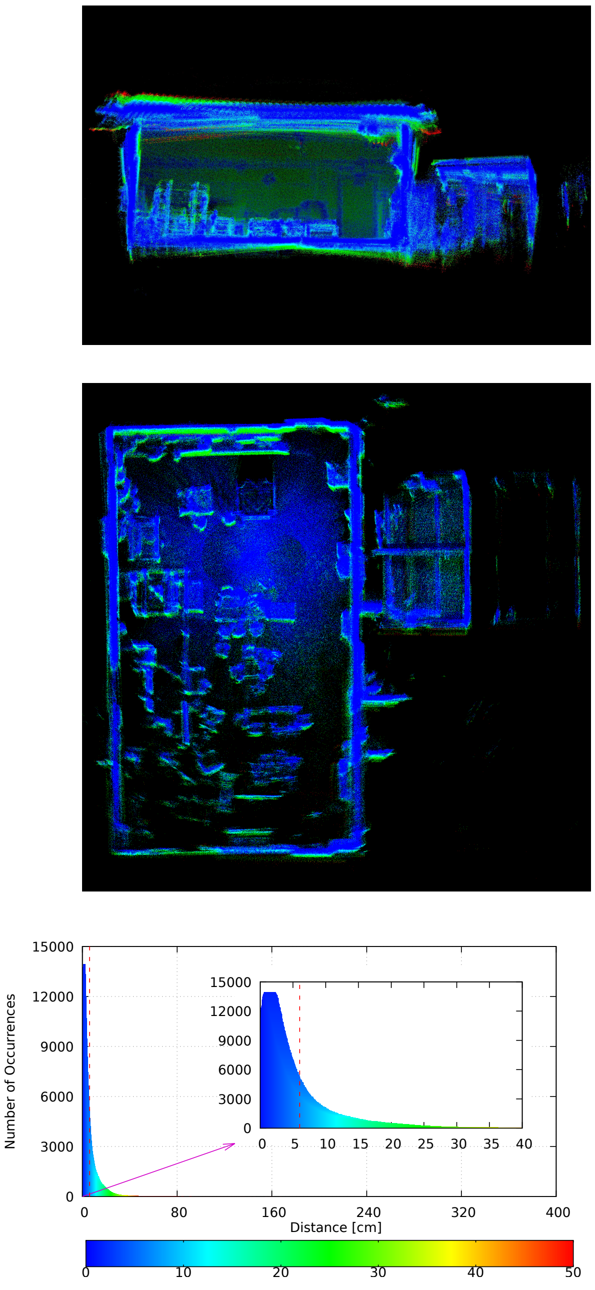

(a) Frontal view **(b)** Side view **(c)** Top view

Figure 7.12: Point cloud, based on the KF pose estimation of the larger trajectory, using dynamically adjusting $\boldsymbol{R}_{\text{cam}}$ and $\boldsymbol{R}_{\text{imu}}$, according to the logarithmic approach (5.21) - (5.26). To make drift and errors more detectable, the green rectangle serves as a reference. The light-blue path represents the estimator's trajectory.

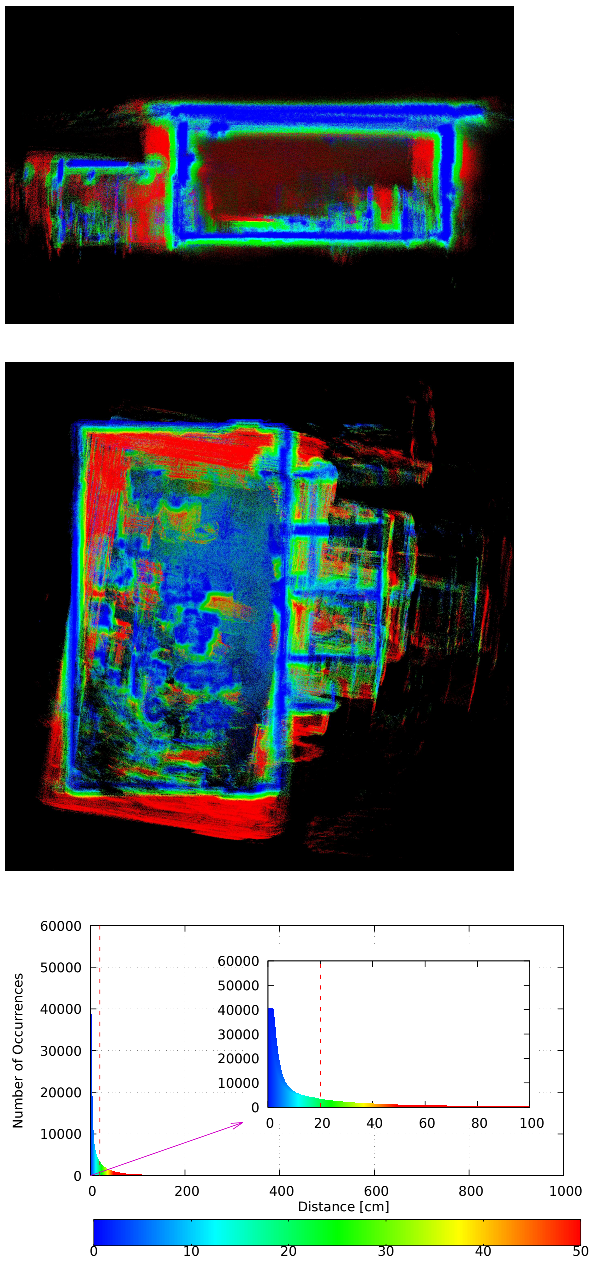


(a) Static KF

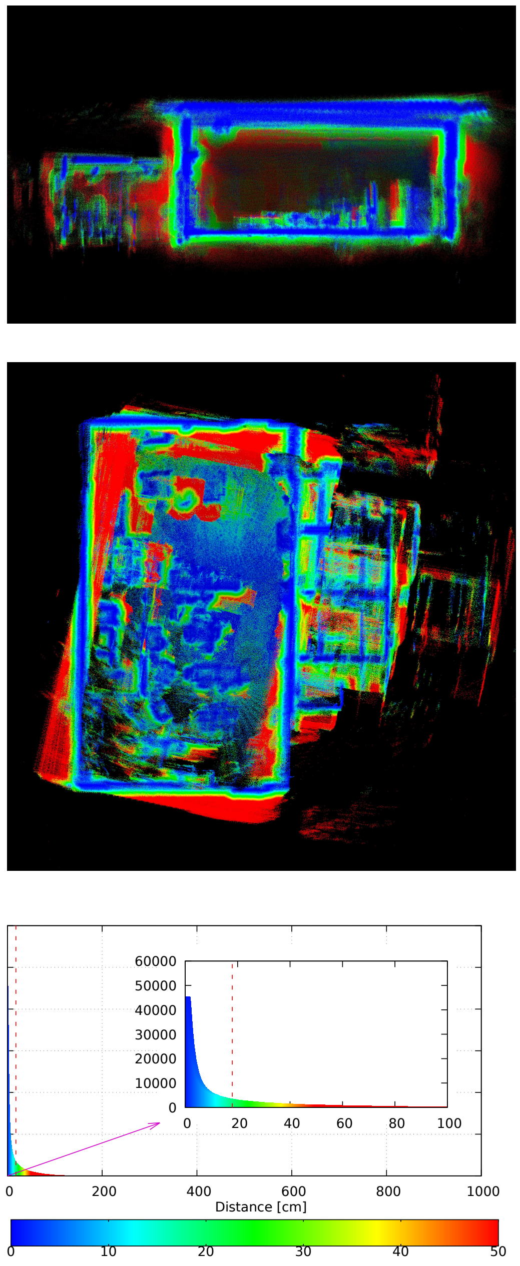


(b) Dynamic KF

Figure 7.13: Comparison of the point-to-point error point clouds, generated by the most accurate static KF implementation (7.2.1.2) and the most accurate dynamic KF implementation (7.2.2.1) for the circular trajectory. The two upper rows depict the point clouds with colored points according to the points' distance to ground truth. Below, a histogram depicts the number of occurrences of different point-to-point errors. These distances are represented by colors, according to the color scale. Note, that the scale of the x-axis is determined by the search radius of the scan2scan_distance tool, described in 6.3, and was chosen by visual analysis in imshow. In the zoomed-in histograms, the number of occurrences of distances to ground truth, within 10 % of the search radius are presented. The MPTPE is marked by the dashed red line and lies at 6.579 cm for the static KF and at 6.027 cm for the dynamic KF.



(a) Static KF



(b) Dynamic KF

Figure 7.14: Comparison of the point-to-point error point clouds, generated by the most accurate static KF implementation (7.2.1.2) and the most accurate dynamic KF implementation (7.2.2.1) for the larger trajectory. The two upper rows depict the point clouds with colored points according to the points' distance to ground truth. Below, a histogram depicts the number of occurrences of different point-to-point errors. These distances are represented by colors, according to the color scale. Note, that the scale of the x-axis is determined by the search radius of the scan2scan_distance tool, described in 6.3, and was chosen by visual analysis in imshow. In the zoomed-in histograms, the number of occurrences of distances to ground truth, within 10 % of search radius are presented. The MPTPE is marked by the dashed red line and lies at 20.117 cm for the static KF and at 17.934 cm for the dynamic KF.

7.2.3 Sources of Error

Now, that the performance of the KF has been evaluated, the sources of pose estimation errors must be discussed. The most notable factor are the measurement noise covariance matrices. The manufacturer of the used sensors, does not provide covariance matrices for these models, which results in lower accuracy of the KF. Only the RealSense T265 camera provides a tracking confidence factor, which provides information about the validity of the measurements. Though this factor is restricted to only four integer values, thus, representing a rather broad confidence description of *Failed, Low, Medium* & *High*. Another aspect to consider is, that the motion model, used for the state prediction assumes, the sensors are centered on the SMMS. However, this is not the case, as seen in section 4.1. This leads to a deviation of the state estimation by the prediction step of the KF and, thus, to a deviation of the entire pose estimation from the actual pose. Furthermore, the motion model also does not consider the slipping of the SMMS on the ground, which is always present to a certain degree. The update step also suffers from the interpolation of the camera measurements. While the interpolation enables the fusion of pose estimates of different frequencies, it also leads to the interpolated measurement not representing the exact value of the current timestep. The evaluation via point clouds is suitable for a SMMS, but also brings some sources of error. For one, the point clouds are reduced before applying the scan2scan_distance tool, which lowers the point cloud's resolution. However, this was necessary, to make further processing of the point clouds possible with limited computing resources. As stated in section 7.1, the environment of the experiments brings many reflective objects, like steel chair legs, shelves and tables, next to many glass elements, which causes less accurate point clouds. For example, when a LiDAR signal passes through glass elements, it gets refracted and possibly reflected to a certain degree, depending on the glass's properties, e.g., its thickness. This is further increased by the fact, that the LiDAR is surrounded by the SMMS's plastic shell. The camera also gets affected by this plastic shell, since it is located at a part of the SMMS, where the two hemispheres are connected, leading to a partly restricted view, which makes visual-inertial pose estimation less accurate. Especially when the transparent shell is not entirely clear, due to dirt and dust particles, collected by rolling on the floor. The LiDAR measurements also suffer from motion distortion, due to the rotational and translational motion of the SMMS, while the LiDAR data is acquired. This further decreases the accuracy of all resulting point clouds. At last, the ground truth trajectories, and therefore the ground truth point clouds, do not represent the actual environment without flaw. Consequently, the point-to-point errors do not equal the actual deviation from the environment, but rather the deviation from a more accurate representation of the environment.

Chapter 8

Conclusions

This thesis proposes a real-time linear Kalman filter for 6 DoF pose estimation of a spherical mobile mapping system, using filtered IMU pose estimates and a tracking camera's pose estimates as measurement sources. The filter is further enhanced by the possibility of adapting the measurement noise covariance matrices, depending on the current angular velocities of the system, as well as the camera's tracking confidence. Therefore, two approaches are applied to calculate scaling factors: one based on the exponential function and one based on an adjusted natural logarithmic function. In particular, the approach, based on the exponential scaling factor, provided the most accurate results. Furthermore, the manual initialization approach of the measurement noise covariance matrices is evaluated against the approach of calculating the entries of the main diagonals, using the sensor's data while standing still. The approach of manual initialization, based on knowledge about the system, proves to provide a more accurate pose estimation. Thus, the conclusion is drawn, that the calculation of the sensor's measurement noise covariance matrices must be done with data, more representative of the system's rotation-based motion model. An evaluation, based on point cloud analysis shows, that adaptive measurement noise covariance matrices improve pose estimation accuracy, compared to the conventional Kalman filter with static measurement noise covariance matrices. In particular, the exponential scaling factor provides the most accurate pose estimation results, exceeding the accuracy of the previously proposed Delta filter [5]. However, the Kalman filter suffers from the inaccuracy of the camera's pose estimation in the recorded trajectories, which hinders the filtered pose estimation from exceeding the accuracy of the filtered IMU pose estimation. By increasing the pose estimation accuracy of the tracking camera, the Kalman filter's pose estimation would be further enhanced. Therefore, the camera shall be mounted at the front of the SMMS instead of the side, since this setup proved to provide better pose estimation results in a previous version of the SMMS.

Consequently, the proposed Kalman filter provides accurate pose estimation results for smaller trajectories, with decreasing accuracy for longer and more complex trajectories. The cause for these consists of multiple aspects, e.g., linearized motion assumptions and manually tuned measurement noise covariance matrices, further described in section 7.2.3. A possible solution for this behavior is to use sensors, which come with previously determined measurement noise covariance matrices, so that errors, caused by manually tuning the variance values are eliminated. For further improvement, the assumption of centered sensors must be eliminated, by providing the Kalman filter pose estimates, that take the non-linear motion of the sensors into account. Therefore, the linear Kalman filter shall be enhanced to an extended Kalman filter or an unscented Kalman filter. Additionally, the attitude representation using quaternions is an alternative to avoid gimbal lock via estimating the delta pose between two timesteps, and could improve the computational performance of the filter.

While indicating challenges for spherical mobile mapping systems and providing design approaches for handling rotational motion, the proposed Kalman filter contributes to the research of spherical systems and builds a foundation for possible future enhancements.

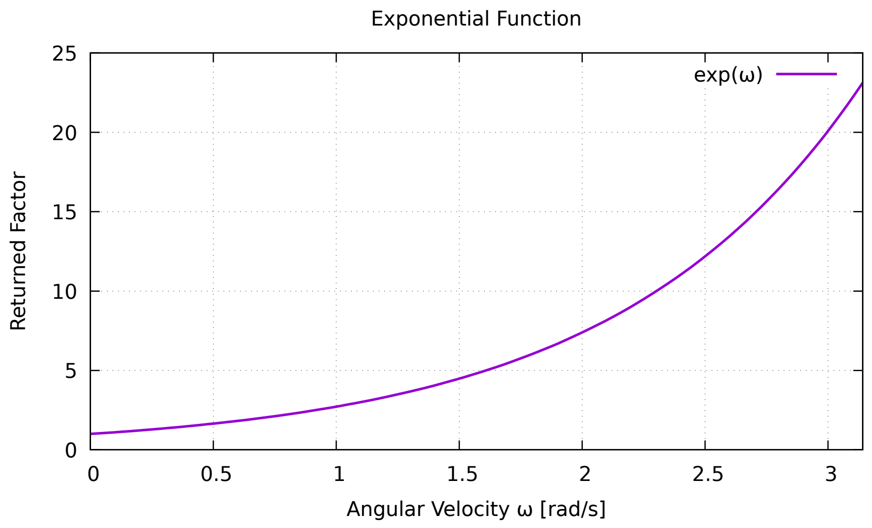
Appendix A

GNU Octave Script for Variance Calculation

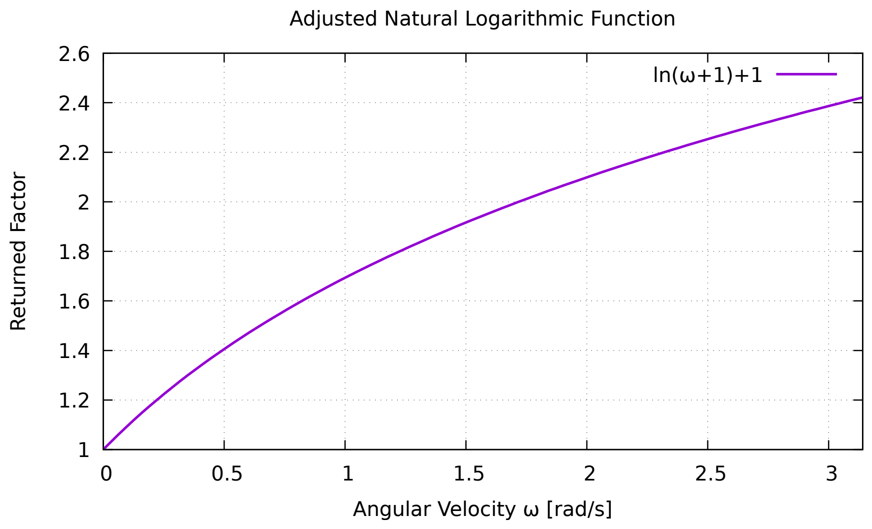
```
% Path to CSV file
    path = '/path/to/csv/file';

% Load CSV file
    data = csvread(path);

    num_entries = 4977; %change value depending on number of measured data
    subset_data = data(2:num_entries, :); % leave out first value (always 0)

% Column indices for paramters of posePub_Merged and camera topics
    x_col = 1;
    y_col = 2;
    z_col = 3;
    roll_col = 4;
    pitch_col = 5;
    yaw_col = 6;
    w_x_col = 7;
    w_y_col = 8;
    w_z_col = 9;

% Extract data
    x_values = subset_data(:, x_col);
    y_values = subset_data(:, y_col);
    z_values = subset_data(:, z_col);
    roll_values = subset_data(:, roll_col);
    pitch_values = subset_data(:, pitch_col);
    yaw_values = subset_data(:, yaw_col);
    w_x_values = subset_data(:, w_x_col);
    w_y_values = subset_data(:, w_y_col);
    w_z_values = subset_data(:, w_z_col);

% Calculate variances
    var_x = var(x_values);
    var_y = var(y_values);
```

```
var_z = var(z_values);
var_roll = var(roll_values);
var_pitch = var(pitch_values);
var_yaw = var(yaw_values);
var_w_x = var(w_x_values);
var_w_y = var(w_y_values);
var_w_z = var(w_z_values);
```

Plots of Functions for Scaling Factor Calculation



(a)



(b)

Figure A.1: Plots of the exponential function and adjusted logarithmic function, taking the current angular velocity in $\frac{rad}{s}$ as input to return a scaling factor. The range of the x-axis is limited to 3.141 $\frac{\mathrm{rad}}{\mathrm{s}}$, which represents 180 $\frac{\mathrm{deg}}{\mathrm{s}}$, since this value is never exceeded in the recorded data.

ROS Frames

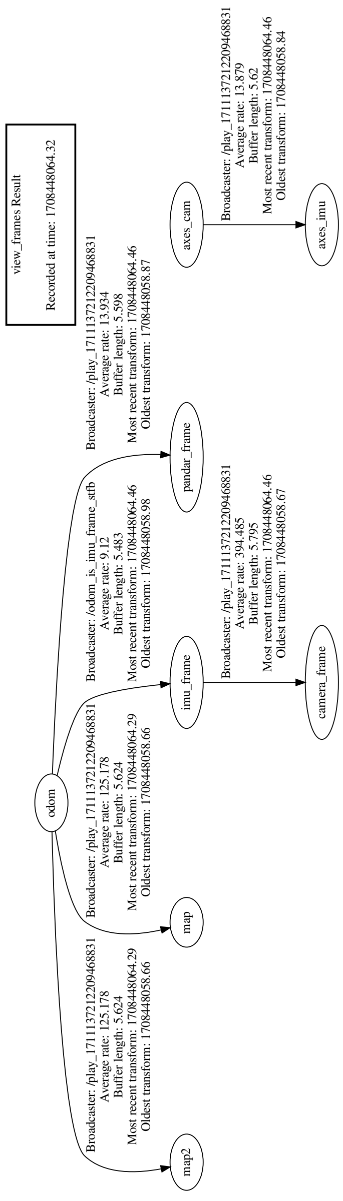

view_frames Result
Recorded at time: 1708448064.32
odom
map2
map
imu_frame
pandar_frame
camera_frame
axes_cam
axes_imu
Broadcaster: /play_1711137212209468831
Average rate: 125.178
Buffer length: 5.624
Most recent transform: 1708448064.29
Oldest transform: 1708448058.66
Broadcaster: /play_1711137212209468831
Average rate: 125.178
Buffer length: 5.624
Most recent transform: 1708448064.29
Oldest transform: 1708448058.66
Broadcaster: /odom_is_imu_frame_stfb
Average rate: 9.12
Buffer length: 5.483
Most recent transform: 1708448064.46
Oldest transform: 1708448058.98
Broadcaster: /play_1711137212209468831
Average rate: 13.934
Buffer length: 5.598
Most recent transform: 1708448064.46
Oldest transform: 1708448058.87
Broadcaster: /play_1711137212209468831
Average rate: 394.485
Buffer length: 5.795
Most recent transform: 1708448064.46
Oldest transform: 1708448058.67
Broadcaster: /play_1711137212209468831
Average rate: 13.879
Buffer length: 5.62
Most recent transform: 1708448064.46
Oldest transform: 1708448058.84

Figure A.2

Bibliography

[1] Shahrokh Akhlaghi, Ning Zhou, and Zhenyu Huang. Adaptive adjustment of noise covariance in kalman filter for dynamic state estimation. In *2017 IEEE Power & Energy Society General Meeting*, pages 1–5, 2017.

[2] Alacam, Burak, and Birsen. *Extended Kalman Filtering for the Modeling and Estimation of ICG Pharmacokinetics in Cancerous Tumors Using NIR Measurements.* 04 2009.

[3] Simone Andolfo, Flavio Petricca, and Antonio Genova. Precise pose estimation of the nasa mars 2020 perseverance rover through a stereo-vision-based approach. *Journal of Field Robotics*, 40(3):684–700, 2023.

[4] Fabian Arzberger and Tim Schubert. Github repository of the kalman filter and delta filter implementation. `https://github.com/fallow24/delta_pose_filter`. Accessed: 02.03.2024.

[5] Fabian Arzberger, Fabian Wiecha, Jasper Zevering, Julian Rothe, Dorit Borrmann, Sergio Montenegro, and Andreas Nuechter. Delta filter – robust visual-inertial pose estimation in real-time: A multi-trajectory filter on a spherical mobile mapping system. Coimbra, Portugal, September 2023. Accepted for publication on the European Conference on Mobile Robots (ECMR) 2023.

[6] A. Benini, M. J. Rutherford, and K. P. Valavanis. Real-time, gpu-based pose estimation of a uav for autonomous takeoff and landing. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3463–3470, 2016.

[7] Jeannette Bohg, Javier Romero, Alexander Herzog, and Stefan Schaal. Robot arm pose estimation through pixel-wise part classification. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3143–3150, 2014.

[8] Danpeng Chen, Shuai Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Hujun Bao, and Guofeng Zhang. Vip-slam: An efficient tightly-coupled rgb-d visual inertial planar slam. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5615–5621, 2022.

[9] Shu-Bo Chen, Alireza Beigi, Amin Yousefpour, Farhad Rajaee, Hadi Jahanshahi, Stelios Bekiros, Raúl Alcaraz Martínez, and Yuming Chu. Recurrent neural network-based robust

nonsingular sliding mode control with input saturation for a non-holonomic spherical robot. *IEEE Access*, 8:188441–188453, 2020.

[10] Giovanni Cioffi and Davide Scaramuzza. Tightly-coupled fusion of global positional measurements in optimization-based visual-inertial odometry. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5089–5095, 2020.

[11] Vincent A Crossley. A literature review on the design of spherical rolling robots. *Pittsburgh, Pa*, pages 1–6, 2006.

[12] James Diebel et al. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[13] Hao Du, Wei Wang, Chaowen Xu, Ran Xiao, and Changyin Sun. Real-time onboard 3d state estimation of an unmanned aerial vehicle in multi-environments using multi-sensor data fusion. *Sensors*, 20(3), 2020.

[14] Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. 6dof semi-rigid slam for mobile scanning. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1865–1870. IEEE, 2012.

[15] Wael Farag. Real-time autonomous vehicle localization based on particle and unscented kalman filters. *Journal of Control, Automation and Electrical Systems*, 32, 01 2021.

[16] Bo Feng, Mengyin Fu, Hongbin Ma, Yuanqing Xia, and Bo Wang. Kalman filter with recursive covariance estimation—sequentially estimating process noise covariance. *IEEE Transactions on Industrial Electronics*, 61(11):6253–6263, 2014.

[17] Anders Grunnet-Jepsen, Michael Harville, Brian Fulkerson, Daniel Piro, Shirit Brook, and Jim Radford. Introduction to Intel® RealSense™ Visual SLAM and the T265 Tracking Camera. `https://dev.intelrealsense.com/docs/intel-realsensetm-visual-slam-and-the-t265-tracking-camera`. Accessed: 22.01.2024.

[18] Loizos Hadjiloizou, Kyriakos M. Deliparaschos, Evagoras Makridis, and Themistoklis Charalambous. Onboard real-time multi-sensor pose estimation for indoor quadrotor navigation with intermittent communication. In *2022 IEEE Globecom Workshops (GC Wkshps)*, pages 154–159, 2022.

[19] Yanlin He, Lianqing Zhu, Guangkai Sun, and Junfei Qiao. Cooperative localization and evaluation of small-scaled spherical underwater robots. *Microsystem Technologies*, 25:573–585, 2019.

[20] ”Hesai”. Hesai-pandarxt32 user manual. `https://www.oxts.com/wp-content/uploads/2021/01/Hesai-PandarXT_User_Manual.pdf`. Accessed: 25.01.2024.

[21] ”Hesai”. Hesai-pandarxt32 website. `https://www.hesaitech.com/product/xt32/`. Accessed: 03.02.2024.

[22] Yanjun Jiang and Xiangyu Nong. Trainable kalman filter based on recurrent neural network and its application in aviation surveillance. volume 1642, page 012010. IOP Publishing, sep 2020.

[23] Soohwan Kim and Minkyoung Kim. Rotation representations and their conversions. *IEEE Access*, PP:1–1, 01 2023.

[24] Youngjoo Kim and Hyochoong Bang. Introduction to kalman filter and its applications. In Felix Govaers, editor, *Introduction and Implementations of the Kalman Filter*, chapter 2. IntechOpen, Rijeka, 2018.

[25] Theodore S Lindsey. *On the Kalman filter and its variations.* PhD thesis, University of Kansas, 2014.

[26] Adrian Manzanilla, Sergio Reyes, Miguel Garcia, Diego Mercado, and Rogelio Lozano. Autonomous navigation for unmanned underwater vehicles: Real-time experiments using computer vision. *IEEE Robotics and Automation Letters*, 4(2):1351–1356, 2019.

[27] Fernando P. Marafão, Diego Colón, and Marcelo S. de Padua. *Kalman Filter on Power Electronics and Power Systems Applications.* 04 2009.

[28] Andreas Nuechter, Kai Lingemann, and Dorit Borrmann. 3d toolkit. `https://slam6d.sourceforge.io/index.html`. Accessed: 02.03.2024.

[29] Lorenzo Pasqualetto Cassinis, Robert Fonod, Eberhard Gill, Ingo Ahrns, and Jesús Gil-Fernández. Evaluation of tightly- and loosely-coupled approaches in cnn-based pose estimation systems for uncooperative spacecraft. *Acta Astronautica*, 182:189–202, 2021.

[30] MG Petovello, ME Cannon, and G Lachapelle. Kalman filter reliability analysis using different update strategies. In *Proceedings of the CASI Annual General Meeting*, pages 1–12, 2003.

[31] Phidget. Phidget Spatial 3/3/3 1044b User Guide. `https://www.phidgets.com/?prodid=1038#Tab_User_Guide`. Accessed: 22.01.2024.

[32] E. Pérez and J. Barros. An extended kalman filtering approach for detection and analysis of voltage dips in power systems. *Electric Power Systems Research*, 78(4):618–625, 2008.

[33] Lothar Reichel. Random vectors and the variance-covariance matrix. `https://www.math.kent.edu/~reichel/courses/monte.carlo/alt4.7d.pdf`. Accessed: 07.02.2024.

[34] ROS. ROS/Introduction. `http://wiki.ros.org/ROS/Introduction`. Accessed: 22.01.2024.

[35] Sahand Sabet, Mohammad Poursina, Parviz E. Nikravesh, Paul Reverdy, and Ali-Akbar Agha-Mohammadi. Dynamic modeling, energy analysis, and path planning of spherical robots on uneven terrains. *IEEE Robotics and Automation Letters*, 5(4):6049–6056, 2020.

[36] J.Z. Sasiadek. Sensor fusion. *Annual Reviews in Control*, 26(2):203–228, 2002.

[37] Tixiao Shan, Brendan Englot, Carlo Ratti, and Daniela Rus. Lvi-sam: Tightly-coupled lidar-visual-inertial odometry via smoothing and mapping. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5692–5698, 2021.

[38] Yuriy S. Shmaliy, Shunyi Zhao, and Choon Ki Ahn. Unbiased finite impluse response filtering: An iterative alternative to kalman filtering ignoring noise and initial conditions. *IEEE Control Systems Magazine*, 37(5):70–89, 2017.

[39] Michael Strohmeier, Thomas Walter, Julian Rothe, and Sergio Montenegro. Ultra-wideband based pose estimation for small unmanned aerial vehicles. *IEEE Access*, 6:57526–57535, 2018.

[40] Y. Sugiyama, A. Shiotsu, M. Yamanaka, and S. Hirai. Circular/spherical robots for crawling and jumping. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3595–3600, 2005.

[41] Michelle Valente, Cyril Joly, and Atnaud de La Fortelle. Deep sensor fusion for real-time odometry estimation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6679–6685. IEEE, 2019.

[42] Brain Voyager. Spatial transformation matrices. `https://www.brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html`. Accessed: 26.01.2024.

[43] Gregory F. Welch. *Kalman Filter*, pages 1–3. Springer International Publishing, Cham, 2020.

[44] ”Wolfram. Rotation matrix. `https://mathworld.wolfram.com/RotationMatrix.html`. Accessed: 26.01.2024.

[45] Jasper Zevering, Anton Bredenbeck, Fabian Arzberger, Dorit Borrmann, and Andreas Nüchter. L.u.n.a. - a laser-mapping unidirectional navigation actuator. In Bruno Siciliano, Cecilia Laschi, and Oussama Khatib, editors, *Experimental Robotics*, pages 85–94, Cham, 2021. Springer International Publishing.

[46] Jasper Zevering, Anton Bredenbeck, Fabian Arzberger, Dorit Borrmann, and Andreas Nuechter. Imu-based pose-estimation for spherical robots with limited resources. In *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 1–8, 2021.

[47] Mingle Zhao, Dingfu Zhou, Xibin Song, Xiuwan Chen, and Liangjun Zhang. Dit-slam: Real-time dense visual-inertial slam with implicit depth representation and tightly-coupled graph optimization. *Sensors*, 22(9), 2022.

Proclamation

Hereby I, Tim Schubert, confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Signature: T. Schubert

Würzburg, April 2024