CHALMERS
UNIVERSITY OF TECHNOLOGY

Dispelling Illusions of Self-Motion

Robust Localisation for Autonomous Robots in Feature-Sparse and Dynamic Indoor Environments

Master's thesis in Systems, Control and Mechatronics

Antonio Bogdanovic
Simon Vading

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Dispelling Illusions of Self-Motion

Robust Localisation for Autonomous Robots in Feature-Sparse and Dynamic Indoor Environments

Antonio Bogdanovic
Simon Vading

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2026

Dispelling Illusions of Self-Motion
Robust Localisation for Autonomous Robots in Feature-Sparse and Dynamic Indoor Environments
Antonio Bogdanovic
Simon Vading

Supervisor: Johannes Persson, Husqvarna Construction
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences, Chalmers

Master's Thesis 2026
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Photograph of a storage basement where the autonomous robot was tested.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Dispelling Illusions of Self-Motion
Robust Localisation for Autonomous Robots in Feature-Sparse and Dynamic Indoor Environments
Antonio Bogdanovic
Simon Vading
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Feature-sparse and dynamic environments pose significant challenges for autonomous robots. This thesis presents an online method for removing dynamic objects from 2D LiDAR measurements. The dynamic filter first segments and tracks objects in the data and uses a Kalman filter to estimate their speed and determine whether each object is static or dynamic. Testing shows that the dynamic filter improves localisation performance in a feature-sparse and dynamic environment. Additionally, a non-holonomic Coordinated Turn (CT) model was implemented in the robot_localization (RL) library for ROS 2 to improve odometry pose estimation. Testing showed no noticeable difference compared with RL's default omnidirectional model.

Keywords: SLAM, Object Detection, Dynamic Object filtering, Feature-sparse Environment, Odometry, Motion model, Localisation.

Acknowledgements

We would like to thank Husqvarna Construction for providing us with the opportunity and resources to conduct this master's thesis. A special thank you goes to our main industrial supervisor, Johannes Persson, whose valuable feedback, guidance, and hands-on support have been greatly appreciated throughout the project. We would also like to extend our gratitude to our examiner, Peter Forsberg, for the valuable input and feedback provided during the course of the thesis. The meetings with you have always been both enjoyable and lighthearted, which we have truly appreciated.

Furthermore, we are grateful to Casper Christiansen for welcoming and introducing us to the workplace during the first week of the project, as well as for supporting us with practical matters along the way. We would also like to thank Alexander Åstrand for introducing us to key components related to the odometry filtering, and for always showing interest in our progress and taking the time to talk to us whenever we crossed paths. Special thanks also go to our fellow master's thesis students, Filip Djäknegren and Konrad Billing, who also conducted their thesis at Husqvarna Construction. Thank you for your company, the enjoyable lunchtime discussions, and the shared commutes throughout the semester.

Lastly, we would like to thank our friends and families for their support and encouragement throughout this work.

Antonio Bogdanovic, Gothenburg, June 2026
Simon Vading, Gothenburg, June 2026

Thesis advisor: Johannes Persson, Husqvarna Construction
Thesis examiner: Peter Forsberg, Mechanics and Maritime Sciences

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CT	Coordinated Turn
CV	Constant Velocity
DBSCAN	Density Based Spatial Clustering of Application with Noise
FOV	Field of View
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
LiDAR	Light Detection and Ranging
POG	Presumption of Guilt
POI	Presumption of Innocence
RL	robot_localization
ROS	Robot Operating System
SLAM	Simultaneous Localisation and Mapping
UKF	Unscented Kalman Filter

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

n,j	Object labels
k	Index for time step

Sets

m	Set of landmarks
$Z_{0:k}$	Set of landmark observations
$U_{0:k}$	Set of control inputs

Parameters

ε	Eps
α	Angle increment
γ	Velocity decay coefficient
w_d	Dynamic weight
w_s	Static weight
L_d	Dynamic weight limit
L_s	Static weight limit
Δt	Time discretisation step (time interval)
T	Sampling interval

Variables

z_k	Landmark observation
u_k	Control input
r_1, r_2	Ranges given by rangefinder sensor
d	Distance between two measured ranges
$p^{\mathrm{n/j}}_{first/last}$	Object edges coordinates
$\mathbf{x}_k$	State vector
$\mathbf{y}_k$	Measurement vector

Contents

List of Figures

List of Tables

1

Introduction

Automation is playing an increasingly important role in modern industry, contributing to improved productivity, safety, and consistency in demanding working environments. As part of this development, Husqvarna Construction is investigating increased levels of automation for industrial robots in large-scale indoor facilities. Such machines have the potential to reduce operator workload while improving robustness and ease of use for potential customers. For these systems to operate reliably without continuous human intervention, accurate and robust localisation is a fundamental requirement.

Localisation enables a robot to estimate its position and orientation within its environment and is a prerequisite for essential functions such as navigation, path planning, mapping, and collision avoidance. In many indoor robotic applications, localisation is achieved using sensors such as Light Detection and Ranging (LiDAR), cameras, and in outdoor or semi-outdoor settings, Global Navigation Satellite System (GNSS) solutions. LiDAR sensors and cameras perform well in environments rich in geometric structures or visual features, such as corridors, furnished rooms, or textured surfaces. However, industrial environments, such as warehouses, factories, or hangars, contain long corridors and large open spaces that impose a significant challenge on localisation.

Large industrial halls, warehouses, and production facilities are typically characterised by wide open spaces, long flat surfaces, and minimal vertical structures. The uniform appearance of concrete floors, combined with limited visual texture and repetitive geometry, makes these environments feature-sparse. In such conditions, LiDAR-based localisation suffers from ambiguous scan matching due to the lack of distinctive geometric features, leading to reduced accuracy and increased drift. Similarly, vision-based systems struggle in environments with uniform surfaces, poor lighting, dust, or low contrast, which degrade feature detection and matching. GNSS solutions, while effective outdoors, are generally unavailable indoors due to signal attenuation.

These limitations pose a significant challenge for autonomous robots that must operate reliably over long durations and in large areas. Without accurate localisation, the quality of navigation degrades, resulting in inefficient coverage, poor performance, or potential safety risks.

1.1 Problem description

When a robot operates in a feature-sparse environment and does not receive sufficient LiDAR measurements, the localisation system will instead rely on odometry to estimate its position. However, a dynamic object, such as a person, can cause localisation to fail because the system cannot distinguish its own motion from the dynamic object's motion. A similar effect occurs in humans and is called vection, or an illusion of self-motion. Hence the title *Dispelling Illusions of Self-Motion*. The illusion comes from assuming one's environment is static. For the robot in this case, it assumes that the measurements from the LiDAR are of static objects.

This problem often occurs in sizeable environments, such as large warehouses. Once the autonomous robot reaches an area of the environment where static objects, such as walls and pillars, no longer remain within the LiDAR's range, the SLAM algorithm starts relying on dynamic objects that are commonly found in working environments, such as workers. The autonomous robot has no way of discerning that these objects are dynamic and should not be used for localisation. This over-reliance on dynamic objects severely hampers the localisation since the SLAM algorithm anchors the autonomous robot's position to said objects, meaning a person moving even slightly signals to the autonomous robot that it is drifting. The autonomous robot's odometry is, however, sufficient to enable navigation for a limited time in feature-sparse environments.

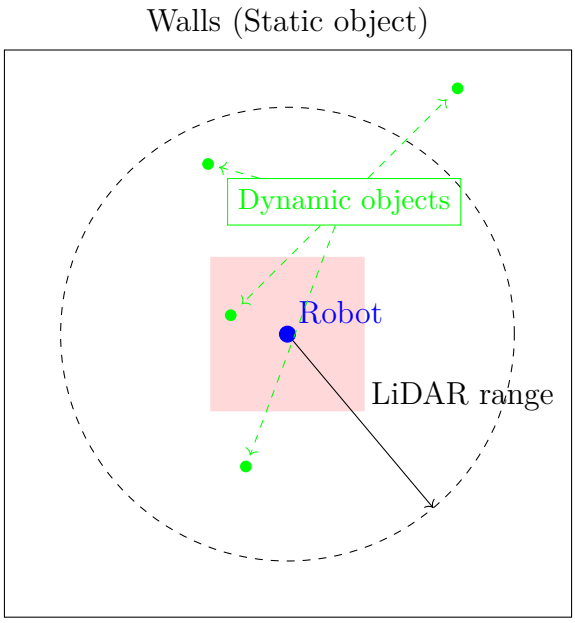

Figure 1.1: The diagram above shows a situation in which the autonomous robot will fail. The walls and other static features are outside the range of LiDAR, and the only features inside the LiDAR's range are dynamic. The red area represents the area where the LiDAR can no longer detect sufficient static features and is at risk of relying on dynamic features.

Current situation

- The autonomous robot operates well in smaller spaces where there are plenty of static features visible at all times.
- The autonomous robot can perform satisfactorily for a limited time using wheel odometry and an IMU.
- The autonomous robot fails when there are too many dynamic features compared to static ones.

1.2 Related work

This thesis is primarily concerned with the implementation of a dynamic filtering algorithm to filter out dynamic objects and a more suitable, constrained motion model. In this section, we present some previous dynamic filtering methods and different approaches to modelling vehicle motion in 2D.

1.2.1 Dynamic object filtering

The removal of dynamic objects from LiDAR scans is a well-researched topic. It is primarily necessary for use in urban environments, as those types of environments tend to be filled with numerous dynamic objects, such as crowds of people and a significant number of vehicles. To classify different objects, a scan first needs to be segmented. This is commonly done with an algorithm such as Density Based Spatial Clustering of Application with Noise (DBSCAN) [1], or the range image-based segmentation proposed in [2]. The algorithm proposed in [2] is especially useful for 3D LiDARs as it simplifies the problem by viewing the LiDAR data as a 2D range image rather than an explicit 3D point cloud.

Kim et al. propose a multistage algorithm for removing dynamic features, which the authors call *Removert* [3]. The *Removert* algorithm first liberally removes points considered dynamic at a high resolution of the range image and then reverts some of those points to static at a coarser resolution. This thesis aims to filter out dynamic objects to improve localisation, which requires online operation. However, as [3] is primarily concerned with creating a clean static map, it is not suited for online operation due to its computational demand and the need for a sequential set of scans.

More closely related to this thesis is the method proposed by Wang et al. [4]. In [4], Wang et al. model an object as a set of rigid sample points, allowing them to represent objects independent of their class or shape. Wang et al. created a framework for estimating the pose of the sensor, the static background, and tracking of dynamic objects in a tightly coupled manner. While this could potentially solve the problem dealt with in this thesis, it has only been evaluated in an urban environment, assumes that there is a visible static background, and has a high average computation time of 336 ms, limiting the scan frequency to around 3 Hz.

1.2.2 Modelling of vehicle dynamics in 2D space

Kinematic and dynamic modelling, navigation, path planning, and the analysis of motion models for mobile robots and vehicle tracking have been studied extensively. Depending on a robot's configuration and area of operation, i.e. whether it is airborne, seaborne, or grounded, its general movement may vary considerably. Pepy et al. draw attention to the importance of implementing realistic vehicle models to reduce navigation errors as well as skidding and slipping, since simpler kinematic car-like models do not consider these properties [5]. However, in [5], Pepy et al. also make it clear that the need for more intricate, dynamic models occurs at high speeds—speeds the autonomous robot considered in this thesis will not reach under any circumstances.

A common way of describing the kinematics of a non-holonomic, two-wheeled robot in a 2D plane, proposed by Uddin et al. [6] and Xu and Yang [7], among others, is with the time derivatives of the robot's x- and y-position, as well as its heading angle. Bill Triggs refers to this model as the *unicycle model* [8].

Lastly, Schubert et al. compare and evaluate myriad motion models for vehicle tracking, ranging from simple models in complexity to advanced ones [9]. They note that simple models, such as the Constant Velocity (CV) model, benefit from linearity. However, the CV model assumes straight-line motion and therefore does not account for rotations. Slightly more complex, non-linear models take the yaw angle into account, such as the *Constant Turn Rate and Velocity* model, at times also referred to as the *Coordinated Turn* (CT) model. Schubert et al. [9] note that this type of model is regularly used in airborne tracking systems, however, as seen in [6] and [7], the unicycle model—being very similar in structure to the CT model—is commonly used for two-wheeled robots as well. Hammarstrand [10] also demonstrates that the CT model is a fitting model for tracking ground vehicles. Nevertheless, while the CT model and similar models are frequently applied to vehicles, they have often been evaluated with rapid movements and at high speeds, as opposed to the particularly slow autonomous robot addressed in this thesis.

1.3 Aim

This thesis aims to improve an autonomous robot's ability to estimate its position and navigate in feature-sparse, potentially dynamic industrial indoor environments. When the robot is in a large open indoor area, such as a hangar or warehouse, with minimal static objects within the LiDAR's range to anchor its position, SLAM will fail when it starts over-relying on dynamic objects. This is addressed through two primary approaches: (1) utilising a more accurate motion model to extend the duration the autonomous robot can localise accurately without LiDAR measurements, and (2) detecting, tracking, and filtering dynamic objects from LiDAR measurements to ensure reliable sensor data. Previous works on dynamic object filtering are primarily concerned with removing an excessive amount of dynamic objects in urban environments, while the aim of this thesis is to investigate the potential of dynamic object filtering in feature-sparse environments to improve localisation robustness.

- How can the removal of dynamic features from the LiDAR measurements improve localisation performance in a feature-sparse environment?
- How much can a more constrained motion model reduce odometry drift?

1.4 Limitations

This thesis is concerned with the improvement of an autonomous robot operating in a feature-sparse industrial environment, focusing on improving the robot's odometry and removing dynamic objects from the LiDAR scan that might confuse the SLAM algorithm. This thesis does not aim to create a new SLAM algorithm, but rather to create a preprocessing step that improves the information (odometry and LiDAR) that is received by the SLAM algorithm.

The project builds upon an existing autonomous robot designed to operate in an indoor industrial environment. The methods described in this thesis have only been evaluated on this platform. The sensors are limited to a 2D LiDAR, wheel encoders, and an IMU. The autonomous robot uses ROS 2-based software, including SLAM Toolbox [11] and the robot_localization ROS package [12]. The robot is a very slow machine with a common operating speed of 3-4 m/min.

2

Theory

This section outlines the key concepts underlying the project and summarises relevant related work from the literature.

2.1 Light Detection and Ranging

Light Detection and Ranging (LiDAR) is an active remote sensing technology that measures distances by emitting laser pulses and estimating the time-of-flight of the reflected light from surrounding objects, thereby enabling accurate range measurements and spatial perception. By continuously scanning the environment, LiDAR sensors generate a geometric representation of the surroundings in the form of point cloud data, which can be used for localisation, mapping, and obstacle detection in autonomous systems. Due to their high measurement accuracy, robustness to illumination changes, and ability to provide depth information, LiDAR sensors are widely used in robotics and autonomous vehicles, often in combination with probabilistic estimation and sensor fusion methods [13].

2.2 SLAM

Simultaneous Localisation and Mapping (SLAM) is a necessary tool in robotics that enables robots to effectively plan and control their motion while simultaneously building an understanding of their environment [14]. Ideally, SLAM is performed online and should not require any previous information about the environment. This is done to allow a robot to perform its task without an information-gathering (mapping) phase [15].

2.2.1 Probabilistic SLAM

It is often useful to view SLAM as a probabilistic problem and solve it using Bayesian estimation. Following the probabilistic SLAM formulation in [15], a given SLAM solution must calculate the probability distribution of the map and the robot's state at each time step k

$$P(\mathbf{x}_k, m | Z_{0:k}, U_{0:k}, x_0) \tag{2.1}$$

where $\mathbf{x}_k$ is the state vector of the robot at time k, m the set of all landmarks or map, $Z_{0:k}$ the set of all landmark observations, and $U_{0:k}$ the history of all control

inputs. The probability distribution defined in Eq. (2.1) describes the joint posterior density of the map and robot state. To compute the posterior recursively, a motion and measurement model needs to be defined.

Measurement model:

$$P(z_k|\mathbf{x}_k, m) \tag{2.2}$$

Motion model:

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, u_k) \tag{2.3}$$

With the measurement and motion model defined, the posterior can be calculated for every iteration using Baye's theorem. The posterior is usually computed sequentially with a prediction and a measurement update step as:

Prediction step

$$\begin{aligned} &P(\mathbf{x}_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \\ &\int P(x_k|\mathbf{x}_{k-1}, u_k) P(\mathbf{x}_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0)\ d\mathbf{x}_{k-1} \end{aligned} \tag{2.4}$$

Measurement update step

$$P(\mathbf{x}_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|\mathbf{x}_k, m) P(\mathbf{x}_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{k-1}, U_{0:k})} \tag{2.5}$$

These equations serve as the basis for the classic EKF SLAM [16], as well as Fast-SLAM [17], which is based on a Rao-Blackwellized filter.

2.2.2 Graph SLAM

A newer approach than the Bayes filter-based method is the graph-based method, first introduced with GraphSLAM [18]. As the name implies, GraphSLAM works by constructing a graph from a set of measurements and control inputs. In the graph, each node represents a pose, and edges are either a motion event or a measurement event. Motion events link two robot poses, and a measurement event links a robot pose and a map feature. The graph representation is very efficient and allows the handling of much larger maps.

2.2.3 Modern SLAM algorithms

There are two popular sensor configurations used for SLAM [19]. Visual SLAM uses a camera to capture an image of its surrounding environment, identifying features within that image, and matching those features to previously captured images to estimate the robot's pose [20]. There exist SLAM methods that rely on a single monocular camera, such as ORB-SLAM [20], as well as methods that rely on stereo camera setups (or RGB-D) such as Nvidia's Isaac ROS Visual SLAM [21]. Estimating the pose of a camera from an image is a common problem within computer vision. This problem has been traditionally solved by matching an image's structural features, such as SIFT [22]. However, Walch et al. [23] show that these SIFT-based methods fail in environments with textureless surfaces and repetitive structures.

More recently, machine learning-based methods are showing more promising results in feature-sparse environments. Zangeneh et al. [24] used a pretrained ResNet-18 [25] network to extract a 2048-dimensional feature vector from an RGB image taken by a monocular camera. They then fed the feature vector through a linear layer to extract a posterior distribution of the camera pose. The posterior is then mapped to a 6-dimensional pose vector by a fully connected network.

One of the most popular sensors for SLAM use is the LiDAR. One of the most influential of these LiDAR-based SLAM algorithms is LOAM [26]. Many other algorithms have been derived from LOAM, such as V-LOAM [27] that combines it with visual odometry, and LeGO-LOAM [28] which leverages ground segmentation for improved accuracy and decreased computation time.

SLAM Toolbox [11], which builds upon the open-source library Open Karto [29], is an open-source SLAM algorithm built for the ROS platform that relies on 2D LiDAR measurements. SLAM Toolbox is a highly efficient graph-based SLAM algorithm that implements many modern features such as multi-robot SLAM, loop closure, and Lifelong mapping. Lifelong mapping is the ability to continuously refine the map over time and multiple sessions [30], and loop closure is the ability to identify that an area has been visited previously to correct for accumulated error [31].

2.3 Inertial Navigation System

An Inertial Navigation System (INS) is a self-contained navigation approach that estimates a system's position, velocity, and orientation by integrating measurements from motion sensors such as accelerometers and gyroscopes, starting from a known initial state. It operates using dead reckoning, meaning that the current state is continuously updated based on previously estimated states and measured motion, without requiring external references [32]. In the context of this thesis, INS-based dead reckoning is essential when LiDAR observations are sparse or absent, enabling the robot to maintain a continuous estimate of its pose using only internal sensor measurements.

2.3.1 IMU-Wheel odometry filter

In addition to a LiDAR sensor, the autonomous robot is also equipped with an IMU as well as wheel encoders that provide raw odometry estimations. The robot's IMU-Wheel odometry filter consists of processing the wheel- and IMU data separately, namely (1) running the IMU's measured yaw-rate data through an Exponential Moving Average filter, and (2) combining each wheel's estimated velocity into a unified velocity vector, before the independent estimates are fused using an Unscented Kalman Filter (UKF) to obtain a pose estimation, as illustrated in Figure 2.1.

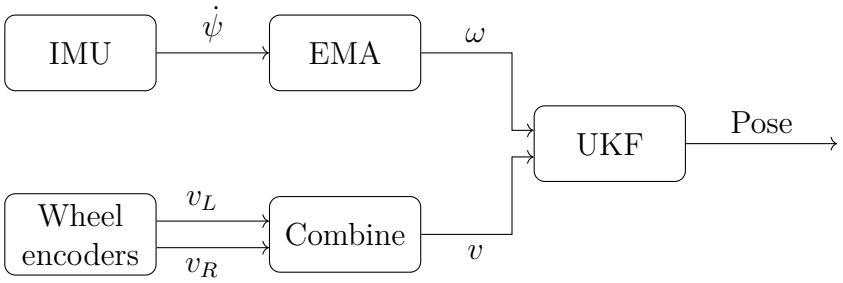

Figure 2.1: A diagram of the IMU-Wheel odometry filter. $\dot{\psi}$ and ω refer to the angular yaw velocity, or heading rate, while v_L and v_R refer to the estimated linear velocity from each wheel respectively, with v being the total linear velocity of the robot in its front-facing direction.

2.3.2 State estimation software

The sensor fusion is implemented using the Robot Operating System (ROS) package *robot_localization* (RL), developed by Moore and Stouch [12]. This package provides a set of non-linear state-estimation nodes for robotic localisation in three-dimensional space. In particular, it includes implementations of both the Extended Kalman Filter and the Unscented Kalman Filter (UKF), which fuse data from multiple sensors to estimate the robot's state. The package supports an arbitrary number of sensor inputs and common ROS message types such as odometry and IMU messages. The filter maintains an estimate of the robot's 15 states, including position, orientation, velocities, and accelerations, and continuously updates this estimate as new sensor measurements arrive [33].

2.3.3 Applicable motion models

The prediction step of a Kalman filter involves estimating the evolution of a state vector by propagating the current state vector through a process or motion model and adding white Gaussian process noise to the estimation. Since Kalman filters can be used in myriad applications with varying dynamics, the complexity of motion models consequently also varies: whether they are translational- or rotational kinematics models, linear or non-linear, and, particularly, *holonomic* or *non-holonomic*. In mobile robotics, holonomic systems may move freely in any planar direction independently of orientation, whereas non-holonomic systems are subject to non-integrable velocity constraints that restrict the allowable instantaneous motion of the system [34].

The autonomous robot considered in this thesis is two-wheeled and differentially driven, meaning it cannot generate instantaneous lateral motion perpendicular to the wheel direction. Consequently, the robot is non-holonomic. However, the default motion model in RL's UKF is omnidirectional, i.e. holonomic. Thus, the necessity of employing a broad omnidirectional motion model for a system with clearly constrained degrees of freedom is moot.

As seen in Figure 3.2, the general intended movement of the robot consists mainly

of straightforward motion followed by U-turns to assume a parallel trajectory in the opposite direction. According to Schubert et al. [9] and their survey of motion models of varying complexity, straight motion can easily be modelled with simple motion models, such as the CV or constant acceleration motion model. However, said models do not account for rotations around the z-axis, thus, more advanced models are required for modelling the robot in this thesis. Schubert et al. further claim that *curvilinear* models describe the motion of road vehicles very accurately, i.e. models that do account for rotations around the z-axis, or yaw angle. One such model is the *Constant Turn Rate and Velocity* model, often also referred to as the CT model. A curvilinear model such as the CT model thus assumes the robot may move in a circular trajectory with constant velocity, which aligns well with its expected movement as seen in Figure 3.2. Hammarstrand [10], who discusses and demonstrates the discretisation of the continuous-time CT model, also emphasises its favourable properties in describing objects that occasionally travel along smooth curves, but also if an object stops and shortly thereafter is likely to resume its course in a similar direction as before. Indeed, curvilinear models are popular when modelling two-wheeled differentially driven mobile robots as also seen in the works of, inter alia, Uddin et al. [6] and Xu and Yang [7].

2.4 Dynamic feature filtering

Dynamic objects can significantly reduce the performance of the SLAM algorithm. This challenge can be overcome by first detecting dynamic objects in the LiDAR measurements and filtering them out before the SLAM algorithm can process them. This prevents the SLAM from trying to anchor the robot's position to a moving object and also signals to the algorithm that a given scan has more noise due to the reduced number of measured points.

Learning-based methods, such as [35], first perform a semantic segmentation and then determine, based on the classification, whether an object is dynamic or static, i.e. humans are dynamic while walls are static. This approach has the ability to perform classification from a single sensors measurement, and in certain applications, it is also necessary to know whether an object is dynamic even if it is currently not moving. For example, an autonomous car needs to be aware that an object is a car even if it is standing still at a stop sign. However, a significant disadvantage of the semantic segmentation approach for filtering out dynamic objects is that it can only classify the object if it has been trained on those types of objects. As such, methods that only classify if an object is static or dynamic are becoming more popular [36].

An alternative approach to the learning-based methods is to first perform object segmentation and then estimate the velocity of each object over the course of several LiDAR scans. There exist many different methods for segmenting a LiDAR scan into objects. [2] proposes a computationally effective method of segmenting objects from a 3D LiDAR scan. [2] gains its efficiency by interpreting the 3D LiDAR data as a 2D range image using a cylindrical coordinate system rather than as a 3D point cloud. A more conventional approach to clustering points in a point cloud is DBSCAN [1]. DBSCAN labels clusters of points within a point cloud based on

two parameters, Eps and $MinPts$. Eps is the maximum distance for two points to be considered neighbours, while $MinPts$ is the minimum number of points in a neighbourhood for it to not be considered noise. Once the objects within a LiDAR scan are determined, an object can be tracked to determine whether it is dynamic. Zeybek uses the timestamp of each range measurement and the fact that static objects tend to be from a similar time frame as the surrounding ground points, whereas dynamic objects are concentrated within a short or distinct time interval to distinguish static and dynamic objects [37]. Alternatively, a Bayes filter can be used to estimate the velocity of the object and classify objects with a velocity above a certain threshold as dynamic [4].

3

Methods

This chapter describes the methodology used in this thesis. It presents the hardware platform on which the solution is evaluated, the object segmentation algorithm, the dynamic filtering algorithm and its associated sub-problems, and the workings of the implemented odometry estimation method.

3.1 Platform

The solution described in this thesis is deployed and evaluated on a mobile autonomous robot designed to operate in an industrial environment while performing work that introduces significant noise (in the form of vibrations) to the system. The software of the robot is built on the ROS 2 framework. The robot is equipped with a 2D LiDAR whose specifications are listed in Table 3.1. The LiDAR also serves as a safety mechanism on the robot; part of the scanning segment is covered by mirrors redirecting the laser to the ground to allow it to detect any edges in front of it. The resulting LiDAR FOV is shown in Figure 3.1. These LiDAR points are ignored so that they do not confuse the SLAM algorithm.

Table 3.1: The specifications of the 2D LiDAR.

Metric	Value
Range	40 m
Scanning angle	275°
Angular resolution	0.17°
Scan rate	20 Hz

The solution is evaluated using the trajectory pattern seen in Figure 3.2. During testing, the autonomous robot covers about 4 m in one minute, i.e. a speed of roughly 0.067 m/s. This testing configuration allows odometry drift to be tested over longer periods and its sensitivity to turns and straightforward motion to be assessed

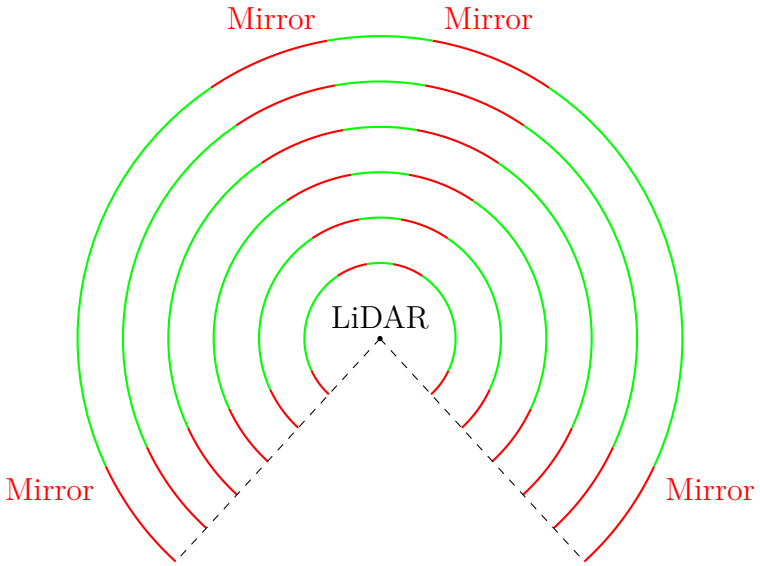

Figure 3.1: Diagram showing the obstructed (red) and non-obstructed (green) parts of the LiDAR FOV. The two non-obstructed side segments have an FOV of around 85°, while the centre segment has an FOV of around 19°.

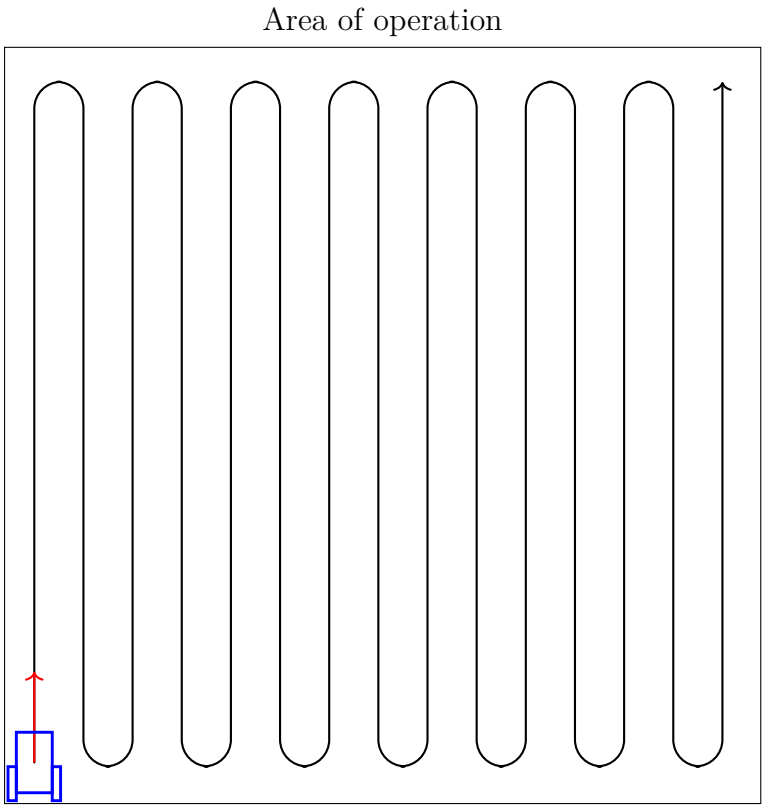

Figure 3.2: The diagram above shows the pattern of the trajectory the autonomous robot will attempt to follow during operation.

3.2 Dynamic object filter

In a feature-sparse environment, the SLAM algorithm normally detects that it is not receiving sufficient LiDAR measurements and adjusts by relying more on the IMU and odometry. However, if dynamic objects are present, the SLAM does not recognise that the measurements are unreliable and continues to use them. As the SLAM algorithm is still receiving measurements from the LiDAR, it does not adjust to relying on odometry. A filter is created between the LiDAR and the SLAM (see Figure 3.3) as a preprocessing step to remove measurements from dynamic objects and prevent SLAM failure.

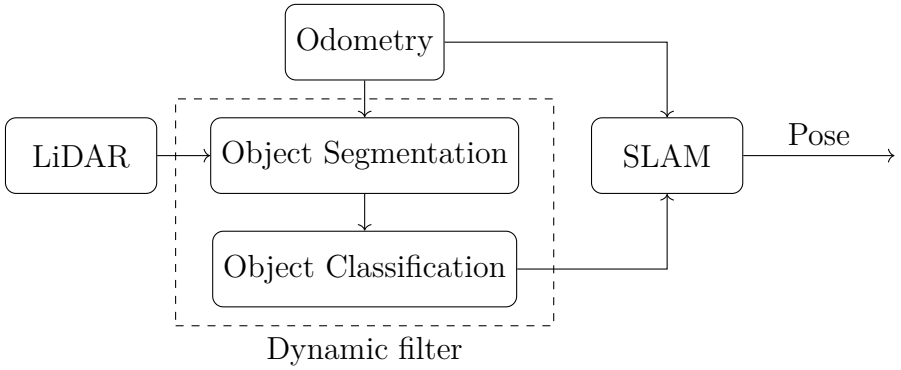

Figure 3.3: Dynamic object filter

3.2.1 Object segmentation

Before an object can be determined as dynamic, the points from the LiDAR need to be grouped into possible discrete objects. This is done with a modified DBSCAN [1] algorithm. DBSCAN works by grouping points within a specified density, i.e. a minimum number of points N_{min} within a given distance ε of each other.

Taking advantage of the fact that the points are generated by a 2D LiDAR, it is not necessary to calculate all of the distances between every point or explicitly convert the LiDAR measurements into a 2D point cloud. Additionally, the list of ranges from the LiDAR is ordered by their bearing, and the angle increment is constant (0.17°). Assuming then that the objects that are to be segmented are contiguous, a point can be clustered by only checking the distance to the next point in the list of ranges given by the LiDAR.

Given two ranges r_1, r_2, and the angle difference α between them, the distance d can be calculated using the law of cosines as:

$$d^2 = r_1^2 + r_2^2 - 2r_1 r_2 \cos\alpha \tag{3.1}$$

The LiDAR specification in Table 3.1 gives an angle increments of: $\alpha = 0.17°$, as such, Eq. (3.1) can be simplified using the approximation $\cos 0.17° \approx 1$. This gives:

$$d^2 = r_1^2 + r_2^2 - 2r_1 r_2 = (r_1 - r_2)^2 \tag{3.2}$$

$$\Longrightarrow$$

$$d = |r_1 - r_2| \tag{3.3}$$

If d is smaller than some specified distance ε, they are given the same object label. This is done for every range found in the list of ranges provided by the LiDAR, changing the label if $d > \varepsilon$ and the previous range was not labelled as noise. With this algorithm, all the data points can be segmented in a single pass through all of the points instead of calculating every distance between every point.

Using this modified DBSCAN algorithm causes issues if there is a significant amount of noise in the range measurements or if there are small gaps in the object for the LiDAR beam to pass through (e.g., between two legs). Ideally, such an object should be considered a single object to better estimate its velocity. Therefore, the Euclidean distance between the edges of each cluster $||p^n_{last} - p^j_{first}||_2$ (see Figure 3.4) is calculated using Eq. (3.1). If the distance is smaller than the specified distance ε, the two clusters are merged. Lastly, if the minimum number of points N_{min} per cluster is greater than two points for it to be considered an object, it goes through the list and removes the labels of the clusters with fewer points than the specified minimum amount.

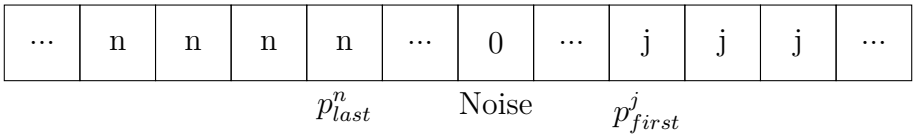

Figure 3.4: A list of the same length as the list of all the ranges from the LiDAR containing the labels of all of the points in the set. The figure shows an example set of labels where n and j are labels for two clusters, and 0 is the label given to a point not clustered into an object. $p^{\text{n/j}}_{first/last}$ are the points of the object that are seen as the edges of an object n/j from the LiDAR's point of view.

An example of this object segmentation algorithm can be seen in Figure 3.5 with the parameters $\varepsilon = 0.3$ and $N_{min} = 6$.

Figure 3.5: Measurement points from a 2D LiDAR in an indoor environment segmented using the modified DBSCAN algorithm. The LiDAR is located near the bottom right corner at *base_frame*. Note that the points that are labelled noise by the algorithm are not shown.

To determine whether an object is dynamic, its position must be tracked. Objects in the current LiDAR scan therefore need to be matched with objects in the previous scan. All object positions are first transformed to world Cartesian coordinates based on odometry estimation. Then the positions of the objects from the current LiDAR frame are compared with the object positions from previous frames. If the closest old object is within a given distance D, the new object is given the same label as the old object. If an object has not been detected in a given number of scans, it is removed from the object history. Old, undetected objects are removed because the object matching computation time scales with the number of stored objects. This prevents memory use and computation time from continually increasing.

3.2.2 Point cloud ambiguity

Before estimating the velocity of an object by looking at its position, it is important to consider that the position of an object can change without the object actually moving. The position of an object is determined by the average position of the points in the point cloud, the positions of which are affected by noise, the LiDAR pose, and other objects in the LiDAR scan.

Consider the example seen in Figure 3.6. Ideally, the left and right segments should be classified as static and the moving object as dynamic. However, as the dynamic object is moving, it affects the point clouds of the objects behind it. In the example seen in Figure 3.6, both the left and right segments' point clouds are lines (see Figure 3.7) so the position of each object will be the centre of that line. It is clear when comparing Figures 3.7 and 3.8 that the centre of each object's point cloud will shift to the right as the dynamic object moves to the right. This change in position will be interpreted as motion by the Kalman filter, and therefore the two segments would be classified as dynamic. A similar effect occurs if an object is near the edges of the LiDAR scan when the robot is moving. As the LiDAR scan area changes, it will see more or less of the object, which will affect its point cloud's centre.

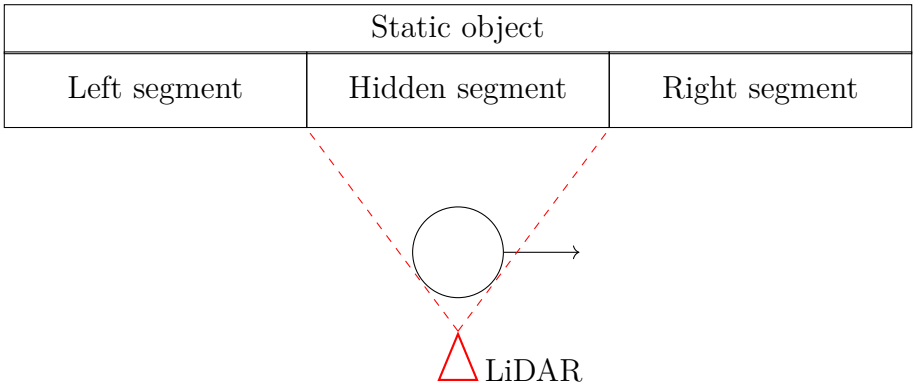

Figure 3.6: An example situation where a dynamic object is moving perpendicular to the LiDAR towards the right, partially obscuring the large static object behind it.

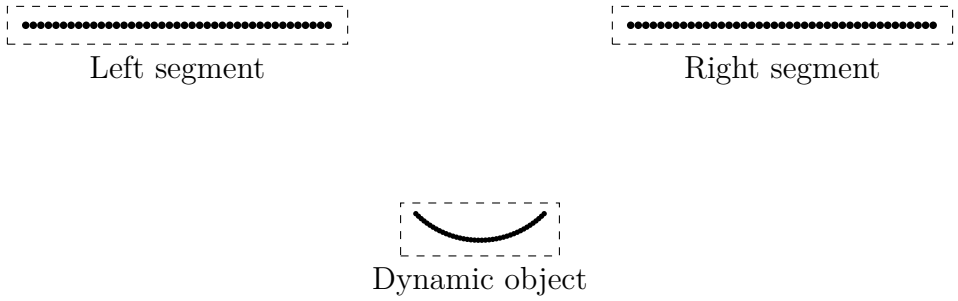

Figure 3.7: Hypothetical LiDAR points of the example situation in Figure 3.6.

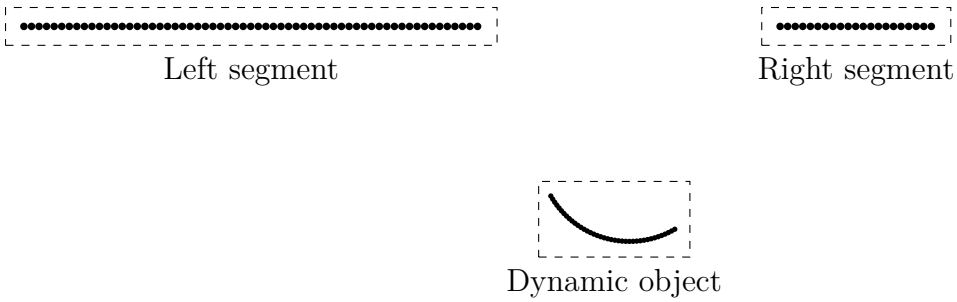

Figure 3.8: The LiDAR points of the example situation in Figure 3.6 after the dynamic object has moved further to the right.

To counter this effect, the two edges (the points with the smallest and largest angles given by the LiDAR) of each object are also tracked, and the velocities are similarly estimated. In the example seen in Figure 3.6, one of the edges of the left segment would be stationary, while the dynamic object's average position and its other edges are all moving in the same direction. All three tracked points of an object need to move in a similar direction to be considered a valid velocity. An invalid velocity will not increase the dynamic weight (described in section *Object classification*).

3.2.3 Object velocity estimation

The velocity of the three tracked points of an object is estimated with a linear Kalman filter using a CV model. The motion model is defined as:

$$\begin{bmatrix} x_k \\ y_k \\ v_{xk} \\ v_{yk} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ v_{xk-1} \\ v_{yk-1} \end{bmatrix} + \mathbf{q}_{k-1} \tag{3.4}$$

where Δt is the time step, $\mathbf{q}_{k-1} \sim \mathcal{N}(0, \mathbf{Q}_{k-1})$ and $\mathbf{Q}_{k-1} = \text{diag}(\sigma_p^2, \sigma_v^2)$. The time step Δt is not constant because an object is not always detected at every iteration. The time an object was last detected is therefore stored and is used at every filter

update to calculate Δt. Using the measurement model:

$$\mathbf{y}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ v_{xk} \\ v_{yk} \end{bmatrix} + \mathbf{r}_{k-1} \tag{3.5}$$

where $\mathbf{r}_{k-1} \sim \mathcal{N}(0, \mathbf{R}_{k-1})$ and $\mathbf{R}_{k-1} = \text{diag}(\sigma_m^2)$.

As noted in the *Point cloud ambiguity* section, the measured point cloud is affected by the LiDAR's pose. If any of the points in the object is close to the edge of the LiDAR's FOV, then any change in the measured position is likely caused by the object entering or exiting the scan range rather than actual motion. In this case, the velocity state is not updated using the Kalman filter but using the formula:

$$\hat{v}_k = \gamma \hat{v}_{k-1} \tag{3.6}$$

where $\gamma \in (0, 1)$. This is an extra safeguard to prevent static objects from appearing dynamic when the LiDAR rotates.

3.2.4 Object classification

The goal of the dynamic filter is to remove the LiDAR points from objects that are associated with moving objects. As such, the class of an object needs to be determined based on its speed, where the speed of the object is defined as the lowest speed of the three tracked points (the two edges and the average position). Since an object's speed is used for classification, its class cannot be determined upon first detection. Multiple detections are therefore required to classify with reasonable confidence. It is an important decision whether an object should be treated as static or dynamic when first detected. Borrowing a term from the judicial sphere, the two choices can be called "innocent until proven guilty" when presuming an object is static, and "guilty until proven innocent" when presuming it is dynamic.

Innocent until proven guilty

When an object is detected for the first time, it is assumed to be static and is assigned an initial dynamic weight of 0. At each iteration, if the speed of the object is estimated to be higher than a specified limit L_v, the dynamic weight w_d is increased by 1. The weight is not increased if the object is very large or if there has been a large change in its size (the number of points associated with the object) between LiDAR frames. When this weight exceeds a specified limit L_d, the object is labelled as dynamic. Once an object has been labelled as dynamic, it will never be labelled as static again. This classification strategy is called Presumption of Innocence (POI), and is summarised in Figure 3.9.

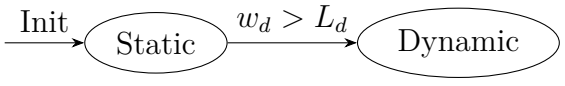

Figure 3.9: A diagram of the classes when assuming an object is static (POI).

Guilty until proven innocent

Since an object labelled dynamic can never be labelled static, a third label called unknown is needed when assuming objects are dynamic. An unknown object, similar to a dynamic object, has its associated LiDAR measurement points removed, but an unknown object can still be labelled as static in a later iteration. The dynamic classification works the same way as when presuming dynamic, but there is now an additional weight w_s to determine if the object is static. The static weight is increased by 1 whenever an object's speed is estimated to be below the specified limit, but the static weight will also be reduced whenever the dynamic weight is increased. The limit L_s to reach with the static weight is also greater than the threshold for the dynamic weight. This classification strategy is called Presumption of Guilt (POG), and is summarised in Figure 3.10.

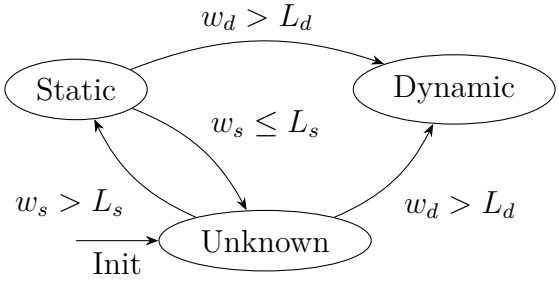

Figure 3.10: A diagram of the classes when assuming an object is dynamic (POG).

3.3 Reduction of state-space model

The default state-space model of RL consists of 15 states, those being Cartesian coordinates $(x,\ y,\ z)$, roll, pitch, and yaw $(\varphi,\ \theta,\ \psi)$, as well as linear velocities $(v_x,\ v_y,\ v_z)$, angular velocities $(\dot{\varphi},\ \dot{\theta},\ \dot{\psi})$, and lastly, linear accelerations $(a_x,\ a_y,\ a_z)$. The prediction step of a Kalman filter can generally be described as:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1} \tag{3.7}$$

where $\mathbf{q}_{k-1} \sim \mathcal{N}(0, \mathbf{Q}_{k-1})$ is white Gaussian process noise and $\mathbf{Q}_{k-1}$ is the process noise covariance. The 15 states in $\mathbf{x}_{k-1}$ are propagated through the holonomic state transition matrix, or non-linear motion model, $f(\cdot)$, which predicts the state of the system, $\mathbf{x}_k$, at time k. Because the 15-state structure is hard-coded in RL, modifying the core state vector would require extensive package rewrites and risk compromising stability. Instead, the non-holonomic motion model is applied only to the relevant states, while the remaining states are propagated with minimal process noise.

3.3.1 CT model implementation

Hammarstrand [10] demonstrates a non-holonomic CT model with a five-dimensional state vector. It assumes that the autonomous robot moves on a two-dimensional surface represented by a Cartesian coordinate system, as shown in Figure 3.11. The

position of the robot at time t is described by the pair $(x(t),\ y(t))$. The heading, or yaw angle, $\psi(t)$ is assumed to evolve according to a CV model, i.e. the yaw rate $\omega(t)$ is expected to be constant. Additionally, the velocity $v(t)$ is modelled as a Wiener process.

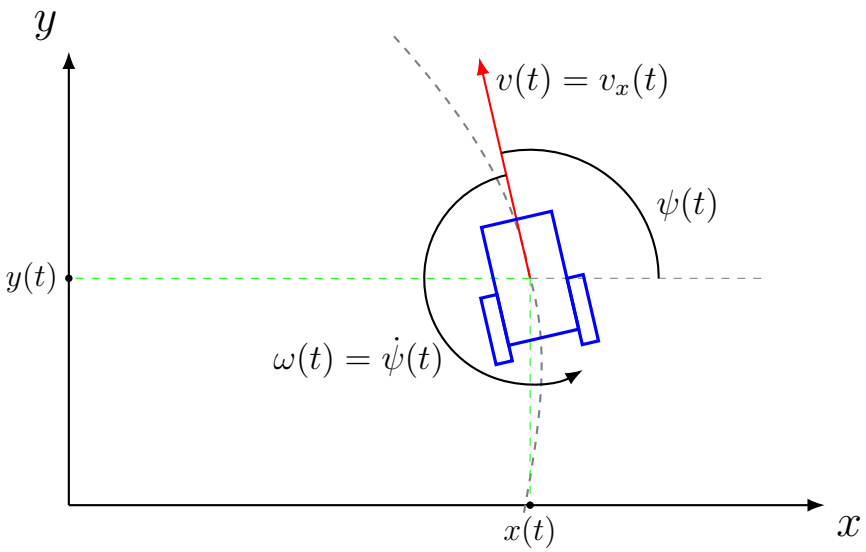

Figure 3.11: A diagram showing the robot in planar space along with the defined states in the CT model.

The state vector in the CT model thus consists of $x(t),\ y(t),\ v(t),\ \psi(t),\ \omega(t)$, meaning the continuous-time CT model can be expressed as:

$$\underbrace{\begin{bmatrix}\dot{x}(t)\\ \dot{y}(t)\\ \dot{v}(t)\\ \dot{\psi}(t)\\ \dot{\omega}(t)\end{bmatrix}}_{\dot{\mathbf{x}}(t)} = \underbrace{\begin{bmatrix}v(t)\ \cos(\psi(t))\\ v(t)\ \sin(\psi(t))\\ 0\\ \omega(t)\\ 0\end{bmatrix}}_{\tilde{f}(\mathbf{x}(t))} + \underbrace{\begin{bmatrix}0 & 0\\ 0 & 0\\ 1 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix}q_c^v(t)\\ q_c^\omega(t)\end{bmatrix}}_{\mathbf{q}_c(t)} \tag{3.8}$$

where $\mathbf{q}_c(t)$ is white Gaussian process noise present in the linear and angular velocity dimensions, respectively.
To implement the model in the ROS package, the model is discretised using both an Euler approximation and an analytical solution of the system. To find the approximation, Hammarstrand makes use of a *Modified Euler method* [10]:

$$\mathbf{x}(t+T) \approx \mathbf{x}(t) + T\ \tilde{f}(\mathbf{x}(t)) + \int_t^{t+T} \tilde{\mathbf{q}}(\tau)\ d\tau \tag{3.9}$$

where T is the sampling interval, such that the discrete-time motion model can be

expressed as:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + T\ \tilde{f}(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1} \tag{3.10}$$

$$\Longleftrightarrow$$

$$\begin{bmatrix} x_k \\ y_k \\ v_k \\ \psi_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + T \cdot v_{k-1} \cos(\psi_{k-1}) \\ y_{k-1} + T \cdot v_{k-1} \sin(\psi_{k-1}) \\ v_{k-1} \\ \psi_{k-1} + T \cdot \omega_{k-1} \\ \omega_{k-1} \end{bmatrix} + \mathbf{q}_{k-1} \tag{3.11}$$

where $\mathbf{q}_{k-1} \sim \mathcal{N}(0, \mathbf{Q}_{k-1})$ and $\mathbf{Q}_{k-1} = T\,\mathbf{G} \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \mathbf{G}^T = \mathrm{diag}(0,\ 0,\ T\sigma_v^2,\ 0,\ T\sigma_\omega^2)$.

The exact expression of the discretisation is found by setting the noise term $\mathbf{G} \cdot \mathbf{q}_c(t)$ to 0 and solving the deterministic part of Eq. (3.8) analytically, i.e. $\dot{\mathbf{x}}(t) = \tilde{f}(\mathbf{x}(t))$. Combined with the noise component acquired from the Modified Euler method, we get:

$$\underbrace{\begin{bmatrix} x_k \\ y_k \\ v_k \\ \psi_k \\ \omega_k \end{bmatrix}}_{\mathbf{x}_k} = \underbrace{\begin{bmatrix} x_{k-1} + \frac{2v_{k-1}}{\omega_{k-1}} \sin\left(\frac{\omega_{k-1} \cdot T}{2}\right) \cos\left(\psi_{k-1} + \frac{\omega_{k-1} \cdot T}{2}\right) \\ y_{k-1} + \frac{2v_{k-1}}{\omega_{k-1}} \sin\left(\frac{\omega_{k-1} \cdot T}{2}\right) \sin\left(\psi_{k-1} + \frac{\omega_{k-1} \cdot T}{2}\right) \\ v_{k-1} \\ \psi_{k-1} + T \cdot \omega_{k-1} \\ \omega_{k-1} \end{bmatrix}}_{f(\mathbf{x}_{k-1})} + \mathbf{q}_{k-1} \tag{3.12}$$

where $\mathbf{q}_{k-1} \sim \mathcal{N}(0, \mathbf{Q}_{k-1})$ and $\mathbf{Q}_{k-1} = T\,\mathbf{G} \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \mathbf{G}^T = \mathrm{diag}(0,\ 0,\ T\sigma_v^2,\ 0,\ T\sigma_\omega^2)$.

For the UKF, the exact discretisation of the motion model (Eq. 3.12) is better suited to producing pose estimations when the robot follows curved trajectories than the Euler approximation (Eq. 3.11), since the Euler approximation fails to compensate for the heading angle changing during the prediction interval, whereas this is accounted for by the terms $\frac{\omega_{k-1} \cdot T}{2}$ in Eq. 3.12 [10]. However, in situations where ω_k is close to 0, i.e., the robot is moving in a straight line, the pose estimation of the UKF with the exact discretised motion model will become undefined. Thus, under such conditions, the UKF will switch to the Euler approximated motion model to keep the filter well-defined.

4

Results

This chapter presents how the dynamic filter improves localisation robustness and compares the performance of a holonomic and a non-holonomic motion model. Additional results concern the effects of dynamic filtering on mapping and the filter's computational load.

4.1 Localisation in a feature-sparse environment

To evaluate whether the dynamic filter can increase the robustness of localisation in a feature-sparse environment, the filter was tested in an area where SLAM would normally perform well, but the LiDAR range was reduced in software to 5.5 m. The reduced LiDAR range emulates a feature-sparse environment and allows the recorded measurement data to be replayed with the full LiDAR range to obtain an approximate ground truth.

In the tests, the robot was driven forward manually without any dynamic objects to generate a map and establish good initial conditions. The robot was then set to follow the pattern in Figure 3.2, but was stopped after the first turn. During this autonomous phase, people walked around the robot to introduce dynamic object disturbances. Figure 4.1 shows all of the estimated positions of the robot from startup to shutdown with the corresponding map generated with the full LiDAR range, whereas Figure 4.2 shows the estimated position of the robot during the autonomous phase.

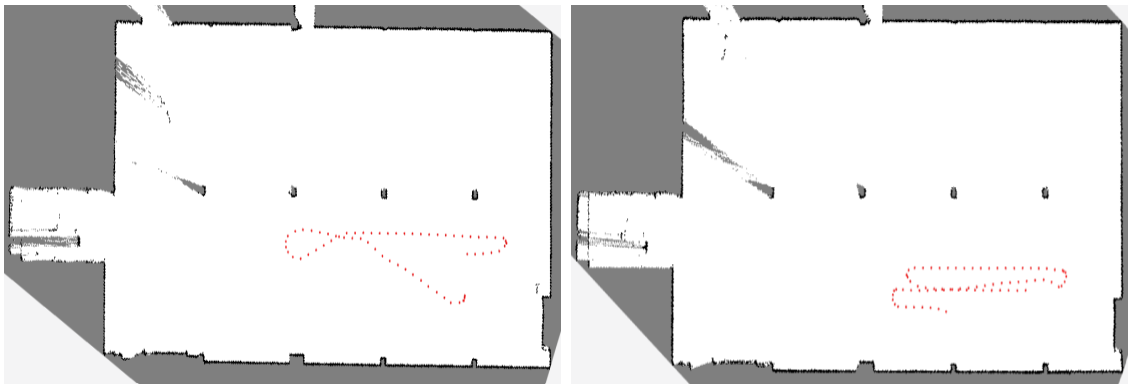

Figure 4.1: Map generated by SLAM Toolbox with the full LiDAR range. The red dots show all of the estimated poses of the robot during the test without the dynamic filter (left) and with the filter (right).

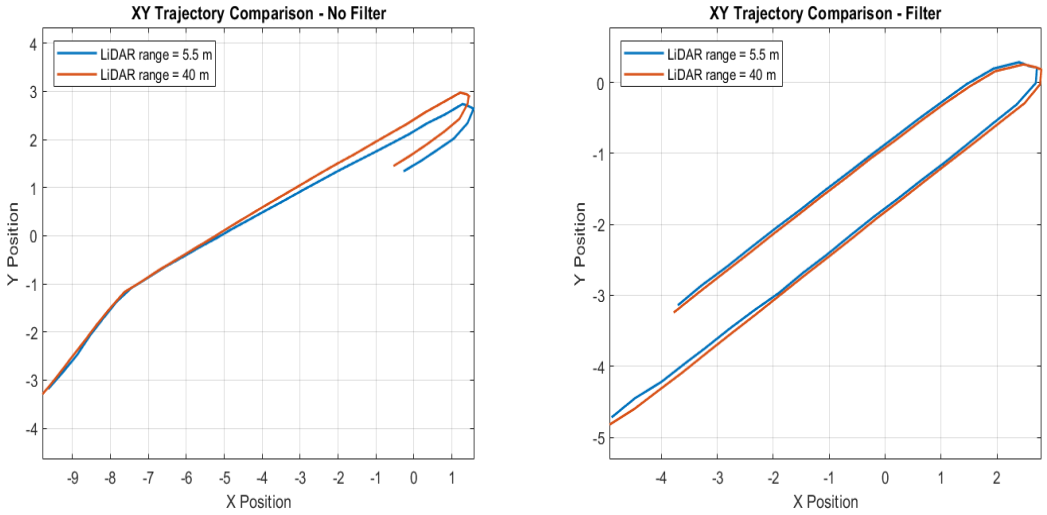

Figure 4.2: Graph comparing the estimated trajectory of the robot with the different LiDAR ranges. Note that the two graphs do not have the same coordinate system, as the origin is determined by the robot's starting position.

The x and y components of the trajectories in Figure 4.2, as well as the yaw error, can be seen in Figures 4.3 and 4.4 with and without the dynamic filter, respectively. The Root Mean Square Error (RMSE) is listed in Table 4.1.

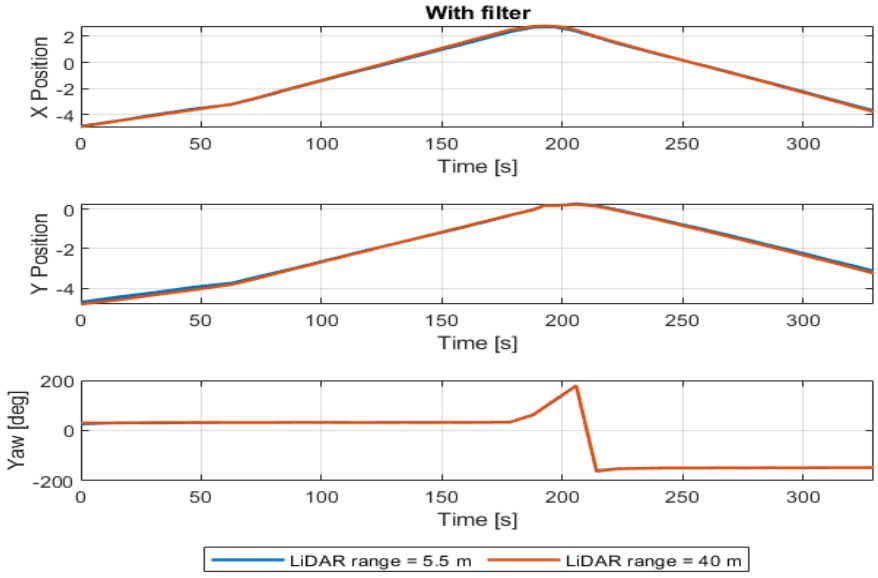

Figure 4.3: Graphs showing the translational position and the yaw of the robot with both LiDAR ranges, when using the dynamic filter.

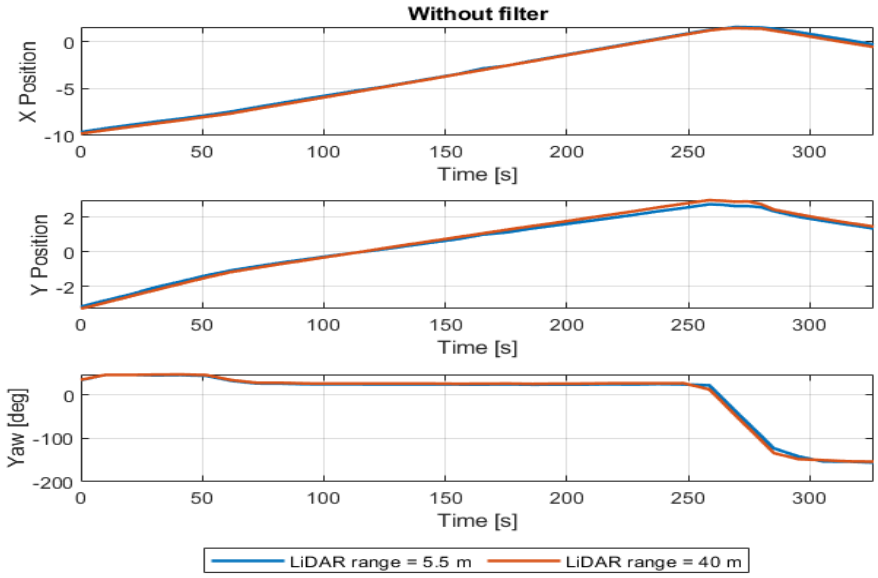

Figure 4.4: Graphs showing the translational position and the yaw of the robot with both LiDAR ranges when not using the dynamic filter.

Table 4.1: Table of the RMSE values of the graphs seen in Figures 4.3 and 4.4 between the estimated pose using the limited and full LiDAR range.

RMSE	x-position	y-position	yaw-orientation
With filter	0.06789 m	0.06549 m	0.93892°
Without filter	0.15026 m	0.14981 m	4.60992°

4.2 Motion model drift comparison

As previously mentioned, localisation robustness was evaluated using real data; therefore, there is no known ground truth. This section presents a comparison of the drift over time in the IMU-Wheel odometry estimations obtained using two different motion models, namely the default motion model found in robot_localization as well as the implemented CT model (described in section *CT model implementation*). The drift is measured by the difference between the SLAM Toolbox pose estimations and the odometry pose estimation. Furthermore, the SLAM output in this feature-dense environment serves as a proxy for the ground truth.

The test for model drift comparison was performed by setting the robot to operate autonomously in an area roughly 41.5 m^2 in size, as shown in Figure 4.5. The robot had a constant speed of 1.5 m/min and the duration of the test was roughly 58 minutes. The position drift, as well as the yaw angle drift, of the recorded data on the robot with factory settings can be observed in Figure 4.8. The recorded data was replayed twice: once with the default motion model and once with the CT model. This ensures a fair comparison, as the initial recording began with a partially built map, whereas the replay starts from an empty map. The respective position- and yaw angle drifts for the motion models can be observed in Figure 4.6, along with the differences between them in Figure 4.7.

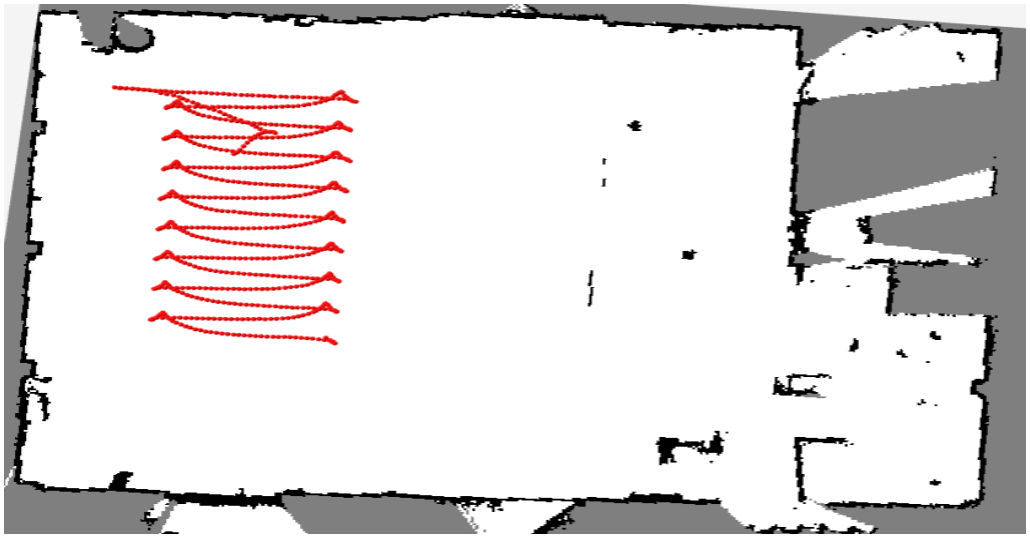

Figure 4.5: Map generated by SLAM Toolbox during the odometry drift test as well as the trajectory the robot drove, coloured in red.

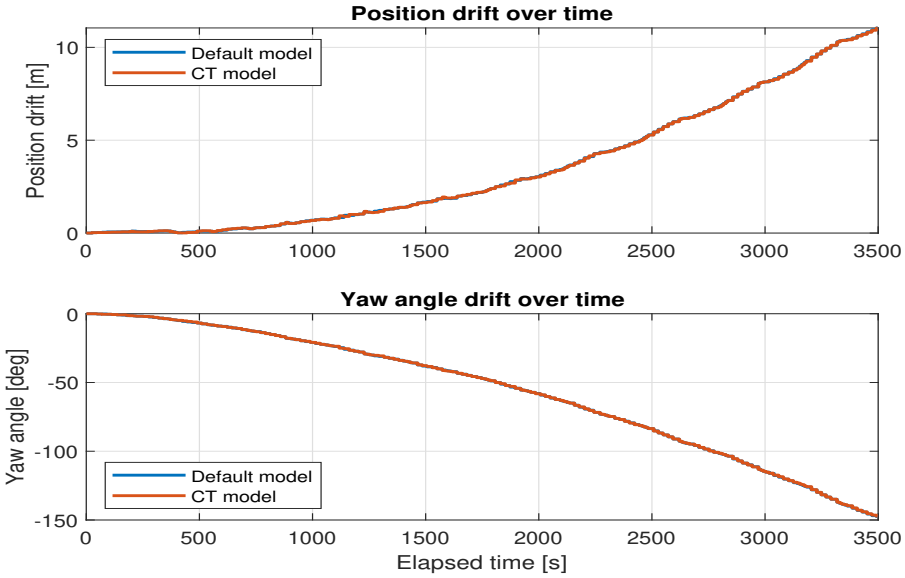

Figure 4.6: A diagram showing the position- and yaw angle drift over the course of the performed odometry drift test for both motion models. The position drift refers to the Euclidean distance between the SLAM Toolbox and odometry pose estimations, whereas the yaw angle drift simply refers to the difference in degrees over time between respective pose estimations.

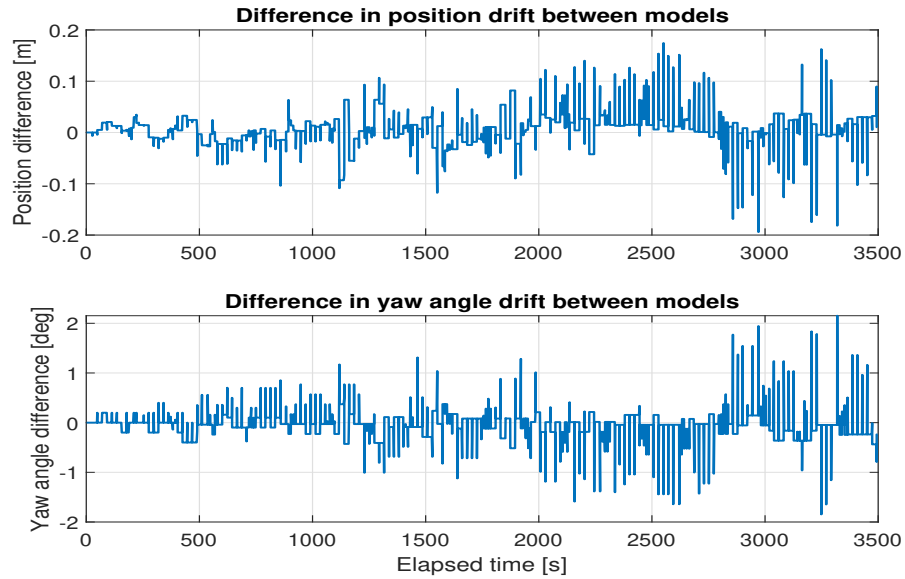

Figure 4.7: A diagram showing the difference between the position and yaw angle drift values between the two motion models over the course of the performed odometry drift test. That is, the difference between the UKF estimations: the pose estimation with the default omnidirectional model subtracted from the estimation with the CT model at each filter output.

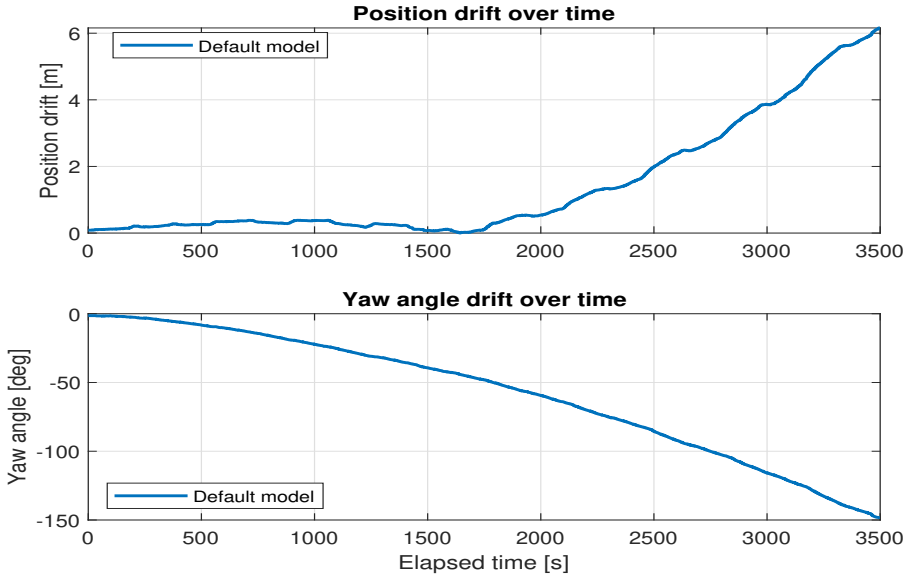

Figure 4.8: A diagram showing the position- and yaw angle drift over the course of the performed odometry drift test during the recording, i.e. the factory setting robot with only the default motion model. The position drift refers to the Euclidean distance between the SLAM Toolbox and odometry pose estimations, whereas the yaw angle drift simply refers to the difference in degrees over time between respective pose estimations.

4.3 Dynamic filter mapping

The goal of the dynamic filter was to increase the robustness of the localisation. However, since dynamic filters are often used offline to create a cleaner map (particularly useful in an urban environment), the effect the dynamic filter has on mapping was also investigated.

A robot was manually driven around in an indoor industrial environment while recording the LiDAR, IMU, and wheel encoder measurements. The measurement data was then used to generate a map using SLAM Toolbox offline with different dynamic filter configurations. Figure 4.9 shows the map generated with no dynamic filter, and Figure 4.10 shows the map generated with the dynamic filter using both POI and POG classification. When the LiDAR measurements are filtered using POG classification, the resulting map is noticeably sparser than the maps generated with POI classification.

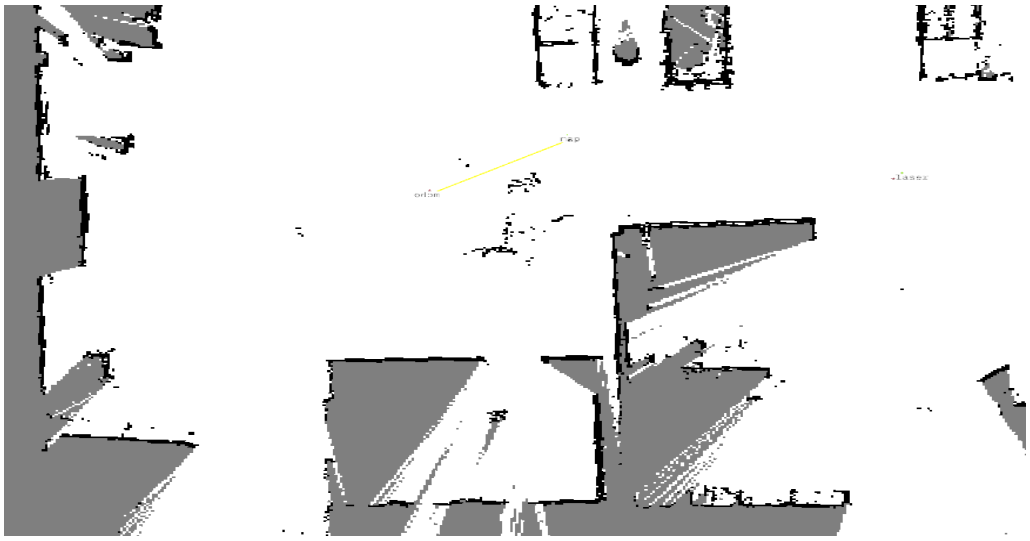

Figure 4.9: Map of an indoor industrial environment generated with SLAM Toolbox.

The centre of the maps in Figures 4.9-4.10 shows a sparse point cloud caused by the people operating the robot, who were mostly stationary during the data recording. The dynamic filter using either POI or POG classification showed no significant difference in the impact of humans on the mapping process compared with the unfiltered map.

Figure 4.10: Map of an indoor industrial environment generated with SLAM Toolbox, utilising the dynamic filter with POG classification on the left and POI classification on the right.

4.4 Dynamic filter computational load

The dynamic filter's computation time was evaluated on the same hardwaren mounted on the robot: an ARM CPU with 8 GB of RAM. The CPU is configured with six cores and a maximum clock speed of 2 GHz. The test was conducted by replaying data and only running the object segmentation and object classification nodes.

Combining the mean computation time of the object segmentation node and object classification seen in Figure 4.11 gives a total average computation time of around 20 ms, which is a potential operating frequency of 50 Hz. The object segmentation not only contains the modified DBSCAN algorithm but also the matching of objects from different LiDAR frames, as well as the conversion of objects' positions to world coordinates. The modified DBSCAN algorithm by itself has an average computation time of 1.3 ms, see Figure 4.12.

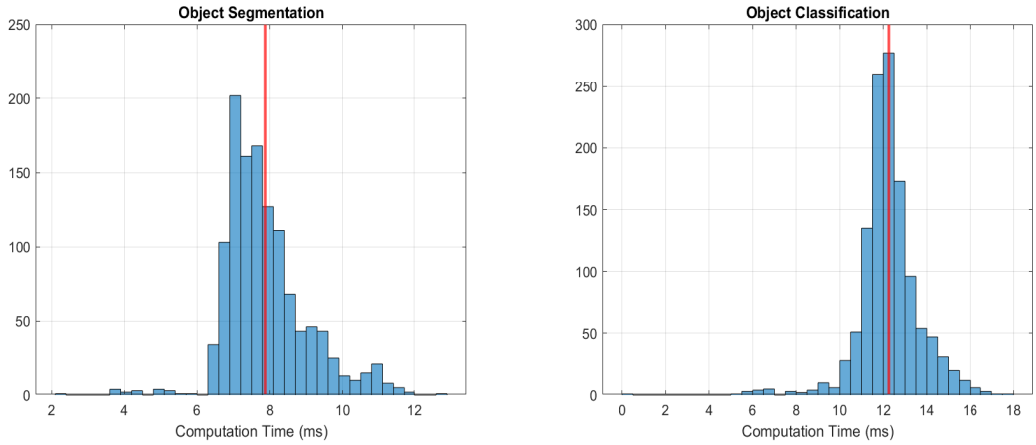

Figure 4.11: Histogram of the object segmentation and classification computation times for each LiDAR scan. The red line shows the mean computation time.

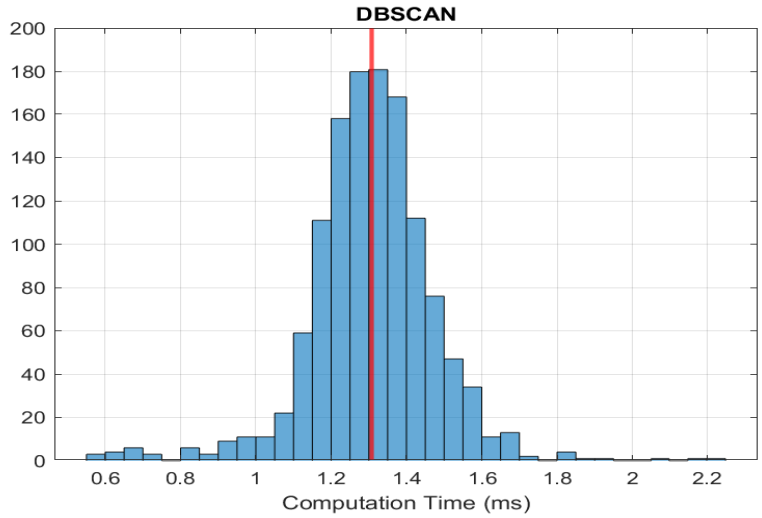

Figure 4.12: Histogram of the computation time for each LiDAR frame for the modified DBSCAN algorithm.

4.4.1 DBSCAN computational complexity

To assess the viability of the object segmentation algorithm for higher resolution LiDARs, the modified DBSCAN algorithm's computation time was measured for

various sets of artificially generated data, ranging from 1 617 to 11 000 points. As seen in Figure 4.13, the computation time of the modified DBSCAN exhibits linear behaviour. The original DBSCAN's computation time scales quadratically with the number of points. However, note that [1] shows DBSCAN only has a time complexity between linear and quadratic.

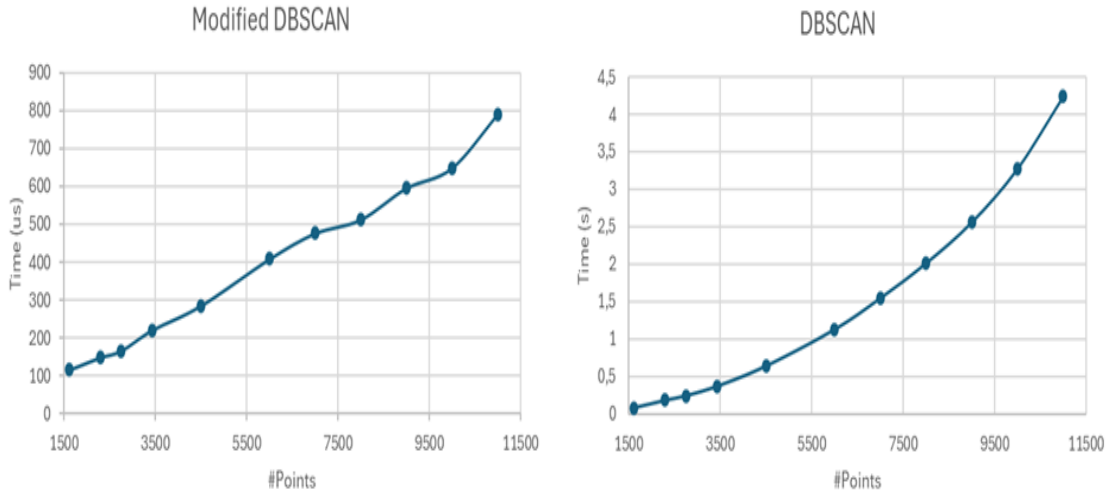

Figure 4.13: A graph of the computation times for the modified DBSCAN algorithm and the regular DBSCAN algorithm, for different numbers of points to segment. Observe that the graph for the modified DBSCAN is in microseconds and the regular DBSCAN is in seconds.

5

Discussion

The goal of this thesis was to create a dynamic filter to improve localisation robustness in a feature-sparse environment and evaluate the effect of motion model choice on localisation. This chapter discusses the results from the previous chapter as well as potential improvements in future work.

5.1 Localisation

As indicated in Table 4.1, the dynamic filter does provide some benefits in a feature-sparse and dynamic environment, as the estimates obtained with the limited LiDAR range were closer to those obtained with the full range when the dynamic filter was used.

It is difficult to quantify the exact conditions where the dynamic filter is effective. SLAM Toolbox with the dynamic filter still requires the presence of some useful static features (feature-sparse, not feature-less) in order to perform adequately. However, it can be stated that the dynamic filter reduces the number of static features required to navigate in a dynamic environment, though the exact limits have not been tested. Determining the limits of this solution is difficult because scan quality depends on more than the number of points per frame. Measurements at longer ranges are more susceptible to noise, and points on corners provide more useful localisation information than points on a flat wall.

There are several tunable parameters for the object segmentation and classification that impact the performance of the dynamic filter. The parameter values (found in Appendix A) in this thesis were tuned based on real data. The ideal parameter values will depend on the robot and the LiDAR used. The high static weight limit L_s used might not be feasible on a faster platform that has less time to observe an object, but this can be mitigated by using a 360° LiDAR rather than the segmented LiDAR used in this thesis, where objects will frequently enter and leave the LiDAR FOV.

5.2 Mapping

The generated maps shown in Figure 4.10 highlight a key issue with using an online dynamic filter to create a cleaner map. Regardless of tuning or the precision of the filter, a dynamic object can always be stationary long enough to appear static. An

offline dynamic filter that has access to all of the measurements can remove the dynamic object from the measurements before it starts moving. POG classification creates a sparser map due to the time it takes for an object to be classified as static. A static object might leave the LiDAR's FOV before the static weight has time to reach the static weight limit. Therefore, the LiDAR points from the object are never passed to the SLAM algorithm and never appear on the map. POG classification can produce a map of a quality similar to that produced by POI classification or the unfiltered case, but it would take more time as the object in the map needs to be visible for a longer period.

One possible solution would be to retroactively remove the parts of the map associated with an object when it is first classified as dynamic. However, this method would make an incorrect dynamic classification more consequential. Such a misclassification is less likely to occur with an offline dynamic filter that has access to all of the measurement data. Semantic segmentation with a camera is a potential solution as it would allow the detection and removal of recognisable dynamic objects such as humans immediately, ensuring a clean map.

An offline dynamic filter will always produce a superior map compared to an online dynamic filter. The choice will depend on the use case. A robot that will operate in the same environment most of the time, such as in a city or a warehouse, could benefit greatly from spending more time creating a high-quality map, as it could be used throughout its operation. However, if the robot would instead operate in numerous locations, such as different construction sites, then it would need to spend a significant amount of time at every new location for mapping. The additional mapping time reduces the overall efficiency of the robot, and if the mapping is done manually, it removes any gain from using an autonomous robot.

5.3 Dynamic filter computational load

Compared to the dynamic filter proposed in [4], with a mean computation time of 336 ms per LiDAR frame, the dynamic filter presented in this thesis is significantly faster with a mean computation time of 20 ms. This computational speed does come with certain trade-offs. The modified DBSCAN, for example, is a less general version of DBSCAN, requiring a list of ranges ordered by their bearing, with a constant and small enough angle increment α such that the approximation $\cos \alpha \approx 1$ is reasonable. Additionally, the matching of objects between LiDAR frames is quite naive, making it difficult to track objects for longer periods of time.

The low computation time of the dynamic filter does leave room for improvements. The current implementation allows an operating frequency of up to 50 Hz, which is significantly greater than the scan rate of most LiDAR sensors, which cap at around 20 Hz. This means that there is room for a more complex and accurate algorithm while still allowing for online operation. The computation time of the modified DBSCAN algorithm could be decreased by disabling the removal of small clusters of points (essentially setting the minimum number of points in a cluster to two) and not merging clusters that are discontiguous.

5.4 Motion model comparison

The results of the motion model comparison indicate that the implemented CT model did not produce a significant improvement over the default omnidirectional motion model in the performed odometry drift test. As seen in Figure 4.6, the position and yaw angle drift curves for the two models are nearly indistinguishable over the duration of the test. This is further supported by Figure 4.7, where the difference between the two pose estimates appears to be small and irregular, akin to white noise. Thus, the difference between the models appears to be closer to small estimation fluctuations than to a systematic improvement. As far as the implementation in RL is concerned, the CT model therefore did not provide a worthwhile improvement in odometry drift under the tested conditions.

One possible explanation is that the default motion model, despite being omnidirectional, is sufficiently general to model the dominant motion of the robot when combined with IMU and wheel odometry measurements. The robot is non-holonomic and cannot generate instantaneous lateral motion, but this does not necessarily mean that the default model will produce significant lateral errors in practice. If the measurement updates sufficiently correct the estimated motion, the additional degrees of freedom in the omnidirectional model may have little influence on the final estimate. In that case, the difference between the default motion model and the CT model becomes small, especially if the measurement updates dominate the prediction step of the filter.

The intended motion of the robot may also help explain the similarity between the models. The robot mainly performs straightforward motion followed by U-turns to assume a parallel trajectory in the opposite direction. During straight motion, the yaw rate is close to 0, and the CT model behaves similarly to a simpler CV model. The main advantage of the CT model is expected during curved motion, where the change in heading over the sampling interval is explicitly accounted for. However, if curved motion only constitutes a limited part of the trajectory, this advantage may not become clearly visible in the final drift results.

Another important factor is the practical implementation within RL. The package uses a fixed 15-dimensional state representation, whereas the CT model considered in this thesis is naturally expressed using a reduced five-dimensional vector. Consequently, the CT model could not be implemented as a fully independent reduced-state UKF without consequential changes to the package architecture. Instead, the CT model was adapted to the existing state-space structure, while states not included in the CT model were propagated in a simplified manner. The results should therefore be interpreted as a comparison between the default RL model and a CT model adapted to the constraints of RL, rather than as a comparison with a filter designed specifically around the CT model.

The physical behaviour of the robot may also limit the practical benefit of a more accurate motion model. During operation, as mentioned in section *Platform*, the robot is exposed to considerable vibrations and motor-induced disturbances from additional onboard actuators. These disturbances may cause the robot to rock slightly and deviate from the ideal planar motion assumed by the CT model. As a result,

the potential accuracy gained by using a more physically appropriate non-holonomic motion model may be masked by process noise, wheel slip, and unmodelled mechanical disturbances. In other words, even if the CT model better represents the ideal motion of a differentially driven robot, the real platform may be noisy enough that this advantage does not significantly affect the final pose estimate.

The difference between the original recording and the playback results should also be interpreted with caution. The drift observed during playback differs from that observed during the original recording, as shown in Figure 4.8. This is due to differences in map initialisation: the robot begins mapping whenever it is powered on and driven, including while it is moved into place before data recording starts. Therefore, the comparison between the CT model and the default model is most meaningful when both models are evaluated under the same playback conditions, while the original recording is included for the sake of completeness and supporting observations.

5.5 Future work

This thesis demonstrates the feasibility of using dynamic filtering and a reduced state-space model, with some aspects left for further investigation and future improvement. The dynamic filter itself is not fully accurate and requires further work for increased robustness. Fortunately, there is room for a more complex and computationally demanding algorithm without sacrificing the LiDAR sampling rate. The current implementation of object matching across different LiDAR frames could be improved by taking object geometry into account. The object segmentation does not account for measurement noise, and the purely distance-based approach implemented means that objects are not consistently segmented. The inconsistent object segmentation can cause a single feature to be treated as multiple different objects, each with its own weights. Increasing segmentation consistency would improve classification accuracy and increase classification time.

As for the reduced state-space model in the IMU-Wheel odometry filter, overall, the results do not indicate that the CT model improves localisation robustness for the tested robot in the performed experiment. However, this does not necessarily mean that the CT model is unsuitable for the platform. The CT model remains theoretically well-motivated, since it explicitly reflects the non-holonomic structure of a two-wheeled, differentially driven robot. Rather, the results suggest that the practical benefit of the CT model was not observable within the current implementation, test setup, and noise conditions. A more conclusive evaluation would require a filter designed specifically for the reduced CT state vector and tests in scenarios where the prediction model has greater influence and longer periods of sparse LiDAR observations, i.e. operation in vast spaces that motivated this thesis.

In summary, further testing is required to fully evaluate the performance of the dynamic filter and the CT model. All tests were performed on a platform with a very low speed and significant process disturbances, which is not representative of most robotic platforms.

6

Conclusion

The goal of this thesis was to investigate the implementation of an online dynamic filter and a non-holonomic motion model to improve localisation robustness in a dynamic and feature-sparse environment. The dynamic filter first segments the measurement data provided by a 2D LiDAR into objects. The objects are tracked using their two edges and the average position of all of the associated points. The velocities of the three tracked points are estimated using a linear Kalman filter, which is used to determine if the object is dynamic or static. The dynamic filter was tested in a dynamic environment with a reduced LiDAR range to emulate a feature-sparse environment. The dynamic filter demonstrated better localisation performance than the unfiltered setup.

The robot's default IMU-Wheel odometry UKF provides pose estimations in areas where LiDAR observations are sparse. The results indicate that the CT model offers no significant improvement over the omnidirectional motion model when implemented in RL. Thus, if the preference is for the filter to remain in RL, there is no merit in altering its motion model. However, if the CT model is investigated further to assess its potential, it should be done separately from the confined architecture of RL.

Bibliography

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231, AAAI Press, 1996.

[2] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 163–169, 2016.

[3] G. Kim and A. Kim, "Remove, then revert: Static point cloud map construction using multiresolution range images," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10758–10765, 2020.

[4] D. Wang, I. Posner, and P. Newman, "Model-free detection and tracking of dynamic objects with 2d lidar," *The International Journal of Robotics Research*, vol. 34, 06 2015.

[5] R. Pepy, A. Lambert, and H. Mounier, "Reducing navigation errors by planning with realistic vehicle model," in *2006 IEEE Intelligent Vehicles Symposium*, pp. 300–307, 2006.

[6] N. Uddin, H. Nugraha, A. Manurung, H. Hermawan, and T. M. Darajat, "Kinematics modeling and motions analysis of non-holonomic mobile robot," in *2022 5th International Conference on Information and Communications Technology (ICOIACT)*, pp. 220–225, 2022.

[7] H. Xu and S. Yang, "Real-time collision-free motion planning of nonholonomic robots using a neural dynamics based approach," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 3, pp. 3087–3092 vol.3, 2002.

[8] B. Triggs, "Motion planning for nonholonomic vehicles: An introduction," Tech. Rep. inria-00548415, INRIA, 1993.

[9] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *2008 11th International Conference on Information Fusion*, pp. 1–6, 2008.

[10] L. Hammarstrand, "5.5 nonlinear motion models." `https://www.youtube.com/watch?v=VUX8L0fuOmU`, Feb 2021. [Online video]. Accessed: Apr. 23, 2026.

[11] S. Macenski and I. Jambrecic, "Slam toolbox: Slam for the dynamic world," *Journal of Open Source Software*, vol. 6, p. 2783, 05 2021.

[12] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[13] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, 2021.

[14] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," 2023.

[15] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[16] M. W. M. G. Dissanayake, P. Newman, H. F. Durrant-Whyte, S. Clark, and M. Csorba, "An experimental and theoretical investigation into simultaneous localisation and map building," in *Experimental Robotics VI*, (London), pp. 265–274, Springer London, 2000.

[17] M. Montemerlo, S. Thrun, D. Koller, and B. Webgreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 593–598, 2002.

[18] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[19] A. Yarovoi and Y. K. Cho, "Review of simultaneous localization and mapping (slam) for construction robotics applications," *Automation in Construction*, vol. 162, p. 105344, 2024.

[20] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[21] NVIDIA Corporation, "Nvidia isaac ros repositories and packages." `https://nvidia-isaac-ros.github.io/repositories_and_packages`, 2026. Accessed: 2026-02-25.

[22] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla, "24/7 place recognition by view synthesis," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1808–1817, 2015.

[23] F. Walch, C. Hazirbas, L. Leal-Taixé, T. Sattler, S. Hilsenbeck, and D. Cremers, "Image-based localization using lstms for structured feature correlation," 2017.

[24] F. Zangeneh, L. Bruns, A. Dekel, A. Pieropan, and P. Jensfelt, "A probabilistic framework for visual localization in ambiguous scenes," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3969–3975, 2023.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[26] J. Zhang and S. Singh, "Loam : Lidar odometry and mapping in real-time," *Robotics: Science and Systems Conference (RSS)*, pp. 109–111, 01 2014.

[27] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: low-drift, robust, and fast," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2174–2181, 2015.

[28] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, 2018.

[29] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 22–29, 2010.

[30] N. Banerjee, D. Lisin, S. R. Lenser, J. Briggs, R. Baravalle, V. Albanese, Y. Chen, A. Karimian, T. Ramaswamy, P. Pilotti, M. L. Alonso, L. Nardelli, V. Lane, R. Moser, A. O. Huttlin, J. Shriver, and P. Fong, "Lifelong mapping in the wild: Novel strategies for ensuring map stability and accuracy over time evaluated on thousands of robots," *Robotics and Autonomous Systems*, vol. 164, p. 104403, 2023.

[31] X. Xu, L. Guan, J. Zeng, Y. Sun, Y. Gao, and Q. Li, "When-to-loop: Enhanced loop closure for lidar slam in urban environments based on scan context," *Micromachines*, vol. 15, no. 10, 2024.

[32] G. Oguntala, R. Abd-Alhameed, S. Jones, J. Noras, M. Patwary, and J. Rodriguez, "Indoor location identification technologies for real-time iot-based applications: An inclusive survey," *Computer Science Review*, vol. 30, pp. 55–79, 2018.

[33] T. Moore, "robot_localization: Ros package documentation." `https://docs.ros.org/en/melodic/api/robot_localization/html/index.html`. Accessed: 2026-03-24.

[34] A. M. Bloch, *Nonholonomic Mechanics and Control*, vol. 24 of *Interdisciplinary Applied Mathematics*. New York, NY: Springer, 2 ed., 2015.

[35] S. Li, X. Chen, Y. Liu, D. Dai, C. Stachniss, and J. Gall, "Multi-scale interaction for real-time lidar data segmentation on an embedded platform," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 738–745, 2022.

[36] X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, and C. Stachniss, "Moving object segmentation in 3d lidar data: A learning-based approach exploiting sequential data," *IEEE Robotics and Automation Letters*, vol. 6, p. 6529–6536, Oct. 2021.

[37] M. Zeybek, "Spatio-temporal detection and filtering of dynamic objects in mobile lidar point clouds," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLVIII-4/W18-2025, pp. 371–378, 2026.

A

Parameter values

A.1 Dynamic filter parameters

Tables of parameter values used for the dynamic filter.

Table A.1: Object segmentation parameters

Parameter	Symbol	Value	Unit
Eps	ε	0.1	m
Minimum points	N_{min}	8	-
Object matching distance limit	D	0.25	m

Table A.2: Object classification parameters

Parameter	Symbol	Value	Unit
Dynamic weight limit	L_d	18	-
Static weight limit	L_s	50	-
Speed limit	L_v	0.08	m/s

Table A.3: Object velocity estimation parameters

Parameter	Symbol	Value	Unit
Position variance	σ_p^2	0.75	-
Velocity variance	σ_v^2	1.5	-
Measurement variance	σ_m^2	15	-
Velocity decay coefficient	γ	0.8	-

A.2 UKF parameters

Table of parameter values used for the UKF.

Table A.4: Pose estimation parameters

Parameter	Symbol	Value	Unit
Linear velocity variance	σ_v^2	0.01	-
Yaw rate variance	σ_ω^2	0.049	-
Spread of sigma points	α	0.001	-
Secondary scaling of sigma points	κ	0.0	-
Characterisation of state distribution	β	2.0	-

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY