LiDAR-based 2D EKF SLAM for Indoor Environments

Devrat Singh, Fikrican Ozgur,
Codrin-Matei Căciuleanu

Bachelor (BSc) in Electronics and Computer Engineering
Group ED6-1-F20

BSc Project (Automation and Control)

This document was typeset using Overleaf, an online LaTeX editor. The simulations were performed in Python 3.7.4. ROS Melodic Morenia was used for information flow among system peripherals and Gazebo, running on Ubuntu 18.04, served as a simulation environment. The figures were created in MATLAB **R**2019**b**, Inkscape 0.92.4 and Python 3.7.4.

Faculty of Engineering and Science
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
LiDAR-based 2D EKF SLAM for Indoor Environments

Theme:
Automation and Control

Project Period:
Spring Semester 2019

Project Group:
ED6-1-F20

Participant(s):
Devrat Singh
Fikrican Ozgur
Codrin-Matei Căciuleanu

Supervisor(s):
Petar Durdevic Løhndorf
Daniel Ortiz-Arroyo

Copies: 1

Page Count: 101

Date of Completion:
June 4, 2020

Abstract:

The present thesis expands upon the famous online SLAM problem predominant in the field of robotics, with a focus on wheeled vehicles in indoor environments. The devised solution includes the use of a 2D LiDAR sensor for perception of the surrounding. The a priori pose estimate of the EKF is determined by the velocity motion model. Line features, extracted based on the Split-and-Merge algorithm, are evaluated for possible data associations on the basis of their maximum likelihoods and utilised to compute the a posteriori state estimate. The performance of localization with midpoint enhancement and SLAM algorithms, coded in Python 3, are first assessed in a custom built test environment. Upon the achievements of successful results, efforts towards ROS-based SLAM simulation using the Gazebo tool are put forward. Ultimately, a real setup implementation has been precluded from happening. Nevertheless, the simulation results evince the success of the thesis.

Table of Contents

List of Figures

Nomenclature

Generic	
$\cdot^{-1}$	Matrix inverse
$\cdot^{\top}$	Matrix transpose
$\cdot_{1:t}$,	Indication of all the past and present values of a variable
$\cdot_{\text{mid}}$	Indication that a function or variable is related to the midpoint of a line landmark
$\cdot_{t-1}, \cdot_t, \cdot_{t+1}$	Indication of past, present and future values of a variable
Δt	Time step
$\mathbb{R}^N$	The N-dimensional real number set
$\varnothing$	The empty set
$\mathbf{I}, \mathbf{0}$	Identity matrix and zero matrix for which dimensions are either evident or changing based on the circumstances
$\mathbf{I}_{N\times N}, \mathbf{0}_{N\times N}$	The $N \times N$ identity matrix and the $N \times N$ zero matrix
h, i, j, k	Indices used for counting
State Estimation. Robot and Environment-Related Parameters	
$\alpha_1, \ldots, \alpha_6$	Robot-specific error parameters utilized for modeling the process noise
$\hat{v}_t, \hat{\omega}_t, \hat{\gamma}_t$	Noisy translational velocity, angular velocity and a noisy signal affecting the final heading
κ	Intersection flag
$\hat{\mathbf{x}}_{\mathcal{B}}, \hat{\mathbf{y}}_{\mathcal{B}}$	Unit vectors of the body (local) reference frame
$\hat{\mathbf{x}}_{\mathcal{G}}, \hat{\mathbf{y}}_{\mathcal{G}}$	Unit vectors of the global reference frame
$\overline{\boldsymbol{\mu}}_t, \boldsymbol{\mu}_t$	A priori and a posteriori state vector mean
$\overline{\boldsymbol{\Sigma}}_t, \boldsymbol{\Sigma}_t$	A priori and a posteriori state covariance matrix
$\boldsymbol{\varepsilon}_t, \boldsymbol{\delta}_t$	Process and observation noise vectors
$\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t$	State-transition, input and observation matrices for a linear state space
$\mathbf{g}, \mathbf{h}$	State-transition and observation vector functions for a nonlinear state space
$\mathbf{G}_t, \mathbf{L}_t, \mathbf{H}_t$	Jacobians of $\mathbf{g}$ w.r.t. $\mathbf{x}_t$ and $\boldsymbol{\varepsilon}_t$ and the Jacobian of $\mathbf{h}$ w.r.t. $\mathbf{x}_t$, all evaluated at the mean
$\mathbf{K}_t$	Optimal Kalman gain
$\mathbf{m}, \mathbf{m}_k$	The feature map of the environment and the k-th landmark
$\mathbf{p}_{\mathcal{G}}$	Global position vector for the robot
$\mathbf{Q}_t, \mathbf{R}_t$	Process and observation noise covariance matrices
$\mathbf{S}_t$	Innovation Covariance Matrix

$\mathbf{u}_t$	Control vector
$\mathbf{x}_t$	Pose vector and state vector in localization
$\mathbf{z}_t^i, \hat{\mathbf{z}}_t^k$	Sensor measurement of the i-th feature and expected measurement of the i-th feature computed from the k-th landmark
r_k, ψ_k	Polar coordinates of the k-th line landmark
v_t, ω_t	Translational and angular robot's velocities
x_c, y_c, r_c	Coordinates of the center of the robot's circular trajectory and its radius
x_k, y_k	Cartesian coordinates of the k-th point landmark
x_t, y_t, θ_t	Robot's position and heading in the global frame
Feature Extraction	
$\mathcal{L}$	List of extracted features
$\mathcal{S}_i$, l_i	The i-th point segment/cluster and the corresponding line feature
$\overline{x}^i, \overline{y}^i$	Sensor-frame Cartesian coordinates of the midpoint of the i-th line feature
ρ_t^i, α_t^i	Polar coordinates of the i-th line feature in the sensor space
$\boldsymbol{\mathcal{A}}_h^i, \boldsymbol{\mathcal{B}}_h^i$	Jacobian matrices of $\mathbf{v}$ and $\mathbf{u}$ w.r.t. to the h-th point in the i-th segment
$\boldsymbol{\mathcal{C}}_h^i, \boldsymbol{\mathcal{D}}_h^i$	Covariance matrices of the Cartesian and polar coordinates for the h-th point in the i-th segment
$\boldsymbol{\mathcal{F}}(\cdot)$	Feature extractor function
$\mathbf{u}$	Function of polar coordinates of a measured point giving its Cartesian coordinates
$\mathbf{v}$	Function of Cartesian coordinates of points in a segment giving the line feature parameters associated with the segment
$\mathbf{z}_t^i, \boldsymbol{\mathcal{F}}_t^i$	The i-th extracted feature
d_k, ϕ_k	Sensor-frame polar coordinates of the k-th point acquired by the sensor
E	Squared sum of distances from all points in the i-th segment to the i-th line feature
M, Z, N	Number of data points acquired by the sensor, number of extracted features, number of landmarks
n_i	Number points in the i-th segment
$r_t^i, \varphi_t^i, s_t^i$	Polar coordinates of the i-th point feature in the sensor space and its signature
$S_{x^2}^i, S_{y^2}^i, S_{xy}^i$	Intermediate variables needed in finding the optimal line feature parameters
Data Association	
γ_1	Validation gate threshold
$\hat{c}_t^i, j(i)$	Maximum likelihood correspondence index for the i-th feature and the j-th landmark
$\mathbf{c}_t, c_t^i$	Correspondence vector and correspondence variable for feature i
A_k	Valid association area for the k-th landmark
D_{ik}	Normalized distance between feature i and landmark k
M_{ik}	Squared Mahalanobis distance between feature i and landmark k
$S_{w,ik}$	Weighted sum of the distances between feature i and landmark k (when

	the midpoint is included)
SLAM	
$\cdot_{\text{tent}}$	Indication that a variable is related to a tentative landmark
$\mathbf{\Lambda}_t, \mathbf{\Lambda}_{t,j}$	Sparse matrices required in inefficient SLAM implementation
$\boldsymbol{\mu}_{t,\mathbf{x}}, \boldsymbol{\mu}_{t,\mathbf{m}}$	A posteriori mean of the pose and map vectors
$\mathbf{\Sigma}_{N+1,N+1}, \mathbf{\Sigma}_{t,N+1,\text{all}}$	Covariance matrices of the new landmark with itself and the new landmark with the pose and all other landmarks
$\mathbf{\Sigma}_{t,\mathbf{mx}}, \mathbf{\Sigma}_{t,\mathbf{mm}}$	A posteriori covariance matrices of the map with the pose and the map with itself
$\mathbf{\Sigma}_{t,\mathbf{m}_i\mathbf{m}_j}$	A posteriori covariance matrix of the i-th landmark with the j-th landmark
$\mathbf{\Sigma}_{t,\mathbf{xx}}, \mathbf{\Sigma}_{t,\mathbf{xm}}$	A posteriori covariance matrices of the pose with itself and the pose with the map
$\mathbf{f}$	Inverse observation vector function
$\mathbf{F}_{t,\mathbf{x}}, \mathbf{F}_{t,\mathbf{w}}$	Inverse observation model Jacobian wrt the pose vector and with respect to $\mathbf{w}_t$
$\mathbf{G}_{t,\text{low}}$	Low-dimensional matrix used in the computation of the Jacobian $\mathbf{G}_t$
$\mathbf{G}_{t,\mathbf{x}}$	Motion model Jacobian wrt the pose vector
$\mathbf{H}_{t,\mathbf{x}}, \mathbf{H}_{t,\mathbf{m}}$	Observation model Jacobian wrt the pose vector and with respect to a certain landmark
$\mathbf{L}_{t,\mathbf{x}}$	Motion model Jacobian wrt the process noise in the pose vector
$\mathbf{m}_{N+1}, \overline{\boldsymbol{\mu}}_{N+1}$	New landmark vector and its estimated position
$\mathbf{w}_t$	Random variable with mean at the value of the current measurement
$\mathbf{y}_t$	State vector in SLAM
A, a, A_k, a_k	Maximum allowed number of iterations for the existence of a tentative landmark, the number of associations needed for the creation of a permanent landmark and the associated counters for the k-th landmark
P	Number of tentative landmarks
Probability	
$\chi^2(\cdot)$	Chi-squared distribution
$\mathcal{N}(\cdot)$	Normal Distribution
$\sigma.$	Variance of a one-dimensional continuous random variable
$p(\cdot \mid \cdot)$	Conditional probability density function
$p(\cdot)$	Probability density function of a continuous random variable

Preface

Acknowledgements

We would like to express our gratitude to the people without whom this thesis and our Bachelor's journey would not have been possible. We thank our supervisors: Dr. Petar Durdevic Løhndorf and Dr. Daniel Ortiz-Arroyo, whose insight and knowledge into the subject matter steered us through this project as well as encouraged us to explore ideas of our own. Our most sincere respect and thanks to all our previous supervisors who have guided us in our work in the last 3 years. Each of you have furnished us with your precious time, energy, and expertise and we are richer for it: Dr. Dil Muhammad Akbar Hussain and Dr. Amin Hajizadeh.

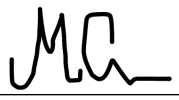
Finally, our journeys to this point would not have been possible without the unconditional support of our families. To our respective parents, who have always held their unyielding trust in us to make our own path. You are the ultimate role models and thank you for all the support that you have shown us.

The Bachelor's thesis entitled "LiDAR-based 2D EKF SLAM for Indoor Environments" was written by three students pursuing the Bachelor (BSc) in Electronics and Computer Engineering at Aalborg University Esbjerg. Hereafter, every mention of "we" refers to the three co-authors listed below.

Aalborg University, June 4, 2020

Devrat Singh
<dsingh17@student.aau.dk>

Fikrican Ozgur
<fyozgy17@student.aau.dk>



Codrin-Matei Căciuleanu
<ccaciu17@student.aau.dk>

To Devika

To İmren and Enes

To Mia and Călin

It's sad that we never get trained to leave assumptions behind.

Sebastian Thrun

Chapter 1

Introduction

In broad terms, the SLAM (Simultaneous Localization and Mapping) problem includes concurrent state estimation of an agent[1] by means of on-board sensors and the creation of a model representing interesting facets of the environment in which the robot performs. The problem emerges in the absence of both a map of the environment and knowledge of robot states. Thus, it comprises two strongly connected robotic problems: localization and mapping. When individually investigated, the former assumes a complete map and tries to estimate robot's pose (position and orientation) whereas the latter presumes known robot pose and constructs a map. On the other hand, SLAM algorithms tackle both localization and mapping simultaneously and constitute a significantly more difficult chicken-and-egg problem. Nevertheless, being a crucial technology enabling mobile robot navigation, SLAM is considered a principal problem for fully autonomous robotics applications [1].

What makes SLAM unique?

Before diving into the specifics of the SLAM problem, we would like to take a step back and discuss what renders it special. The answer to the question *What makes SLAM unique?* is the loop closure, which means nullifying the accumulated drift in the robot trajectory over time by making use of additional information obtained by revisiting a location [2]. Figure 1.1 illustrates the effect of loop closure in accurate state estimation by showing the significant improvement it brings to the pose estimation. Its place in autonomous robotics is unmitigated since it enhances the overall performance with its estimate correction. If loop closure is excluded, SLAM reduces to odometry, which is the robot pose estimation method that is implemented by integrating the output of the wheel encoders [3]. However, odometry-based pose estimation is not accurate at all after a few meters due to its drift [4]. This lack of certainty is indeed what has actually kick-started the research on the idea of reducing drift by means of landmark observations in the late 20^{th} century [3, 5].

[1] In the context of this thesis, it will also be referred to as a robot.

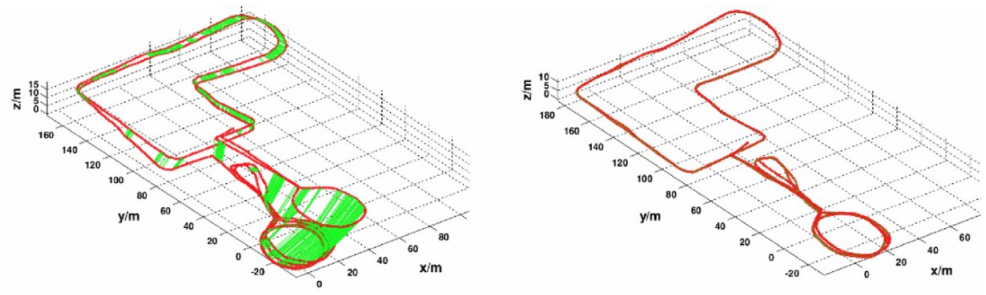


Figure 1.1: *Left:* Pose-graph from visual odometry (red) with the loop-closures (green); *Right:* Final result after pose-graph optimization (Copyright © 2014, IEEE)

History of SLAM

The period from 1984 to 2004, which marks the early days of SLAM research, is referred to as the *classical age*. In this epoch, the main probabilistic SLAM approaches such as Extended Kalman Filter (EKF), Rao-Blackwellised Particle Filters (RBPF) and maximum likelihood estimation were introduced. Moreover, challenges in efficiency and data association were outlined. The results obtained during the classical age have been succinctly reviewed in [6] by Durrant-Whyte and Bailey. A more detailed description of classical age SLAM algorithms can be found in the *Probabilistic Robotics* book [7] written by Thrun, Burgard, and Fox, which also constitutes the primary reference for this thesis. Subsequently, the *algorithmic analysis age* between 2004 and 2015 has witnessed the study of certain SLAM properties such as convergence, observability and consistency. Furthermore, open-source SLAM libraries were released and the positive effect of sparse sensing in the SLAM algorithm efficiency has been better understood [3].

This period of continuous research on SLAM has led to various technological developments, including visual and inertial odometry. These more recent odometry algorithms produce smaller drift compared to conventional encoder-based odometry. Their drift is equal or smaller than 0.5% the trajectory length [8]. Considering this acceptable minor drift, the question *Is SLAM really needed anymore?* becomes valid and there are a couple of strong arguments to it.

Is SLAM really needed?

First of all, the state-of-the-art visual-inertial odometry algorithms such as [9] and [10] are the fruits of extensive research done to solve the SLAM problem over the last 10 years. For instance, VIN (Visual-Inertial Navigation) is regarded as a reduced SLAM algorithm where loop closure is not implemented. Its development is mainly thanks to the fact that more efforts towards developing sensor fusion algorithms with low-quality sensors are put together in contrast to what has been formerly considered in the literature.

A second notable benefit of SLAM algorithms is that they are able to construct the correct topology of the environment with the help of loop closure. Performing

odometry alone builds an "infinite corridor" world model where place recognition does not occur [3]. Figure 1.2 shows two separate maps of the same environment constructed with and without the loop closure feature. Apart from interpreting the topology inexactly and building a faulty map, the absence of loop closure also renders the implementation of efficient navigation techniques almost impossible. For instance, when revisited places are not recognised as in Figure 1.2-left, locations B and C on the map will be thought of as being far away, whereas they are actually next to each other as in Figure 1.2-right.

One counter argument to it could be that place recognition alone is also able to interpret the right topology of the environment. Nonetheless, SLAM's metric information of the environment brings robustness and efficiency to the problem by preventing false data association (perceptual aliasing) where two identical looking places are observed at separate locations. In this regard, SLAM is superior to place recognition with its ability to predict and validate its measurements [3].

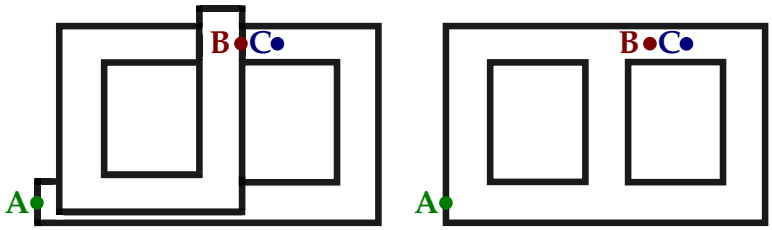


Figure 1.2: *Left:* Odometry based map; *Right:* Map built with SLAM

Is SLAM a solved problem?

Having already discussed the uniqueness and neediness of SLAM technology, a natural question that comes to our mind is *Is SLAM a solved problem?*. The answer is two-fold: yes and no, because the question as it is posed at the moment is too broad and needs further details, regarding environment, robot, performance and so forth, to be answered [11]. For instance, if the question is asked for 2D mapping of an indoor environment via a robot with a laser scanner and wheel encoders on-board, the answer is definitely a yes. Kuka Navigation Solution [12] is one of many solutions to the SLAM problem in the given context with sufficient efficiency and robustness, enabling it to autonomously perform in various industrial environments. Likewise, vision-based SLAM for robots with slow dynamics, like Mars rovers [13], and visual-inertial odometry [14], are well-studied research topics.

On the other side, the present SLAM algorithms fail or perform worse when exposed to fast robot dynamics or executed in very dynamic environments. Therefore, the research on SLAM problem still continues today and the era we are in now is referred to as *robust perception age* in [3]. It challenges researchers all around the world to better address the following four aspects of the SLAM problem [3]: robust performance, high level understanding, resource awareness and task-driven perception.

In essence, there are still open challenges with regards to the SLAM problem that comprise multifarious aspects at the intersection of numerous research fields. At the front-end level, which includes feature extraction and data association, signal processing and computer vision techniques are intensively involved. The back-end level where map estimation occurs is a mix of geometry, optimization and probabilistic state estimation methods. A review of state-of-the-art techniques will be given in Chapter 2.

Why is SLAM a hard problem?

Having already obtained a historical background on SLAM and its research, now we would like to continue our discussion with a focus more on the problem itself. In this section, we elucidate on what makes SLAM a hard problem.

The difficulty of the problem stems from the fact that the estimates of the path and the map posteriors are both unknown and correlated [15]. Figure 1.3 is a fine example to explain this chicken-and-egg nature of the problem and its complications. When the robot first starts its operation, it assumes to be located at the origin of the global frame with zero heading angle. Since it has not moved yet, it is completely certain as to where it is. Thus, no uncertainty ellipse has been drawn around it. However, this is not the case for landmark locations because the sensor measurements come with a certain level of noise and this leads to uncertainty about their true locations. Then, the robot moves to another location in the map. Even though its movement is measured or the given motion command is known, it is hard to pinpoint where it exactly is due to imperfect execution of the motion. For this reason, the robot's pose now has uncertainty. As the motion has increased the uncertainty of the system, currently measured landmarks will have a larger uncertainty than the previously measured ones despite the fact that the sensor noise is left unchanged. Next time the robot moves forward, it re-observes a landmark and obtains a better estimate of its location thanks to additional information from re-observed landmark. Consequently, the uncertainty ellipse of landmark $L4$ shrinks. Now that the robot is more certain about the location of landmark four, it also gets more certain about its previous poses since the map and the pose estimates are correlated. This leads to better location and map estimates. Given the dependencies between the map and the poses, the mapping and localization problem cannot be solved separately. Ergo, the solution (and the difficulty of it) is to simultaneously tackle both problems.

The second reason why SLAM is a hard problem is the unknown data association. It involves identification of sensed features to formerly observed landmarks in the map. In the example of Figure 1.3, the position correction of $L4$ depended not only on re-observing it, but also on making the correct data association. If there were more landmarks in the neighbouring region of landmark four, a wrong data association could have been done and this would have led to divergence in the estimation process. Furthermore, the uncertainty of the pose estimate may also affect data association adversely [15]. As illustrated in Figure 1.4, given that the robot observes

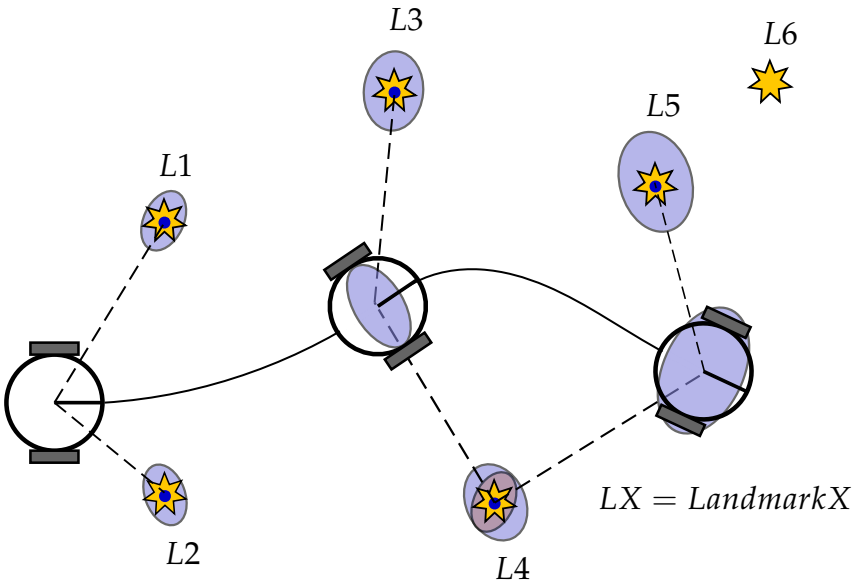


Figure 1.3: Visual illustration of the SLAM problem

two landmarks and it is equally likely to be at positions 1 and 2, correct data association becomes even harder. In continuation of our discussion on SLAM, a short introduction to the main three SLAM paradigms is given in the next section.

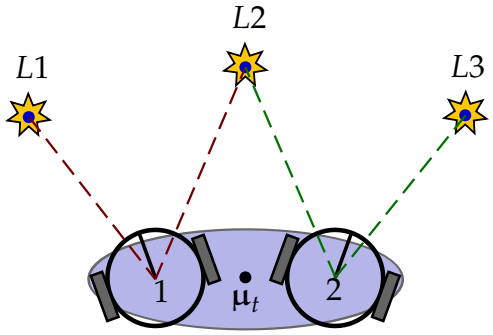


Figure 1.4: Correct data association difficulty due to large pose estimate uncertainty

SLAM Paradigms

There are fundamentally three SLAM paradigms: Extended Kalman Filters (EKF), Graph-based optimization techniques and Particle methods, from which other formulations have derived [6]. EKF SLAM estimates the robot and landmark locations through a single state vector and the corresponding covariance matrix. It builds a metrical and feature-based model of the environment. In Graph-based SLAM, the landmarks and the robot position are thought of as nodes in a graph. Successive location pairs are connected together with an arc according to the odometry data. Other ties among locations and landmarks are also drawn when they are observed. The connections in Graph-based SLAM represent soft constraints. The map and the robot's trajectory are best estimated by relaxing these constraints. Graphical SLAM

techniques are better than EKF in the sense that they can model larger environments since they are not limited by the covariance matrix whose memory and update time expands quadratically with the number of landmarks. In Particle filters, posteriors are represented by sampled particles from the distribution. It is a recently popularised non-parametric representation that can easily capture multimodal distributions. However, it suffers from scaling exponentially with the state space of the robot and the map. Having briefly touched upon the main three SLAM paradigms, now we would like to inform the reader about various SLAM classifications and explain their characteristics [16].

Rich taxonomy of the SLAM problem

In the literature, there exist two distinguishable forms of the problem that are equally important: full and online SLAM. The most notable difference between the two is that the full SLAM computes the posterior of the complete robot path, whereas online SLAM only involves current pose estimation. Recovering a map of the environment is a common target for both. Full SLAM algorithms are classified as batch, referring to processing of all data at the same time. This points out to gathering of sensor information in advance, and then constructing the map and estimating the path of the robot in it. On the other hand, online SLAM algorithms are generally incremental (called filters) and deal with the most recent data iteratively at fixed intervals [16].

Solving either problem requires two mathematical models: a motion model and a measurement model for state and map estimation respectively. Even though deterministic approaches are available, these models are usually probabilistic. In fact, nearly all state-of-the-art SLAM algorithms implement probabilistic motion and sensor models due to indisputable noisy nature of executed motion commands and sensor measurements. Acknowledging the presence and effect of noise and incorporating it into the SLAM algorithms has proven its efficiency and is now being considered as the sole approach to the problem [17].

It is also possible to categorize the SLAM problem across various dimensions besides the already mentioned full versus online distinction. Volumetric SLAM enables a photo-realistic high-resolution representation of the environment, in contrast to featured-based SLAM, where only certain features are mapped. One needs to consider available processing power on-board while deciding between the two, since volumetric SLAM involves more computations due to the increased dimensionality of the map. For this reason, featured-based SLAM algorithms are usually more efficient. However, their efficiency also comes with a drawback, that is the loss of information from the sensor measurements [15].

Another assumption that differentiates SLAM algorithms is the static nature of the environment, which is the default supposition of the vast majority of the literature. While dynamic methods permit changes in the environment, static techniques considers such changes as measurement outliers. In addition, the SLAM literature has mainly been focused on single robot applications. SLAM techniques defined for a

swarm of robots are also gaining more popularity among the researchers [18].

Lastly, in passive SLAM, which constitutes the wide majority of all algorithms, the robot is navigated arbitrarily by another entity and the SLAM algorithm is solely responsible for state and map estimation. Its disadvantage is that the random motion control of the robot usually leads to a longer operation time needed for accurate localization and mapping. On the contrary, active SLAM algorithms yield results faster as they execute purposeful motion commands that help the robot explore the environment in a shorter span of time [19]. Hybrid techniques that control the pointing direction of on-board sensors and thus address the robot motion constraint of active SLAM algorithms also exit [3].

Final Remarks and Our Motivation

With the tremendous advancement it has seen over the course of a couple of decades, the SLAM problem has been better understood and addressed, meanwhile giving birth to many other technologies such as visual-inertial odometry. Even though some SLAM problems with certain environment-robot-performance configurations have been mostly solved, there are still many other challenges that the researchers are now working on to enhance the robustness, efficiency and autonomy of the current systems.

SLAM is a fundamental topic in the field of robotics, as it renders robot operation possible in the absence of a localization infrastructure. The demand for such a technology is indeed high, for it is at the core of a large range of indoor (augmented, virtual reality and vacuum cleaners), outdoor (self-driving cars, lawn mower), aerial (surveillance, inspection), underwater (reef monitoring), underground (mine explorations) and space (terrain mapping for localization) applications [15]. One of the first instances of SLAM applications that have gained great momentum to the ongoing research in the field has been Stanley, the autonomous self-driving car (see Figure 1.5) developed by Sebastian Thrun and the Stanford Racing Team for the 2005 DARPA Grand challenge. The aforementioned SLAM applications are recent and foreshadow integration of more advanced robotic applications into our everyday lives. In the future, we are motivated to be a part of their realization. Thus, we chose to study the SLAM problem for our Bachelor's thesis and learn as much theory as possible and develop practical skills needed for complete system integration, establishing the foundations we need to contribute to the field in our future careers.

Over the years, researchers have developed numerous specific solutions to the SLAM problem, but, as of now, there is not a single SLAM algorithm that works well in all environmental conditions. It might be that a new perspective on the problem is needed to address this wide range, or the little research improvements being accumulated over the time will take us there. Regardless of how it happens, it is still true that learning the classical SLAM techniques is relevant and necessary in our journey towards learning more advanced methods, just like it is essential and important to know the Newton's law of universal gravitation to understand Albert Einstein's gen-

eral theory of relativity. Therefore, we feel strongly motivated to take our first steps in this thesis. We believe that the effort and hard work we put in it will pave the way for us towards more advanced SLAM algorithms and even maybe beyond those, as we move up our career ladders.

Figure 1.5: Self-driving car Stanley is competing in 2005 DARPA Grand challenge

Thesis Organization

This report pursues the following structure. In Chapter 1, an extensive introduction to SLAM problem is given and our motivation to work on this topic is expressed. Chapter 2 provides a review of the state-of-the-art SLAM techniques and the use of recent sensor technologies and learning methods. Chapter 3 gives a system overview and lists the components used in this thesis. Chapter 4 elucidates on the famous Kalman filter state estimation method in mathematical terms and explains the implemented extended version of it. Chapter 5 discusses the modeling of the robot motion and the sensor measurements, as well as the extraction of features from raw sensor data. Chapters 6 and 7 give the EKF (Extended Kalman Filter) localization and SLAM algorithms in detail together with the corresponding simulation results. Chapter 8 presents the ROS implementation in relation to simulations with Gazebo. Finally, Chapters 9 and 10 conclude the thesis with a brief discussion of supplemental topics, followed by the conclusion.

Chapter 2

State of the Art

Numerous types of sensors including but not limited to encoder, laser range scanner (also referred to as LiDAR), inertial measurement unit, GPS, RADAR and camera have been leveraged for SLAM applications over the last 35 years of its history. LiDAR was the main sensor used in the early days of SLAM research [20]. However, nowadays the trend is either to use a camera or to fuse it with a depth sensor because of its ease of configuration, low price, large measurement range and its ability to gather substantial amount of information [21, 22, 23]. Nevertheless, SLAM performed with laser scanners is still a popular choice due to its simplicity and accuracy [22]. In addition, point clouds delivered by laser scanning are easily interpreted to execute SLAM algorithms. Considering this, a literature review regarding various aspects of trending SLAM algorithms with a focus on laser range finders and cameras has been conducted with the aim of gaining insight into the most recent techniques used widely. The section is divided into two subcategories: LiDAR-based and visual SLAM, where corresponding state of the art techniques involved in solving the SLAM problem are mentioned.

2.1 LiDAR-based SLAM

LiDAR-based algorithms work depending on the point clouds obtained in the environment, see Figure 2.1 for a map built with a laser range scanner. In spite of the fact that LiDAR-based SLAM applications are broad, the methods used for it have remained the same for the last ten years [22]. Two most prevalent approaches are particle and optimization-based graph techniques [20].

Particle Filters

The benefit of particle filter approaches such as the ones in [24, 25] is that they can deliver accurate positioning and mapping even for strongly nonlinear systems, unlike EKF. On the other hand, a large number of particles correspond to an increased amount of calculations to be performed. Gmapping (refer to [24]) based upon the

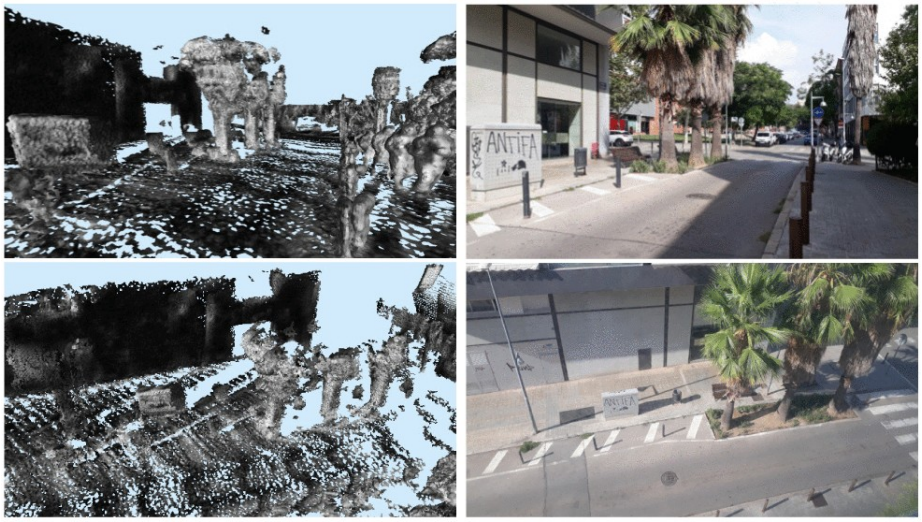


Figure 2.1: *Left:* Mesh reconstructed from LiDAR; *Right:* Photo taken some days after the capture (Copyright © 2018, IEEE)

Rao-Blackwellized particle filter (RBPF) applies adaptive re-sampling and lowers the computational complexity. Work presented in [26] achieves high accuracy 2D SLAM with particle filter using one order of magnitude less particles than conventional approaches. A more famous work of particle filter based SLAM is FastSLAM, where the robot's trajectory and the data associations are represented as samples for the particle filter [27, 28].

Optimization-based Techniques

These days, optimization-based approaches serve as effective alternatives to probabilistic methods. Different optimization techniques such as the Levenberg–Marquardt optimizer are applied to pose graphs where nodes and edges respectively represent sensor measurements and constraints from the observations. Karto-SLAM [25], Hector SLAM [29] and Google's Cartographer [30] are prevalent examples of such techniques [20].

Motion Modeling

State of the art SLAM algorithms prefer utilising simple probabilistic kinematics to model the motion of a robot. In spite of the fact that more advanced kinematic models also exit, it is decided not to include them in this review, considering the higher success and prevalence of the aforementioned simpler ones [31].

The two most common motion models are velocity and odometry models [31, 32]. The former takes commanded translational and rotational velocities into consideration to calculate the pose of the robot. On the other hand, odometry models make use of measured wheel revolutions. Even though both types of models process the same sort of information, the results of odometry model are more accurate given that robots

cannot execute the given velocity commands with the same accuracy that encoders can measure wheel revolution. Though, one drawback of odometry models is that encoder information is available only after executing the command which hinders implementation of motion planning and restricts their use mostly to state estimation [7]. Nevertheless, most wheeled robots make use of odometry model. Velocity models are typically used for flying vehicles as they are not usually equipped with encoders. They are also common for humanoids or legged robots, even though they usually have encoders in their joints because it is hard to estimate how big their step size is [31].

Measurement Modeling

One efficient and simple measurement model, thus widely used, is the so-called beam-endpoint model. It ignores the map information along the measurement line and only takes the end point of the beam into consideration. It calculates the distance to the nearest obstacle in the map. If there is no obstacle in close proximity, the measurement will have low a likelihood. Beam-endpoint model lacks a plausible physical explanation, since there might actually be an object closer than the measured distance. On the other side, it performs better in cluttered environments and it is computationally more efficient [33, 32].

An expensive to compute but still frequently used measurement model is called the ray-cast model. Compared to the beam-endpoint model, it has a plausible physical explanation but its disadvantage is its computational complexity. It suffers from lack of smoothness in cluttered environments, meaning that neighbouring states might have very different likelihoods [31].

There are also feature-based measurement models that reduce the high-dimensional measurement space to a low-dimensional feature space and decrease computational complexity. This type of measurement models require preprocessing of the raw sensor data with appropriate algorithms for extracting the features. The likelihood of measuring a landmark at a location is then calculated with basic geometry [33].

2.2 Visual SLAM

Visual SLAM (vSLAM) algorithms are classified as feature-based, direct and RGB-D camera-based approaches [21]. As compared to feature-based systems, direct methods use complete images without extracting any features from them. They register [1] successive images using photometric consistency instead of geometric positions of the detected features. This enables direct methods to cope with featureless or textureless environments.

[1] Image registration is the process of transforming multiple images into one coordinate system.

Feature-based approaches

The most recent feature-based algorithm is ORB-SLAM [34, 35]. It can be implemented using monocular, stereo and RGB-D cameras. Its biggest disadvantage is the large number of parameters to be tuned to make it functional [22]. Works such as [36] have been carried out to eliminate the need of parameter tuning in order to make it robot and environment independent. However, the performance of ORB-SLAM is still not reached by these approaches. Figure 2.2 illustrates an example of the feature extraction process and the built map for a feature-based vSLAM algorithm.

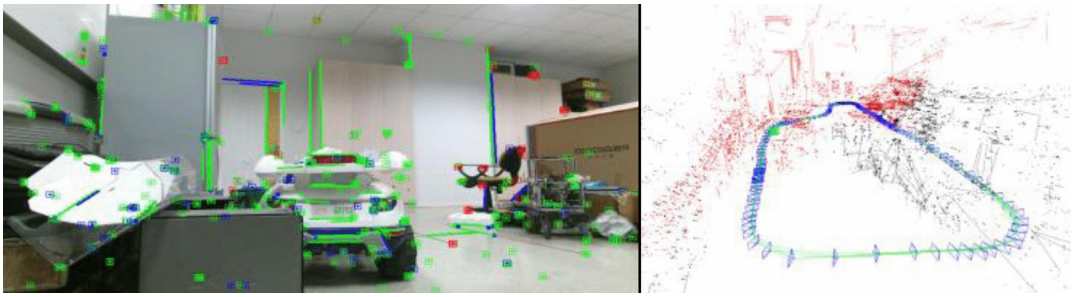


Figure 2.2: *Left:* The extracted points and lines are displayed on each frame; *Right:* The map is built by SLAM which shows the trajectory with extracted points and lines. (Copyright © 2018, IEEE)

Direct Methods

State of the art direct methods are DTAM (Dense Tracking and Mapping) [37], LSD-SLAM (Large-scale direct monocular SLAM) [38], SVO (Semi-direct visual odometry) [39] and DSO (Direct sparse odomery) [40, 41]. Drawback of direct methods is that they often require a GPU for real-time processing [22] due to high dimensionality of the measurements. Figure 2.3 demonstrates a 3D map built using the LSD-SLAM algorithm.

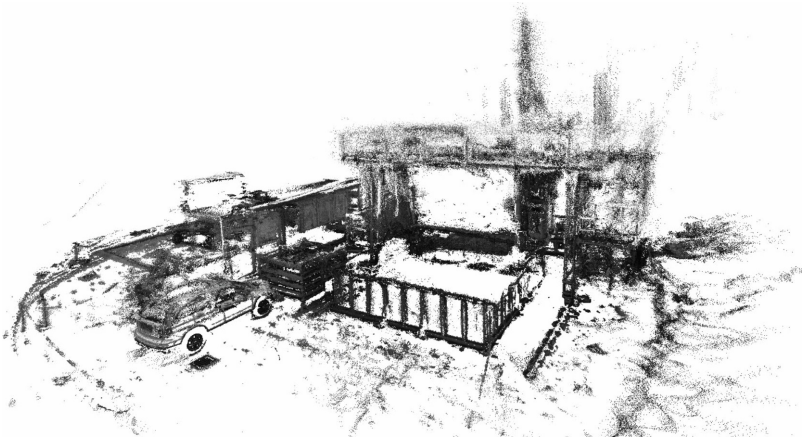


Figure 2.3: 3D reconstruction obtained in real time by LSD SLAM with omnidirectional cameras (Copyright © 2015, IEEE)

RGB-D camera-based methods

RGB-D cameras provide both image and distance information, thus giving the 3D structure of the environment directly. This leads to easier determination of scale of the coordinate system in comparison to monocular vSLAM. Most popular RGB-D camera-based SLAM approaches are Kinect Fusion [42] and SLAM++ [43, 44]. They are mostly used for indoors applications since RGB-D cameras have an approximate range of five meters and they are sensitive to sunlight. Iterative closest point (ICP) algorithm is widely used to estimate the camera motion for these methods [21]. In addition, pose-graph optimization (counterpart of loop closure) and deformation graph optimization are implemented to refine camera motion and to obtain a more geometrically consistent model of the environment respectively [21]. A 3D model built with RGB-D camera is given in Figure 2.4.

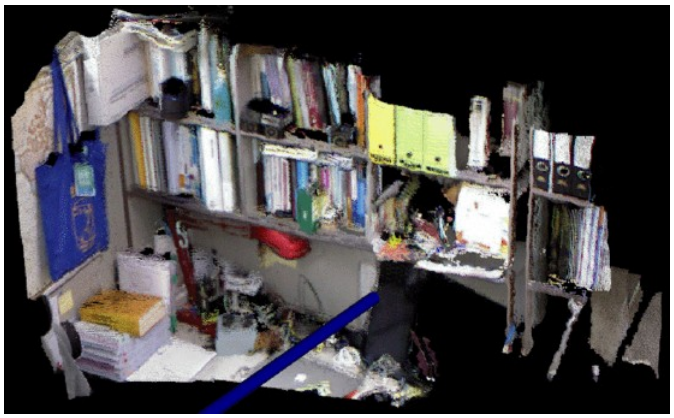


Figure 2.4: Optimized RGB-D 3D model (Copyright © 2012, IEEE)

Camera Tracking and Scene Mapping

Visual SLAM algorithms estimate 6DoF (degrees of freedom) motion of the used camera by matching image features under different poses. Most common approaches are photometric and geometric alignments. The former estimates the motion by minimizing pixel intensity difference of two frames based on the assumption that the same points will have the same color in different images [23]. The latter, on the other side, tries to estimate the camera motion by minimizing Euclidean distances between corresponding points in 2D or 3D. Data from encoders and inertial measurement units (IMUs) can also fused to provide additional information for motion estimation and to enhance the robustness of the SLAM systems.

Two most common scene representations are point-based and volumetric maps. Photometric and geometric bundle adjustment (BA) techniques are widely used in construction of point-based maps. They apply a nonlinear optimization of camera poses and points.

Feature Detection

Feature detection is an essential element of feature-based vSLAM approaches. Prevalent feature extractors are FAST [45], SURF [46], BRIEF [47] and HARRIS [48] detectors. Later, the features are tracked to produce optical flow from which motion is estimated. A frequently used feature point tracker is the KLT tracker [49]. Some instances of feature detecting and tracking are given in [50, 51]. Figure 2.5 is an instance of SURF feature points matching.

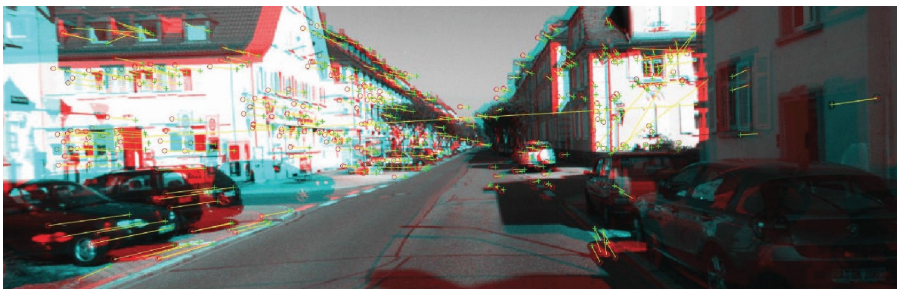


Figure 2.5: An example of SURF feature points matching (Copyright © 2019, IEEE)

2.3 Recent Trends in SLAM

Sensors

Novel sensor technology has always been one of the catalysers of advanced robotics applications. For example, with the development of 3D laser range finders, autonomous car technology has been enabled. Research on vision sensors has resulted in applications such as augmented reality and vision-based navigation. As for SLAM, light-field and event-based cameras are now in trend. Ergo, a short research on the characteristics of these sensors has been carried out with a focus on their impact on SLAM applications [3].

Light-field cameras or plenoptic cameras record not only the intensity of light beams but also their direction. This technology enables depth estimation, video stabilization and removal of specular reflectivity. It formulates motion estimation as a linear optimization problem and gives more accurate results. However, it comes with increased computational requirements due to the extra information obtained [52].

Even-based cameras send local pixel-level changes at the time they occur as opposed to conventional cameras that send full images at fixed intervals. Event-based cameras have faster update rate, lower power consumption and storage requirements than frame-based cameras. These characteristics ease the development of new SLAM algorithms for faster motion and higher dynamic range. Some examples are [53, 54].

SLAM algorithms also benefit from measuring thermal, sound, pressure and magnetic stimuli in the environment. Two pieces of research on the use of these sensors are [55, 56]. The former proposes using local anomalies of the ambient magnetic field and the latter elucidates on navigation using already-installed wireless networks.

Deep Learning

Deep learning applications in SLAM are showing promise and are becoming more popular [57]. Below are some recent instances of how deep learning can be used for enhancing SLAM systems.

The work presented in [24] trains a deep neural network to obtain inter-frame pose between successive images instead of using the conventional geometry of visual odometry. Furthermore, regression forest and deep convolutional neural networks (CNNs) are used to obtain 6DoF of a camera (see [58, 59]) and to estimate the map from a single image (as in [60, 61, 62]). In order to reduce the computational complexity of direct vSLAM algorithms, numerous attempts of applying deep learning for visual odometry have been carried out. Vikram's and his colleagues' work [63] presents a deep learning approach for monocular visual odometry. In addition, Wang et al. worked on end-to-end visual odometry with deep recurrent convolutional neural networks in [64]. A relative camera pose estimation using CNNs has been proposed in [65] as well.

Apart from these supervised methods, unsupervised methods are also applied. SfM-Net presented by Vijayanarasimhan et al. in [66] is a geometry aware neural network for structure and motion estimation from video data. Besides, learning depth and egomotion from monocular camera is focused on in [67]. Other proposed unsupervised learning frameworks in the literature that we found interesting are [68, 69].

A recent trend in computer vision is to use CNNs and deep learning in order to tackle the loop closure problem. Works such as [69, 70] illustrate the use of pretrained CNNs as feature generators to create whole image representations. In these papers, it is concluded that features produced using CNNs are more robust to viewpoint, illumination and scale variations of the environment. CNN features can also be preprocessed and projected into a lower-dimensional space so that the loop detection process is more efficient [71].

Chapter 3

System Description

Our investigation of the state of the art SLAM techniques has provided us the knowledge to outline our own solution to the problem of localization and mapping in indoor environments. The scope of it the thesis has been selected in a way to get us familiar with the most fundamental and well-documented techniques in the realm of SLAM. Before presenting the specifications of the intended project, it is thought to be beneficial to provide a high-level description to the reader as shown in Figure 3.1. The idea of this thesis, in its most basic form, is to give a mobile robot the ability to map its environment and localize itself, while enabling the visualization of the obtained map and the knowledge of location on a separate remote computer which is also used to move the robot arbitrarily in the environment.

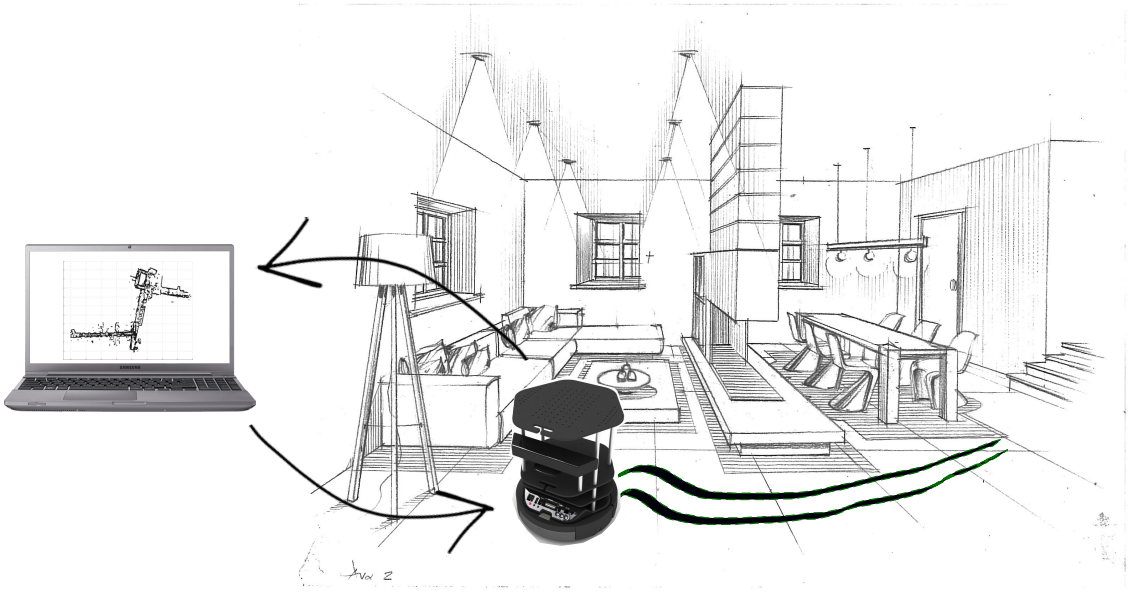


Figure 3.1: High-level system description

As stated in Chapter 2 as well, visual SLAM is nowadays the most ubiquitous approach to solve the SLAM problem. As for LiDAR-based techniques, they are slowly losing their popularity. Still, they are occasionally implemented, or at least LiDAR

scan information is combined with visual sensors to enhance the performance of the system. In light of this information, even though the literature review strongly points towards visual SLAM methods, in this thesis it is decided to continue with LiDAR-based SLAM after taking the following into consideration. This project represents the authors' first contact to the world of SLAM algorithms and acquiring a fundamental knowledge of the classic SLAM solutions is believed to be more appropriate than studying one of the avant-garde techniques.

A block diagram illustrating the components of the system and their corresponding functionalities is shown in Figure 3.2. A Raspberry Pi 4 has been found convenient to employ as a processing unit both for its computational capabilities and compatibility with the Kobuki TurtleBot mobile robot base available in the repository of the Aalborg University Esbjerg. The local point cloud of the environment obtained via the RPLiDAR A3 sensor will be shared with the Raspberry Pi 4 for information processing purposes. This is also where the SLAM algorithm will be executed iteratively. Furthermore, the Raspberry Pi 4 will play an important role in acting as a gateway for the user to send control inputs to the Kobuki TurtleBot for its movement, as well as for the TurtleBot to send back odometry data for performance evaluation purpose.

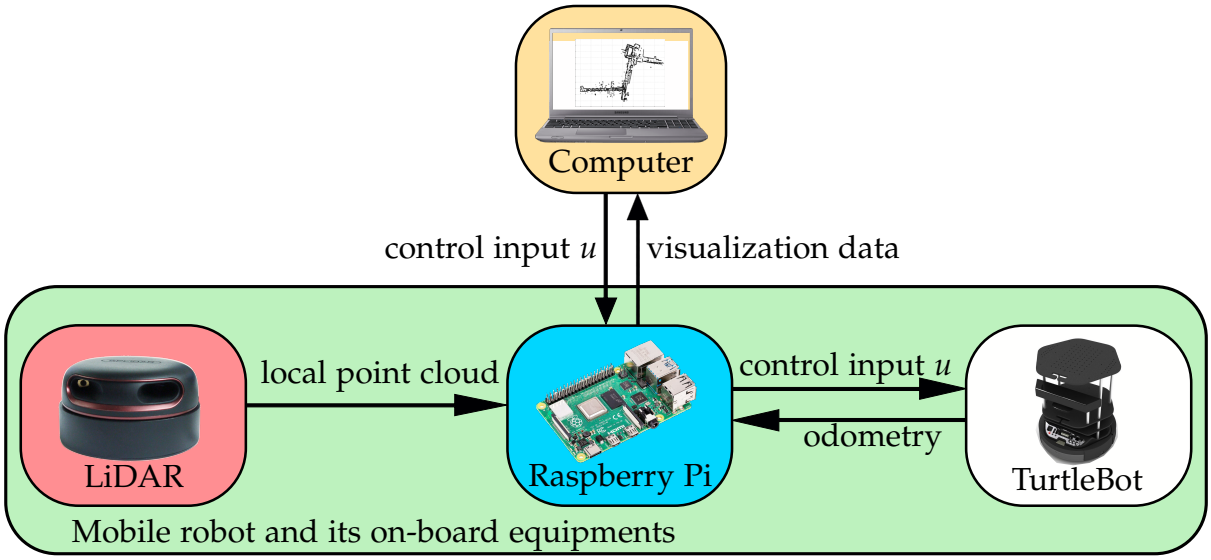


Figure 3.2: Simplified block diagram of the overall system

Regarding the state estimation aspect of the project, an Extented Kalman Filter will be implemented for estimating the important states of the system and the environment. The 2D map recurrently built as the robot navigates will be based on the features extracted from the LiDAR scans. Among numerous feature extraction algorithm options to pick from, it is decided to detect line features due to the natural excessive existence of such beacons[1] at indoor environments. In order to ease the information flow among all the mentioned elements of the system, the standard robotic software writing framework, namely the Robotic Operating System (ROS), will be

[1] "Beacon" is just a term that will be interchangeably used with "landmark"

used. Last but not least, the chosen programming language is Python 3 due to its prevalence and dominance in the field of robotics.

The block diagram shown in Figure 3.2 is in consideration to the circumstances when the SLAM algorithm is ready to be deployed on the Kobuki TurtleBot. That being said, the final implementation in a real environment will be approached in a series of developments leading towards it. These developments mostly relate to simulations using different platforms. First, as the algorithm code is being developed using Python, our attempts would be to test, debug and optimize this code to the utmost level, so as to avoid problems in the later steps. Ergo, prior to testing the algorithm using a professional simulator (e.g. Gazebo), the written code would be first tested against the scenarios created by us, using Python[2]. We will, by ourselves, artificially generate the sensor readings, the indoor structure and the noise elements regarding perception of surroundings and motion of robot. Once satisfactory results are achieved, we proceed to testing the algorithm with ROS and Gazebo. The second step, i.e. the implementation with ROS and Gazebo, will be driven towards a preparatory task of simulating and creating a medium which will see its usage in practical implementation. Through this step, the motive will be to emulate the algorithmic implementation as close to the real life situation as possible. Naturally, this stage would see the utilization of more complex simulators, as compared to the Python simulation step, corroborated by the ROS implementation strategy. Notwithstanding, if the implementation with ROS and Gazebo yields a competent performance, the plausibility of the developed algorithm working on the Kobuki TurtleBot is reinforced. The final step, obviously, would be the testing in an real indoor area, with the set-up shown in Figure 3.2. This piecemeal way of development was adopted to isolate the problems within steps, allowing for a comparatively easier identification of errors.

[2]This step is later referred to as *"Python Simulations"* in upcoming chapters

Chapter 4

State Estimation

The crux of many robotics problems is state estimation from sensor readings. There would be no localization or SLAM algorithms if the location of robot and its surrounding objects could be known directly form sensor measurements. Sensors provide partial information that must be inferred to get the states of our systems. In addition, perceptual data is laced with noise. For these reasons, state estimators are a common sight in problems involving robotics and are utilized to extract valuable state information from the available unrefined data. This chapter thereupon introduces the state estimating algorithms crucial to our approach on Simultaneous Localization and Mapping.

4.1 The Kalman Filter

The Kalman filter prevails as one of the best studied methods for state estimation in concern to a discrete-time controlled process. The significance of the Kalman filter Equations will later become apparent as the the backbone of other relatively complex algorithms for localization and SLAM.

Before proceeding into the details of this filter it is crucial to mention its underlying assumptions and conditions. As it is a Bayesian filter, it is subject to the Markov assumption (also known as *complete state assumption*). According to which, the past and the future data is independent if the current state $\mathbf{x}_t$ is known [7]. The representation of beliefs in Kalman filter is done through the mean and covariance and such representation is called the moment parameterization.

A primary characteristic of Kalman filter is its confinement to problems entailing linear Gaussian systems. Therefore, achieving a Gaussian Posterior is critical. To ensure this, the state transition probability $p(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x}_{t-1})$ and the measurement probability $p(\mathbf{z}_t \mid \mathbf{x}_t)$ should have linear functions in their arguments along with their added respective Gaussian noise. The linear motion model and the observation model are shown in Equations 4.1 and 4.2 respectively. Lastly, it should also adhere to condition that the initial belief must be a normal distribution [7].

$$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\varepsilon}_t \tag{4.1}$$

$$\mathbf{z}_t = \mathbf{C}_t\mathbf{x}_t + \boldsymbol{\delta}_t \tag{4.2}$$

In the Equations above, $\mathbf{x}_t$ is a state vector at time step t and $\mathbf{u}_t$ symbolizes the control vector. For the sake of definitions, it is assumed here that the size of the state vector is n (signifying n states) while the control vector is of dimension m. Similarly, $\mathbf{z}_t$ expresses the measurement, thus dubbed as the measurement vector with length k.

Furthermore, considering the state transition Equation 4.1, $\mathbf{A}_t$ is a $n \times n$ matrix which serves the functionality of relating the state from the previous time step to the state at the current step such that there is neither a driving function nor any process noise present. The $\mathbf{B}_t$ matrix on the other hand is of dimension $n \times m$ and is included with the purpose of relating the control input $\mathbf{u}_t$ to the state $\mathbf{x}_t$. The last element in the state transition equation is the added Gaussian random vector $\boldsymbol{\varepsilon}_t$. Labelled as the *process noise* in literature [72], it models the uncertainty in state transition. A defining factor in this noise is its white nature (i.e. zero mean) and as expected, a normal probability distribution. Additionally, its covariance is denoted by $\mathbf{Q}_t$ [72]. From now on, the probability distribution for the process noise will be expressed using the notation in Equation 4.3.

$$\boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, \mathbf{Q}_t) \tag{4.3}$$

Similarly, the measurement Equation 4.2 has a $k \times n$ matrix $\mathbf{C}_t$ for relating state to the measurement. In other words, it specifies the mapping from state space to measurement space. The added vector $\boldsymbol{\delta}_t$ represents the measurement noise with similar properties to the process noise such as zero mean and normal distribution. The covariance in this case is given by the $\mathbf{R}_t$ matrix and thus can be represented using familiar notation depicted in Equation 4.4. It is worth mentioning that both the process noise and the measurement noise are independent of each other [7][72].

$$\boldsymbol{\delta}_t \sim \mathcal{N}(0, \mathbf{R}_t) \tag{4.4}$$

4.1.1 Kalman Filter Algorithm

With the preliminary details handled in the section above, the focus can now be shifted towards the Kalman filter algorithm. The objective here is to demystify the Kalman filter Equations and the whole process that is undertaken for state estimation. The well known Equations shown in 4.5-4.9 represent the Kalman filter. Anyhow, it seems beneficial to first present a general overview of the filter process before descending into the mechanics of its constituent equations. The algorithm can easily be compartmentalized into two parts: Prediction and Update. These two parts work together in a fashion similar to feedback control. The Equations from 4.5-4.6 encompass the prediction section of the algorithm and the Equations 4.7-4.9 represent the measurement update section.

$$\overline{\boldsymbol{\mu}}_t = \mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t \mathbf{u}_t \tag{4.5}$$
$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\top + \mathbf{Q}_t \tag{4.6}$$
$$\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t \mathbf{C}_t^\top (\mathbf{C}_t \overline{\boldsymbol{\Sigma}}_t \mathbf{C}_t^\top + \mathbf{R}_t)^{-1} \tag{4.7}$$
$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t (\mathbf{z}_t - \mathbf{C}_t \overline{\boldsymbol{\mu}}_t) \tag{4.8}$$
$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \overline{\boldsymbol{\Sigma}}_t \tag{4.9}$$

Primarily, the equations that form the prediction section estimate the process states at a certain time and then a feedback is obtained in terms of (noisy) measurements. The prediction equations are actually projecting forward in time, the current state and the error covariance to determine priori estimates for the next time step. In continuation, the update equations take into account the measurement and the priori estimate to procure the posteriori estimate.

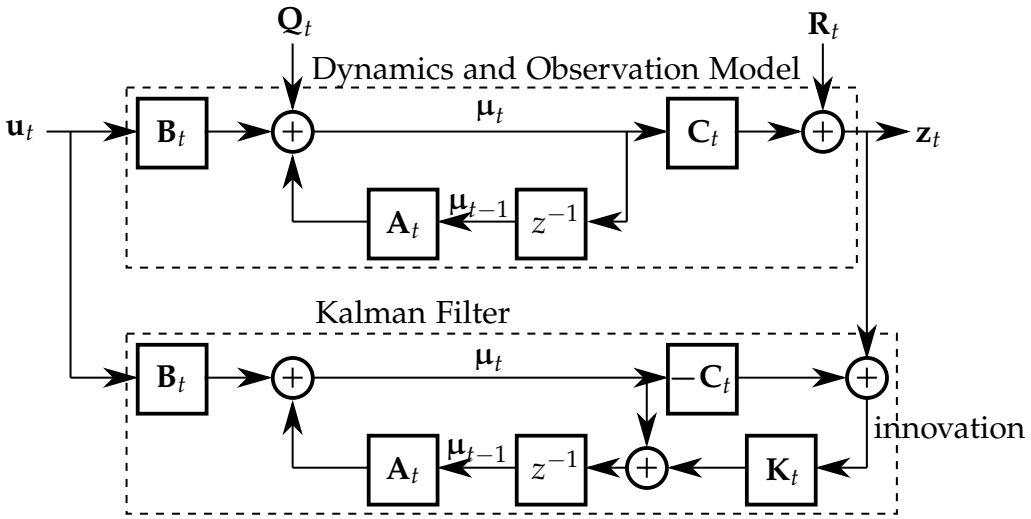


Figure 4.1: Kalman Filter block diagram

Figure 4.1 illustrates the elements of the Kalman filter in a block diagram fashion. As stated earlier, the Kalman filter represents beliefs using the mean and the covariance. Notation wise, the belief $bel(\mathbf{x}_t)$ at time t is represented by mean $\boldsymbol{\mu}_t$ and the covariance $\boldsymbol{\Sigma}_\mathbf{t}$. The input to the filter is bel_{t-1} consisting of $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$. The belief bel_{t-1}, for the first iteration is part of the initialization but in subsequent iterations, it is the posteriori calculated in the last time step.

Prediction Equations

The prediction step calculates the estimated belief $\overline{bel}(\mathbf{x}_t)$ which is represented by the mean $\overline{\boldsymbol{\mu}}_t$ and covariance $\overline{\boldsymbol{\Sigma}}_t$. The estimated mean $\overline{\boldsymbol{\mu}}_t$ is deterministically found using the Equation 4.5 where the matrix multiplication $\mathbf{A}_t \boldsymbol{\mu}_{t-1}$ computes the state for the next time step which is then added to the contribution of the control input to the state.

The second Equation 4.6 determines the estimated error covariance $\overline{\mathbf{\Sigma}}_t$. Basically, this equation symbolizes that the new uncertainty is equal to the old uncertainty ($\mathbf{\Sigma}_{t-1}$) plus the uncertainty $\mathbf{Q}_t$ due to the motion or process.

Update Equations

The update step most briefly can be described as the product of two Gaussians: the prediction and observation (i.e. measurement). The result of this multiplication of Gaussians is also a Gaussian with its mean equal to the weighted mean of the multiplied Gaussians. The first Equation 4.7 in this part is for the calculation of the term $\mathbf{K}_t$, known as the Kalman gain. It is determined such that it minimizes the posteriori error covariance. Accordingly, it is later shown mathematically how the Kalman gain leans towards the Gaussian with the lower uncertainty. However, it is helpful to first introduce the rest of the update equations. The Equation 4.8, deals with the objective of providing posteriori mean ($\boldsymbol{\mu}_t$) as a linear combination of priori estimate ($\overline{\boldsymbol{\mu}}_t$) and a weighted difference between the actual measurement ($\mathbf{z}_t$) and the predicted measurement ($\mathbf{C}_t\boldsymbol{\mu}_t$). This difference is key to the algorithm, as it reflects the discrepancy between the above mentioned measurements. It is known as innovation or residual. Lastly, the Equation 4.9 lets us adjust the error covariance $\mathbf{\Sigma}_t$ according to the new information obtained from the measurement [72].

The degree to which the measurement is included in the new state estimate computation is controlled by the Kalman gain. This can easily be shown by considering two extreme cases for measurement. The first case that will be undertaken is of the perfect sensor, meaning there is no error in the sensor data and thus uncertainty is zero. The assessment of the Kalman gain in such a scenario is presented mathematically through limits where the measurement covariance $\mathbf{R}_t$ approaches zero (depicted in Equation 4.10).

$$\lim_{\mathbf{R_t}\to 0} \mathbf{K_t} = \lim_{\mathbf{R_t}\to 0} \frac{\overline{\mathbf{\Sigma}}_t\mathbf{C}_t^\top}{\mathbf{C}_t\overline{\mathbf{\Sigma}}_t\mathbf{C}_t^\top + \mathbf{R}_t} = \frac{\overline{\mathbf{\Sigma}}_t\mathbf{C}_t^\top}{\mathbf{C}_t\overline{\mathbf{\Sigma}}_t\mathbf{C}_t^\top + \mathbf{0}} = \mathbf{C_t^{-1}} \tag{4.10}$$

As it can be seen in Equation 4.11, the resulting Kalman gain when substituted into Equation 4.8 makes the posteriori mean estimate equal to $\mathbf{C}_t^{-1}\overline{\mathbf{z}}_t$.

$$\boldsymbol{\mu}_t = \mathbf{C}_t^{-1}\mathbf{z}_t \tag{4.11}$$

This makes sense, as the inverse of the $\mathbf{C}_t$ matrix is now mapping from measurement space to the state space. Therefore, Equation 4.11 is implicitly stating that the posteriori mean is just equivalent to the measurement and has no influence from the predicted estimate. Logically speaking, when we have a hypothetically perfect sensor, it would be a sound decision to just rely on the sensor data for state estimation.

The other extreme case is the situation in which there is no measurement i.e. the measurement covariance can be considered infinity. This is because with no measurement, the uncertainty becomes infinite. This situation is shown in the Equation 4.12 below.

$$\lim_{\mathbf{R_t}\to\infty} \mathbf{K_t} = \frac{\overline{\mathbf{\Sigma}}_t \mathbf{C}_t^\top}{\mathbf{C}_t \overline{\mathbf{\Sigma}}_t \mathbf{C}_t^\top + \infty} = \mathbf{0} \tag{4.12}$$

The Kalman gain, now zero, results in the posteriori mean estimate to solely rely on the prediction and discard the measurement (illustrated in Equation 4.13).

$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t \tag{4.13}$$

Through these extreme cases it is inherently clear that Kalman gain regulates the influence of the priori estimate and the measurement on the determination of the posteriori estimate.

4.1.2 Kalman Filter Example

To finally drive home the point, this section will look at one iteration of a 1D Kalman filter example. The example in consideration here is of a robot's movement in one-dimensional space and the Kalman filter is used to provide an estimation of the robot's location. The blue line in Figure 4.2 shows the priori estimate (also referred to as prediction) of robot's location in terms of the Gaussian distribution. To remind, the priori estimate is computed using the prediction Equations 4.5-4.6. Now, the red line illustrates another Gaussian. This Gaussian distribution is obtained from the measurements, where its width concerns the uncertainty in measurements and its mean is centered at the peak of this normal distribution. Moreover, it could be seen that the priori estimate Gaussian is wider than the observation distribution. Therefore, when the information from the measurement is integrated in the update step, the mean of posteriori distribution (obtained using Equation 4.8) is positioned in between the previous two means, although it is located more towards the mean of the measurement. Furthermore, the width of the posteriori is less as compared to the previous distributions. From this, it could be clearly seen that the posteriori (dashed line) calculated by the Kalman filter is done in a way that the uncertainty is reduced such that the Gaussian with less uncertainty has more influence on the posteriori mean.

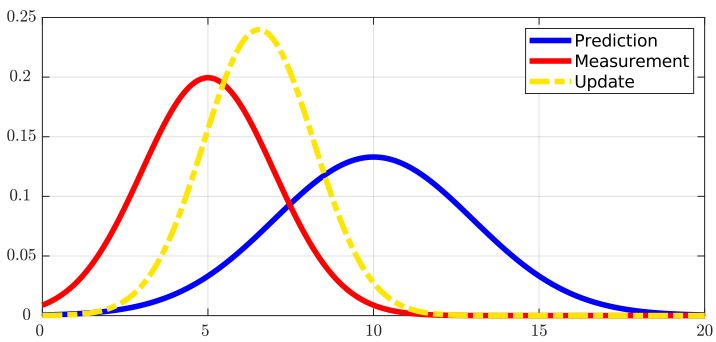


Figure 4.2: Kalman Filter example

4.2 Extended Kalman Filter

It has already been emphasised that for the Kalman filter to be applicable, both the measurement and motion models should be linear functions of the state. This detail forms the basis for the reasoning to explore the Extended Kalman Filter. This idea here is key because the Kalman filter depends on its random variable distributions to be Gaussian and when such a random variable undergoes a transformation through a linear function the result itself is a Gaussian. However, in case of a non-linear function, the shape for the density of the random variable is distorted by the non-linearities and thus resulting in a non-Gaussian distribution. It may be clear that non-linear state transitions and measurements are impediments to the application of Kalman Filter. Unfortunately, it is too much of wishful thinking to hope for linear transformations, as in practice the state transitions and measurements are rarely linear. The commonality of non-linear state transition and measurement function can be illustrated through the case of a robot with constant translational and rotational velocities resulting in a circular trajectory, such motion cannot be described using a linear state transition. Furthermore, it is conventional to include orientation in the state vector for localization problems, this leads to non-linearities in the modelled difference equations in the form of sines and cosines.

With the above discussion in mind, we now consider a more general case for the Kalman Filter by relaxing the linearity assumption. For a winder range of applications, we assume that the process and measurement are governed by non-linear stochastic difference equations, the state vector still is $\mathbf{x}$. The state transition and measurement equations are shown in 4.14 and 4.15, where $\mathbf{g}$ and $\mathbf{h}$ represent the non-linear functions.

$$\mathbf{x}_t = \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}) + \boldsymbol{\varepsilon}_t \tag{4.14}$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \boldsymbol{\delta}_t \tag{4.15}$$

Equations 4.14 and 4.15 are direct replacements to the linear Equations 4.1 and 4.2. If the Equation 4.14 is considered, the function $\mathbf{g}$ is replacing the matrices $\mathbf{A}_t$ and $\mathbf{B}_t$ but with same arguments in question i.e.: $\mathbf{x}_{t-1}$ and $\mathbf{u}_t$. Similarly, the function $\mathbf{h}$ replaces $\mathbf{C}_t$. The non-linear functions perform the same functionalities as their linear counterparts, therefore it is not worthwhile to delve into each of their responsibilities, in order to avoid repetition.

4.2.1 Linearization

An ad-hoc solution to the problem of non-linearity is through linearization. The idea is to approximate the non-linear function by a linear function at a certain point. This is achieved through the first order Taylor expansion. However, a crucial fact to note is the point at which we linearize. The question here is, what is the most likely state at the time of linearization in order to obtain a good approximation? The state

in question would be at the mean $\boldsymbol{\mu}_{t-1}$. The Taylor expansion for state transition function $\mathbf{g}$ is shown in Equation 4.16.

$$\begin{aligned} \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}) &\approx \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) + \frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1})}{\partial \mathbf{x}_{t-1}}(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \\ &\approx \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) + \mathbf{G}_t(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \end{aligned} \tag{4.16}$$

Accordingly, this linear approximation is a result of the function's value and its slope, which is given by the partial derivative w.r.t $\mathbf{x}_{t-1}$. The partial derivative denoted by $\mathbf{G}_t$, due to its multidimensional nature, is actually a Jacobian matrix. So, $\mathbf{G}_t$ is a matrix of size $n \times n$ which differs with time due to the varying nature of $\boldsymbol{\mu}_{t-1}$ and $\mathbf{u}_t$. Moreover, to give an instance of a Jacobian matrix, the Equation 4.17 shows a general Jacobian matrix for a vector valued function.

$$\mathbf{G}_t = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \dots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \dots & \frac{\partial g_n}{\partial x_n} \end{bmatrix} \quad \text{where} \quad \mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_n(\mathbf{x}) \end{bmatrix} \tag{4.17}$$

Similar to the case of state transition function, the measurement function is also linearized, this time at $\overline{\boldsymbol{\mu}}_t$ (the estimated priori mean). The Taylor expansion could be seen in Equation 4.18. Here $\mathbf{H}_t$ is a time varying Jacobian matrix as well.

$$\begin{aligned} \mathbf{h}(\mathbf{x}_t) &\approx \mathbf{h}(\overline{\boldsymbol{\mu}}_t) + \frac{\partial \mathbf{h}(\overline{\boldsymbol{\mu}}_t)}{\partial \mathbf{x}_t}(\mathbf{x}_t - \overline{\boldsymbol{\mu}}) \\ &\approx \mathbf{h}(\overline{\boldsymbol{\mu}}_t) + \mathbf{H}_t(\mathbf{x}_t - \overline{\boldsymbol{\mu}}) \end{aligned} \tag{4.18}$$

With the effects of linearization, the Kalman Filter (KF) algorithm transforms into the EKF algorithm with just minor changes. The EKF algorithm comprises of the Equations shown in 4.19 - 4.23.

$$\overline{\boldsymbol{\mu}}_t = \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) \tag{4.19}$$

$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^\top + \mathbf{Q}_t \tag{4.20}$$

$$\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\top (\mathbf{H}_t \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \tag{4.21}$$

$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t(\mathbf{z}_t - \mathbf{h}(\overline{\boldsymbol{\mu}}_t)) \tag{4.22}$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t)\overline{\boldsymbol{\Sigma}}_t \tag{4.23}$$

On comparison of EKF and KF equations, there are certain differences. The line 1 (or Equation 4.19) of EKF is responsible for estimating the priori mean ($\overline{\boldsymbol{\mu}}_t$) using the non-linear process function (also referred to as the state transition function) around the posteriori mean calculated in the last iteration of the algorithm ($\boldsymbol{\mu}_{t-1}$) along with the control vector ($\mathbf{u}_t$). Moving forward, the objective of the second line in EKF is same

as before but the noticeable changes are in replacement of the linear system matrix $\mathbf{A}_t$ by the Jacobian matrix $\mathbf{G}_t$. The remaining is the update part of the algorithm, here the Jacobian matrix $\mathbf{H}_t$ is utilized instead of $\mathbf{C}_t$ and the non-linear observation function sits in place of the linear measurement system. It should be noted that the value of the non-linear measurement function and the point of linearization is now at the estimated priori mean ($\overline{\mu}_t$) which is determined in the prediction part.

From above, it is obvious that there are minor distinctions between the two state estimaters. However, it would be wrong to define the EKF as an exact replacement for KF but just for non-linearities. Although, like Kalman filter, EKF uses the mean and covairiance to represent the belief, the diverging point is the fact that in case of EKF this belief is merely an approximation as opposed to an exact representation when concerning the KF. Therefore, the goal of the Extended Kalman Filter is different. Its focus is on computing the most efficient estimate of the posteriori mean and covariance instead of their exact closed form solutions. This will present some challenges which are discussed in the next section. Although repetitive, the figure 4.3 succinctly summarizes all the key points that have been the concern of the Kalman Filter sections.

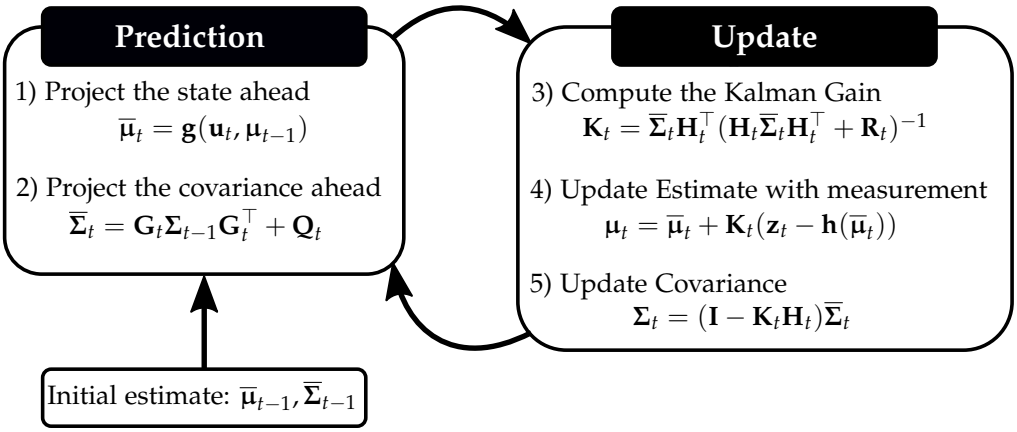


Figure 4.3: Extended Kalman filter algorithm in flow chart form

4.2.2 Drawbacks of EKF

Linearizing the measurement and motion models using Taylor expansions is how these functions are incorporated into the inherently linear Kalman filtering procedure. However, being able to work with nonlinear functions in the filtering process does not necessarily mean that EKF will compute the posterior belief with enough accuracy. It has been mentioned before that EKF only provides an approximation of the true Gaussian. The goodness of the filter approximation will depend on two elements: the local nonlinearities of the non-linear function and the prior uncertainty of the random variable being estimated [7].

The first of the mentioned two factors affecting the quality of the linear Gaussian approximations is depicted in Figure 4.4. The more accurate Monte-Carlo es-

timates of the Gaussian distributions $p(y_1)$ and $p(y_2)$ are shown roughly for comparison purposes. When two random variables x_1 and x_2 with identical variances are passed through separate regions of the same nonlinear function $g(x)$ (due to dissimilar means) which exhibit different levels of nonlinearities, the obtained Gaussian distributions will differ. The random variable x_1 whose mean falls into a more nonlinear local region of the function $g(x)$ possesses a larger approximation error than that of x_2.

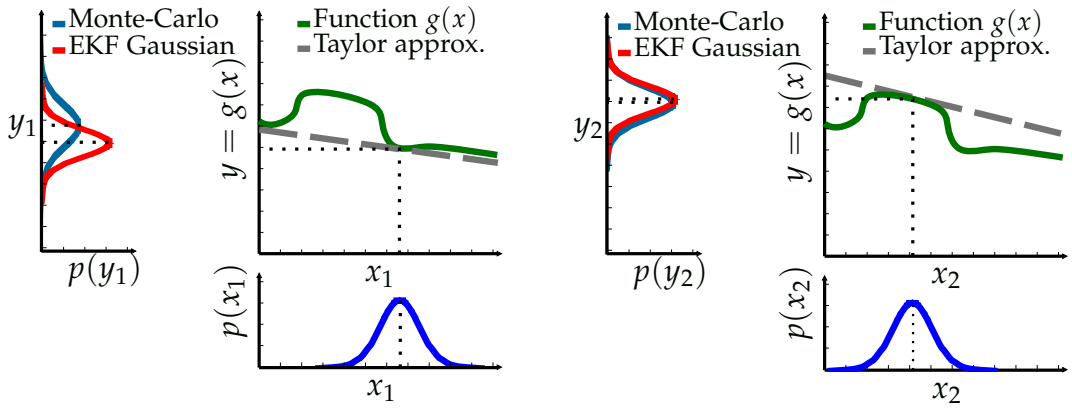


Figure 4.4: Plot illustrating the effect of linearization point in obtaining accurate posterior approximations

Regarding the second dependency of EKF (illustrated in Figure 4.5), the prior uncertainty of the random variables x_3 and x_4 after being passed through the linear Taylor expansion of $g(x)$ will determine the accuracy of the calculated posterior beliefs $p(y_3)$ and $p(y_4)$. The wider Gaussian $p(x_3)$ produces a more distorted Gaussian with less accurate estimates of the mean and covariance than that of x_4.

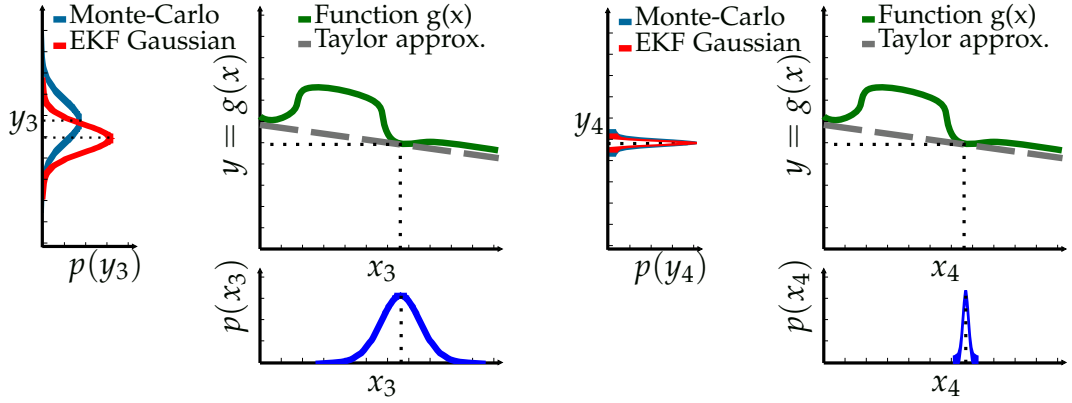


Figure 4.5: Plot illustrating the effect of having a small uncertainty in obtaining accurate posterior approximations

To sum up, if the nonlinear functions exhibit more or less linear behaviours at their points of linearization, then the posterior beliefs produced by the EKF will be accurate approximations. Furthermore, random variables with larger variances will be exposed more to the nonlinearities of the function, even though the latter is quite

linear at the mean of the belief. For these reasons, it is important to maintain the covariances of random variable as small as possible at all times.

4.2.3 EKF Alternatives

Linearizing nonlinear motion and measurement functions by Taylor series expansion in EKF yields good enough approximations as long as the two factors worsening the filtering performance mentioned in the previous section are not dominantly present. In order to enhance the process of linear Gaussian transformation needed in KF, there exist two other methods one can use that have proved to give superior results than Taylor series. The first technique, employed in the Assumed Density Filter (ADF), is called moments matching: the true mean and the true covariance of the posterior are preserved unlike EKF. The second other approach, applied in Unscented Kalman Filter (UKF) is to perform a stochastic linearization using weighted statistical linear regression process. ADF and UKF are better at estimating Gaussians thanks to their unique methods of linearization, and thus can be preferred over EKF [7]. Figure 4.6 demonstrates inaccuracies that occur when a Gaussian is estimated using EKF and UKF by comparing the results of both methods with the true distribution.

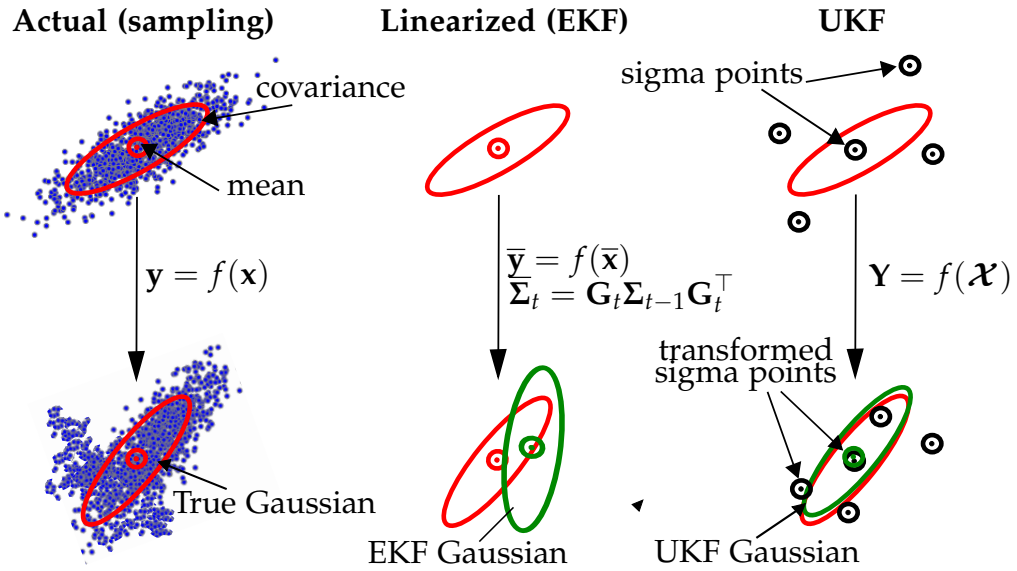


Figure 4.6: Mean and covariance propagation of actual, EKF and UKF transformations

Another alternative to KF, that is subject to the same assumptions, is the Information Filter (IF). It uses a different Gaussian parameterization composed of an information matrix and an information vector instead of a mean and a covariance. This dissimilar parameterization leads to different update equations with varying computational complexities between the techniques. However, both filtering procedures produce the same results in the end and are thus referred to as the dual of each other [7]. Assimakis et al. in [73] conclude that IF is executed faster if $Z > 0.75N$ for time-invariant systems where Z is the measurement vector dimension and N is the state vector dimension.

Chapter 5

Modeling

The contents of this sections will cover the remaining essential subject matter that form the prerequisites for the upcoming algorithms. From the preceding section, the critical role of both motion and measurement models for state estimation should be unequivocally clear. Consequently, this chapter will explore our take on them, along side the reasoning to justify those preferences. It will dig into the intricacies of feature based observation model as well as reconnoitre one of its pivotal components i.e. the split and merge algorithm for line extraction. Needless to say, the forthcoming chapters rely heavily on the models set up here and thus signify the importance of this chapter to the undertaken problem.

5.1 Motion Model

The probability density function $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$ associated with the state transition and required in the prediction step of the Kalman filter can be described using distinct motion models. In the specialized literature, the two prevalent approaches are the velocity model [7, 74, 75] and the odometry model [76, 77, 78], with the first one being more frequently preferred. Odometry, albeit very accurate, has the shortcoming of being available only after motion commands are executed, which renders its use for motion planning impossible. For this reason, and with a desire of following the clearly explained method in [7], the velocity model was selected.

Regardless of the chosen model, the following notation is made. There are two considered reference frames. First and foremost, there is a global reference frame ($\mathcal{G}$), attached to the earth (assumed stationary and flat) and with the origin at the point O, where the robot is located at time $t_0 = 0$. As the vehicle moves, its position and orientation obviously change and it is thus helpful to define an additional coordinate system centered on the robot and moving together with it. We refer to this as the body frame ($\mathcal{B}$), but we may also call it the local frame or the sensor frame. Let us also define a state vector $\mathbf{x}_t = [x_t \quad y_t \quad \theta_t]^\top$ representing the vehicle's pose at time t. Pose comprises the Cartesian global coordinates and the bearing or heading direction, i.e: the angle between the unit vectors $\hat{\mathbf{x}}_\mathcal{G}$ and $\hat{\mathbf{x}}_\mathcal{B}$. For a visualization of the preceding

explanation, refer to the left side of Figure 5.1.

Velocity Motion Model

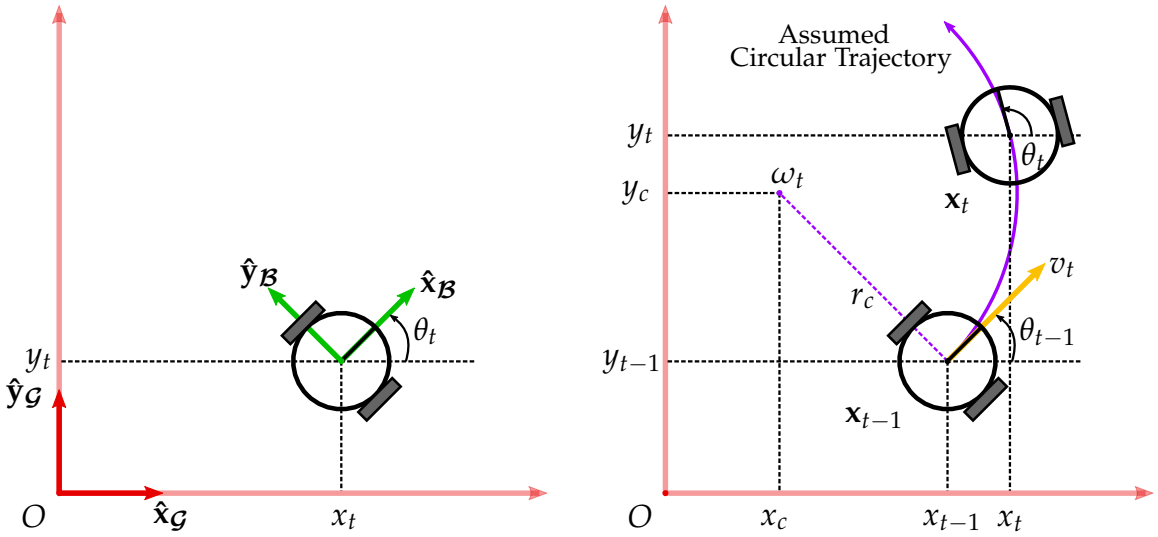


Figure 5.1: *Left:* Pose of the robot within the global reference frame; *Right:* Circular motion in the velocity model (this particular depiction assumes negligible state uncertainty)

In the selected approach to modeling two-dimensional stochastic motion, it is assumed that between any two successive states $\mathbf{x}_{t-1} = [x_{t-1} \quad y_{t-1} \quad \theta_{t-1}]^\top$ and $\mathbf{x}_t = [x_t \quad y_t \quad \theta_t]^\top$, the robot moves on a circular trajectory of radius r_c, with translational velocity v_t of unchanging magnitude and constant, non-zero, angular speed ω_t. We define the control given at time t to be $\mathbf{u}_t = [v_t \quad \omega_t]^\top$. A circular motion is taking place during the time interval Δt, and the coordinates $(x_c,\ y_c)$ of the point representing the center of the circle on which the locomotion transpires can be described by simple trigonometry using either the initial or the final point of the depicted arc. The right side of Figure 5.1 illustrates this movement, while the two sides of Equation 5.1 give the two possible representation of the circle's center:

$$\begin{aligned} x_c &= x_{t-1} - \frac{v_t}{\omega_t}\sin(\theta_{t-1}) & \qquad x_c &= x_t - \frac{v_t}{\omega_t}\sin(\theta_{t-1} + \omega_t \Delta t) \\ y_c &= y_{t-1} + \frac{v_t}{\omega_t}\cos(\theta_{t-1}) & \qquad y_c &= y_t + \frac{v_t}{\omega_t}\cos(\theta_{t-1} + \omega_t \Delta t) \end{aligned} \tag{5.1}$$

Substituting x_c and y_c given by the left side of the equation into the right-hand side yields the expression in Equation 5.2, which links $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ assuming an ideal motion, i.e.: the robot moves strictly according to the fixed commanded velocities.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -\frac{v_t}{\omega_t}\sin(\theta_{t-1}) + \frac{v_t}{\omega_t}\sin(\theta_{t-1} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos(\theta_{t-1}) - \frac{v_t}{\omega_t}\cos(\theta_{t-1} + \omega_t \Delta t) \\ \omega_t \Delta t \end{bmatrix} \tag{5.2}$$

Notwithstanding, as real-world processes are subjected to noise, it is necessary to model the ineluctable uncertainty by considering some inexact velocities $\hat{v}_t$ and $\hat{\omega}_t$ that are actually responsible for the change in the vehicle's pose. Furthermore, even with the noise taken into account, it is easy to notice that, as a consequence of the circular motion, the final pose of the robot is constrained to a two-dimensional manifold, whereas the real pose space is actually three-dimensional. This causes the posterior distribution $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$ to become degenerate, something that will damage state estimation. A correction can be made to account for this issue, correction which consists of assuming that the vehicle performs a rotation upon its arrival to the final pose. In view of this observation, we introduce an additional angular-velocity-like noisy parameter $\hat{\gamma}_t$, which will play its role in the heading at time t [7, 31]. The previous remarks are mathematically expressed in the form of Equation 5.3:

$$\begin{bmatrix} \hat{v}_t \\ \hat{\omega}_t \\ \hat{\gamma}_t \end{bmatrix} = \begin{bmatrix} v_t \\ \omega_t \\ 0 \end{bmatrix} + \begin{bmatrix} \varepsilon_{\alpha_1 v_t^2 + \alpha_2 \omega_t^2} \\ \varepsilon_{\alpha_3 v_t^2 + \alpha_4 \omega_t^2} \\ \varepsilon_{\alpha_5 v_t^2 + \alpha_6 \omega_t^2} \end{bmatrix} \tag{5.3}$$

where ε_{σ^2} denotes a zero-mean normal random error variable with variance σ^2. In the preceding equation, it is obvious that the assumption is that the variance of the error depends on both commanded velocities and some robot-specific error parameters α_i, $i \in \{1, \ldots, 6\}$. Larger parameters are an illustration of a less accurate system [7]. Combining the introduction of uncertainty with the previously-devised exact motion described by Equation 5.2 produces the stochastic motion model given in Equation 5.4:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{g}(\mathbf{x}_{t-1}, \mathbf{u}_t, \boldsymbol{\varepsilon}_t) \\ \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} &= \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}_t}{\hat{\omega}_t} \sin(\theta_{t-1}) + \frac{\hat{v}_t}{\hat{\omega}_t} \sin(\theta_{t-1} + \hat{\omega}_t \Delta t) \\ \frac{\hat{v}_t}{\hat{\omega}_t} \cos(\theta_{t-1}) - \frac{\hat{v}_t}{\hat{\omega}_t} \cos(\theta_{t-1} + \hat{\omega}_t \Delta t) \\ \hat{\omega}_t \Delta t + \hat{\gamma}_t \Delta t \end{bmatrix} \end{aligned} \tag{5.4}$$

5.2 Measurement Model

Sensing is the second stochastic process incorporated in the Bayes filter and it is described by the continuous probability model $p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m})$. With so many existing sensor technologies, it is natural that numerous ways of defining this probability density function have been developed. A very prevalent approach for data acquisition using laser rangefinders in the field of robot localization is represented by feature-based methods, in which only specific information about the environment is extracted from the raw measurements of the sensing device at hand. This information can be eventually utilized for creating feature-based maps of the surroundings. This property has been proven to be quite advantageous, explaining why features have been traditionally chosen as the backbone of sensor models [7].

5.2.1 Feature-Based Observation Model for Point Landmarks

The present subsection describes arguably the simplest feature-based observation model, employed in [7]. Here, the authors consider the case in which landmarks extracted from the environment are discrete points with fixed coordinates[1] in the global reference (x_j, y_j) frame and append to them an additional optional identifier called signature s_j (e.g.: numerical value indicating average colour). It is subsequently assumed that the robot is equipped with a range and bearing sensor capable of extracting the i-th feature in the form $[r_t^i \;\; \varphi_t^i \;\; s_t^i]^\top$. This vector holds a landmark's signature and position expressed in polar coordinates in the robot's local frame. From the entire measurement data set $\mathbf{z}_t$ supplied by the said sensor at a specific time, only the sought features are kept. Feature extraction maps the high-dimensional measurement space to a feature space of a much lower dimension. It can thus be perceived as a means of filtering excess data and reducing the size of the measurement vector, being beneficial from a computational perspective [7]. This is generally true, irrespective of the type and shape of the extracted landmarks. Denoting the feature extractor as the surjective vector-valued function $\boldsymbol{\mathcal{F}}(\mathbf{z}_t)$, its output is shown in Equation 5.5:

$$\boldsymbol{\mathcal{F}}(\mathbf{z}_t) = \{\boldsymbol{\mathcal{F}}_t^1, \; \boldsymbol{\mathcal{F}}_t^2, \dots\} = \left\{ \begin{bmatrix} r_t^1 \\ \varphi_t^1 \\ s_t^1 \end{bmatrix}, \; \begin{bmatrix} r_t^2 \\ \varphi_t^2 \\ s_t^2 \end{bmatrix}, \dots \right\} \tag{5.5}$$

where the superscripts denote a feature's number in the entire feature vector.

If we accept that the i-th feature corresponds to the j-th landmark in the map, then an estimation of the position of this landmark relative to the robot can be devised through straightforward trigonometry applied to the configuration in Figure 5.2. In this illustration, we depict the ideal scenario of a noiseless sensor. The blue dot indicates the point measurement, while the yellow seven-point star symbolizes a landmark.

From the belief of the robot about its location and the knowledge of the position of a landmark within the global reference frame, the estimated polar coordinates and the signature of the landmark in the body frame are given by Equation 5.6. Notice the introduction of noise to accommodate the real-life imperfect measurements corrupted by noise.

$$\begin{bmatrix} r_t^i \\ \varphi_t^i \\ s_t^i \end{bmatrix} = \begin{bmatrix} \sqrt{(x_j - x_t)^2 + (y_j - y_t)^2} \\ \arctan 2(y_j - y_t, x_j - x_t) - \theta_t \\ s_j \end{bmatrix} + \begin{bmatrix} \delta_{\sigma_r^2} \\ \delta_{\sigma_\varphi^2} \\ \delta_{\sigma_s^2} \end{bmatrix} \tag{5.6}$$

where the δ_{σ^2} denotes a zero-mean Gaussian error variable with standard deviation σ.

[1] The case of moving landmarks is not addressed in this report

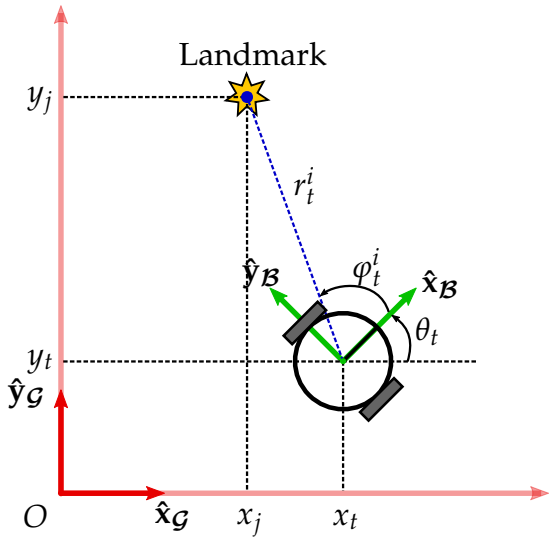


Figure 5.2: Illustration of the robot's onboard sensor measuring the relative position of a point landmark

5.2.2 Feature-Based Observation Model for Line Landmarks

The model utilized here for line landmarks is the same as the one proposed by the authors of [77]. Just like in this paper, we denote rectilinear features by l_j, and represent them in the global frame by the polar coordinates $(r_j,\ \psi_j)$ of the point on the line that, when joined with the global origin, gives a segment perpendicular to l_j. Since such lines will be used to build and/or describe a map $\mathbf{m}$, l_j is represented as an element of the map by the vector $\mathbf{m}_j = [r_j \quad \psi_j]^\top$. A signature parameter can be added if needed, though it is not crucial for feature extraction. We again assume, as in Section 5.2.1, that a range and bearing sensor on the robot can supply measurements from which features of the form $[\rho_t^i \quad \alpha_t^i]^\top$ can be extracted. These are the body-frame polar coordinates of the point on line l_j that, when joined with the local origin[2], gives a segment perpendicular to l_j. We can proceed and write the feature vector as shown in Equation 5.7.

$$\boldsymbol{\mathcal{F}}(\mathbf{z}_t) = \{\boldsymbol{\mathcal{F}}_t^1,\ \boldsymbol{\mathcal{F}}_t^2, \dots\} = \left\{ \begin{bmatrix} \rho_t^1 \\ \alpha_t^1 \end{bmatrix},\ \begin{bmatrix} \rho_t^2 \\ \alpha_t^2 \end{bmatrix}, \dots \right\} \tag{5.7}$$

Henceforth, we shall drop the $\boldsymbol{\mathcal{F}}$-notation and write $\mathbf{z}_t$ for the vector of features, and we may sometimes alternatively refer to it as the measurement vector directly. In addition, we may also still denote by $\mathbf{z}_t$ the entire vector of points supplied by the sensor, and the reader will be told when that is the case. Let us also denote the i-th feature $\mathbf{z}_t^i$. It should be stated that we impose the restrictions $\rho_t^i,\ r_t^j \geq 0$ and $\alpha_t^i,\ \psi_j \in (-\pi,\ \pi]$.

The observation model utilized is contingent on the intersection of the vector joining the origin of the global coordinate system with the robot's current position - we

[2]Just to clarify, by "local origin" we mean the origin of the local (or body) frame

denote this vector by $\mathbf{p}_{\mathcal{G}}$ - and the extension of the j-th line feature l_j. There are hence two cases that have to be considered, as it can be readily seen from Figure 5.3. Once again, the blue circle is associated with the sensor data in the local frame and the star shows the point that is saved as a landmark. Notice that in this figure the extracted feature and the landmark it is associated with are basically represented by the same line. In a real scenario, the feature extraction from sensor data leads to lines that are not superimposed to the landmarks in the map they are supposed to represent.

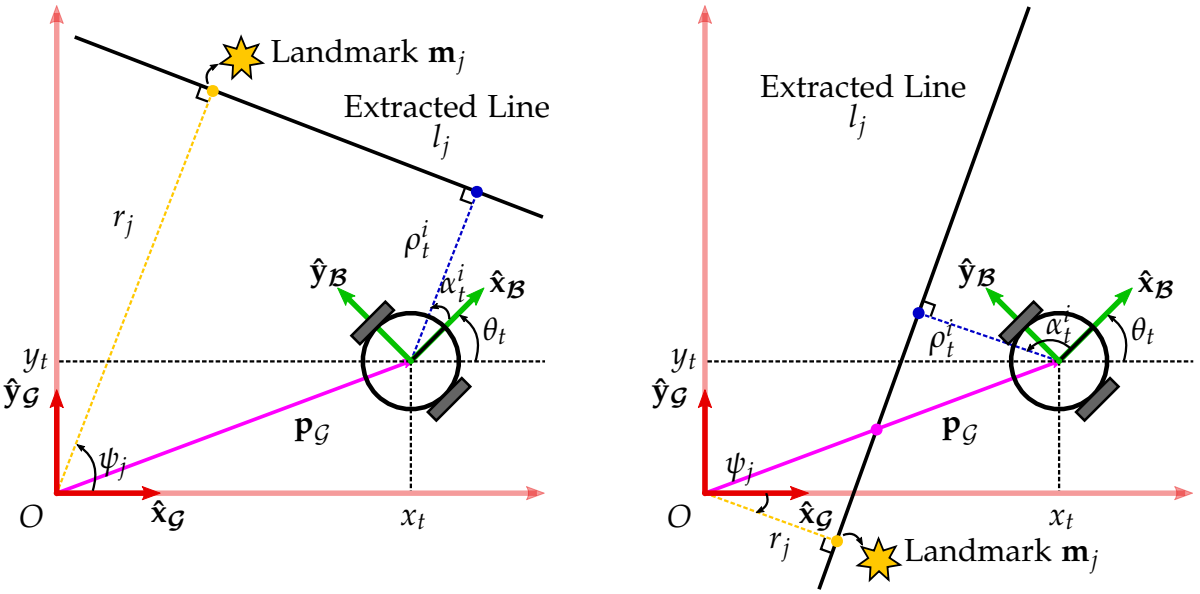


Figure 5.3: Illustration of the robot's onboard sensor measuring the relative position of a line landmark. *Left:* $\mathbf{p}_{\mathcal{G}} \cap l_j = \varnothing$; *Right:* $\mathbf{p}_{\mathcal{G}} \cap l_j \neq \varnothing$ (see the pink point)

Starting with the presumption that the i-th feature corresponds to the j-th landmark in the map, we obtain, with the aid of a simple trigonometrical approach described in Appendix A, Equation 5.8, which constitutes the stochastic measurement model for line landmarks:

$$\begin{bmatrix} \rho_t^i \\ \alpha_t^i \end{bmatrix} = \begin{cases} \begin{bmatrix} -r_j + x_t \cos(\psi_j) + y_t \sin(\psi_j) \\ \psi_j - \theta_t + \pi \end{bmatrix} + \begin{bmatrix} \delta_{\sigma_\rho^2} \\ \delta_{\sigma_\alpha^2} \end{bmatrix} & \text{if } \mathbf{p}_{\mathcal{G}} \cap l_j \neq \varnothing \\ \begin{bmatrix} r_j - x_t \cos(\psi_j) - y_t \sin(\psi_j) \\ \psi_j - \theta_t \end{bmatrix} + \begin{bmatrix} \delta_{\sigma_\rho^2} \\ \delta_{\sigma_\alpha^2} \end{bmatrix} & \text{if } \mathbf{p}_{\mathcal{G}} \cap l_j = \varnothing \end{cases} \tag{5.8}$$

Taking advantage of the mathematical similarity between the two expressions above, we can encompass both sensor models in the formula given in Equation 5.9:

$$\begin{aligned} \mathbf{z}_t^i &= \mathbf{h}(\mathbf{x}_t, j, \mathbf{m}, \kappa) + \boldsymbol{\delta}_t \\ \begin{bmatrix} \rho_t^i \\ \alpha_t^i \end{bmatrix} &= \begin{bmatrix} (-1)^\kappa(-r_j + x_t \cos(\psi_j) + y_t \sin(\psi_j)) \\ \psi_j - \theta_t + (1-\kappa)\pi \end{bmatrix} + \begin{bmatrix} \delta_{\sigma_\rho^2} \\ \delta_{\sigma_\alpha^2} \end{bmatrix} \end{aligned} \tag{5.9}$$

where $\kappa \in \{0, 1\}$. We set $\kappa = 0$ if an intersection occurs, and we let it be 1 if there is no intersection. The variables of type δ_{σ^2} have been already defined in the previous subsection.

At this point, the ability to identify the intersections that occur between landmarks and the position vector $\mathbf{p}_{\mathcal{G}}$ becomes quite desirable. The simple reason is that only one of the two sensor models should be chosen accordingly in each case. The easiest and most elegant means of making the correct decision would be to choose either variation, say the first one, in Equation 5.8, and check the resulting measurement estimation. If ρ_t^i is positive, the initial guess was correct and κ would be 0. For a negative distance, ρ_t^i should be simply multiplied with -1, π should be added to α_t^i and κ should be set to 1.

5.3 Extraction of Line Features

Obtaining a useful representation of rectilinear environmental features is inherently involving a processing stage when it comes to the raw measurements supplied by the onboard sensory equipment. As noted in the previous section, a sensor supplies a vector of range and bearing pairs of high dimensionality for all the points within its visual range. We denote said M pairs by $[d_k \;\; \phi_k]^\top$, $k \in \{1, \ldots, M\}$. These are the polar coordinates in the robot-centered reference frame of all the points sensed at a certain time. Extraction is synonymous to deducing the $[\rho_t^i \;\; \alpha_t^i]^\top$ parameters belonging to a real-world linear-shaped feature.

5.3.1 Segmentation

In the specialized literature, several line extraction algorithms have been employed with acceptable degrees of success. An in-depth comparison of such algorithms in the context of indoor mobile robotic navigation using a 2D laser sensor has been carried out in [79]. All the methods discussed there have in common the aims of identifying the number of lines in the environment, assigning clusters of points to specific lines and finally estimating the sought parameters based on the individual groups of points. These goals are attained in a two-step manner, by first segmenting the points that are found to belong to a common feature and then fitting lines to the individual point clusters. Based on concrete empirical data, the authors of the aforementioned paper inferred that possibly the best algorithm for real-time applications related to SLAM is Split-and-Merge, which provides "superior speed and correctness" [79] as compared to the other investigated approaches. In view of these findings, Split-and-Merge has been selected in this project.

The segmentation process consists of examining all the scanned data at time t, sorted by the angular coordinate. Starting form an all-encompassing segment $\mathcal{S}_1$, points are iteratively being grouped together in smaller subsegments. Following this procedure, merging similar (collinear) clusters is performed using a sequence of steps

that mirrors the one employed during segmentation. Algorithm 1 gives the directions for implementing Split-and-Merge [79, 80].

Algorithm 1: Split-and-Merge

1 $Split\_and\_Merge(\mathcal{S}_1)$:
2 $\mathcal{S}_1$ consists of all the M points scanned at time t. Place $\mathcal{S}_1$ in a list $\mathcal{L}$;
3 Create a line by uniting the first and last point of the next segment $\mathcal{S}_i$ in the list $\mathcal{L}$;
4 Find the most distant point p_k relative to the fitted line, if the current segment has more than 2 points;
5 If the distance is smaller than a predefined threshold value, go back to Step 3;
6 Else, divide $\mathcal{S}_i$ at p_k into two subsegments, $\mathcal{S}_{i,1}$ and $\mathcal{S}_{i,2}$, and let them substitute $\mathcal{S}_i$ in $\mathcal{L}$. Go to Step 3;
7 After checking all the segments in $\mathcal{L}$, merge the collinear ones;
8 Discard segments that are too short or consist of less than a certain number of points;
9 **return** $\mathcal{L}$;

Identifying collinear segments that should be merged is a matter of uniting the same point of a segment with the last point of the successive segment in the list and again searching for the most distant point to this new line. If the distance is found to not exceed a threshold (i.e.: it is considered satisfactorily small), the segments are merged. After merging, the segments that are too short are discarded, for they are deemed unreliable [77]. The Split-and-Merge variation in which fitting (Step 3) is done by simply uniting the first and last points in a segment is called Iterative-End-Point-Fitting.

A visualization of the algorithm is given in Figure 5.4, where a simple point configuration is segmented in only 3 iterations. At each step, a line shown in red indicates the next segment in the list to be segmented, while segments in blue are a sign that two successive clusters have been merged. The maximum distances from a point in a cluster to the line segment related to that cluster are marked by dashed lines.

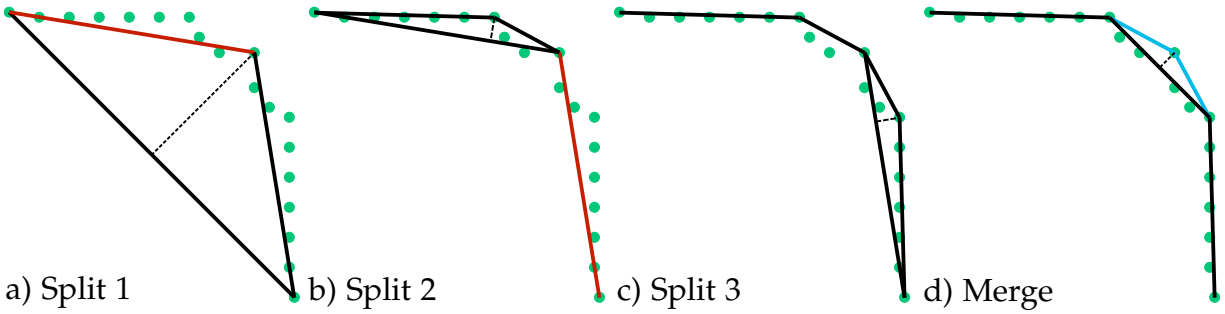


Figure 5.4: Illustration of Split-and-Merge, the Iterative-End-Point-Fitting version. Splitting stage is done in a), b) and c). In d), collinear segments are merged

5.3.2 Line Model Parameter Estimation

After the entire M-dimensional measurement space has been split into several Z clusters (segments) of points belonging to the same line, a line fitting method based on total least-squares is employed to estimate the line parameters $[\rho_t^i \quad \alpha_t^i]^\top$. To get a sense for the desired fitting, an illustration is provided in Figure 5.5.

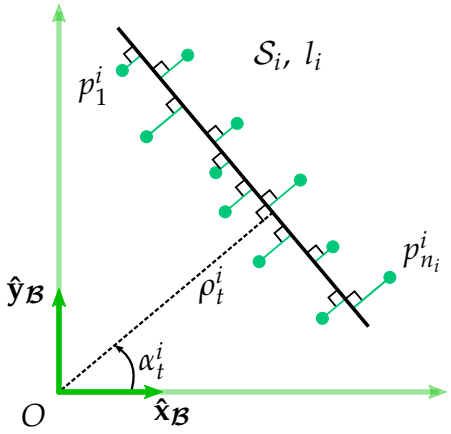


Figure 5.5: Illustration of total least-squares line fitting to a cluster of points in the context of line extraction

First, we need an expression for the distance from p_h^i, the h-th point with local Cartesian coordinates $[x_h^i \quad y_h^i]^\top$ (notice that the $\cdot_t$ notation was dropped in this case for convenience) belonging to segment $\mathcal{S}_i$, to a line l parameterized by $[\rho \quad \alpha]^\top$. The derivation is quite straightforward once one becomes aware of the similarity between this problem and the one discussed in Subsection 5.2.2. It is easily seen that the robot's body frame in Figure 5.5 is a substitute of the global frame in Figure 5.3, and a location of the point in Figure 5.5 corresponds to the location of the robot in Figure 5.3. In Subsection 5.2.2, we used simple geometry (see also Appendix A for details) to derive an expression for ρ_t^i. We can use exactly the same approach to estimate the distance we are seeking, due to the situations being essentially equivalent. The result is given by Equation 5.10:

$$\text{dist}(p_h^i, l) = |\rho - x_h^i \cos(\alpha) - y_h^i \sin(\alpha)| \tag{5.10}$$

We also note that the aforementioned rectangular coordinates can be found from the distance-angle information the sensor outputs if the simple equality in Equation 5.11 is used:

$$\begin{bmatrix} x_h^i \\ y_h^i \end{bmatrix} = \begin{bmatrix} d_h^i \cos(\phi_h^i) \\ d_h^i \sin(\phi_h^i) \end{bmatrix} = \begin{bmatrix} u_1(d_h^i, \phi_h^i) \\ u_2(d_h^i, \phi_h^i) \end{bmatrix} \tag{5.11}$$

The squared sum $E(\rho, \alpha)$ of all the distances from each of the n_i points to the line l_i has to be minimized to get the feature vector associated with the line. This is the well-known total least-squares constrained optimization inquiry, stated formally in Equation 5.12:

$$\begin{bmatrix}\rho_t^i\\ \alpha_t^i\end{bmatrix} = \begin{bmatrix}\rho*\\ \alpha*\end{bmatrix} = \underset{\rho,\alpha}{\operatorname{argmin}}\, E(\rho,\alpha) = \sum_{h=1}^{n_i}[\rho - x_h^i\cos(\alpha) - y_h^i\sin(\alpha)]^2 \tag{5.12}$$

The preceding problem has been proven to have the closed-form solution (see [77, 80, 81, 82, 83]) given by Equation 5.13:

$$\begin{bmatrix}\rho^*\\ \alpha^*\end{bmatrix} = \begin{bmatrix}\overline{x}^i\cos(\alpha^*) + \overline{y}^i\sin(\alpha^*)\\ \frac{1}{2}\arctan 2(-2S_{xy}^i, S_{y^2}^i - S_{x^2}^i)\end{bmatrix} = \begin{bmatrix}v_1(x_1^i, y_1^i, \ldots, x_{n_i}^i, y_{n_i}^i)\\ v_2(x_1^i, y_1^i, \ldots, x_{n_i}^i, y_{n_i}^i)\end{bmatrix} \tag{5.13}$$

where we utilize the notation given by Equation 5.14:

$$\begin{aligned}\overline{x}^i &= \frac{1}{n_i}\sum_{h=1}^{n_i} x_h^i & \overline{y}^i &= \frac{1}{n_i}\sum_{h=1}^{n_i} y_h^i\\ S_{x^2}^i &= \sum_{h=1}^{n_i}(x_h^i - \overline{x}^i)^2 & S_{y^2}^i &= \sum_{h=1}^{n_i}(y_h^i - \overline{y}^i)^2\\ S_{xy}^i &= \sum_{h=1}^{n_i}(x_h^i - \overline{x}^i)(y_h^i - \overline{y}^i)\end{aligned} \tag{5.14}$$

It is worth mentioning that if the solution given in Equation 5.13 produces $\rho_t^i < 0$, then the true line parameters are $[-\rho_t^i \quad \alpha_t^i + \pi]^\top$ [81]. Notice also that the optimal distance ρ_t^i from the origin of the local frame to the line l_i given in Equation 5.13 shows that the mean of all the points in $\mathcal{S}_i$ belongs to the fitted line and it is in fact the point closest to the origin of the body frame.

Once a line is fitted to a specific segment of scanned points, a finite line section (a segment, in the geometrical sense) can be further chosen for visualization purposes. This is very useful in building maps that are easier to understand for humans. The details of the necessary steps needed for such a representation are explored in Appendix A.

5.3.3 Line Feature Covariance Estimation

In the preceding section, it was demonstrated that the two parameters representing a line feature in the robot-centered reference frame can be computed analytically given raw sensor measurements grouped based on Split-and-Merge. Sensors are imperfect, and thus, in practice, for any of the line feature $\mathbf{z}_t^i$, $i \in \{1, \ldots, Z\}$, each and every scanned point $p_h^i(d_h^i,\ \phi_h^i)$, $h \in \{1, \ldots, n_i\}$ is actually subjected to uncertainty. Therefore, the equations shown in the extraction process actually apply to random Gaussian variables. However, they are of course valid for the expected values of those variables. We can think of the measurements as the means of the Gaussian RVs representing the range and bearing of points in the environment, and in order to avoid additional notation we shall utilize the same symbols for the random variables and their expected (measured) values.

What we desire at this juncture is to identify the covariance $\mathbf{R}_t^i$ of a certain feature $\mathbf{z}_t^i$. The form of the the matrix is given in Equation 5.15:

$$\mathbf{R}_t^i = \begin{bmatrix} \sigma_\rho^2 & \sigma_\rho \sigma_\alpha \\ \sigma_\alpha \sigma_\rho & \sigma_\alpha^2 \end{bmatrix} \tag{5.15}$$

This matrix is essentially obtained by fusing the covariances of all the points belonging to the same line feature. The problem we are dealing with is known as error propagation and arises when multiple uncertain measurements need to be combined [80].

Following the derivations done in [77] and the observations in [80, 84], the mapping we are looking for is obtained if the Cartesian coordinate covariance matrices of individual points are known. For the h-th point of the i-th feature, this matrix is denoted as $\boldsymbol{C}_h^i$. An illustration of the covariance propagation is available in Figure 5.6, which shows again the idea behind total-least-squares fitting in Figure 5.5, but this time the covariances of the points and the covariance of the line feature are explicitly shown as ellipses.

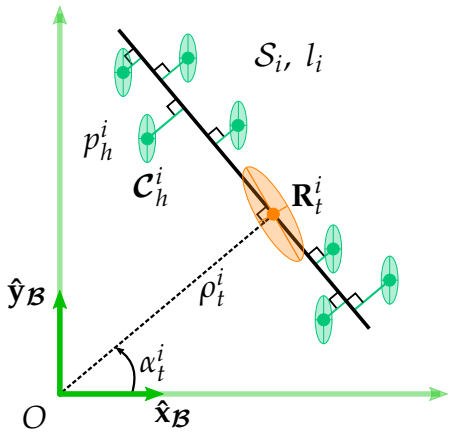


Figure 5.6: Illustration of error propagation for total-least-squares line fitting

Some of the points of the original figure have been removed to make room for the depiction of the line covariance. It should be emphasized that the ellipses plotted are meant to only convey an idea of uncertainty to the reader, and may not represent a reliable indicator of the covariance values in reality. Similarly, the fact that all points have the same covariance in the figure is not to be understood as if that should be the case in a practical setting.

Now, given the distinct uncertainties of the points in a segment, a line feature's covariance matrix $\mathbf{R}_t^i$ is obtained from Equation 5.16:

$$\mathbf{R}_t^i = \sum_{h=1}^{n_i} \boldsymbol{\mathcal{A}}_h^i \boldsymbol{C}_h^i [\boldsymbol{\mathcal{A}}_h^i]^\top \tag{5.16}$$

in which $\boldsymbol{\mathcal{A}}_h^i$ is the Jacobian matrix - taken with respect to the h-th point and

evaluated at its mean[3] - of the vector function $\mathbf{v}$, with the element functions[4] given in Equation 5.13. From [77], $\boldsymbol{\mathcal{A}}_h^i$ is as shown in Equation 5.17:

$$\boldsymbol{\mathcal{A}}_h^i = \begin{bmatrix} \frac{\partial v_1}{\partial x_h^i} & \frac{\partial v_1}{\partial y_h^i} \\ \frac{\partial v_2}{\partial x_h^i} & \frac{\partial v_2}{\partial y_h^i} \end{bmatrix}_{x_h^i,\, y_h^i} = \begin{bmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} \\ \mathcal{A}_{21} & \mathcal{A}_{22} \end{bmatrix} \tag{5.17}$$

where we evaluate the entries separately, in Equation 5.18 [77]:

$$\begin{aligned} \mathcal{A}_{11} &= \frac{\cos(\alpha_t^i)}{n_i} - \mathcal{A}_{21}(\overline{x}^i \sin(\alpha_t^i) - \overline{y}^i \cos(\alpha_t^i)) \\ \mathcal{A}_{12} &= \frac{\sin(\alpha_t^i)}{n_i} - \mathcal{A}_{22}(\overline{x}^i \sin(\alpha_t^i) - \overline{y}^i \cos(\alpha_t^i)) \\ \mathcal{A}_{21} &= \frac{(\overline{y}^i - y_h^i)(S_{y^2}^i - S_{x^2}^i) + 2S_{xy}^i(\overline{x}^i - x_h^i)}{(S_{y^2}^i - S_{x^2}^i)^2 + 4(S_{xy}^i)^2} \\ \mathcal{A}_{22} &= \frac{(\overline{x}^i - x_h^i)(S_{y^2}^i - S_{x^2}^i) + 2S_{xy}^i(\overline{y}^i - y_h^i)}{(S_{y^2}^i - S_{x^2}^i)^2 + 4(S_{xy}^i)^2} \end{aligned} \tag{5.18}$$

However, the sensor gives readings in polar coordinates, not in Cartesian, and therefore $\boldsymbol{\mathcal{C}}_h^i$ is unknown and it must be obtained from the covariance matrix of the polar coordinates for a single point. We label this matrix $\boldsymbol{\mathcal{D}}_h^i$ and show its contents in Equation 5.19, assuming that distance and angle are independent and hence uncorrelated:

$$\boldsymbol{\mathcal{D}}_h^i = \begin{bmatrix} \sigma_{d_h^i}^2 & 0 \\ 0 & \sigma_{\phi_h^i}^2 \end{bmatrix} \tag{5.19}$$

This new mapping is completed by writing $\boldsymbol{\mathcal{C}}_h^i$ as shown in Equation 5.20:

$$\boldsymbol{\mathcal{C}}_h^i = \boldsymbol{\mathcal{B}}_h^i \boldsymbol{\mathcal{D}}_h^i [\boldsymbol{\mathcal{B}}_h^i]^\top \tag{5.20}$$

where the Jacobian $\boldsymbol{\mathcal{B}}_h^i$ of the vector function $\mathbf{u}$, with the element functions given in Equation 5.11, was taken with respect to the h-th point and evaluated at its mean. The entries of this Jacobian are shown in Equation 5.21:

$$\boldsymbol{\mathcal{B}}_h^i = \begin{bmatrix} \frac{\partial u_1}{\partial d_h^i} & \frac{\partial u_1}{\partial \phi_h^i} \\ \frac{\partial u_2}{\partial d_h^i} & \frac{\partial u_2}{\partial \phi_h^i} \end{bmatrix}_{d_h^i,\, \phi_h^i} = \begin{bmatrix} \cos(\phi_h^i) & -d_h^i \sin(\phi_h^i) \\ \sin(\phi_h^i) & d_h^i \cos(\phi_h^i) \end{bmatrix} \tag{5.21}$$

[3] We remind the reader again that in this subsection we use the same notation for a random variable and its expected value

[4] The vector functions $\mathbf{u}$ and $\mathbf{v}$ and their element functions should not be confused with the control vector $\mathbf{u}_t$ or the translational speed v_t

Finally, substituting Equation 5.20 into Equation 5.16 yields Equation 5.22, which provides a means of computing the covariance associated with a line from the individual covariances of its element points:

$$\mathbf{R}_t^i = \sum_{h=1}^{n_i} \boldsymbol{\mathcal{A}}_h^i \boldsymbol{\mathcal{B}}_h^i \boldsymbol{\mathcal{D}}_h^i [\boldsymbol{\mathcal{A}}_h^i \boldsymbol{\mathcal{B}}_h^i]^\top \tag{5.22}$$

5.3.4 Split-and-Merge Implementation with LiDAR

The mechanics and implications of the Split-and-Merge algorithm have already been elucidated upon in the previous sections. However, there is merit to this discussion only if the said algorithm provides the desired result for the actual sensory data.

In order to obtain the data points from the range finder we utilize ROS. In the present situation, ROS usage is limited to a single publisher and subscriber. As expected, the LiDAR node plays the role of a publisher, publishing the data points over the topic and a subscribing node is enacted which receives the information. The subscriber then feeds those points into the Split-and-Merge algorithm. For the purpose of testing the algorithm, the scenario employed is kept simple, with just the LiDAR scanning the walls and other surrounding objects inside a room that fall within its plane of rotation. The incoming data from the LiDAR comprise the distance to the scanned surfaces (range) and the angle of rotation corresponding to those distances. Accordingly, through this information, a scanned point can be represented using the polar coordinate system or can easily be converted into its Cartesian counterpart. As such, from the scan, the points for a single rotation of the LiDAR are shown in the left graph in Figure 5.7. Due to their large number, the points altogether might look like lines themselves, but still are distinctly placed. The output of the Split-and-Merge algorithm is shown in the right half of Figure 5.7.

On side by side comparison of the right and left halves of Figure 5.7, the difference is instantly recognizable. The identifiable point cluster from the left half are now replaced by the line segments in the right half. From the results of this small experiment, it suffices to say that the outcome of the Split-and-Merge algorithm is up to the expectations and thus suitable for inclusion in the upcoming intricate processes.

5.3.5 LiDAR Noise Distribution

The analysis in Subsection 5.3.3 was exclusively focused on identifying the covairance matrix $\mathbf{R}_t^i$ of a certain feature. The significance of this covariance will become more apparent in the later sections. As such, an expression was presented to calculate $\mathbf{R}_t^i$, however, the said expression (given in Equation 5.22) is reliant on the covariance matrix $\boldsymbol{\mathcal{D}}_h^i$. The matrix $\boldsymbol{\mathcal{D}}_h^i$ is essentially governed by the noise distribution over the measurements of the LiDAR sensor. Therefore, an experiment was performed to acquire the corresponding noise distribution. Before we proceed further, it should be notified that the experiment was conducted using a simulated LiDAR sensor within

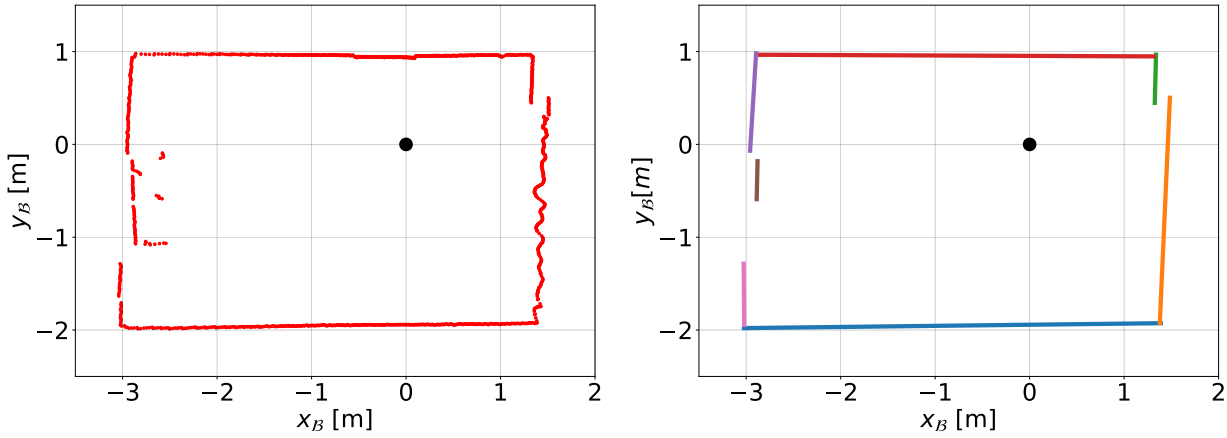


Figure 5.7: Split-and-Merge algorithm implementation in a test area. *Left:* Raw LiDAR points from the test area. *Right:* Point Clusters Converted to Points

Gazebo[5]. The details of the simulated sensor and the platform will be made clear later in Chapter 8.

The $\mathcal{D}_h^i$ matrix requires the variance of both range and bearing measurements from the sensor. Having said that, the LiDAR sensor only provides the bearing data in terms of a constant angular increment, therefore it can be assumed that the variance for the angles is zero (i.e. $\sigma_\phi^2 = 0$). As for the range measurements, the experiment was a simple task of obtaining multiple distance readings for the same scanned point. As a result, the acquired measurements could be plotted as histograms and are shown in Figure 5.8. It should be evident that the shape of the histogram plot is emulating a Gaussian distribution. Therefore, a normal distribution curve is fitted to the histogram (shown by the red line in Figure 5.8). Hence, by extracting the standard deviation (σ_d) of this normal distribution, we can calculate the variance for range measurements.

The experiment done here is only for one of many points that a LiDAR can scan. Therefore, it is possible that variance might change for points at different distances, depending on the specifications of the LiDAR sensor. Moreover, conducting experiment for each point is non-viable, hence we will be operating under the assumption that all LiDAR points have the same covariance matrix $\mathcal{D}$, as shown in Equation 5.23.

$$\mathcal{D} = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\phi^2 \end{bmatrix} = \begin{bmatrix} 1.055 \cdot 10^{-4} & 0 \\ 0 & 0 \end{bmatrix} \tag{5.23}$$

5.4 The Data Association Problem

One of the main obstacles in the way of developing and implementing localization and mapping algorithms goes by the name of the data association problem (also known

[5]The procedure for the actual LiDAR sensor would remain the same.

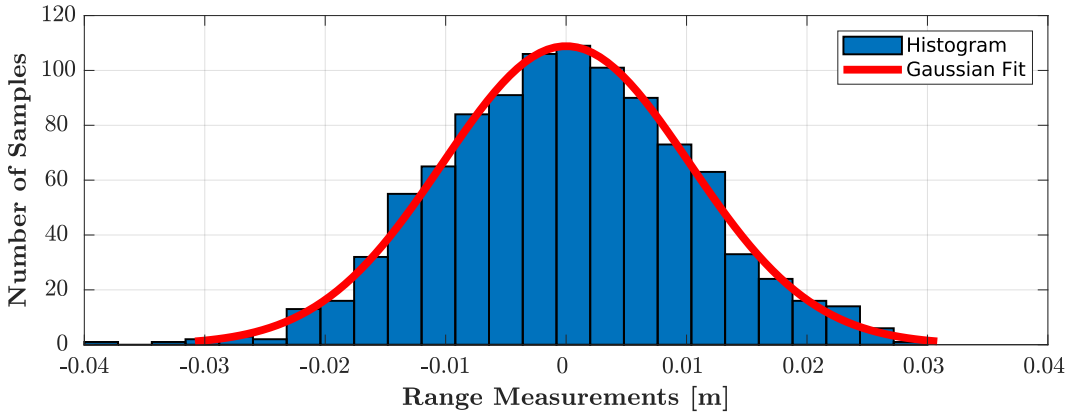


Figure 5.8: Histograms and probability density functions of the LiDAR measurements with mean value subtracted

as the correspondence or matching problem) [6, 7, 77]. If the landmarks the robot is using to estimate its position in the environment cannot be uniquely identified, then a question arises regarding the identity of a landmark given an observed feature. Moreover, [6] avers the following: "The standard formulation of the EKF-SLAM solution is especially fragile to incorrect association of observations to landmarks".

Remember that the equations and explanations provided in Section 5.2 are based on the presumption that the i-th extracted feature corresponds to the j-th landmark in the map. In reality, we need a means for going from assumption to verifiable certainty. Let us introduce a correspondence variable c_t^i, $i \in \{1 \dots Z\}$ linking the feature $\mathbf{z}_t^i$ with a certain landmark. Supposing that at time t the map comprises N landmarks, we have that $c_t^i = j \in \{1 \dots N\}$, if the feature is found to correspond to an already-existing landmark $\mathbf{m}_j$. If, however, $c_t^i = N + 1$, the i-th feature is found to not correspond to any of the recorded landmarks and therefore it is interpreted as an unobserved landmark and it is appended to $\mathbf{m}$, N being updated accordingly.

The correspondences are crucial in the context of robot localization and mapping, because landmarks are essentially the only information the vehicle can ever possess about the surroundings. Hence, ensuring accurate position estimation is tightly conditioned on the correct identification of landmarks. As explained, the robot should be able to either correctly assign features sightings to previously observed landmarks or, on the contrary, decide that a feature should be elevated to the rank of new landmark.

Solutions for tackling the matching problem will be discussed in Chapters 6 and 7.

Chapter 6

Localization

Mobile robot localization is arguably the most fundamental problem in the world of robotic perception [7, 78]. Furthermore, in the endeavour of implementing SLAM, localization represents the prime milestone. It is for these reasons why it was found appropriate to dedicate an entire chapter of the report to this central aspect. What is referred to as localization in this report is in fact the position tracking problem, in which the initial pose is relatively well known. Multimodal distributions of random variables are required for performing the more general global localization, which is not possible with the Gaussian-reliant EKF utilized here.

6.1 Definition of the Localization Problem

The robot localization problem can be graphically summarized in a diagram such as the one in Figure 6.1. The shaded nodes correspond to values that are known, namely, the controls, the measurements and the map, while the arrows depict the influences and dependencies existing between the variables in the system. The objective in localization is to deduce the next pose in the global reference frame, with respect to the map (see the non-dashed bubbles).

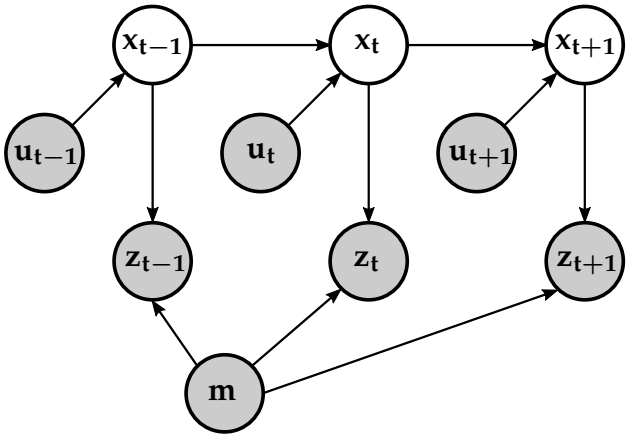


Figure 6.1: Graphical model of the localization problem

From a mathematical standpoint, the goal of localization is the estimation of the present pose $\mathbf{x}_{t+1}$ of the robot, while having access to knowledge regarding the present and all the past controls $\mathbf{u}_{1:t+1}$ and measurements $\mathbf{z}_{1:t+1}$. The vehicle also has complete awareness of the full map $\mathbf{m}$. Probabilistically, the conditional PDF $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_{1:t+1}, \mathbf{z}_{1:t+1}, \mathbf{m})$ needs to be identified at every time step t.

This definition accurately conveys the aim of robot position estimation, but only in the case when the current feature-to-map data associations $\mathbf{c}_{t+1}$ have been identified or are known a priori. With no knowledge of the correspondence variables, a small amendment can be made to the density function just mentioned, so that the necessity of discovering the right matchings is not ignored. The new sought PDF is the joint conditional $p(\mathbf{x}_{t+1}, \mathbf{c}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_{1:t+1}, \mathbf{z}_{1:t+1}, \mathbf{m})$.

6.2 Mathematical Derivation

Prediction Step

Referring back to Section 4.2, the EKF localization starts with the computation of the predicted state estimate $\overline{\boldsymbol{\mu}}_t$ and the predicted estimate covariance $\overline{\boldsymbol{\Sigma}}_t$. The function responsible for the state transition is the motion model in Section 5.1, which has as one of its arguments the zero-mean noise vector $\boldsymbol{\varepsilon}_t$. The standard extended Kalman filter represents noise as an additive vector and, therefore, slight modifications of the EKF prediction equations are required. The new expressions can be deduced by rewriting the first-order Taylor linearization of $\mathbf{g}$ at $\boldsymbol{\mu}_{t-1}$, $\mathbf{u}_t$ and $\mathbf{0}$, and including the differentiation with respect to the noise (see Equation 6.1) [72, 85, 86].

$$\begin{aligned}\mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}, \boldsymbol{\varepsilon}_t) &\approx \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}, \mathbf{0}) + \frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}, \boldsymbol{\varepsilon}_t)}{\partial \mathbf{x}_{t-1}}(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) + \frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}, \boldsymbol{\varepsilon}_t)}{\partial \boldsymbol{\varepsilon}_t}\boldsymbol{\varepsilon}_t \\ &\approx \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}, \mathbf{0}) + \mathbf{G}_t(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) + \mathbf{L}_t\boldsymbol{\varepsilon}_t\end{aligned} \tag{6.1}$$

The differentiations are of course evaluated at the linearization point. With this new linearized state space, it can be proven (see [85]) that prediction should be performed according to Equations 6.2 and 6.3:

$$\overline{\boldsymbol{\mu}}_t = \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}, \mathbf{0}) \tag{6.2}$$

$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{G}_t\boldsymbol{\Sigma}_{t-1}\mathbf{G}_t^\top + \mathbf{L}_t\mathbf{Q}_t\mathbf{L}_t^\top \tag{6.3}$$

Equation 6.4 shows the Jacobian of the nonlinear process function $\mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}, \boldsymbol{\varepsilon}_t)$, evaluated at $\mathbf{u}_t$, $\boldsymbol{\mu}_{t-1}$ and $\mathbf{0}$:

$$\mathbf{G}_t = \frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}, \boldsymbol{\varepsilon}_t)}{\partial \mathbf{x}_{t-1}}\bigg|_{\mathbf{u}_t,\, \boldsymbol{\mu}_{t-1},\, \mathbf{0}} = \begin{bmatrix} 1 & 0 & \frac{v_t}{\omega_t}(-\cos(\theta) + \cos(\theta + \omega_t \Delta t)) \\ 0 & 1 & \frac{v_t}{\omega_t}(-\sin(\theta) + \sin(\theta + \omega_t \Delta t)) \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

It should be noted that we let $\theta = \mu_{t-1,\theta}$ in the preceding expression to simplify the notation. We shall keep this convention throughout the rest of the chapter.

The non-additive noise lies embedded in the process function and contributes to obtaining the actual, noise-corrupted velocities from the noise-free commanded ones. We can restate the formula in Equation 5.3 as in Equation 6.5:

$$\begin{bmatrix} \hat{v}_t \\ \hat{\omega}_t \\ \hat{\gamma}_t \end{bmatrix} = \begin{bmatrix} v_t \\ \omega_t \\ 0 \end{bmatrix} + \mathcal{N}(0, \mathbf{Q}_t) \tag{6.5}$$

in which $\mathbf{Q}_t$ is the state transition covariance matrix in the control space (see [7]), having the form in Equation 6.6:

$$\mathbf{Q}_t = \begin{bmatrix} \alpha_1 v_t^2 + \alpha_2 \omega_t^2 & 0 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 \omega_t^2 & 0 \\ 0 & 0 & \alpha_5 v_t^2 + \alpha_6 \omega_t^2 \end{bmatrix} \tag{6.6}$$

The right-most term in the new predicted covariance estimate formula (Equation 6.3) can be interpreted as an additional calculation required to map the noise from the control space to the state space and make the model suitable for EKF implementation [7]. As observed, this is done through a linear approximation utilizing the Jacobian $\mathbf{L}_t$, with respect to the noise, of the motion function. The contents of this matrix are listed in Equation 6.7:

$$\begin{aligned} \mathbf{L}_t &= \frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}, \boldsymbol{\varepsilon}_t)}{\partial \boldsymbol{\varepsilon}_t}\bigg|_{\mathbf{u}_t,\, \boldsymbol{\mu}_{t-1},\, \mathbf{0}} \\ &= \begin{bmatrix} \frac{-\sin(\theta)+\sin(\theta+\omega_t\Delta t)}{\omega_t} & \frac{v_t(\sin(\theta)-\sin(\theta+\omega_t\Delta t))}{\omega_t^2} + \frac{v_t \Delta t \cos(\theta+\omega_t\Delta t)}{\omega_t} & 0 \\ \frac{\cos(\theta)-\cos(\theta+\omega_t\Delta t)}{\omega_t} & -\frac{v_t(\cos(\theta)-\cos(\theta+\omega_t\Delta t))}{\omega_t^2} + \frac{v_t \Delta t \sin(\theta+\omega_t\Delta t)}{\omega_t} & 0 \\ 0 & \Delta t & \Delta t \end{bmatrix} \end{aligned} \tag{6.7}$$

Update Step

The Kalman filter demands the existence of a single measurement vector $\mathbf{z}_t$ that can be modelled stochastically based on a known mathematical representation $p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{c}_t, \mathbf{m})$. However, it should be evident from the discussion in Section 5.2 that $\mathbf{z}_t$ is actually a concatenation of multiple features denoted $\mathbf{z}_t^i$ (Equation 5.7), and the

described observation model concerns only individual measurements, that is, Equation 5.9 gives $p(\mathbf{z}_t^i \mid \mathbf{x}_t, c_t^i, \mathbf{m})$. To use the standard EKF, additional assumptions are needed when processing multiple measurements at the same time. In a static environment, a reasonable assumption that is very helpful in addressing the introduced issue is that of conditional independence of measurement probabilities. According to [78, 87], the assumption holds as long as the noise in one measurement is independent of the noise in any other feature. Following the definition of conditional independence given in [88], the expression in Equation 6.8 results:

$$p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{c}_t, \mathbf{m}) = \prod_i p(\mathbf{z}_t^i \mid \mathbf{x}_t, \mathbf{c}_t, \mathbf{m}) = \prod_i p(\mathbf{z}_t^i \mid \mathbf{x}_t, c_t^i, \mathbf{m}) \tag{6.8}$$

The supposition allows features extracted from the sensor information to be included in the Kalman update step in a an incremental fashion. This is desirable, since implementation can now be carried out by looping through the feature vector in software, and integrating each feature individually into the filter [7].

We can now safely start analyzing the EKF update stage by looking at the i-th feature only. Equation 6.9 gives the Jacobian of the observation model (Equation 5.9) required in the EKF linearization presented in Equation 4.18:

$$\begin{aligned} \mathbf{H}_t^i &= \left.\frac{\partial \mathbf{h}(\mathbf{x}_t, j, \mathbf{m}, \kappa)}{\partial \mathbf{x}_t}\right|_{\overline{\boldsymbol{\mu}}_t,\, j,\, \mathbf{m}} \\ &= \begin{bmatrix} (-1)^\kappa \cos(\psi_j) & (-1)^\kappa \sin(\psi_j) & 0 \\ 0 & 0 & -1 \end{bmatrix} \end{aligned} \tag{6.9}$$

Choosing the value of κ should be as per the observation made in Subsection 5.2.2. The covariance matrix $\mathbf{R}_t^i$ of the measurement is obtained according to Equation 5.22, discussed in the last part of Section 5.3.

Additionally, taking inspiration from as done in [80], it is convenient to introduce here some new notation. In particular, $\hat{\mathbf{z}}_t^i$ will be used to refer to the expected measurement that is calculated based on the observation model, while $\mathbf{S}_t^i$ will denote the innovation covariance matrix needed in computing the optimal Kalman gain, meaning that it will be computed as in Equation 6.10:

$$\mathbf{S}_t^i = \mathbf{H}_t^i \overline{\boldsymbol{\Sigma}}_t [\mathbf{H}_t^i]^\top + \mathbf{R}_t^i \tag{6.10}$$

It should be mentioned that it is crucial to update the a priori state estimate $\overline{\boldsymbol{\mu}}_t$ and the a priori estimate covariance $\overline{\boldsymbol{\Sigma}}_t$ at every iteration of the update stage, according to the last two equations of the extended Kalman filter. The explanation is that every landmark sighting gives the robot more information regarding its position within the map.

6.3 Data Association

One should keep in mind that, in reality, the correspondences c_t^i are rarely known with absolute certainty prior to the deployment of the algorithm. More often that not, they have to be determined along the localization process. This is precisely the matching problem briefly introduced in Section 5.4, where the importance of correct data associations has been emphasized.

6.3.1 Maximum Likelihood Association

As maintained by the authors of [7, 89, 90], the most straightforward way to tackle this issue goes by the name of maximum likelihood correspondence and is aiming to maximize the data likelihood $p(\mathbf{z}_t \mid \mathbf{c}_{1:t}, \mathbf{m}, \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t})$. Continuing with the assumption of feature independence, maximization of the preceding PDF is equivalent to the maximization of the density functions corresponding to individual extractions $\mathbf{z}_t^i$, as stated in Equation 6.11, whose proof is discussed in [7]. The result of the optimization is the best correspondence $\hat{c}_t^i$:

$$\begin{aligned} \hat{c}_t^i &= \underset{c_t^i}{\operatorname{argmax}}\, p(\mathbf{z}_t^i \mid \mathbf{c}_{1:t}, \mathbf{m}, \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) \\ &\approx \underset{c_t^i}{\operatorname{argmax}}\, \mathcal{N}(\mathbf{z}_t^i; \mathbf{h}(\overline{\boldsymbol{\mu}}_t, c_t^i, \mathbf{m}), \mathbf{H}_t \overline{\boldsymbol{\Sigma}} \mathbf{H}_t^\top + \mathbf{R}_t^i) \\ &\approx \underset{c_t^i}{\operatorname{argmax}}\, \mathcal{N}(\mathbf{z}_t^i; \hat{\mathbf{z}}_t^i, \mathbf{S}_t^i) \end{aligned} \tag{6.11}$$

For each $\mathbf{z}_t^i$ extracted at a given time, the EKF localization algorithm shall loop through all the landmarks in $\mathbf{m}$ and evaluate and store the expected measurement and the innovation matrix at every iteration k. Equation 6.11 comes into play when the probability densities are being compared, at the end of the aforementioned loop. Thus, the best association $\hat{c}_t^i$ is the index $j(i)$, which is precisely computed through Equation 6.12 [7, 89]:

$$j(i) = \underset{k}{\operatorname{argmax}}\, \frac{1}{\sqrt{\det(2\pi \mathbf{S}_t^k)}} \exp\left(-\frac{1}{2}(\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k)^\top [\mathbf{S}_t^k]^{-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k)\right) \tag{6.12}$$

An equally valid metric, though less computationally cumbersome, is the minimization of the normalized distance D_{ik}. When this is done, the matching technique is referred to as nearest neighbour associations [89, 90]. Following the directions given in Appendix B, we get the optimization requirement in Equation 6.13:

$$j(i) = \underset{k}{\operatorname{argmin}}\, D_{ik} = \underset{k}{\operatorname{argmin}}\, (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k)^\top [\mathbf{S}_t^k]^{-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k) + \ln(\det \mathbf{S}_t^k) \tag{6.13}$$

6.3.2 Validation Gating

It is widely accepted in the literature that even the simplest form of matching in localization or SLAM is actually a two-step process (see [80, 89, 90, 91]). Solely considering the most likely data association, found through either of the two preceding formulas, would overlook the issue of spurious measurements. In the case of unforeseen changes in the map (e.g.: sudden appearance of a new object) or sensor malfunction (e.g.: extraction of an absurd feature), an association would still be selected, even though the sensor observation is not related to the known map in reality. As a preventive measure, validation gating is introduced first, eliminating the statistically unlikely associations before the best candidate is chosen. In fact, an approach such as maximum likelihood is considered only an ambiguity management method, while validation gating is regarded as an ambiguity reduction scheme. While neither method should be independently utilized for acceptable selection of correspondences, when combined, they complement each other and offer a fairly straightforward solution to a very convoluted problem [89, 90].

Let us now elaborate on the concept of validation gating. The normalised innovation squared strategy, alternatively called Mahalanobis distance, is a ubiquitous validation gate that dictates the maximum allowable disparity, in the measurement space, between an extracted feature $\mathbf{z}_t^i$ and a landmark $\mathbf{m}_k$. In the context of our discussion, the squared Mahalanobis distance between the random vector $\mathbf{z}_t^i$ and the mean $\hat{\mathbf{z}}_t^i$ of the observation multivariate Gaussian distribution has the form presented in Equation 6.14 [90, 92]:

$$M_{ik} = (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k)^\top [\mathbf{S}_t^k]^{-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k) \tag{6.14}$$

As observed, the preceding equation can be used to gauge, through M_{ik}, the similarity between an actual (i-th) measurement and the measurement likelihood of any k-th landmark. The resulting number is subsequently compared to a predefined gate threshold γ_1, and the measurement is accepted for association only if the squared Mahalanobis distance does not exceed the threshold [89, 90].

For each landmark, this validation principle allows potential correspondences only with measurements falling within the area Area_k defined by Equation 6.15:

$$\text{Area}_k = \{\mathbf{z}_t^i \mid (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k)^\top [\mathbf{S}_t^k]^{-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^k) \leq \gamma_1\} \tag{6.15}$$

which, in our case, represents an ellipse in the two-dimensional observation space, as shown in the left side of Figure 6.2. The rationale why validation gating is not guaranteed to provide unequivocal data association is apparent from the right side of the same figure. While no measurement outside the yellow acceptable areas is taken into consideration, ambiguity arises when a single measurement falls within more than one gate (see that $\mathbf{z}_t^2$ is in both the right and the left ellipses) or when there are multiple observations within the same validation gate (notice that $\mathbf{z}_t^1$ and $\mathbf{z}_t^2$ are both in the left ellipse).

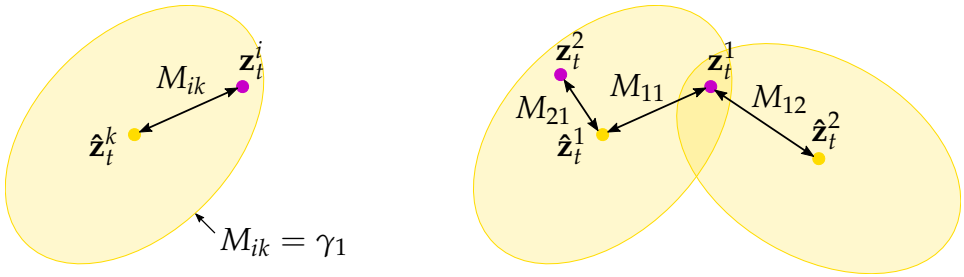


Figure 6.2: *Left*: Graphical representation of the validation gate; *Right*: Ambiguity in data association

The first of the aforementioned problems is solved by applying the nearest neighbour association (Equation 6.13). The latter, however, is much more difficult to deal with, due to the incremental structure of the algorithm: features are compared to all landmarks one at a time. In theory, always choosing the nearest neighbor could lead to assigning multiple observations to the same landmark, whereas the features may actually correspond to different locations is space. To prevent the confusion, it was decided to not associate an observation to its nearest map element if a previous measurement has been already associated with the same map element. The closest previously unassociated landmark, if any, will be utilized for correspondence.

For implementation, a specific value of the gate γ_1 should be selected based on reasoning. It can be shown that, for a multivariate Gaussian distribution, the squared Mahalanobis distance, if treated as a random variable, is χ^2-distributed (the proof is not very relevant and is quite complex; it can be found in [93]). Such distributions depend on the number of degrees of freedom, which are really just the dimensions of the random vector for which the Mahalanobis distance has been computed, so 2 in our case (remember $\mathbf{z}_t^i$ is comprised of a distance and an angular parameter). Figure 6.3 depicts the PDF of a χ^2-distribution for a continuous random variable X and 2 degrees of freedom:

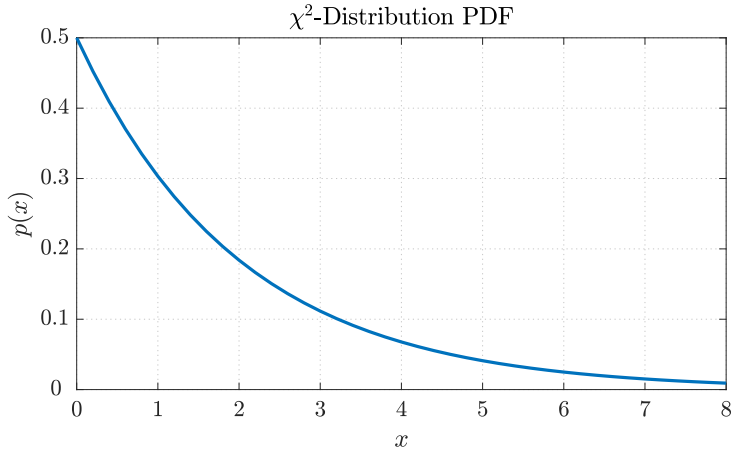


Figure 6.3: Plot of the PDF of a χ^2-distribution with 2 degrees of freedom

The value of the validation gate will determine the probability that, if an observa-

tion $\mathbf{z}_t^i$ is an actual measurement of the k-th landmark, a potential association between the two is allowed. This probability is the area under the curve in Figure 6.3, between 0 and γ_1. A common choice in the literature is $\gamma_1 = 6$, for which almost 95% of the true associations are accepted [90] (other authors use 9 [94]).

6.3.3 Correspondence Enhancement for Line Features

In the particular instance of using lines for robot localization, associating only based on validation gating and nearest neighbour, as described thus far, may not lead to successful matching. That is because the k-th line landmark is stored in the map as $[r_k \;\; \psi_k]^\top$, creating additional ambiguity when multiple lines lie on the same line (that is, they form an interrupted line). Imagine, for example, the ubiquitous real-world indoor scenario of a corridor between two aligned walls. The walls, although situated at distinct locations in space, will be represented through very similar distance and angle parameters. When a feature that should be matched to the first wall is extracted by the sensor, an assignment of the feature to the second wall could be performed instead due to noise. The wrong assignment has detrimental effects on the localization process [77, 94].

An enhancement to the correspondence stage can be done in a similar fashion to what the authors of [77] propose. For localization, the midpoints of collinear landmarks have unique known global-frame Cartesian coordinates and thus they could be stored and utilized as an additional tool in the association process[1]. In Figure 6.4, the challenge of selecting an association for collinear landmarks is depicted. If the robot would be aware of the locations of the midpoints for the feature and the two landmarks, it would correctly select the orange line, and not the purple one, as the map element "closest" to the extracted segment.

Our objective is to define a new metric that can be calculated for each feature-landmark pair and can aid us in making a decision for assigning observations to map elements. Because we can find such a metric using concepts already discussed in detail in this thesis, we shall skip the definitions and proofs and focus on the results.

In Equation 6.16, a slightly modified measurement model for point features is given. An observation $\mathbf{z}_{t,\text{mid}}^i$ of a feature's midpoint is modeled as a function of the robot's pose and the location of the landmark midpoints $\mathbf{m}_{k,\text{mid}} = [x_{k,\text{mid}} \;\; y_{k,\text{mid}}]^\top$, with $k \in \{1, \ldots, N\}$:

$$\begin{aligned} \mathbf{z}_{t,\text{mid}}^i &= \mathbf{h}_{\text{mid}}(\mathbf{x}_t, k, \mathbf{m}_{\text{mid}}) + \boldsymbol{\delta}_{t,\text{mid}} \\ \begin{bmatrix} \rho_{t,\text{mid}}^i \\ \alpha_{t,\text{mid}}^i \end{bmatrix} &= \begin{bmatrix} \sqrt{(x_{k,\text{mid}} - x_t)^2 + (y_{k,\text{mid}} - y_t)^2} \\ \text{arctan2}(y_{k,\text{mid}} - y_t, x_{k,\text{mid}} - x_t) - \theta_t \end{bmatrix} + \begin{bmatrix} \delta_{\sigma^2_{\rho_{\text{mid}}}} \\ \delta_{\sigma^2_{\alpha_{\text{mid}}}} \end{bmatrix} \end{aligned} \tag{6.16}$$

The Jacobian of this measurement model with respect to the pose and evaluated at the mean can be found in Equation 6.17:

[1] By "midpoint" we here refer to the average location of all the points in an extracted segment $\mathcal{S}_i$

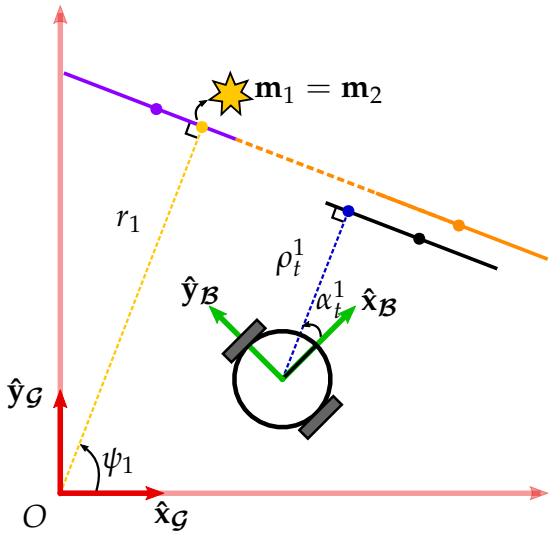


Figure 6.4: Illustration of the matching ambiguity arising due to landmark collinearity. An enhancement consists of utilizing the midpoint of the features and landmarks for data association

$$\begin{aligned}\mathbf{H}_{t,\text{mid}}^{k} &= \left.\frac{\partial \mathbf{h}_{\text{mid}}(\mathbf{x}_t, k, \mathbf{m}_{\text{mid}})}{\partial \mathbf{x}_t}\right|_{\overline{\boldsymbol{\mu}}_t,\, k,\, \mathbf{m}_{\text{mid}}} \\ &= \begin{bmatrix} -\frac{x_{k,\text{mid}}-\overline{\mu}_{t,x}}{\sqrt{(x_{k,\text{mid}}-\overline{\mu}_{t,x})^2+(y_{k,\text{mid}}-\overline{\mu}_{t,y})^2}} & -\frac{y_{k,\text{mid}}-\overline{\mu}_{t,y}}{\sqrt{(x_{k,\text{mid}}-\overline{\mu}_{t,x})^2+(y_{k,\text{mid}}-\overline{\mu}_{t,y})^2}} & 0 \\ \frac{y_{k,\text{mid}}-\overline{\mu}_{t,y}}{(x_{k,\text{mid}}-\overline{\mu}_{t,x})^2+(y_{k,\text{mid}}-\overline{\mu}_{t,y})^2} & -\frac{x_{k,\text{mid}}-\overline{\mu}_{t,x}}{(x_{k,\text{mid}}-\overline{\mu}_{t,x})^2+(y_{k,\text{mid}}-\overline{\mu}_{t,y})^2} & -1 \end{bmatrix}\end{aligned} \tag{6.17}$$

Let the rectangular coordinates of the midpoint of the i-th observed feature be $[\overline{x}^i \;\; \overline{y}^i]^\top$. This is obtained from the sensor measurements, on which Split-and-Merge is executed. The local polar coordinates representing $\mathbf{z}_{t,\text{mid}}^i$ are computed through Equation 6.18:

$$\begin{bmatrix}\rho_{t,\text{mid}}^i \\ \alpha_{t,\text{mid}}^i\end{bmatrix} = \begin{bmatrix}\sqrt{[\overline{y}^i]^2+[\overline{x}^i]^2} \\ \arctan2(\overline{y}^i, \overline{x}^i)\end{bmatrix} = \begin{bmatrix}v_1(x_1^i, y_1^i, \ldots, x_{n_i}^i, y_{n_i}^i) \\ v_2(x_1^i, y_1^i, \ldots, x_{n_i}^i, y_{n_i}^i)\end{bmatrix} \tag{6.18}$$

For the midpoint covariance which we label $\mathbf{R}_{t,\text{mid}}^i$, error propagation applied in the preceding relation leads to Equation 6.19:

$$\mathbf{R}_{t,\text{mid}}^i = \sum_{h=1}^{n_i} \boldsymbol{\mathcal{A}}_{h,\text{mid}}^i \boldsymbol{\mathcal{B}}_h^i \boldsymbol{\mathcal{D}}_h^i [\boldsymbol{\mathcal{A}}_{h,\text{mid}}^i \boldsymbol{\mathcal{B}}_h^i]^\top \tag{6.19}$$

with the Jacobian $\boldsymbol{\mathcal{A}}_{h,\text{mid}}^i$ shown in Equation 6.20:

$$\boldsymbol{\mathcal{A}}_{h,\text{mid}}^i = \left.\begin{bmatrix}\frac{\partial v_1}{\partial x_h^i} & \frac{\partial v_1}{\partial y_h^i} \\ \frac{\partial v_2}{\partial x_h^i} & \frac{\partial v_2}{\partial y_h^i}\end{bmatrix}\right|_{x_h^i,\, y_h^i} = \begin{bmatrix}\frac{\overline{x}^i}{n_i\sqrt{[\overline{y}^i]^2+[\overline{x}^i]^2}} & \frac{\overline{y}^i}{n_i\sqrt{[\overline{y}^i]^2+[\overline{x}^i]^2}} \\ -\frac{\overline{y}^i}{n_i([\overline{y}^i]^2+[\overline{x}^i]^2)} & \frac{\overline{x}^i}{n_i([\overline{y}^i]^2+[\overline{x}^i]^2)}\end{bmatrix} \tag{6.20}$$

The normalized distance between midpoints is represented by Equation 6.21:

$$D_{ik,\text{mid}} = (\mathbf{z}^i_{t,\text{mid}} - \hat{\mathbf{z}}^k_{t,\text{mid}})^\top [\mathbf{S}^k_{t,\text{mid}}]^{-1} (\mathbf{z}^i_{t,\text{mid}} - \hat{\mathbf{z}}^k_{t,\text{mid}}) + \ln(\det \mathbf{S}^k_{t,\text{mid}}) \tag{6.21}$$

in which one shall utilize the midpoint innovation covariance $\mathbf{S}^k_{t,\text{mid}}$, having the expression in Equation 6.22:

$$\mathbf{S}^k_{t,\text{mid}} = \mathbf{H}^k_{t,\text{mid}} \overline{\mathbf{\Sigma}}_t [\mathbf{H}^k_{t,\text{mid}}]^\top + \mathbf{R}^i_{t,\text{mid}} \tag{6.22}$$

The maximum likelihood data association policy from Subsection 6.3.1 can be modified to include the midpoints. Accordingly, the minimization for the nearest neighbor will be applied to the weighted sum $S_{w,ik}$ of D_{ik} and $D_{ik,\text{mid}}$, as indicated by Equation 6.23:

$$j(i) = \underset{k}{\text{argmin}} \; S_{w,ik} = \underset{k}{\text{argmin}} \; w_1 D_{ik} + w_2 D_{ik,\text{mid}} \tag{6.23}$$

6.4 EKF Localization Algorithm

Algorithm 2 illustrates the discussed EKF localization with unknown data correspondences. The map $\mathbf{m}$ consists of N stationary line landmarks, the extraction of features from sensory data is done according to Algorithm 1 and the estimation of line parameters is precisely carried out as specified in Section 5.3. The $\mathbf{z}_t$ in the prototype of the functions denotes the full measurement information the sensor supplies. Notice that some of the formulas appearing in the mathematical derivation are not restated in the algorithm.

6.5 Simulation of EKF Localization

In order to evaluate the correctness and the performance of the devised localization algorithm complemented with the midpoint tracking feature, a simple test bench environment that imitates real world characteristics such as imperfect motion, sensor noise and limited LiDAR vision has been developed in Python. With continuous testing and improvement of the Algorithm 2, coded in Python as well, done in this custom built environment, the code has been debugged and satisfactory results highlighting successful implementation and execution of the mentioned algorithm have been obtained. In spite of the fact that localization alone is not an application that we are going to implement and test in the real world, we regard it as a milestone achieving which will take us a step closer to solving the SLAM problem in the consecutive chapters. It is also important to mention that this simulation has served us as a tool to verify and improve other elements of the system, such as the feature extraction algorithm. Once more, the reader should note that this test bench has been developed for the sole purpose of debugging the code and endorsing the devised localization

Algorithm 2: EKF Localization with Unknown Correspondences

EKF_Localization$(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t, \mathbf{m})$:
Compute $\mathbf{G}_t, \mathbf{L}_t$ and $\mathbf{Q}_t$;
$\overline{\boldsymbol{\mu}}_t = \boldsymbol{\mu}_{t-1} + \begin{bmatrix} -\frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta}) + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta}) - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{bmatrix}$;
$\overline{\boldsymbol{\Sigma}}_t = \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^\top + \mathbf{L}_t \mathbf{Q}_t \mathbf{L}_t^\top$;
Extract all the visible line features $\mathbf{z}_t^i$ and compute the midpoints $\mathbf{z}_{t,\text{mid}}^i$;
Compute the line covariances $\mathbf{R}_t^i$ and the midpoint covariances $\mathbf{R}_{t,\text{mid}}^i$;
for *all extracted features* $\mathbf{z}_t^i = [\rho_t^i \;\; \alpha_t^i]^\top$, *from* $i = 1$ *to* $i = Z$ **do**
  for *all landmarks in* $\mathbf{m}$, *from* $k = 1$ *to* $k = N$ **do**
    Find the value of κ to decide the correct measurement model;
    $\hat{\mathbf{z}}_t^k = \begin{bmatrix} (-1)^\kappa(-r_k + \overline{\mu}_{t,x}\cos(\psi_k) + \overline{\mu}_{t,y}\sin(\psi_k)) \\ \psi_k - \overline{\mu}_{t,\theta} + (1-\kappa)\pi \end{bmatrix}$;
    $\mathbf{H}_t^k = \begin{bmatrix} (-1)^\kappa\cos(\psi_k) & (-1)^\kappa\sin(\psi_k) & 0 \\ 0 & 0 & -1 \end{bmatrix}$;
    $\mathbf{S}_t^k = \mathbf{H}_t^k \overline{\boldsymbol{\Sigma}}_t [\mathbf{H}_t^k]^\top + \mathbf{R}_t^i$;
    Follow a similar procedure for $\hat{\mathbf{z}}_{t,\text{mid}}^i$, $\mathbf{H}_{t,\text{mid}}^k$ and $\mathbf{S}_{t,\text{mid}}^k$;
    Compute M_{ik}. If $M_{ik} > \gamma_1$, discard M_{ik};
    Calculate $S_{w,ik}$;
  end
  Find $j(i)$, the correspondence of the nearest unmatched neighbor of $\mathbf{z}_t^i$;
  if $j(i)$ *exists* **then**
    $\mathbf{K}_t^i = \overline{\boldsymbol{\Sigma}}_t [\mathbf{H}_t^{j(i)}]^\top [\mathbf{S}_t^{j(i)}]^{-1}$;
    $\overline{\boldsymbol{\mu}}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t^i (\mathbf{z}_t^i - \hat{\mathbf{z}}_t^{j(i)})$;
    $\overline{\boldsymbol{\Sigma}}_t = (\mathbf{I} - \mathbf{K}_t^i \mathbf{H}_t^{j(i)}) \overline{\boldsymbol{\Sigma}}_t$;
  end
end
$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t$;
$\boldsymbol{\Sigma}_t = \overline{\boldsymbol{\Sigma}}_t$;
return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$;

algorithm before proceeding further towards tackling the SLAM problem. The results achieved have made sure that the localization algorithm is functional on its own and necessary amendments can be made to address the SLAM problem in later chapters.

6.5.1 Simulation Setup

Before demonstrating the achieved results, a subsection explaining the test bench setup has been considered appropriate to familiarize the reader with the robot and environment specifications that were simulated. The reader should note that this subsection is equally important and valid for the simulation of EKF SLAM in Section 7.6 and no other preamble will be given later again.

First and foremost, a 2D point cloud representation of an 13 m by 8 m indoor environment with nine identical 0.25 m^2 square objects consisting of 4680 points is digitally obtained. Considering the existing objects in the environment with each having four sides and the four walls of the room, the map consists of 40 line landmarks in total. This imaginary environment is shown in Figure 6.5. As one might think, the placement of the square objects in the figure is not arbitrary. In fact, they are purposefully positioned in the given configuration to introduce five pairs of collinear landmarks, thus giving us the opportunity to check if the midpoint tracking extension added to the localization algorithm to enhance the data association really augments the number of correct correspondences done during the execution of the program.

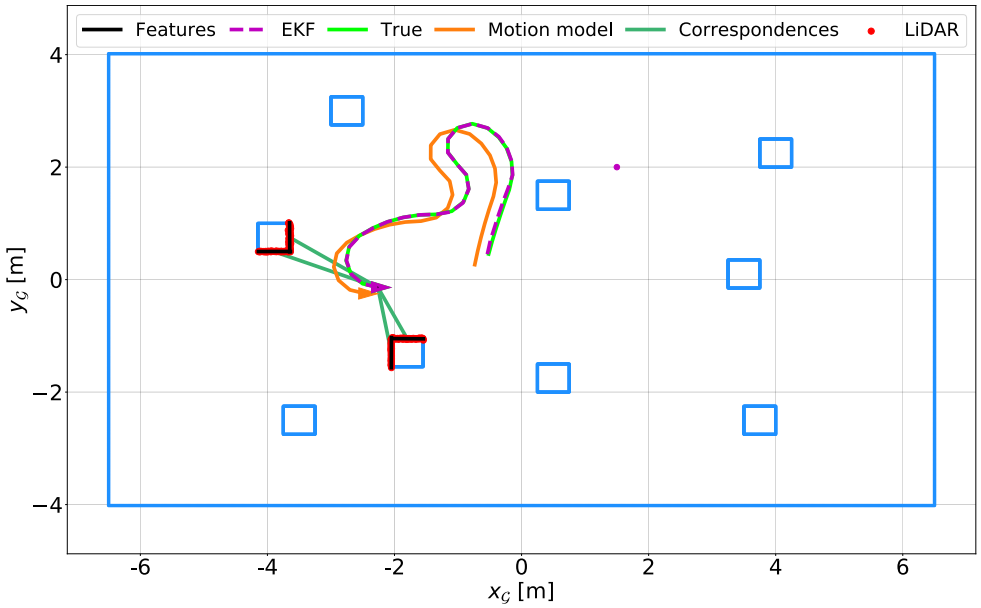


Figure 6.5: A Snapshot of The 2D Simulation Test Bench Environment Showing The Trajectory of The Robot With The Extracted Features And Acquired Correspondences

A closer look at Figure 6.5 reveals other important elements of the simulation as well. One of the first aspects one notices at the first glance is the three robot trajectories plotted in different colors and line styles. The solid orange path is based on the motion model given in Section 5.1 and is calculated with the assumption that the given control input $\mathbf{u}$ at each iteration is executed perfectly by the robot. It is an idealistic assumption and thus obviously a wrong way to estimate the robot trajectory, therefore the discrepancy with the other two routes. It is still included in the simulation to illustrate the necessity of EKF in accurate pose estimation.

The solid line in lime color is the true trajectory that the robot follows. It is again computed using the motion model, but this time with added noise on the control inputs, simulating imperfect motion execution of a real agent. Even though the difference between the true and motion model trajectories is indistinguishable when the simulation first starts, they diverge from each other more and more as a result of the continuous noise integration at each time step. Regarding the added noise on linear and angular velocity commands, they were randomly picked from two different Gaussian distributions about zero with the first one having 0.0125 and the other 0.01 standard deviation. The added noise characteristics were thoughtfully determined with the aim of simulating a realistic robot motion. Considering the 1Hz update rate of the simulation (a time period long enough for EKF localization algorithm to complete one iteration) that points out execution of the given velocity commands for one second, it is plausible to assume that the motion of a vehicle in the real world might be 5 cm off of the actual position in x and y axis and 0.05 rad (3°) off of the actual heading for velocity commands of 0.25 m/s and 0.5 rad/s (29°/s). This level of inaccuracy in motion is equal to 20% error. The previously mentioned noise characteristics also model similar levels of uncertainty for our fictitious robot. They reflect the possibility of it actually being within a 6 cm circular range of 5σ-bound for the mentioned linear velocity command and within the 0.05 rad (2.9°) range of 5σ bound for the said angular velocity command. The additional angular-velocity-like noisy parameter $\hat{\gamma}_t$, that is included in $\mathbf{u}_t$ as a third element to compensate for the constrained final heading of the robot as a consequence of the circular motion, is picked from a random distribution with 0.005σ around zero mean. These noise characteristics were ultimately utilised in the simulation.

Considering the fact that we know the level of noise on the control inputs (since we put it ourselves in the simulation), there was no need to tune the robot-specific error parameters $\alpha_i,\ i \in \{1, \ldots, 6\}$ given in Equation 5.3. Furthermore, it was tried to increase the noise magnitude more to obtain a larger discrepancy between the true and ideal motion trajectories in order to obtain a less accurate state transition belief through the deterministic motion model and to test the performance of the EKF localization algorithm under such conditions. However, this was rendered impossible as more noise added on $\mathbf{u}$ have led to a more unpredictable motion of the agent that was no longer constrained within the dimensions of the custom built environment since there was no measure in the simulation to not let it go through the square objects or the outer walls.

Last but not least, the dashed line colored in magenta depicts the trajectory estimated by EKF localization algorithm. The current pose of the vehicle for all three trajectories are illustrated via three triangles in their respective colors with each pointy tip indicating the heading of the robot. The position covariance ellipse of the robot limited to 5σ bound has also been drawn on top of the most recent EKF-estimated pose location at each iteration. Even though the error ellipse is not visible in Figure 6.5 due to very small uncertainty in position, its evolution during the simulation time will be illustrated more clearly later when we discuss the achieved results. Fi-

nally, the Cartesian coordinates of the robot's initial position in the global frame are kept visible at all times during the simulation with a magenta point placed at (1.5,2).

The LiDAR measurements obtained in the local frame of the mobile vehicle are restricted within a circle with radius of 2.25 m for the reason that it would not be appropriate to assume the same covariance matrix $\boldsymbol{\mathcal{D}}$, that was obtained in Section 5.3 by gathering data for a point at a specific range, for scans further away than 2.25m. Doing so would mean assuming a more optimistic measurement covariance matrix that does not reflect the real nature of the LiDAR sensor for larger measurement distances. The same argument can also be made for readings of points at a closer proximity than 2.25 m. However, for the mentioned short range of 0 to 2.25 m, the LiDAR exhibits a similar noise distribution behaviour and thus using the same covariance matrix was thought appropriate for all measurements in this range. The covariance $\mathbf{R}_t^i$ of a certain feature $\mathbf{z}_t^i$ was then continuously computed based on Equation 5.22 as the localization algorithm was executed iteratively. In Figure 6.5, the restricted LiDAR measurements are shown as a point cloud of red dots around the current location of the robot. It is also crucial to not get any LiDAR scans behind any object in the environment even though they are within the 2.25 m zone. The reason is that seeing through nontransparent items would be physically implausible in a real scenario. This mentioned feature of not getting unreasonable measurements through square objects is implemented in the simulation.

With regards to the feature extraction technique given in Algorithm 1, certain parameters needed to be tuned in accordance with the custom built environment characteristics in order to enhance its performance. First, the threshold for splitting and merging line segments is adjusted based on the LiDAR measurement noise. Second, extracted features of very short length and few points are discarded since they are found to be unreliable. Lastly, spurious line segments that are unexpectedly lengthy for the number of points that they possess are deleted since these are understood to be bogus. Figure 6.5 illustrates the extracted features as solid black lines on top of the red point cloud from the LiDAR sensor. Later in the algorithm after having found the correspondence for a line feature, if any, a dark green line between the current robot pose and the feature for which an association has been found is drawn for clear visualization of the correspondences obtained at each iteration.

As it is already clear from the previous chapters, the localization Algorithm 2 assumes a known landmark map of the environment and works towards estimating the true position of the mobile robot in it based on the motion model and the correspondences made between extracted line features and the landmarks. Therefore, a point landmark representation of the environment as discussed in Subsection 5.2.1 should be provided to the localization algorithm for it to successfully run. In Figure 6.6, the reader can see the point landmark map of the blueprint given in Figure 6.5. Point landmarks plotted in green mark the collinear ones while the others are painted in black and red for easy interpretation of the figure. An obvious observation about this map is the horizontal and vertical alignment of the point landmarks. This is due to the fact that all line beacons found in the custom built environment (the sides of nine

square objects and the four walls of the room) are at either 0 rad, $\pm\pi/2$ rad or π with respect to x_G. The close proximity of adjacent landmarks and existing collinear beacons will be a challenge in correct data association later.

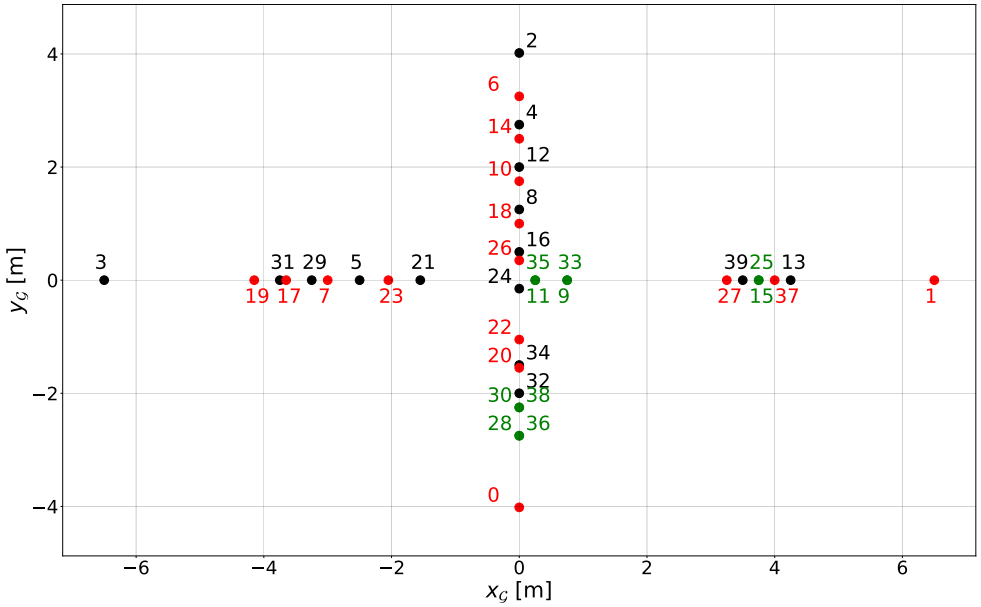


Figure 6.6: Point landmark map of the custom built environment

6.5.2 Simulation Results

A simulation of Algorithm 2 (this includes the midpoints) was conducted in the developed Python test bench environment for 220 iterations to evaluate the performance of EKF localization with unknown correspondences. The fictitious robot is placed in the previously mentioned starting position in the environment with the knowledge of the point landmarks' global coordinates.

The performance of the EKF in terms of localization is first assessed by observing the three trajectories: the true (noisy motion of the robot), the ideal motion (not disrupted by any noise - solely based on the motion model) and the Kalman estimation. Figure 6.7 shows these paths iteratively generated by the algorithm. There is a plain resemblance between the estimated trajectory (dashed magenta line) and the route supposed to imitate the real motion of the agent (solid lime line). A significant dissimilitude is noted due to the lack of noise when looking at the trajectory based on the motion model, an observation that proclaims the need for a stochastic observer capable of handling a noisy process such as the motion of our fictitious vehicle.

The performance of position tracking can be further evaluated by plotting the observed error between the estimated and the real pose during the simulation. In Figure 6.8, the absolute errors in x_t, y_t and θ_t are accompanied by their respective 5σ-bounds. It is observed that, most of the times, the error lies within the defined 5σ limit while occasionally exceeding it. Furthermore, the position error in x_t and y_t

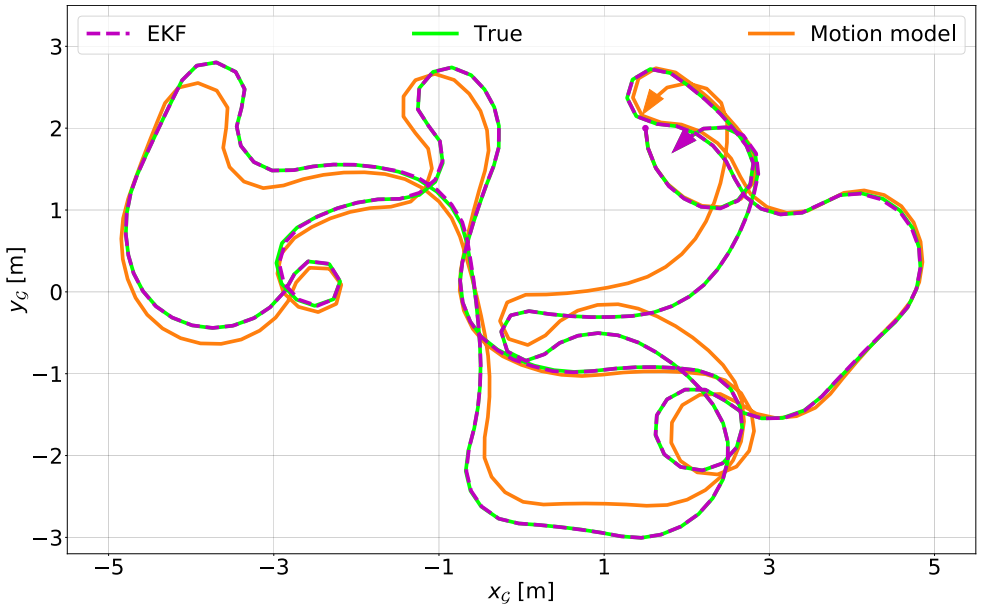


Figure 6.7: True, estimated and noise-free trajectories resulted from the localization simulation

is for the most part smaller than 1 cm, while the difference in heading only briefly surpasses 0.025 rad, with the spike around iteration 74 occurring as a result of the angular nonliterary at π and $-\pi$.

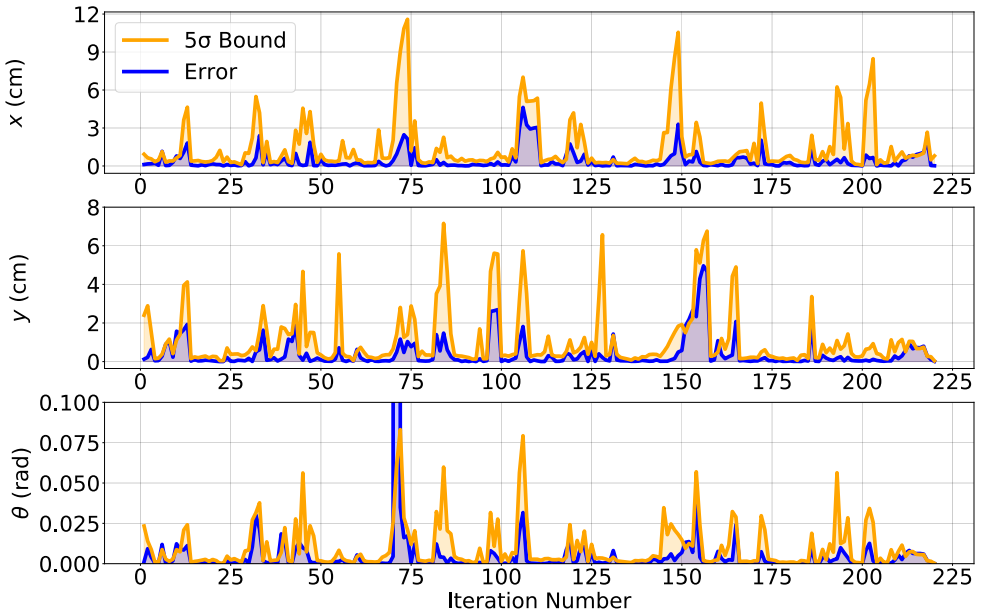


Figure 6.8: Absolute error between EKF and true trajectories and the covariance bound during the localization simulation

Another important performance indicator worth investigating is the deviation between the true trajectory and the noiseless path computed by the deterministic motion model. Figure 6.9 plots the evolution of the absolute error between the two poses. The first three plots correspond to the elements of the pose vector, while the fourth one

shows the Euclidean distance between the positions at each time step. Easily distinctive errors are accumulated by the end of the simulation, reaching peaks of 46 cm, 63 cm, 0.19 rad and 75 cm for x_t, y_t, θ_t, and Euclidean distance, respectively. The nonlinearity in angle at π and $-\pi$ causes five high peaks in the plot of θ_t in this case as well. The accumulation of errors emphasizes once again the importance of EKF.

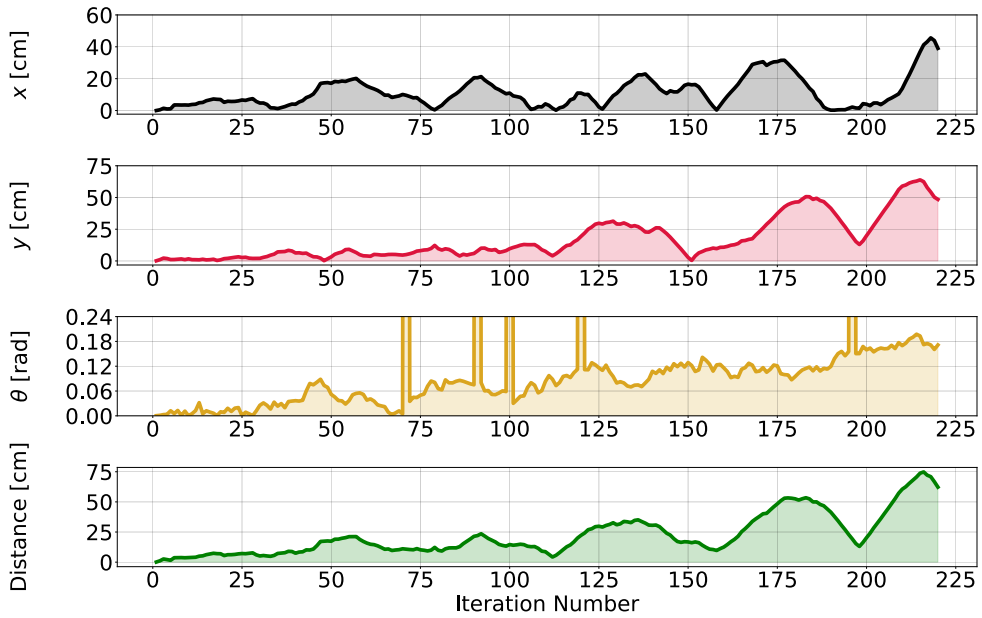


Figure 6.9: Absolute error between the true and ideal trajectories during the localization simulation

Figure 6.10 illustrates the full history of the robot's estimated pose while also showing the 20σ uncertainties in position estimated by the EKF as yellow ellipses.

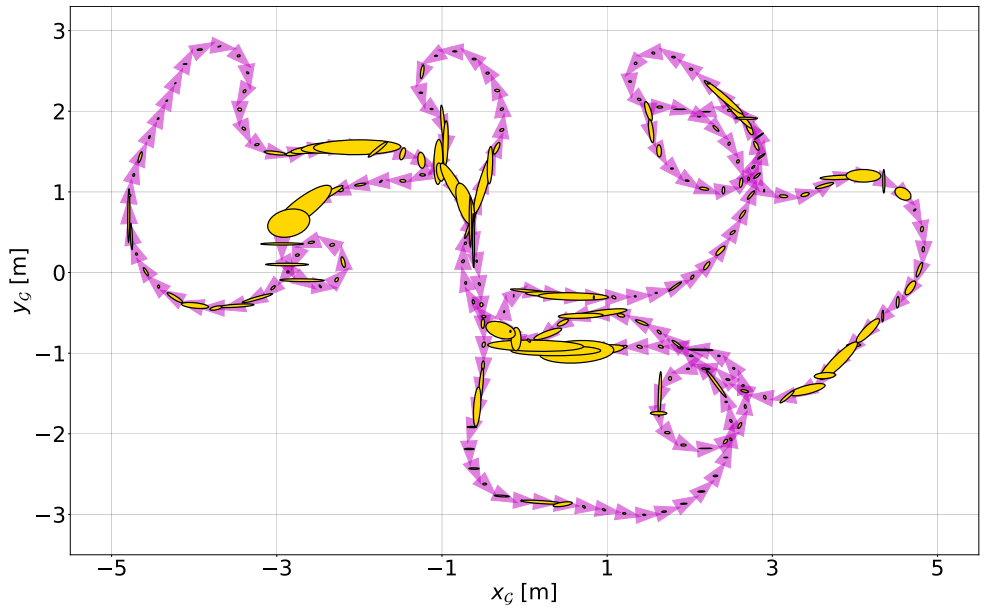


Figure 6.10: Position and pose uncertainty history resulted from the localization simulation

First of all, the calculated uncertainty in pose is very small since all the landmarks are known and the initial position belief of EKF is same as the true position of the mobile robot as a requirement of position tracking nature of the EKF localization algorithm. That is also why 20σ uncertainty ellipses were drawn instead of 5σ. At the beginning of the simulation, the robot is very sure about its whereabouts, but as it traverses some distance the area of the uncertainty ellipse grows as a consequence of the noise in the control inputs. On the other hand, observing a landmark leads to a decrease in the pose uncertainty.

Figure 6.11 plots yet another significant performance measure: the accumulated number of correct and wrong correspondences found between the extracted features and the known point landmarks in the map for every ten iterations. It clearly illustrates the level of improvement the midpoint tracking extension brings to the EKF localization algorithm. While the average of found correspondences for all extracted features is a little bit above 50% for both versions, midpoint tracking data association enhancement prevents the algorithm from detecting the fifty wrong correspondences its counterpart commits.

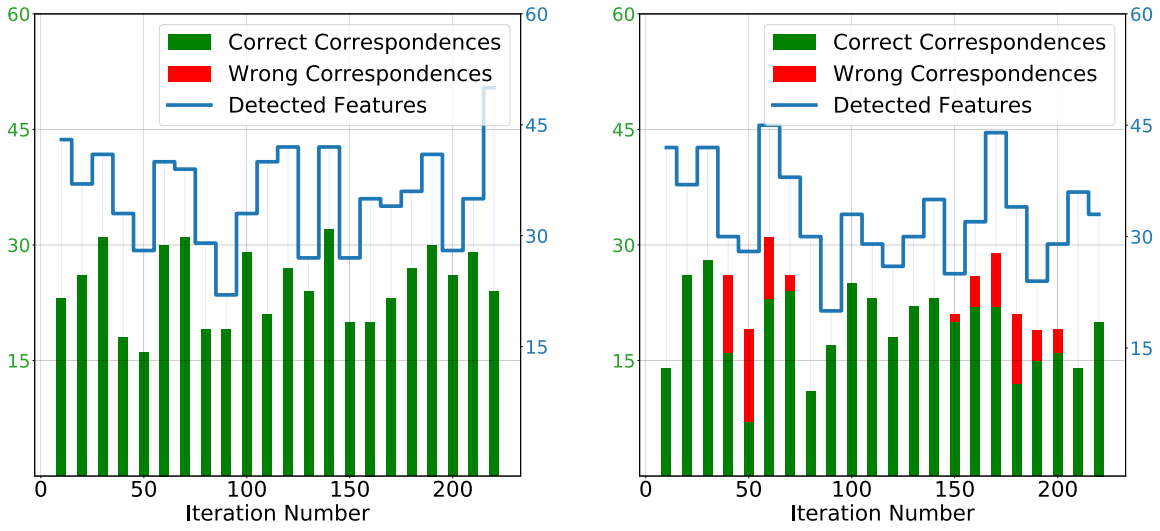


Figure 6.11: *Left:* Number of correct and wrong correspondences for the detected features with midpoint data association enhancement; *Right:* Without midpoint data association enhancement

In light of the obtained results, the simulation of EKF localization with unknown correspondences is considered successful, as the mobile robot maintained its true trajectory and is able to keep its uncertainty small. The gif file of the Python localization simulation can be watched by accessing `https://bit.ly/3gOBSNr`.

Chapter 7

Simultaneous Localization and Mapping

This chapter introduces the famous SLAM problem from a mathematical standpoint, building upon the information presented in all the foregoing chapters. In the sections that follow, an algorithm relying on the Extended Kalman Filter meant to solve the SLAM problem for the two-dimensional case is delineated. The solution put forward addresses the scenario in which a robot equipped with a rotating LiDAR sensor is operating in an indoor environment and builds a map consisting of line features. Without a shadow of a doubt, this chapter represents the culmination of the present thesis.

7.1 Definition of the SLAM Problem

The particular SLAM form of greatest practical importance is, in the opinion of the authors of [7], the online SLAM problem, which concerns the estimation of the posterior over the current pose and the map (until the present moment in time). What is given is represented by all the present and previous controls and landmark observations. In reality, the objective includes the identification of the correspondences $\mathbf{c}_{t+1}$, in which case the PDF of the SLAM posterior becomes $p(\mathbf{x}_{t+1}, \mathbf{m}, \mathbf{c}_{t+1} \mid \mathbf{u}_{1:t+1}, \mathbf{z}_{1:t+1}, \mathbf{c}_{1:t})$. Figure 7.1 illustrates through a graph the goal of online SLAM, as well as the dependencies and influences that one needs to be aware of in pursuing the solution.

7.2 Mathematical Derivation

First and foremost, since both the current pose $\mathbf{x}_t$ and the map $\mathbf{m}$ need to be estimated in simultaneous localization and mapping, it is convenient to define a combined state vector $\mathbf{y}_t$ that covers all the random variables we are aiming to approximate. Equation 7.1 gives the variables forming this new state vector:

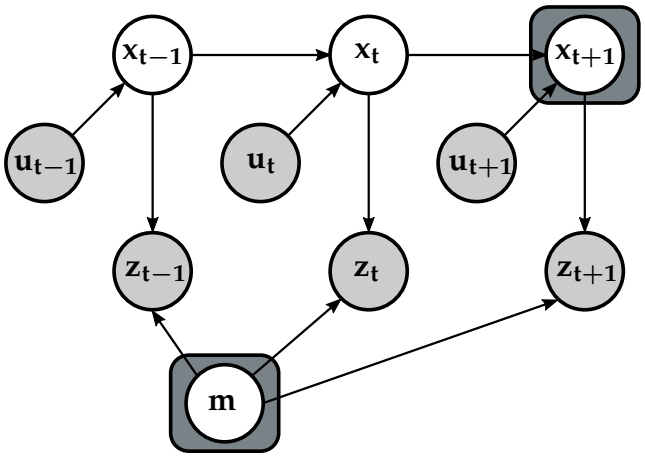


Figure 7.1: Graphical model of the online SLAM problem

$$\mathbf{y}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} x_t & y_t & \theta_t & r_1 & \psi_1 & \dots & r_N & \psi_N \end{bmatrix}^\top \in \mathbb{R}^{2N+3} \tag{7.1}$$

where the assumption is that there are N landmarks in the map. We also let the mean of the state vector be denoted by $\boldsymbol{\mu}_t$, with the constituent components shown in Equation 7.2:

$$\boldsymbol{\mu}_t = \begin{bmatrix} \boldsymbol{\mu}_{t,\mathbf{x}} \\ \boldsymbol{\mu}_{t,\mathbf{m}} \end{bmatrix} = \begin{bmatrix} \mu_{t,x} & \mu_{t,y} & \mu_{t,\theta} & \mu_{r,1} & \mu_{\psi,1} & \dots & \mu_{r,N} & \mu_{\psi,N} \end{bmatrix}^\top \tag{7.2}$$

Augmenting the state vector has inherent ramifications vis-á-vis the dimensions and contents of the state covariance matrix $\boldsymbol{\Sigma}_t$. With N landmarks in the map, the number of dimensions is now $(2N+3) \times (2N+3)$, and the matrix has the variances of state variables on the main diagonal and the covariances between those as off-diagonal entries (see Equation 7.3).

$$\boldsymbol{\Sigma}_t = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y & \sigma_x\sigma_\theta & \sigma_x\sigma_{m_{r,1}} & \sigma_x\sigma_{m_{\psi,1}} & \dots & \sigma_x\sigma_{m_{r,N}} & \sigma_x\sigma_{m_{\psi,N}} \\ \sigma_y\sigma_x & \sigma_y^2 & \sigma_y\sigma_\theta & \sigma_y\sigma_{m_{r,1}} & \sigma_y\sigma_{m_{\psi,1}} & \dots & \sigma_y\sigma_{m_{r,N}} & \sigma_y\sigma_{m_{\psi,N}} \\ \sigma_\theta\sigma_x & \sigma_\theta\sigma_y & \sigma_\theta^2 & \sigma_\theta\sigma_{m_{r,1}} & \sigma_\theta\sigma_{m_{\psi,1}} & \dots & \sigma_\theta\sigma_{m_{r,N}} & \sigma_\theta\sigma_{m_{\psi,N}} \\ \sigma_{m_{r,1}}\sigma_x & \sigma_{m_{r,1}}\sigma_y & \sigma_{m_{r,1}}\sigma_\theta & \sigma_{m_{r,1}}^2 & \sigma_{m_{r,1}}\sigma_{m_{\psi,1}} & \dots & \sigma_{m_{r,1}}\sigma_{m_{r,N}} & \sigma_{m_{r,1}}\sigma_{m_{\psi,N}} \\ \sigma_{m_{\psi,1}}\sigma_x & \sigma_{m_{\psi,1}}\sigma_y & \sigma_{m_{\psi,1}}\sigma_\theta & \sigma_{m_{\psi,1}}\sigma_{m_{r,1}} & \sigma_{m_{\psi,1}}^2 & \dots & \sigma_{m_{\psi,1}}\sigma_{m_{r,N}} & \sigma_{m_{\psi,1}}\sigma_{m_{\psi,N}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{m_{r,N}}\sigma_x & \sigma_{m_{r,N}}\sigma_y & \sigma_{m_{r,N}}\sigma_\theta & \sigma_{m_{r,N}}\sigma_{m_{r,1}} & \sigma_{m_{r,N}}\sigma_{m_{\psi,1}} & \dots & \sigma_{m_{r,N}}^2 & \sigma_{m_{r,N}}\sigma_{m_{\psi,N}} \\ \sigma_{m_{\psi,N}}\sigma_x & \sigma_{m_{\psi,N}}\sigma_y & \sigma_{m_{\psi,N}}\sigma_\theta & \sigma_{m_{\psi,N}}\sigma_{m_{r,1}} & \sigma_{m_{\psi,N}}\sigma_{m_{\psi,1}} & \dots & \sigma_{m_{\psi,N}}\sigma_{m_{r,N}} & \sigma_{m_{\psi,N}}^2 \end{bmatrix} \tag{7.3}$$

As mentioned in [15, 86], the preceding matrix can be thought of as comprising four smaller matrix blocks, as observed from Equation 7.4:

$$\boldsymbol{\Sigma}_t = \begin{bmatrix} \boldsymbol{\Sigma}_{t,\mathbf{xx}} & \boldsymbol{\Sigma}_{t,\mathbf{xm}} \\ \boldsymbol{\Sigma}_{t,\mathbf{mx}} & \boldsymbol{\Sigma}_{t,\mathbf{mm}} \end{bmatrix} \tag{7.4}$$

This fact requires further clarifications. Since $\mathbf{y}_t$ is a result of the concatenation of the pose and map random vectors, one can divide the covariances extant between these constituent elements into four groups. Thus, we have one submatrix for the covariances between: 1) the pose vector and itself ($\mathbf{\Sigma}_{t,\mathbf{xx}}$), 2) the pose vector and the map ($\mathbf{\Sigma}_{t,\mathbf{xm}}$), 3) the map and the pose vector ($\mathbf{\Sigma}_{t,\mathbf{mx}}$), 4) the map and itself ($\mathbf{\Sigma}_{t,\mathbf{mm}}$). Furthermore, according to the basic property of symmetry of the covariance matrix, it follows that $\mathbf{\Sigma}_{t,\mathbf{mx}} = \mathbf{\Sigma}_{t,\mathbf{xm}}^{\top}$ [86]. The four covariance blocks and the state mean are also illustrated graphically in Figure 7.2:

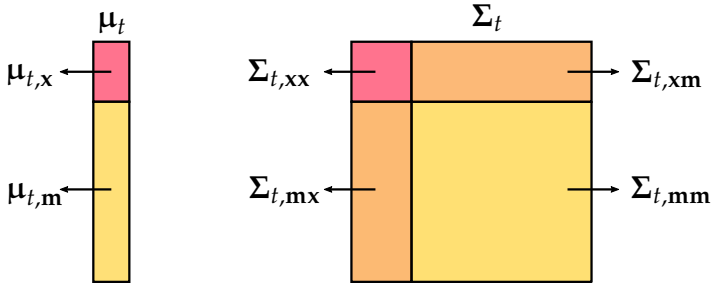


Figure 7.2: State mean and state covariance in EKF SLAM

From the previous equations and, it is of course clear that one can compartment the map covariance matrix based on the statistical relationships between landmarks: variances on the main diagonal and covariances arranged symmetrically over and below the diagonal, as done in [95] and as Equation 7.5 indicates:

$$\mathbf{\Sigma}_{t,\mathbf{mm}} = \begin{bmatrix} \mathbf{\Sigma}_{t,\mathbf{m}_1\mathbf{m}_1} & \mathbf{\Sigma}_{t,\mathbf{m}_1\mathbf{m}_2} & \dots & \mathbf{\Sigma}_{t,\mathbf{m}_1\mathbf{m}_N} \\ \mathbf{\Sigma}_{t,\mathbf{m}_2\mathbf{m}_1} & \mathbf{\Sigma}_{t,\mathbf{m}_2\mathbf{m}_2} & \dots & \mathbf{\Sigma}_{t,\mathbf{m}_2\mathbf{m}_N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Sigma}_{t,\mathbf{m}_N\mathbf{m}_1} & \mathbf{\Sigma}_{t,\mathbf{m}_N\mathbf{m}_2} & \dots & \mathbf{\Sigma}_{t,\mathbf{m}_N\mathbf{m}_N} \end{bmatrix} \tag{7.5}$$

An interesting property of EKF SLAM is that during the filter's covariance update step, the total state uncertainty cannot increase. Even more powerful result was published by the authors of [96], which proved that the uncertainty in any given landmark's position decreases monotonically with each new reobservation, but also that the preceding properties are in fact true for any submatrix of $\mathbf{\Sigma}_{t,\mathbf{mm}}$. A measure of uncertainty is the determinant and we can thus write, for the particular case of the j-th diagonal entry in Equation 7.5, the relation given in Equation 7.6:

$$\det(\Sigma_{t,\mathbf{m}_j\mathbf{m}_j}) \leq \det(\Sigma_{t-1,\mathbf{m}_j\mathbf{m}_j}) \tag{7.6}$$

Given the notation adopted in this section, one can redefine the SLAM problem as the calculation of the posterior $p(\mathbf{y}_t \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t})$, fully characterized by the mean $\boldsymbol{\mu}_t$ and covariance $\mathbf{\Sigma}_t$ [7]. The rest of the section describes the mathematical details behind the EKF SLAM algorithm. In showing the steps of the derivation, we shall proceed in a similar fashion to the localization case presented in Chapter 6, by discriminating between the two stages of the extended Kalman filter.

Prediction Step

Using the extended state notation, the velocity motion model of Equation 5.4 takes the form in Equation 7.7 (we keep the notation $\theta = \mu_{t-1,\theta}$ throughout this chapter as well). An important observation that should be made here is that the map is not modified at this juncture. Only the elements of $\mathbf{x}_t$ are affected as a result of motion, while a change in the knowledge about the environment inherently depends on sensing, which is dealt with in the correction step. As a corollary, the present state vector $\mathbf{y}_t$ and the past one, $\mathbf{y}_{t-1}$, have the same size. The observation extends to the state covariance matrices.

$$\begin{aligned} \mathbf{y}_t &= \mathbf{g}(\mathbf{y}_{t-1}, \mathbf{u}_t, \boldsymbol{\varepsilon}_t) \\ \mathbf{y}_t &= \mathbf{y}_{t-1} + \boldsymbol{\Lambda}_t^\top \begin{bmatrix} -\frac{\hat{v}_t}{\hat{\omega}_t}\sin(\theta) + \frac{\hat{v}_t}{\hat{\omega}_t}\sin(\theta + \hat{\omega}_t\Delta t) \\ \frac{\hat{v}_t}{\hat{\omega}_t}\cos(\theta) - \frac{\hat{v}_t}{\hat{\omega}_t}\cos(\theta + \hat{\omega}_t\Delta t) \\ \hat{\omega}_t\Delta t + \hat{\gamma}_t\Delta t \end{bmatrix} \end{aligned} \tag{7.7}$$

Here, $\boldsymbol{\Lambda}_t^\top$ is a matrix converting a 3×1 vector into an $(2N+3) \times 1$ vector, introduced for the purpose of preserving a shortness in the notation. Then, $\boldsymbol{\Lambda}_t$ must have the form in Equation 7.8:

$$\boldsymbol{\Lambda}_t = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 2N} \end{bmatrix} \tag{7.8}$$

As expected, appending the map to the pose vector does not affect the non-additive nature of the motion noise. Hence, the linearization Equation 6.1 remains valid and necessary, and so do the modified Kalman prediction equations from Equations 6.2 and 6.3, as long as one replaces $\mathbf{x}_t$ by $\mathbf{y}_t$. The differences that emerge with the use of a larger state vector in SLAM are related to the computation of the Jacobians $\mathbf{G}_t$ and $\mathbf{L}_t$. For the former, its evaluation at the means of the state, control and noise is given in Equation 7.9:

$$\begin{aligned} \mathbf{G}_t &= \left.\frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{y}_{t-1}, \boldsymbol{\varepsilon}_t)}{\partial \mathbf{y}_{t-1}}\right|_{\mathbf{u}_t,\, \boldsymbol{\mu}_{t-1},\, \mathbf{0}} \\ &= \mathbf{I}_{(2N+3)\times(2N+3)} + \boldsymbol{\Lambda}_t^\top \begin{bmatrix} 0 & 0 & \frac{v_t}{\omega_t}(-\cos(\theta) + \cos(\theta + \omega_t\Delta t)) & \mathbf{0}_{1\times 2N} \\ 0 & 0 & \frac{v_t}{\omega_t}(-\sin(\theta) + \sin(\theta + \omega_t\Delta t)) & \mathbf{0}_{1\times 2N} \\ 0 & 0 & 0 & \mathbf{0}_{1\times 2N} \end{bmatrix} \\ &= \mathbf{I}_{(2N+3)\times(2N+3)} + \boldsymbol{\Lambda}_t^\top \mathbf{G}_{t,\text{low}} \boldsymbol{\Lambda}_t \end{aligned} \tag{7.9}$$

where we utilize the notation $\mathbf{G}_{t,\text{low}}$ for the lower-dimensional matrix in the above equation. For the Jacobian with respect to the process noise $\boldsymbol{\varepsilon}_t$, its SLAM form is

shown given in Equation 7.10 and it can be computed from $\mathbf{L}_{t,\mathbf{x}}$, denoting here the 3×3 Jacobian of Equation 6.7.

$$\begin{aligned}
\mathbf{L}_t &= \left.\frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{y}_{t-1}, \boldsymbol{\varepsilon}_t)}{\partial \boldsymbol{\varepsilon}_t}\right|_{\mathbf{u}_t,\, \boldsymbol{\mu}_{t-1},\, \mathbf{0}} \\
&= \boldsymbol{\Lambda}_t^{\top} \begin{bmatrix} \frac{-\sin(\theta)+\sin(\theta+\omega_t \Delta t)}{\omega_t} & \frac{v_t(\sin(\theta)-\sin(\theta+\omega_t \Delta t))}{\omega_t^2} + \frac{v_t \Delta t \cos(\theta+\omega_t \Delta t)}{\omega_t} & 0 \\ \frac{\cos(\theta)-\cos(\theta+\omega_t \Delta t)}{\omega_t} & -\frac{v_t(\cos(\theta)-\cos(\theta+\omega_t \Delta t))}{\omega_t^2} + \frac{v_t \Delta t \sin(\theta+\omega_t \Delta t)}{\omega_t} & 0 \\ 0 & \Delta t & \Delta t \end{bmatrix} \\
&= \boldsymbol{\Lambda}_t^{\top} \mathbf{L}_{t,\mathbf{x}}
\end{aligned} \tag{7.10}$$

From Equation 7.10, and given the control-space process covariance matrix $\mathbf{Q}_t$ of Equation 6.6, the mapped covariance to the state space is obtained.

Update Step

Maintaining the measurement conditional independence assumption (Equation 6.8) allows for the incremental implementation of the Kalman update step. Therefore, we shall only analyze this second filtering stage for a single feature $\mathbf{z}_t^i$, using the measurement model for line landmarks from Equation 5.9, in which we only need to replace $\mathbf{x}_t$ by $\mathbf{y}_t$. Linearizing now the nonlinear function $\mathbf{h}(\mathbf{y}_t, j, \kappa)$ according to Equation 4.18, it is obvious that its Jacobian with respect to the extended state vector is needed. We show the contents of this matrix in Equation 7.11:

$$\mathbf{H}_t^i = \left.\frac{\partial \mathbf{h}(\mathbf{y}_t, j, \kappa)}{\partial \mathbf{y}_t}\right|_{\overline{\boldsymbol{\mu}}_t,\, j} = \begin{bmatrix} \mathbf{H}_{t,\mathbf{x}}^i & \mathbf{0}_{2\times(2j-2)} & \mathbf{H}_{t,\mathbf{m}}^i & \mathbf{0}_{2\times(2N-2j)} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{t,\mathbf{x}}^i & \mathbf{H}_{t,\mathbf{m}}^i \end{bmatrix} \boldsymbol{\Lambda}_{t,j} \tag{7.11}$$

Due to space limitations, we wrote $\mathbf{H}_t^i$ as the concatenation of the matrices $\mathbf{H}_{t,\mathbf{x}}^i$ and $\mathbf{H}_{t,\mathbf{m}}^i$, multiplied with a third matrix $\boldsymbol{\Lambda}_{t,j}$. The former is simply the Jacobian computed during the mathematical derivation of the EKF localization (see Equation 6.9), in which the mean of the location of the j-th landmark takes the place of the precise coordinates assumed in the previous chapter. The second is also a Jacobian matrix, but it corresponds to the partial derivatives of $\mathbf{h}$ taken with respect to the j-th landmark $\mathbf{m}_j$ and evaluated at the mean, as in Equation 7.12:

$$\begin{aligned}
\mathbf{H}_{t,\mathbf{m}}^i &= \left.\frac{\partial \mathbf{h}(\mathbf{y}_t, j, \kappa)}{\partial \mathbf{m}_j}\right|_{\overline{\boldsymbol{\mu}}_t,\, j} \\
&= \begin{bmatrix} (-1)^{\kappa+1} & (-1)^{\kappa}(-\overline{\mu}_{t,x} \sin(\overline{\mu}_{j,\psi}) + \overline{\mu}_{t,y} \cos(\overline{\mu}_{j,\psi})) \\ 0 & 1 \end{bmatrix}
\end{aligned} \tag{7.12}$$

Regarding $\mathbf{\Lambda}_{t,j}$, it represents a matrix meant to convert the 2×5 matrix $\begin{bmatrix} \mathbf{H}^i_{t,\mathbf{x}} & \mathbf{H}^i_{t,\mathbf{m}} \end{bmatrix}$ into the $2 \times (2N+3)$ Jacobian $\mathbf{H}^i_t$. Its form is shown in Equation 7.13, which follows from Equation 7.11:

$$\mathbf{\Lambda}_{t,j} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times(2j-2)} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times(2N-2j)} \\ \mathbf{0}_{2\times 3} & \mathbf{0}_{2\times(2j-2)} & \mathbf{I}_{2\times 2} & \mathbf{0}_{2\times(2N-2j)} \end{bmatrix} \tag{7.13}$$

It should be also specified that the filter is implemented sequentially, on a feature-by-feature basis. For each feature, the algorithm loops through all the landmarks by incrementing an index k. We maintain the same notations for the estimated measurement and the innovation covariance: $\hat{\mathbf{z}}^k_t$ and $\mathbf{S}^k_t$, respectively. Provided that maximum likelihood correspondence occurs at $k = j(i)$ the standard Kalman correction should happen as specified in Algorithm 2.

Efficient Implementation

As it was surely observed from the above equations, in both filtering steps, the matrices involved in the calculations are sparse. In practice, it is not necessary, nor advisable, to use the matrices $\mathbf{\Lambda}_t$ and $\mathbf{\Lambda}_{t,j}$ in computing many of the Kalman filter variables from their standard formulas. According to [90, 97], there are more efficient ways to reduce the computational cost of these operations, by exploiting the relationships between the random variables in the state vector, as well as their arrangement.

For the prediction step, the map is invariant when the robot changes its position and thus $\mathbf{\Sigma}_{t,\mathbf{mm}}$ does not change. One can simply integrate motion into the filter as shown in Equations 7.14-7.17:

$$\mu_{t,\mathbf{x}} = \mathbf{g}_{\mathbf{x}}(\mu_{t,\mathbf{x}}, \mathbf{u}, \mathbf{0}) \tag{7.14}$$

$$\overline{\mathbf{\Sigma}}_{t,\mathbf{xx}} = \mathbf{G}_{t,\mathbf{x}} \mathbf{\Sigma}_{t-1,\mathbf{xx}} \mathbf{G}^\top_{t,\mathbf{x}} + \mathbf{L}_{t,\mathbf{x}} \mathbf{Q}_t \mathbf{L}^\top_{t,\mathbf{x}} \tag{7.15}$$

$$\overline{\mathbf{\Sigma}}_{t,\mathbf{xm}} = \mathbf{G}_{t,\mathbf{x}} \mathbf{\Sigma}_{t-1,\mathbf{xm}} \tag{7.16}$$

$$\overline{\mathbf{\Sigma}}_{t,\mathbf{mx}} = \overline{\mathbf{\Sigma}}^\top_{t-1,\mathbf{xm}} \tag{7.17}$$

where $\mathbf{G}_{t,\mathbf{x}}$ is the Jacobian of the process function with respect to the pose only (as in Equation 6.4) and $\mathbf{g}_{\mathbf{x}}$ denotes the motion model for the case when the state is $\mathbf{x}_t$.

Moving to the update stage, the innovation covariance concerns only the pose of the robot, the observation noise and the landmark which the extracted feature has been matched with. The other landmarks are therefore not involved in its calculation. However, the Kalman gain is not sparse due to the fact that gaining knowledge about a landmark should not only influence the belief about the robot's position and heading, but also the estimation of the other map elements. In view of these observations, the

SLAM-specific modifications in Equations 7.18-7.20 are made to the Kalman correction formulas utilized in localization:

$$\mathbf{S}_t^k = \begin{bmatrix} \mathbf{H}_{t,\mathbf{x}}^k & \mathbf{H}_{t,\mathbf{m}}^k \end{bmatrix} \begin{bmatrix} \overline{\boldsymbol{\Sigma}}_{t,\mathbf{xx}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{x}\mathbf{m}_k} \\ \overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}_k\mathbf{x}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}_k\mathbf{m}_k} \end{bmatrix} \begin{bmatrix} [\mathbf{H}_{t,\mathbf{x}}^k]^\top \\ [\mathbf{H}_{t,\mathbf{m}}^k]^\top \end{bmatrix} + \mathbf{R}_t^i \tag{7.18}$$

$$\mathbf{K}_t^i = \begin{bmatrix} \overline{\boldsymbol{\Sigma}}_{t,\mathbf{xx}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{x}\mathbf{m}_{j(i)}} \\ \overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}\mathbf{x}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}\mathbf{m}_{j(i)}} \end{bmatrix} \begin{bmatrix} [\mathbf{H}_{t,\mathbf{x}}^{j(i)}]^\top \\ [\mathbf{H}_{t,\mathbf{m}}^{j(i)}]^\top \end{bmatrix} [\mathbf{S}_t^{j(i)}]^{-1} \tag{7.19}$$

$$\boldsymbol{\Sigma}_t = \overline{\boldsymbol{\Sigma}}_t - \mathbf{K}_t^i \begin{bmatrix} \mathbf{H}_{t,\mathbf{x}}^{j(i)} & \mathbf{H}_{t,\mathbf{m}}^{j(i)} \end{bmatrix} \begin{bmatrix} \overline{\boldsymbol{\Sigma}}_{t,\mathbf{xx}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{x}\mathbf{m}} \\ \overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}_{j(i)}\mathbf{x}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}_{j(i)}\mathbf{m}} \end{bmatrix} \tag{7.20}$$

The reader should be aware that an index $\mathbf{m}_k$ for the state covariance indicates the covariance submatrix between the k-th landmark and either itself, the pose or the entire map, depending on the other index lying next to $\mathbf{m}_k$. The meaning is the same in the case of $\mathbf{m}_{j(i)}$, which replaces $\mathbf{m}_k$ only to show that the Kalman gain and the a posteriori covariance are computed only for the landmark associated with the measurement.

7.3 Landmark Initialization

The observations about the structure of the algorithm made in the preceding chapter remain valid for EKF SLAM. In spite of that, there is a further issue that needs to be handled in SLAM and has not been previously encountered. This problem concerns the fact that, unlike before, when all the landmarks and their number were known with exact precision, now the robot is facing the challenge of deciding whether an extracted feature should be associated to a visited beacon or it actually represents a new landmark. The decision process itself is elaborated on in the next section, but here we would like to give the expressions that need to be utilized when the robot indeed chooses to add a new landmark to its current state vector $\mathbf{y}_t$.

The problem just described is known in the literature as landmark initialization [77, 86, 90, 94]. In [86], Solá deals with the aspect of incorporating a new landmark $\mathbf{m}_{N+1}$ through an inverse observation model that outputs the new landmark's location based on the current pose and the measurement $\mathbf{z}_t^i$ that could not be assigned to any existing landmark. Note that, in practice, the use of this model replaces the Kalman update step, which should only be performed if a feature is matched to a known landmark. To define the inverse observation model for line features, we first introduce the random vector $\mathbf{w}_t = [w_{t,\rho} \;\; w_{t,\alpha}]^\top$, as Equation 7.21 indicates:

$$\mathbf{w}_t = \mathbf{h}(\mathbf{x}_t, \mathbf{m}_{N+1}, \kappa) \tag{7.21}$$

Substituting the preceding equality in the standard observation model and solving for $\mathbf{w}_t$ yields the result in Equation 7.22:

$$\mathbf{w}_t = \mathbf{z}_t^i - \boldsymbol{\delta}_t \tag{7.22}$$

where we think of the senor's output $\mathbf{z}_t^i$ as having known entries, which means that $\mathbf{w}_t$ is a random vector with variance $\mathbf{R}_t^i$ and mean at $\mathbf{z}_t^i$. When one solves Equation 7.21 for $\mathbf{m}_{N+1}$, what results is the explicit inverse observation model defined by the nonlinear function $\mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa)$ in Equation 7.23:

$$\begin{aligned} \mathbf{m}_{N+1} &= \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa) \\ \begin{bmatrix} r_{N+1} \\ \psi_{N+1} \end{bmatrix} &= \begin{bmatrix} (-1)^{\kappa+1} w_{t,\rho} + x_t \cos(\psi_{N+1}) + y_t \sin(\psi_{N+1}) \\ w_{t,\alpha} + \theta_t - (1-\kappa)\pi \end{bmatrix} \end{aligned} \tag{7.23}$$

Through this new model, we are aiming to identify a suitable way of augmenting the covariance $\overline{\boldsymbol{\Sigma}}_t$ such that $\boldsymbol{\Sigma}_t$ includes the correlations between the newly-added landmark and the pose and between the said landmark and the rest of the map and the pose. We keep following Solá's approach to achieve this [86]. Expanding Equation 7.23 by a Taylor series, truncated after the first-order terms, around the means of $\mathbf{x}_t$ and $\mathbf{w}_t$, we have the relationship in Equation 7.24:

$$\begin{aligned} \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa) &\approx \mathbf{f}(\overline{\boldsymbol{\mu}}_{t,\mathbf{x}}, \mathbf{z}_t^i, \kappa) + \frac{\partial \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa)}{\partial \mathbf{x}_t}(\mathbf{x}_t - \overline{\boldsymbol{\mu}}_{t,\mathbf{x}}) + \frac{\partial \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa)}{\partial \mathbf{w}_t}(\mathbf{w}_t - \mathbf{z}_t^i) \\ &\approx \mathbf{f}(\overline{\boldsymbol{\mu}}_{t,\mathbf{x}}, \mathbf{z}_t^i, \kappa) + \mathbf{F}_{t,\mathbf{x}}(\mathbf{x}_t - \overline{\boldsymbol{\mu}}_{t,\mathbf{x}}) + \mathbf{F}_{t,\mathbf{w}}(\mathbf{w}_t - \mathbf{z}_t^i) \end{aligned} \tag{7.24}$$

where the Jacobians $\mathbf{F}_{t,\mathbf{x}}$ and $\mathbf{F}_{t,\mathbf{w}}$ are given in Equations 7.25 and 7.26, respectively:

$$\mathbf{F}_{t,\mathbf{x}} = \frac{\partial \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa)}{\partial \mathbf{x}_t}\bigg|_{\overline{\boldsymbol{\mu}}_{t,\mathbf{x}}, \mathbf{z}_t^i} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & -\overline{\mu}_{t,x}\sin(\psi) + \overline{\mu}_{t,y}\cos(\psi) \\ 0 & 0 & 1 \end{bmatrix} \tag{7.25}$$

$$\mathbf{F}_{t,\mathbf{w}} = \frac{\partial \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa)}{\partial \mathbf{w}_t}\bigg|_{\overline{\boldsymbol{\mu}}_{t,\mathbf{x}}, \mathbf{z}_t^i} = \begin{bmatrix} (-1)^{\kappa+1} & -\overline{\mu}_{t,x}\sin(\psi) + \overline{\mu}_{t,y}\cos(\psi) \\ 0 & 1 \end{bmatrix} \tag{7.26}$$

Here, we used ψ as a shorthand notation for the mean $\overline{\mu}_{N+1,\psi}$, found from the evaluation of $\mathbf{f}(\overline{\boldsymbol{\mu}}_{t,\mathbf{x}}, \mathbf{z}_t^i, \kappa)$.

With the inclusion of a new landmark, three covariance matrices should be computed and appended to $\overline{\boldsymbol{\Sigma}}_t$. Firstly, the $(N+1)$-th landmark's covariance $\overline{\boldsymbol{\Sigma}}_{N+1,N+1}$ results from Equation 7.27 [86]:

$$\overline{\boldsymbol{\Sigma}}_{N+1,N+1} = \mathbf{F}_{t,\mathbf{x}}\overline{\boldsymbol{\Sigma}}_{t,\mathbf{xx}}\mathbf{F}_{t,\mathbf{x}}^\top + \mathbf{F}_{t,\mathbf{w}}\mathbf{R}_t^i\mathbf{F}_{t,\mathbf{w}}^\top \tag{7.27}$$

Secondly, the correlations of the new landmark with the rest of the variables in the state vector are gathered in a matrix calculated according to Equation 7.28 [86]:

$$\overline{\boldsymbol{\Sigma}}_{N+1,\text{all}} = \mathbf{F}_{t,\mathbf{x}}\begin{bmatrix} \overline{\boldsymbol{\Sigma}}_{t,\mathbf{xx}} & \overline{\boldsymbol{\Sigma}}_{t,\mathbf{xm}} \end{bmatrix} \tag{7.28}$$

By remembering the symmetry property of the covariance matrix, we also have $\boldsymbol{\Sigma}_{\text{all},N+1} = \boldsymbol{\Sigma}_{N+1,\text{all}}^{\top}$. The two preceding equations are actually the efficient implementation of SLAM covariance expansion. A proof that unveils the precise mathematical origins of the shown formulas can be consulted in Appendix C.

To conclude the section, when a new landmark is detected, the indications in Equation 7.29 should be followed [86]. The graphical representation of the necessary additions is revealed in Figure 7.3:

$$\boldsymbol{\mu}_t = \begin{bmatrix} \overline{\boldsymbol{\mu}}_t \\ \overline{\boldsymbol{\mu}}_{N+1} \end{bmatrix}, \qquad \boldsymbol{\Sigma}_t = \begin{bmatrix} \overline{\boldsymbol{\Sigma}}_t & \overline{\boldsymbol{\Sigma}}_{N+1,\text{all}}^{\top} \\ \overline{\boldsymbol{\Sigma}}_{N+1,\text{all}} & \overline{\boldsymbol{\Sigma}}_{N+1,N+1} \end{bmatrix} \tag{7.29}$$

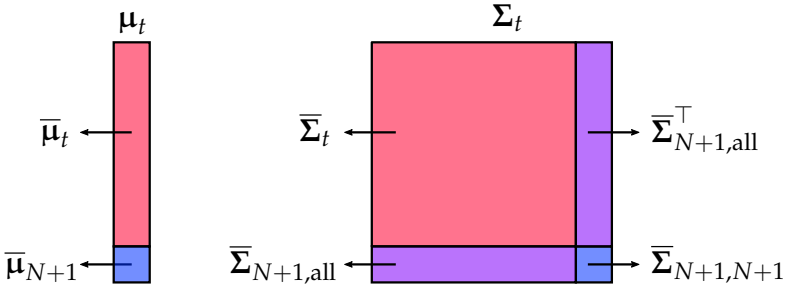


Figure 7.3: State mean and state covariance after landmark initialization

7.4 Data Association

In the derivation carried out so far, the discussion of the crucial feature-landmark matching problem has been avoided. As far as correspondences are concerned, SLAM is somewhat different compared to localization, because new landmarks have to be added to the map until all the cartographic data is gathered. Therefore, in case a feature does not fall within the validation ellipse of a known landmark, it cannot be associated with any map element and may therefore actually indicate an unobserved landmark. Notwithstanding, it may also simply constitute a spurious feature. Distinguishing between outliers and features with high landmark potential is in many cases done by keeping an evidence of possible future landmarks in a data structure very much similar to $\mathbf{m}$. Out of those, only the features deemed sufficiently reliable, are added to $\mathbf{y}_t$ [94, 98, 96].

7.4.1 Tentative Landmark List

Dissanayake et al. develop in [96] a landmark initialization algorithm that solves the correspondence problem by defining a tentative (or provisional) landmark list in the form of the vector $\mathbf{m}_{\text{tent}}$, with elements $\mathbf{m}_{j,\text{tent}} = [r_{j,\text{tent}} \;\; \psi_{j,\text{tent}}]^{\top}$, where $j \in \{1, \ldots, P\}$.

There are thus now two groups of elements every observed feature $\mathbf{z}_t^i$ should be compared against. In consequence, we define sensor-space validation gates for each

$\mathbf{m}_{j,\text{tent}}$ as well, all having the threshold γ_1, and for each feature-landmark combination we calculate the squared Mahalanobis distance $M_{ik,\text{tent}}$. Regardless of the nature of the landmark, a certain measurement falling within one or more validation gates has to be matched based on a nearest neighbour approach. For a correspondence to $\mathbf{m}_j$, the entire state $\mathbf{y}_t$ (i.e.: pose and map) needs to be updated. When matching is done for a tentative beacon $\mathbf{m}_{j,\text{tent}}$, the ideal thing to do would be to update the mean and the covariance of this segment such that it can be detected and correctly matched in future iterations. Apart from these significant additional computational and storage requirements, the robot should understand when a tentative landmark becomes reliable enough or, on the contrary, when it should be discarded for inconsistency reasons. It is easiest to tackle all the details in a structured manner, when matching hypotheses are individually discussed for a particular feature.

Association with a Confirmed Landmark

The EKF SLAM algorithm is examining all the elements of $\mathbf{m}$ and finds several feasible landmarks. The index of the statistically closest landmark is selected and labeled as $j(i)$ by implementing a nearest neighbor technique. In this case and this case only is the Kalman update step carried out for the entire state, because both the pose and the cartographic estimations are contingent on landmarks rediscoveries.

Association with a Tentative Landmark

For any given feature, a series of comparisons is made with all the P elements $\mathbf{m}_{k,\text{tent}}$ of the provisional map after checking the confirmed landmark list. Let the data association method generate the maximum likelihood correspondence index $j(i)$ as a result of the comparisons. At this stage, the feature should be matched with the $j(i)$-th provisional landmark, and the mean and covariance of the latter should be theoretically updated through Kalman correction.

The important thing to realize is that all $\mathbf{x}_t$, $\mathbf{m}_t$ and $\mathbf{m}_{k,\text{tent}}$ (for $k \neq j(i)$) affect the uncertainty of a provisional landmark. Thus, if one aims to correctly compute the a posteriori tentative map mean and covariance, one should take all the correlations into account. Notwithstanding, the update of a tentative feature's position should happen without affecting the pose of the robot or the rest of the map. The literature fails to provide a detailed strategy for dealing with tentative landmarks and only describes the maintenance of the provisional list in equivocal broad terms (see all [7, 77, 90, 96, 94]). A possible interpretation of the described methods is the least computationally expensive technique of choosing not to update the locations of the provisional map at all, avoiding the use of a second filter. Furthermore, it was decided to even completely disregard the correlations between any $\mathbf{m}_{k,\text{tent}}$ and the state $\mathbf{y}_t$ and keep only the provisional landmarks' variances in a non-sparse matrix $\mathbf{\Sigma}_{t,\mathbf{m}_{\text{tent}}\mathbf{m}_{\text{tent}}}$, in order to reduce the storage requirements. Vis-à-vis the matching process itself, it is adapted to our approach by simply writing adequately sized zero matrices instead of the pose-map covariances appearing in Equation 7.18.

If the k-th tentative landmark is successfully associated with an extracted feature more than a predefined number of times a during A successive iterations, it is moved to the permanent landmark list $\mathbf{m}$. A variable a_k is incremented every time an association transpires and another variable A_k keeps track of the number of iterations since the addition of $\mathbf{m}_{k,\text{tent}}$ to the provisional list. We have already discussed at the end of Section 7.3 the procedure for adding a new element to the map, but two things should be mentioned in the special case of provisional landmarks. Firstly, when adding a feature to the tentative list we disregard Equation 7.28. Secondly, if $\mathbf{m}_{j,\text{tent}}$ becomes a confirmed map element, one should delete the $(2j+2)$-th and the $(2j+3)$-th rows of $\boldsymbol{\mu}_{t,\mathbf{m}_{\text{tent}}}$ and columns of $\boldsymbol{\Sigma}_{t,\mathbf{m}_{\text{tent}}\mathbf{m}_{\text{tent}}}$. The deletion (but without the insertion into $\mathbf{m}$) will also happen when the robot fails to detect again the same provisional landmark sufficiently often during A iterations. The described procedure ensures a desired landmark reliability.

No Association Found

It can of course happen that the feature does not correspond to any element in either $\mathbf{m}$ or $\mathbf{m}_{\text{tent}}$. In this scenario, the measurement $\mathbf{z}_t^i$ is assumed to be associated with a new provisional landmark, whose mean and covariance are appended to $\boldsymbol{\mu}_{t,\mathbf{m}_{\text{tent}}}$ and $\boldsymbol{\Sigma}_{t,\mathbf{m}_{\text{tent}}\mathbf{m}_{\text{tent}}}$, respectively, through the inverse observation model introduced in Section 7.3. Notice the high level of resemblance between this situation and the recently discussed one, in which a tentative landmark is raised to the rank of confirmed landmark. However, the difference is that, in the latter case, the augmentation concerns $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ instead and correlations with the pose and the other landmarks are initialized with nonzero values.

7.4.2 Correspondence Enhancement for Line Features

Ambiguity in data association is inherent to the use of line features, as shown in Section 6.3, and it may affect the performance of EKF SLAM. In the previous chapter, a solution for reducing the probability of wrong correspondences was to consider the landmark and feature midpoints. In localization, the locations of the landmarks in the global frame do not vary and it is fairly straightforward to incorporate midpoints in data association. The same thing cannot be said about SLAM, as uncertainty shrouds the true positions of the map elements. Midpoints need therefore to be updated together with the EKF correction. The extra computational overhead is apparent. Although research papers such as [77, 94] declare the use of endpoints (even more computationally costly to maintain) in SLAM as successful, implementing something similar in this project was not found effective and absolutely necessary for the deployment of a basic, yet functional, SLAM algorithm.

Midpoints aside, it should be specified that data association is handled as discussed in Section 6.3, following these three principles: validation gating, maximum likelihood association (based on the landmarks' estimated positions with respect to the robot) and avoiding associations of features to already assigned landmarks.

7.5 EKF SLAM Algorithm

We show in Algorithm 3 our solution to the online SLAM problem. Note that the pseudocode is very high level and significant modifications are clearly a necessity for implementation on a real machine. We show only one iteration in which we assume the existence of N confirmed landmarks and P tentative ones. Although not explicitly shown, the variables that count the number of associations and iterations for tentative landmarks, i.e.: a_k, A_k, where $k \in \{1, \ldots, P\}$, must be passed at the function call, but also returned at the end of every iteration.

Algorithm 3: EKF SLAM with Unknown Correspondences

$EKF\_SLAM(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \boldsymbol{\mu}_{t-1,\mathbf{m}_{\text{tent}}}, \boldsymbol{\Sigma}_{t-1,\mathbf{m}_{\text{tent}}\mathbf{m}_{\text{tent}}}, \mathbf{u}_t, \mathbf{z}_t, N, P)$:
Compute $\mathbf{G}_{t,\mathbf{x}}$, $\mathbf{L}_{t,\mathbf{x}}$ and $\mathbf{Q}_t$;
Efficiently compute $\overline{\boldsymbol{\mu}}_{t,\mathbf{x}}$ and append to it $\boldsymbol{\mu}_{t-1,\mathbf{m}}$;
Efficiently compute $\overline{\boldsymbol{\Sigma}}_t$;
Extract all the visible line features $\mathbf{z}_t^i$ and find the covariances $\mathbf{R}_t^i$;
Increment A_k $\forall k \in \{1, \ldots, P\}$. If $A_k > A$, resize $\overline{\boldsymbol{\mu}}_{t,\mathbf{m}_{\text{tent}}}$ and $\overline{\boldsymbol{\Sigma}}_{t,\mathbf{m}_{\text{tent}}\mathbf{m}_{\text{tent}}}$. $P--$;
for all extracted features $\mathbf{z}_t^i = [\rho_t^i \;\; \alpha_t^i]^\top$, from $i = 1$ to $i = Z$
    for *all landmarks in* $\mathbf{m}$, *from* $k = 1$ *to* $k = N$ **do**
        Find the value of κ to decide the correct measurement model;
        Compute $\hat{\mathbf{z}}_t^k$, $\mathbf{H}_{t,\mathbf{x}}^k$ and $\mathbf{H}_{t,\mathbf{m}}^k$;
        Efficiently compute $\mathbf{S}_t^k$;
        Compute M_{ik}. If $M_{ik} > \gamma_1$, discard M_{ik};
    end
    Find $j(i)$, the correspondence of the nearest unmatched neighbor of $\mathbf{z}_t^i$;
    if $j(i)$ *exists* **then**
        Efficiently compute $\mathbf{K}_t^i$;
        $\overline{\boldsymbol{\mu}}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t^i(\mathbf{z}_t^i - \hat{\mathbf{z}}_t^{j(i)})$;
        Efficiently compute $\overline{\boldsymbol{\Sigma}}_t$;
    else
        for *all landmarks in* $\mathbf{m}_{\text{tent}}$, *from* $k = 1$ *to* $k = P$
            Find the value of κ to decide the correct measurement model;
            Compute $\hat{\mathbf{z}}_{t,\text{tent}}^k$ and $\mathbf{H}_{t,\mathbf{m}_{\text{tent}}}^k$;
            Efficiently compute $\mathbf{S}_{t,\text{tent}}^k$;
            Compute $M_{ik,\text{tent}}$. If $M_{ik,\text{tent}} > \gamma_1$, discard $M_{ik,\text{tent}}$;
        end

Algorithm 3: EKF SLAM with Unknown Correspondences (*continued*)

```
...
    ...
        Find j(i), the correspondence of the nearest unmatched neighbor of z_t^i;
        if landmark j(i) exists then
            a_{j(i)} + +;
            if a_{j(i)} ≥ a then
                Find κ to decide the correct inverse measurement model;
                Compute μ̄_{N+1}. Extend μ̄_t;
                Compute F_{t,x} and F_{t,w};
                Efficiently compute Σ̄_{N+1,N+1} and Σ̄_{N+1,all}. Extend Σ̄_t;
                Resize μ̄_{t,m_tent} and Σ̄_{t,m_tent m_tent};
                P − −. N + +;
            end
        else
            Find κ to decide the correct inverse measurement model;
            Compute μ̄_{N+1} from z_t^i. Extend μ̄_{t,m_tent};
            Compute F_{t,x} and F_{t,w};
            Efficiently compute Σ̄_{N+1,N+1}. Extend Σ̄_{t,m_tent m_tent};
            P + +. A_P = 0. a_p = 0;
        end
    end
end
μ_t = μ̄_t. Σ_t = Σ̄_t;
μ_{t,m_tent} = μ̄_{t,m_tent}. Σ_{t,m_tent m_tent} = Σ̄_{t,m_tent m_tent};
return μ_t, Σ_t, μ_{t,m_tent}, Σ_{t,m_tent m_tent}, N, P;
```

7.6 Simulation of EKF SLAM

A simulation of Algorithm 3 was carried out in a Python environment to evaluate the performance of EKF SLAM with unknown correspondences. The fictitious robot is placed in the same environment described previously in Section 6.5, but this time it is deprived of the knowledge of the landmarks' coordinates. These are progressively acquired through repeated observations of features (kept as tentative landmarks) that are being promoted to the landmark rank after 3 detections during 10 consecutive iterations. The time step, initial pose, controls, noise, range of the simulated LiDAR and Split-and-Merge thresholds are not altered, with the algorithm also being executed for 220 iterations.

7.6.1 Localization Performance

Just as in Section 6.5, the performance of the EKF in terms of localization was assessed for SLAM as well, by observing three different trajectories: the true noisy motion of the robot, the ideal motion not disrupted by any noise, and the Kalman estimation. Figure 7.4 shows these trajectories iteratively generated by the algorithm. There is a crystal clear similitude between the estimated trajectory (dashed magenta line) and the path supposed to mimic the real motion of the vehicle (solid lime line). A substantial disparity is noticed due to the lack of noise when looking at the orange trajectory, a fact that evinces the need for a stochastic observer capable of handling noisy processes.

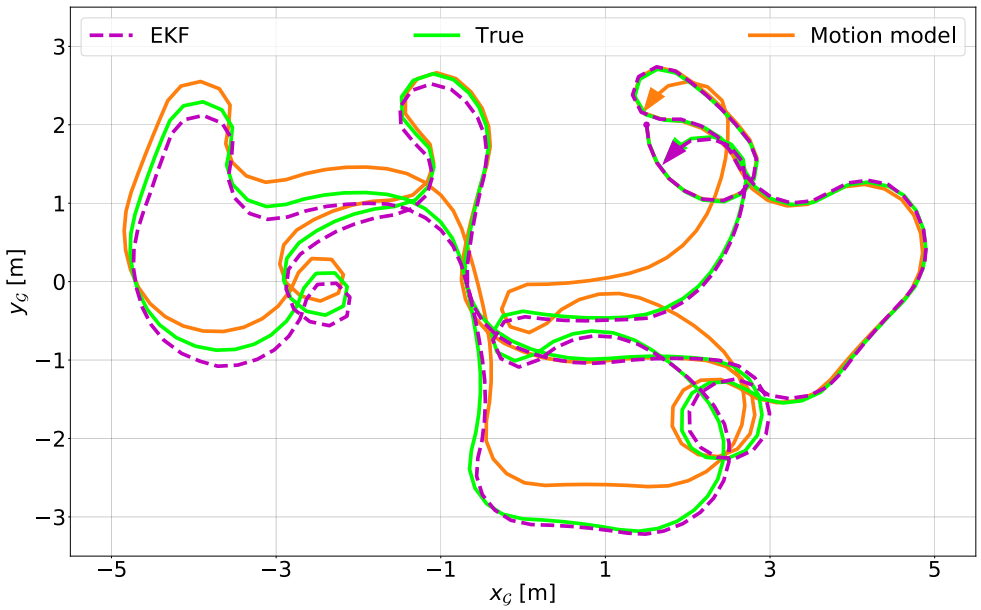


Figure 7.4: True, estimated and noise-free trajectories resulted from the SLAM simulation

The results of position tracking can be further examined by plotting the error between the estimated and the real pose during execution. In Figure 7.5, the absolute errors in x_t, y_t and θ_t are complemented by their respective 5σ covariance bounds. What is observed is that the error invariably lies within the defined uncertainty limit. Moreover, the position error in x_t and y_t is for the most part smaller than 10 cm, while the difference in heading only briefly surpasses 0.05 rad, with the spike around iteration 74 appearing as a consequence of the angular nonliterary at π and $-\pi$.

Yet another performance indicator worth a closer investigation is the discrepancy between the true trajectory and the noiseless path generated by a non-stochastic motion model. Four absolute error plots between the two pose evolutions can be seen in Figure 7.6. Three of them correspond to the components of the pose vector, while the fourth shows the Euclidean distance between the locations at each time step. Visible errors are accumulated by the end of iteration 220, reaching peaks of 35 cm, 77 cm, 0.18 rad and 80 cm for x_t, y_t, θ_t, and Euclidean distance, respectively. The nonlinearity in angle causes high peaks in the plot of θ_t in this case too.

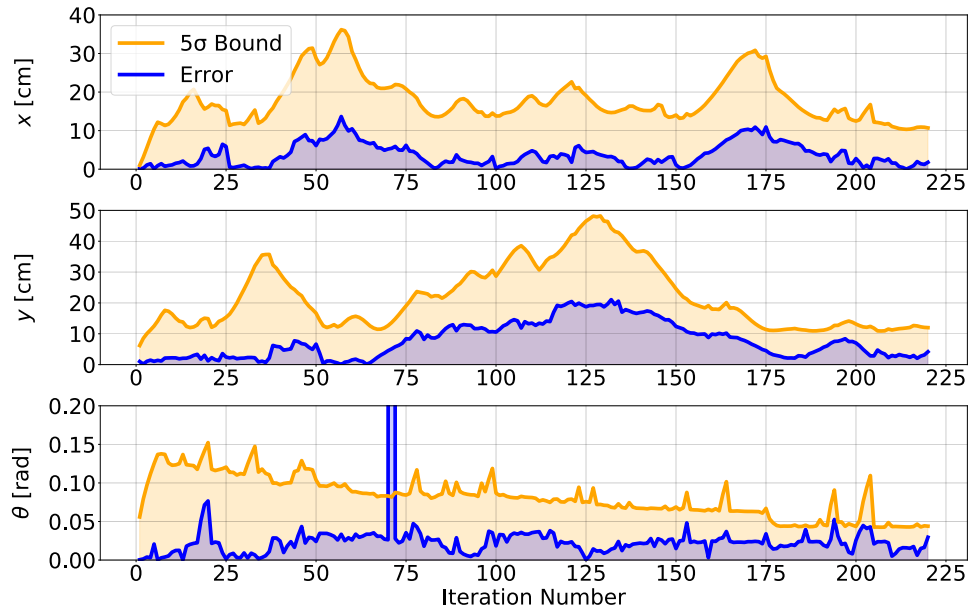


Figure 7.5: Absolute error and covariance bound during SLAM simulation

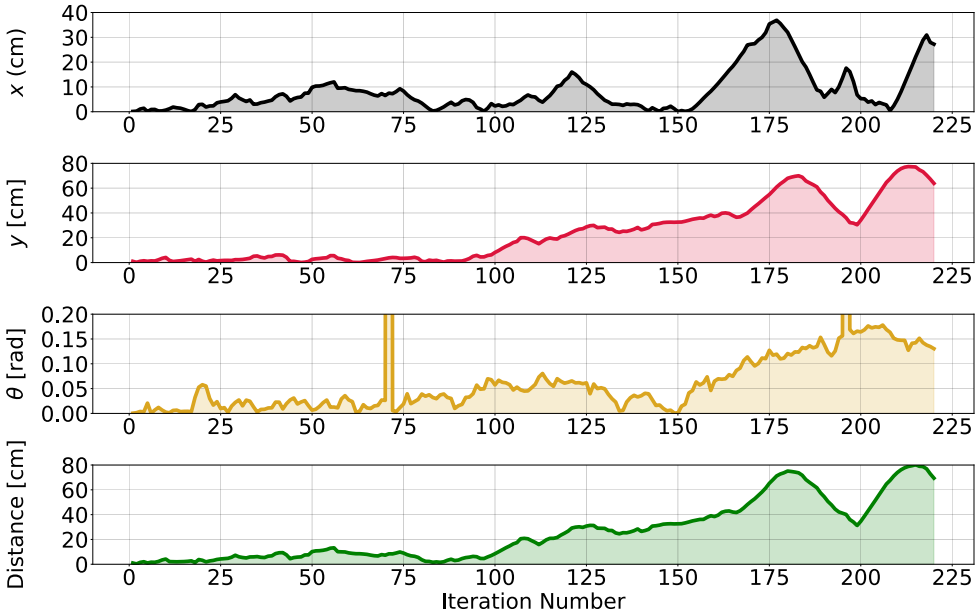


Figure 7.6: Absolute error between the true and ideal trajectories during the SLAM simulation

7.6.2 Mapping Performance

Let us first illustrate the mathematical property of uncertainty reduction of landmark variances. Figure 7.7 shows the evolution of the determinants of $\mathbf{\Sigma}_{t,\mathbf{m}_j\mathbf{m}_j}$, $j \in \{1, \dots N\}$, with N being the number of map elements at each iteration. The trend is undeniably a decreasing one and, in the limit, the determinants will converge to a very small value, proving that the vehicle becomes more certain about its surroundings as exploration unfolds. One comment we would like to make here is that, notwithstanding the

diminution of the determinants, the uncertainty of the map is in fact always small, even at landmark initialization. The phenomenon is due to many factors: the relatively small control noise, the even smaller observation noise, and the chosen motion sequence of the robot, which allows for immediate reobservation of landmarks. Despite these, the theory seems to accurately predict the behaviour in simulation.

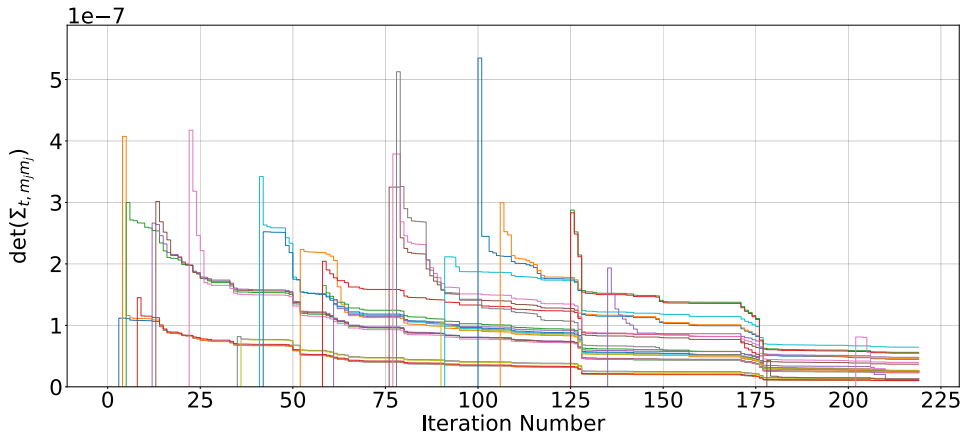


Figure 7.7: Evolution of the determinants of landmark variances

In order to gauge the accuracy of the acquired map, the scatter plot displayed in Figure 7.8 containing both the locations of the true landmarks and the means of the estimated landmarks' positions was created (the polar to Cartesian conversion has been applied for illustration purposes). Although not exactly superimposed, the acquired and real map elements lie quite close to each other. The robot was able to map 27 out of the total of 40 created landmarks, or approximately 68% of the map. When taking into account the fact that there are five pairs of collinear lines in the real map, the success rate goes up to 78%, since no mechanism to distinguish between landmarks with the same polar coordinates has been implemented. The reason for not mapping the entire environment is that the utilized trajectory is somewhat arbitrary and it was not designed with the objective of optimizing the mapping process.

An additional interesting visual representation of the developed SLAM solution is the plot in Figure 7.9, in which the full history of the robot's estimated pose is shown. The yellow ellipses are the 5σ-uncertainties in position found through EKF. First of all, as opposed to localization (see Figure 6.10), the uncertainty is more significant, which was expected, as no landmark is initially known. When the algorithm starts, the robot is very sure about where it is, but as it moves, the area of the ellipse grows due to the noise in the control. Measurements are taken and eventually landmarks are discovered and initialized with their own uncertainties that include the observation inaccuracies. When reobserved, a landmark produces a decrease in the current pose uncertainty (but also in the uncertainty of other map elements; this is not shown here explicitly). Since at iteration 220, the robot is in a previously explored area (marked with 1 in the figure), the pose uncertainty ellipse is not that apparent. On the other hand, when entering an uncharted area with low known landmark density (like area

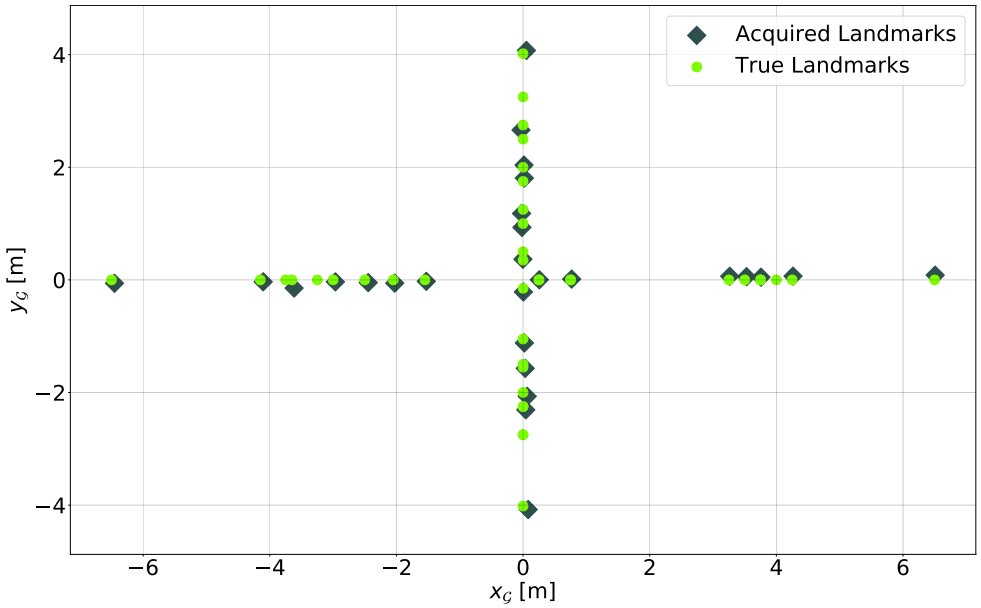


Figure 7.8: Cartesian coordinates of the estimated and true landmarks

2), the yellow ellipse increases in size.

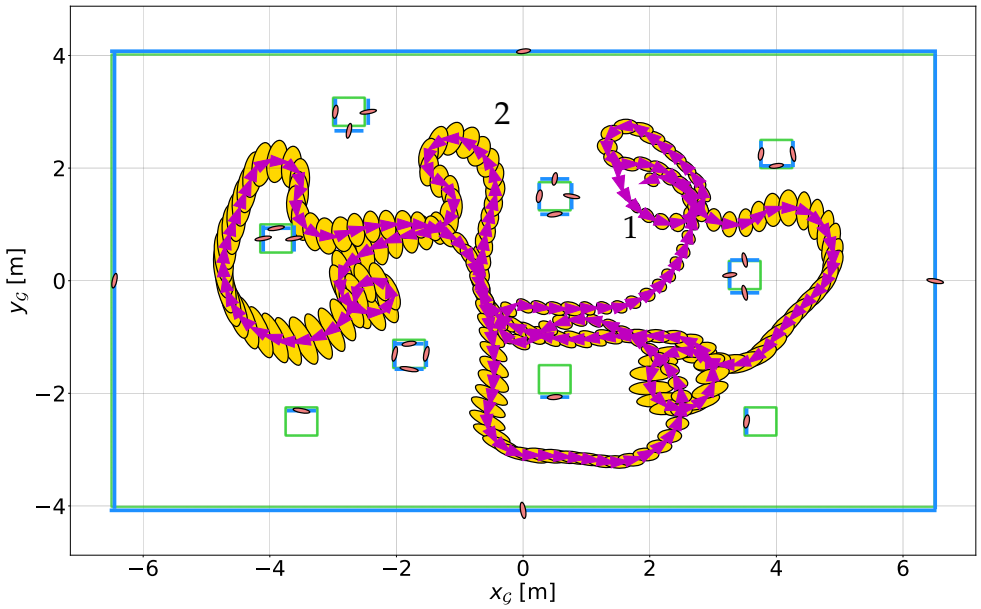


Figure 7.9: Position and pose uncertainty history, along with the true and final estimated line landmarks

What Figure 7.9 also shows is the real map (see the dim green lines), along with the final configuration of the estimated map elements, which are plotted in blue over the actual landmarks. It is obvious that the acquired landmarks match the true ones satisfactorily, and their uncertainties are small when the robot completes the journey around the environment (see the thin coral ellipses).

In view of all the showcased results, the simulation of EKF SLAM with unknown

correspondences is deemed successful, as the robot maintains the true trajectory and is able to create an accurate cartographic representation of the most part of its surroundings. The gif file of the Python SLAM simulation can be watched by accessing `https://bit.ly/3gOBSNr`.

Chapter 8

ROS Implementation

Throughout the entirety of this report, we have explored subject matters that serve as building blocks and are crucial to the SLAM problem. Especially, considering sections 6.5 and 7.6 where we have simulated the localization and SLAM algorithms on a custom built environment. In the above mentioned sections, as we know, the LiDAR sensor scans at different times stamps were artificially generated along with the movement of the robot. While this served as an essential step in developing and debugging the algorithm code, there are scenarios that could not be simulated through these means. After all, the goal of such algorithms is to be deployed on a machine operating in the real world. The platform of choice in our case was the Kobuki TurtleBot whose details are presented in system overview (Chapter 3). However, there would be a considerable gap in between the implementation done in Sections 6.5 and 7.6 and the actual implementation on Kobuki. The Python implementations were simply done with the motive of testing and debugging the SLAM and localization algorithms as opposed to the actual implementation.

When considering the actual deployment, there are additional factors to be considered such as peripherals that are crucial to a functioning system. These peripherals include the processing units as well as the sensory units. In consideration to the actual implementation, a Raspberry Pi micro-computer would be responsible for executing the algorithm while it communicates with the LiDAR sensor and then the results can be plotted separately on a different computer. Moreover, even with all the necessary steps taken, testing the developed algorithms directly on the robot could lead to unpredictable scenarios. Albeit, such unpredictability could never be completely removed, it could be considerably reduced through prior robust testing on simulation platforms, which provide close to real life conditions. One of such platforms in our case was Gazebo. On the surface, it may appear that testing with Gazebo or through the methods in Sections 6.5 and 7.6 present the same situations, as ultimately both are simulations. Having said that, the functionalities provided by Gazebo far exceed what we could achieve through our own code. Gazebo is a robotics simulator equipped with complex physics engines, which is crucial to simulating the dynamics and kinematics of a robot and other models. Furthermore, its strength lies in its

ability to include custom built structures that emulate the indoor and outdoor scenarios for a robot and its interaction with such structures through the use of simulated sensors. To give an example, an indoor environment such as a house could be built and implemented in Gazebo, with the robot being operated in this environment. As for simulated sensors, Gazebo takes into account even the properties such as how readings from the sensor are impacted by interaction with different kinds of surfaces. Moreover, what makes Gazebo even more of an obvious choice is its compatibility with Robot Operating System [99, 100].

ROS will provide the framework for the development of software required for implementation. The main concept behind ROS is its philosophy to divide large systems into multiple independent subsystems which communicate with each other through a semi peer-to-peer network. This became one of the enticing factors to choose ROS, as it allowed individual development of the functionalities needed to implement the SLAM/localizatoin algorithm practically. On top of that, it keeps the door open for future enhancement of this project, since the base structure could be built upon by adding new subsystems to it. Additionally, a key aspect of ROS is its inclination towards re-use of code. This would allow us to keep substantial amount of code that has been implemented for simulation with Gazebo and, with introduction of minor changes, essentially the same structure could be utilized to control the robot in real life. The subsystems mentioned in the beginning of this paragraph are referred to as nodes, which are basically processes that serve distinct objectives. Accordingly, the most fundamental, yet crucial processes (nodes) in our system would be those that execute the SLAM or localization algorithm and those that extract the sensor readings. The other nodes for the ROS implementation would be listed eventually with the whole structure. Prior to that, some ROS terminology needs to be introduced in order to keep the forthcoming discussion cogent.

ROS Terminology

Continuing the line of thought from the previous paragraph, as stated before, nodes represent the processes in the ROS structure. Simply put, any code that we develop will be wrapped inside a node in addition to certain specifiers that set up the respective node. Correspondingly, they come in different varieties out of which the most frequently used in our case are the publishing and the subscribing nodes. As mentioned earlier, the ROS framework is heavily dependent on the peer-to-peer communications between its nodes. The data that needs to sent in-between nodes is transferred in the form of messages. In a ROS-dependent robotic system, there are multitude of message types being utilized depending on the kind of data being transferred. Whereas, the links through which these message streams travel to the other nodes are referred to as topics. Correspondingly, as the name suggests, a publishing node transfers the data (message stream) to the subscribing node via a topic. The publisher is responsible for establishing a topic to which a subscriber could latch on to. It is worth mentioning that topic names must be unique, as the subscriber is only

privy to the name of the topic and is otherwise unaware of the publisher. Another kind of relation between nodes that will be employed is that of a server and a client. This sort of communication is handled through ROS services to facilitate the demand for request/reply communication, which is unachievable through one way publisher/subscriber network. The client sends a service request, which is simply a request to call a function (callback function) in the server node. Gazebo provides a variety of ROS services, some of which will be called upon during the implementation [99, 101, 100].

There are multitude of other components as well that contribute to the ROS architecture. Accordingly, to manage such a compartmentalized system, ROS is built on top of a comprehensive filesystem. Out of all, the ROS package is the most significant to our discussion. It is commonly referred to as the atomic unit of ROS, on top of which everything else is constructed. Although a package contains a wide variety of files, the only relevant attribute to our discussion is that all executable code is organized inside a package, as well as the dependencies on which the node relies. Apart from the packages built by us, there are thousands of publicly available packages developed by the ROS community. It is simply impractical, as well as time consuming, to write all the code by ourselves, therefore we will be utilizing some outside packages that are needed to achieve the SLAM/localization implementation. To end this discussion, we introduce the final term: workspace. When controlling a robotic system using ROS, multiple packages could be used at the same time, however, all the packages should be built and contained inside the same ROS workspace [99, 101, 100].

Gazebo Models

Even though, the ROS structure was built around the simulations in Gazebo, efforts were still inspired towards making a structure which could directly be implemented on the actual robot. It could be thought of as Gazebo being a place holder for Kobuki TurtleBot in the real environment. Therefore, a notable part of the simulated world is the robot to be controlled. Although Gazebo provides tools to construct a bespoke robot model and fit it with all the necessary sensors, we opted to go for models which were already available. Out of all, the TurtleBot3_Burger robot model was the perfect choice because of its similarities to our actual robot [1]. The robot model could be seen in Figure 8.1, where it is placed inside an empty Gazebo world. This model is extracted form the TurtleBot3 Gazebo package, which also includes gazebo_ros_pkgs packages, necessary for the integration of Gazebo with ROS. Apart from the robot's body, the key aspect of the Turltlebot in simulation is the attached LiDAR sensor on top. The simulated sensor is a 360° rotation LiDAR, capable of fetching 360 points per revolution [102, 103].

Besides the TurtleBot, other notable models were the structures that formed the

[1] From now on, any mentions of the "TurtleBot" refers to the "TurtleBot3_Burger" model in simulation, unless otherwise notified

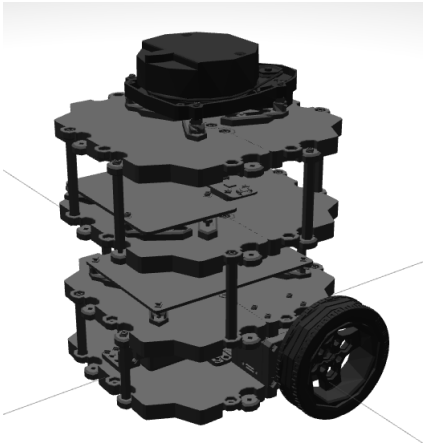


Figure 8.1: TurtleBot3 Burger model inside Gazebo environment

surroundings. The world in which the robot navigates has a critical impact on the performance of the algorithm, as the collected landmarks depend on the objects within the sensor range. Gazebo is equipped with the tools to generate any kind of environment, be it outdoor or indoor. Complying with our needs, we chose an indoor environment without the complexities of outdoors (e.g. wind) as well as devoid of any moving objects other than the robot itself. This indoor environment is just a collection of randomly placed walls inside a closed square space, constructed to avoid the robot from wandering away from the built area. A picture of the Gazebo world is shown in Figure 8.2.

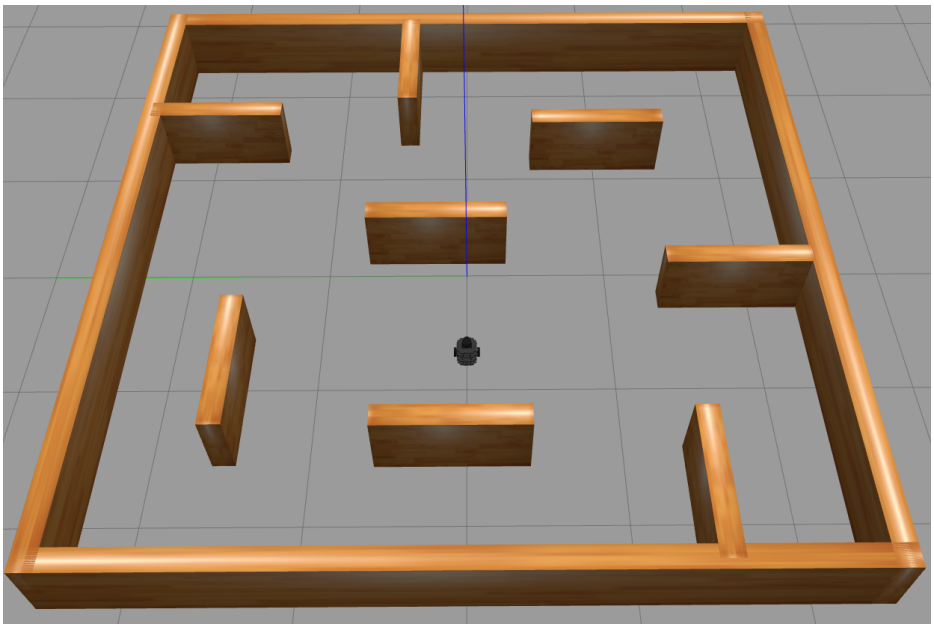


Figure 8.2: Gazebo structure with the TurtleBot inside it

ROS Implementation Structure

At this point, we have all the key components required to emulate the conditions of the real world, specific to our case. Gazebo simulations should be able to present a reflection of the scenarios encountered in an actual indoor structure. Correspondingly, the ROS implementation will be built around the flow of critical information to and from Gazebo. On that account, Gazebo will be treated as a node that runs the processes for the simulation of TurtleBot and its world. The graphical node arrangement shown in Figure 8.3 is the ROS solution to the given problem.

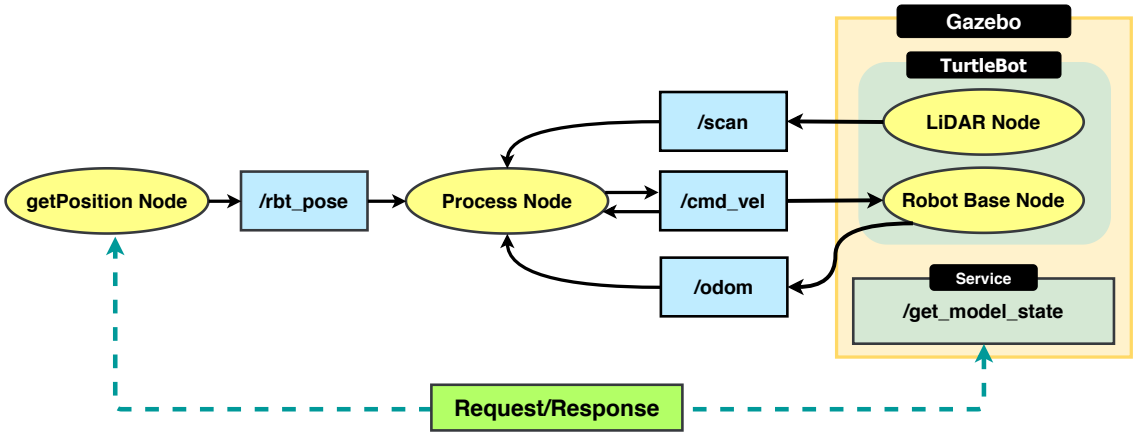


Figure 8.3: ROS graph structure for Gazebo implementation

The node graph in Figure 8.3 is an archetypal example of a ROS structure illustrating a collection of nodes and a variety of topics being published/subscribed to. However, it should be noted that some information has been omitted or presented in a way that best describes the whole system. For example, the Gazebo node is shown containing the simulated TurtleBot, which has its own separate nodes but in reality it's just one of the models inside Gazebo. Nonetheless, the rest of the discussion will invest in elucidating the roles played by each node in the respective graph. A good starting point will be the Process Node, that represents the main processing part of this graph, as it could either be housing the SLAM or the localization code, of course with suitable alterations. The node in itself is both a publisher and a subscriber and as it iteratively parses through the code, it publishes commands on a topic whereas also utilizing the information gained through subscribed topics. At each iteration, the Process Node should publish control commands to the TurtleBot in the form of linear and angular velocities. It does so by using the topic cmd_vel to which the TurtleBot Base (Robot Base Node) has been configured to subscribe and move accordingly. In the case of actual robot, it should also have a node that gives input to control the wheel motors. Moreover, the algorithm is heavily dependent on the LiDAR scanned points. Consequently, the LiDAR sensor mounted on top of the bot (represented by LiDAR Node in Figure 8.3) is made to publish its data over the scan topic. The message type for scan holds more than just the ranges and bearings, it also includes entities such as time stamp, frame id, increment angle etc.

In practice, these are all the basic nodes that would be essential to implement SLAM or localization using a LiDAR on a wheeled robot. Naturally, additional nodes could be added if new peripherals are attached. As this is a simulation, it gives us dominion over everything inside Gazebo - viz., we have access to all kinds of information in relation to the TurtleBot model. In the real world, we are not privy to the exact actual position of the robot but only an estimate, yet while simulating, it is worthwhile to check the credibility of our own algorithm by comparing its results against the exact values acquired from Gazebo. Congenitally, Gazebo provides a ROS service which when requested, acquires a model's pose and sends it to the client node. In this situation, the client is the getPosition Node node which calls the get_model_state service and then publishes the received pose information on the rbt_pose topic. To clear any confusion, the obtained exact position is solely used for comparative reasons and has no contribution in the algorithm. Moreover, in relation to wheeled robots, it is also customary to contemplate the use of odometry to generate a robot's pose. A deterministic odometry model is hampered by accumulation of position error with time, resulting in a drift in pose estimate. Nevertheless, it would prove for a good comparative study, if the effectiveness of the proposed algorithms in this report are juxtaposed against the odometry results. As such, the TurtleBot base node publishes the odometry messages, from which the most significant detail is the two dimensional pose of the robot.

The discussion heretofore has made it inherent that communication between nodes is the glue that keeps the whole ROS architecture functional. Still, just like in any form of communication, there is a rate at which the nodes communicate and this rate is not necessarily equal for every topic. Expectedly, there is a disparity between the publishing frequencies of the nodes in Figure 8.3. For instance, the LiDAR node transmits at 5 Hz while the control velocities are being published at a slower rate of 2 Hz. This becomes a significant problem as the sensor values and the control command are expected to be from the same time step in the proposed algorithms. Increasing the pace of the slower node is non-viable since the rate is restricted by the calculation time for necessary operations. Alternatively, slowing the faster node may seem like an obvious solution, but oftentimes, sensors have unchangeable rates at which they provide the readings. And, even if it were possible to slow down the sensor node, there will still be no guarantee that the values utilized are all from the same time. Therefore, a synchronization approach is needed to solve the issue.

On that account, message_ filters is a utility library commonly utilized by the ROS community for message synchronization [104]. As stated, we utilize this library to synchronize the incoming messages from different publishers using their time stamps. Each message has a header field that contains the time stamp at which that particular message was published. Consequently, in the receiving node the time stamps of incoming messages (from different topics) are compared and only the messages with (approximately) equal time stamps are allowed to pass to the subscriber. This brings us to an aspect of Process node which was left unexplained - subscribing to a topic published by itself. This was done to synchronize the time at which command ve-

locities are given and the LiDAR readings are taken, as the message_filter functions are only applicable to incoming data. Consequently, in the algorithm, at the start of each iteration we send the control commands and then receive the LiDAR readings corresponding to same time.

During the testing with Gazebo and ROS, the ROS structure was found to be properly set up, as we were able to provide the TurtleBot with command velocities and synchronously receive the LiDAR data along with the actual robot pose from Gazebo. Furthermore, the SLAM algorithm was made to iterate for a certain number of times and was observed to be executing without any communication problems between the nodes. However, even though the setup seemed to be functional, the results of the SLAM algorithm were erroneous. As the final SLAM implementation with ROS was approached at the end, after thorough testing with simulations in Python (discussed in Sections 6.5 and 7.6), the limited remaining time frame to implement the algorithm with ROS and to fix the issues, was found to be inadequate. Notwithstanding, ROS implementation remains to be an important aspect for this project and thus efforts will be directed later on towards solving its underlying issues.

Chapter 9

Discussion

The following section is a miscellaneous collection of additional keynotes that relate to this project. It explores topics such as suggestions for future improvement of the employed methods, problems incurred during the project as well as the direction towards which the project could be taken next.

Motion Model

The motion model given in Section 5.1 assumes a nonzero angular velocity command, thus restricting the mobile robot to always following a circular path. Albeit very straightforward to comprehend and implement, the equations required for modeling a purely linear motion were omitted in this project on purpose, due to the extra Jacobian matrix computation necessity. It is for this reason that a sole linear motion of the mobile robot is not achievable with the current version of the algorithm. Nevertheless, this did not rise any difficulty for the simulation purpose. By giving a tad of angular velocity besides the desired linear speed, a linear enough movement to the human eye was obtained when necessary.

Multi Hypothesis Tracking

The basic EKF implemented in this project can be extended further by incorporating a multi-hypothesis tracking filter (MHT) that enables multimodal posteriors. Doing so would allow pursuit of multiple correspondences using a mixture of Gaussians to form the posterior. This enhancement would bring further robustness to the data association problems faced in the challenging environment of the test bench where there exist five sets of collinear landmarks. In the presence of an MHT filter, the need of midpoint tracking for decreasing the number of incorrect data associations would be questionable and maybe even obsolete. Even though MHT is not able solve the kidnapped robot problem, it is appropriate for tackling the global localization problem, an improvement we would like to include in our solution in the future [7].

Implementation

The approach towards SLAM and localization discussed in this report still remains to be tested practically in a real indoor environment. Initially the project priorities were inclined towards practical execution. However, the unique circumstances arising from the global pandemic shifted the project's direction towards a more theoretical approach. Along with the simulations in Python, the ROS and Gazebo implementation were considered to be adequate verification tools for the developed algorithms, in place of the actual deployment on TurtleBot. Moreover, as previously stated, the ROS structure was still built with practical implementation in mind. The line of thought behind this modus operandi was inspired by the reasoning to use simulations for thoroughly debugging, testing and optimizing the algorithms, before jumping into practical experimentation, where the complexities of the problem increase significantly. On that account, the ROS structure although already constructed and seemingly functional, still has to be fixed for some underlying issues. The development of SLAM and localization algorithms was naturally assigned a higher priority, henceforth, they were being worked on until the final moment. Consequently, it was simply not feasible within the remaining time frame to fix and test ROS in combination with the latest version of code.

Data Visualization

Data visualization is a key requirement when developing a system, it lets us verify that the operation of a designed algorithm is as intended. In the Python simulations, the data plotting was part of the algorithm, therefore, the plotting was an additional process alongside the other critical functions. Plotting added to the time that the process took as a whole for calculations, thus impacting the rate of the whole operation. In the case of ROS, the same method was being followed for plotting. However, to avoid this unnecessary processing inside the main node, an extra ROS node can be created which is merely responsible for the visualization of data coming from the main process node. This way, the data visualization process is completely separate from the main node, which now only sends the data to be plotted. The result would be a significant reduction in workload for the main process of SLAM. Furthermore, this method becomes crucial when the processing node is in a remotely operating machine and the data needs to be visualized on a separate device away from the main computer. In this case, the information could be wirelessly sent to the visualization node in the other device. One such commonly used plotting environment is Rviz. It works as a general purpose 3D visualization tool for robots, sensors, and algorithms. Needless to say, Rviz was considered as an option when building the ROS structure. Howbeit, due to complete lack of any prior experience with ROS, the idea of adopting Rviz was discarded as a way of reducing the complexities in an already challenging task.

Drone Application

The inspiration to take on this project was aspired by our last project involving a quadcopter. During that project, it became inherently clear how crucial it is for drones to localize themselves when operating in unknown environments. However, due to the abundance of control variables and complex nature of drones, it was decided that we tackle a simpler problem but with the same end goals. Accordingly, when researching the subject matter for this project, it was kept in mind to maintain an approach to SLAM as independent as possible from the type of robot platform. Therefore, the concepts developed in this report, for a wheeled robot in 2D plane, can be extended to a drone in 3D plane. In this scenario, the robot now has 6 DoF (degrees of freedom), resulting in a addition of three more variables in our state vector. The elements inside the pose vector for a drone will be namely: the translational position x, y and z, along with the angular pose given by roll (ϕ), pitch (θ), and yaw (ψ). In addition to that, a drone is equipped with comparatively more sensors than our present case. These sensors for a drone may include: LiDAR, IMU, ultrasonic sensor, RGB-D camera etc. As such, there will be a need for a sensor fusion algorithm. Kalman Filter, the crux of this project, is also commonly utilized as sensor fusion algorithm in drones. Adding to the complications, a drone is inherently unstable. Therefore, the SLAM algorithm should be implemented on top of a comprehensive control algorithm (e.g. LQG). Also, the maps generated by SLAM for this scenario will be three dimensional, henceforth, the number of landmarks identified will most likely be significantly higher, unless a more selective approach to feature extraction is utilized. Additionally, the computation to run the SLAM algorithm for drones would be enormous and could not be implemented on any run of the mill processor. In the end, our decision to first tackle the problem for planar implementation of SLAM seems appropriate, and serves as a step towards more complex algorithms and situations.

Midpoint Tracking

The midpoint tracking extension included in the localization algorithm to enhance correct data association has served its purpose and were successful in differentiating collinear landmarks from each other. A similar improvement that could have been implemented instead of it is the inclusion of the landmark endpoints. Apart from supplementing feature matching even further, endpoint tracking would also enable easy plotting of the detected landmarks in the case of SLAM. Its implementation would not be very troublesome either. Instead of tracking a single point in the midpoint extension case, we would have the two endpoints of the extracted features. In spite of the doubled calculation cost, it is worth incorporating in the algorithms.

Submaps

In the scenario of executing the SLAM algorithm in a large area where there exist a lot more landmarks than how many we had in the custom environment we designed,

it would not be possible to build a global landmark map due to computational cost it would bring. Instead, keeping track of submaps and doing calculations on them would decrease the execution time of the SLAM algorithm [90].

Autonomous Navigation

It is mentioned in the introduction chapter that SLAM is a fundamental problem in the field of robotics, enabling autonomous applications such as self-driving cars. With the fulfilment of this thesis, we have acquired a foundation for SLAM research and we would very much like to continue our investigation in more advanced techniques currently being developed during our Master's programmes. However, autonomous navigation is also an interesting topic that we have set our eyes on. It builds upon Simultaneous Localization and Mapping, thus seems to be an appropriate path for us to follow.

Visual SLAM

Research on the state of the art SLAM techniques have shown us the prevalence and superiority of visual SLAM methods in the literature. Building upon that, the next step in pursuit of training ourselves in the field of SLAM should be towards studying image processing methods and working on SLAM solutions that are based on visual input from the environment. We can always make use of our LiDAR-based SLAM knowledge and complement a camera-based technique with it for superior results.

Performance Evaluation

Even though the results obtained from the Python simulation have pointed out the successful working of the implemented SLAM algorithm, a better way of investigating its performance would be making a comparison of the achieved results with the ones of a well-known open-source SLAM library. This way, the validity of our simulation results would have also been inspected. Another performance evaluation metric, we missed in this thesis, is the odometry data. If we have implemented the concepts of this thesis on a real mobile robot and possessed odometry information from it, we would be able to better underline the importance and neediness of Kalman filtering.

SLAM Paradigm

In this thesis, the chosen SLAM paradigm has been the Kalman filter family, the EKF more specifically. In Chapter 4, the drawbacks of EKF and its alternatives from the Kalman filter paradigm have been discussed. On top of that, another approach to the SLAM problem can be investigating other SLAM paradigms such as the graph-based optimization techniques and the particle methods. These techniques remain undiscovered from our point of view.

Chapter 10

Conclusion

Simultaneous Localization and Mapping is an abundantly studied and profoundly complex problem in robotics, whose solutions find countless relevant applications in modern society. As autonomous machines are indelibly morphing into a cornerstone of global industries, it is becoming increasingly clear that SLAM is not, by any means, merely a subject of mathematical curiosity for researchers, but a central component of future human-robot cooperation. As we are witnessing the prominent emergence of self-driving cars, there is no question why there seems to be an endless chain of enhancements being constantly put forward to enable a more wide-spread usage of SLAM technologies.

The present thesis constitutes our humble first step in a journey towards gaining insight into advanced robot navigation and perception techniques. The work described throughout the preceding chapters embodies an interpretation of the well-known fundamentals, with an imperfect, but fruitful outcome. The aim was to analyze and develop a SLAM solution for indoor environment applications and, in the process of attaining it, several building blocks have been carefully dissected, comprehended and described on an individual level.

The extended flavour of the Kalman Filter optimal estimator constitutes the very backbone of the project, for it represents the means of iteratively computing the belief of the robot about its state. Stochastic modeling of the vehicle's interaction with the environment was done via a probabilistic velocity motion model and a feature-based measurement function. What we defined as useful features to be extracted from the environment with a 2D LiDAR sensor were geometrical primitives in the form of lines and points. In the particular case of rectilinear features, the extraction method was Split-and-Merge clustering, an algorithm evaluated as adequate in a real indoor test area. A first challenge in our SLAM mission was developing a localization program for achieving robot position tracking, where a data associations policy based on squared Mahalanobis distance validation gating and nearest neighbour matching was employed. An enhancement using feature and landmark midpoints has been proposed to reduce correspondence ambiguity. Building on top of this, the EKF SLAM solution follows, relying on maximum likelihood association, a landmark initializa-

tion strategy and the maintenance of confirmed and tentative landmark lists. For both EKF localization and Simultaneous Localization and Mapping, the mathematical derivations are delineated.

The performance of the localization and the SLAM algorithms has been assessed by running Python simulations in a designed environment. In both cases, low-uncertainty high-fidelity estimates of the true position have been achieved, substantiating algorithmic correctness in tracking a noisy path. Regarding mapping, around three fourths of the true landmarks have been discovered without controls for optimal map acquisition being provided. As far as simulation is concerned, the results were more than satisfactory.

Furthermore, a communication network has been constructed in the ROS meta-operating system, comprising nodes that supplied the measurements, controls and offered a very realistic simulation environment in the form of Gazebo. Despite relentless efforts, work is still needed to bring the system to a functioning state, before real-world experiments can be carried out on the robot platform.

Overall, the Bachelor's thesis does honour to the intricate, yet fascinating, SLAM problem, by showcasing successful results of the proposed approach.

Bibliography

[1] M. Montemerlo et al. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem*. http://robots.stanford.edu/papers/montemerlo.fastslam-tr.pdf.

[2] Y. Latif, C. Cadena, and J. Neira. "Robust Loop Closing Over time for Pose Graph SLAM". In: *The International Journal of Robotics Research* (2013).

[3] C. Cadena et al. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". In: *IEEE Transactions on Robotics* (2016).

[4] A. Kelly. *Mobile Robotics: Mathematics, Models, and Methods*. Cambridge, 2014.

[5] P. Newman et al. "Explore and return: experimental validation of real-time concurrent mapping and localization". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation*. 2002.

[6] H. Durrant-Whyte and T. Bailey. "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms". In: *IEEE Robotics and Automation Magazine* (2006).

[7] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. 2^{nd} ed. The MIT Press, 2006.

[8] C. Forster et al. "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry". In: *IEEE Transactions on Robotics* (2017).

[9] S. Lynen et al. "Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization". In: *Proceedings of Robotics: Science and Systems Conference (RSS)*. 2015.

[10] A.I. Mourikis and S.I. Roumeliotis. "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2007.

[11] U. Frese. "Interview: Is SLAM Solved?" In: *KI - Künstliche Intelligenz* 24.3 (2010), pp. 255–257. DOI: 10.1007/s13218-010-0047-x.

[12] *Kuka Navigation Solution*. Kuka Robotics. URL: https://www.kuka.com/en-se/products/mobility/navigation-solution (visited on 04/23/2020).

[13] M. Maimone, Y. Cheng, and L. Matthies. "Two Years of Visual Odometry on the Mars Exploration Rovers". In: *Journal of Field Robotics* (2007).

[14] *ProjectTango*. Google. URL: https://www.youtube.com/watch?v=Qe10ExwzCqk (visited on 04/23/2020).

[15] *Robot Mapping*. Albert-Ludwigs-Universität Freiburg. 2013. URL: http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/ (visited on 04/23/2020).

[16] B. Siciliano and O. Khatib. *Handbook of Robotics*. 1^{st} ed. Springer, 2008.

[17] S. Thrun. *Robotic Mapping: A Survey*. 2002.

[18] J. Kshirsagar, S. Shue, and J.M. Conrad. "A Survey of Implementation of Multi-Robot Simultaneous Localization and Mapping". In: *SoutheastCon 2018*. 2018, pp. 1–7.

[19] C. Leung, S. Huang, and G. Dissanayake. "Active SLAM using Model Predictive Control and Attractor based Exploration". In: *International Conference on Intelligent Robots and Systems*. 2006.

[20] J. Guolai et al. "A Simultaneous Localization and Mapping (SLAM) Framework for 2.5D Map Building Based on Low-Cost LiDAR and Vision Fusion". In: *Applied Sciences* 9.10 (May 2019), p. 2105. ISSN: 2076-3417. DOI: 10.3390/app9102105. URL: http://dx.doi.org/10.3390/app9102105.

[21] T. Taketomi, H. Uchiyama, and S. Ikeda. "Visual SLAM algorithms: a survey from 2010 to 2016". In: *IPSJ Transactions on Computer Vision and Applications* (2017).

[22] C. Debeunne and D. Vivet. "A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping". In: *MDPI Sensors* (2020).

[23] J. Civera and S. H. Lee. "RGB-D Odometry and SLAM". In: *RGB-D Image Analysis and Processing*. Cham: Springer International Publishing, 2019, pp. 117–144. DOI: 10.1007/978-3-030-28603-3_6.

[24] G. Grisetti, C. Stachniss, and W. Burgard. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters". In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46.

[25] K. Konolige et al. "Efficient Sparse Pose Adjustment for 2D mapping". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 22–29.

[26] G. Grisetti, C. Stachniss, and W. Burgard. "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 2432–2437.

[27] S. Thrun. "Particle Filters in Robotics". In: *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*. 2002.

[28] M. Montemerlo and S. Thrun. *FastSLAM*. Springer, 2007.

[29] S. Kohlbrecher et al. "A flexible and scalable SLAM system with full 3D motion estimation". In: *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. 2011, pp. 155–160.

[30] W. Hess et al. "Real-time loop closure in 2D LIDAR SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1271–1278.

[31] *Introduction to Mobile Robotics*. Albert-Ludwigs-Universität Freiburg. 2019. URL: http://ais.informatik.uni-freiburg.de/teaching/ss19/robotics/ (visited on 04/10/2020).

[32] R. Yagfarov, M. Ivanou, and I. Afanasyev. "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth". In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2018, pp. 1979–1983.

[33] P. Abbeel. *Scan Matching*. https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/ScanMatching.pdf.

[34] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* (2015).

[35] R. Mur-Artal and J. D. Tardós. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* (2017).

[36] D. Vivet, A. Debord, and G. Pagès. "PAVO: A Parallax based Bi-Monocular VO Approach For Autonomous Navigation In Various Environments". In: *DISP Conference*. 2019.

[37] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. "DTAM: Dense tracking and mapping in real-time". In: *2011 International Conference on Computer Vision*. 2011, pp. 2320–2327.

[38] J. Engel, T. Schöps, and D Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *European Conference on Computer Vision*. 2014, 834–849.

[39] C. Forster, M. Pizzoli, and D. Scaramuzza. "SVO: Fast semi-direct monocular visual odometry". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 15–22.

[40] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.3 (2018), pp. 611–625.

[41] R. Wang, M. Schworer, and D. Cremers. "Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras". In: *IEEE International Conference on Computer Vision*. 2017, 3903–3911.

[42] R. A. Newcombe et al. "KinectFusion: real-time dense surface mapping and tracking". In: *Proceedings of International Symposium on Mixed and Augmented Reality*. 2011, pp. 127–136.

[43] R. F. Salas-Moreno et al. "SLAM++: Simultaneous Localisation and Mapping at the Level of Objects". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 1352–1359.

[44] K. Tateno, F. Tombari, and N. Navab. "When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 2295–2302.

[45] D. Galvez-López and J. D. Tardos. "Bags of Binary Words for Fast Place Recognition in Image Sequences". In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1188–1197.

[46] C. Hsu et al. "Depth measurement based on pixel number variation and Speeded Up Robust Features". In: *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*. 2014, pp. 228–229.

[47] E. Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571.

[48] C. Harris and M. Stephens. "A combined corner and edge detector". In: *Alvey Vision Conference*. 1988, 147–52.

[49] J. Shi and C. Tomasi. "Good Features to Track". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 1994, 593–600.

[50] A. E. Johnson et al. "Robust and Efficient Stereo Feature Tracking for Visual Odometry". In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 39–46.

[51] L. Matthies. "Dynamic stereo vision". PhD thesis. Carnegie Mellon University, 1989.

[52] R. Ng et al. *Light Field Photography with a Hand-held Plenoptic Camera*. `https://graphics.stanford.edu/papers/lfcamera/lfcamera-150dpi.pdf`. 2005.

[53] D. Weikersdorfer, D. Adrian D. B. Cremers, and J. Conradt. "Event-based 3D SLAM with a depth-augmented dynamic vision sensor". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 359–364.

[54] H. Kim, S. Leutenegger, and A.J. Davison. "Real-time 3D reconstruction and 6-DoF tracking with an event camera". In: *European Conference on Computer Vision*. 2016, pp. 349–364.

[55] I. Vallivaara et al. "Simultaneous localization and mapping using ambient magnetic field". In: *2010 IEEE Conference on Multisensor Fusion and Integration*. 2010, pp. 14–19.

[56] R. Elbasiony and W. Gomaa. "WiFi Localization for Mobile Robots Based on Random Forests and GPLVM". In: *2014 13th International Conference on Machine Learning and Applications*. 2014, pp. 225–230.

[57] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828.

[58] J. Valentin et al. "Exploiting uncertainty in regression forests for accurate camera relocalization". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4400–4408.

[59] A. Kendall, M. Grimes, and R. Cipolla. "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2938–2946.

[60] A. Cadena, A. Dick, and I. D. Reid. "Multi-modal Auto-Encoders as Joint Estimators for Robotics Scene Understanding". In: *Robotics: Science and Systems Conference (RSS)*. 2016, 377–386.

[61] D. Eigen and R. Fergus. "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2650–2658.

[62] F. Liu et al. "Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.10 (2016), pp. 2024–2039.

[63] V. Mohanty et al. "DeepVO: A Deep Learning approach for Monocular Visual Odometry". In: (Nov. 2016).

[64] S. Wang et al. "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2043–2050.

[65] X. Ruan, F. Wang, and J. Huang. "Relative Pose Estimation of Visual SLAM Based on Convolutional Neural Networks". In: *2019 Chinese Control Conference (CCC)*. 2019, pp. 8827–8832.

[66] S. Vijayanarasimhan et al. *SfM-Net: Learning of Structure and Motion from Video*. `https://arxiv.org/pdf/1704.07804.pdf`. 2017.

[67] R. Mahjourian, M. Wicke, and A. Angelova. "Unsupervised Learning of Depth and Ego-Motion from Monocular Video Using 3D Geometric Constraints". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5667–5675.

[68] V. Casser et al. "Unsupervised Monocular Depth and Ego-Motion Learning With Structure and Semantics". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 381–388.

[69] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-scale Image Recognition*. `https://arxiv.org/pdf/1409.1556.pdf`. 2015.

[70] N. Sunderhauf et al. "Place Recognition with ConvNet Landmarks: Viewpoint-Robust, Condition-Robust, Training-Free". In: *Robotics: Science and Systems (RSS)*. 2015.

[71] X Zhang, Y. Su, and X. Zhu. "Loop closure detection for visual SLAM systems using convolutional neural network". In: *2017 23rd International Conference on Automation and Computing (ICAC)*. 2017, pp. 1–6.

[72] G Welch and G Bishop. *An Introduction to the Kalman Fiter*. `https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf`. 2006.

[73] N. Assimakis, M. Adam, and A. Douladiris. "Information Filter and Kalman Filter Comparison: Selection of the Faster Filter". In: *International Journal of Information Engineering* (2012).

[74] R. P. Saputra. "Implementation 2D EKF SLAM for Wheeled Mobile Robot". MA thesis. University of New South Wales, 2015.

[75] M. Korkmaz, N. Yilmaz, and A. Durdu. "Comparison of the SLAM algorithms: Hangar experiments". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation*. 2002.

[76] Paz L. M., J. D. Tardos, and J Neira. "Divide and Conquer: EKF SLAM in $O(n)$". In: *IEEE Transactions on Robotics* (2008).

[77] A. Garulli et al. "Mobile robot SLAM for line-based environment representation". In: *Proceedings of the 44th IEEE Conference on Decision and ControlProceedings of the 44th IEEE Conference on Decision and Control*. 2005.

[78] R. Negenborn. "Robot Localization and Kalman Filters: On finding your position in a noisy world". MA thesis. Utrecht University, 2003.

[79] V. Nguyen et al. "A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005.

[80] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. 2^{nd} ed. The MIT Press, 2011.

[81] L. R. Munoz, M. G. Villanueva, and C. G. Suarez. "A tutorial on the total least squares method for fitting a straight line and a plane". In: *Revista de Ciencia e Ingeniería del Instituto Tecnológico Superior De Coatzacoalcos* (2014).

[82] S. Liu et al. "Adaptive Covariance Estimation Method for LiDAR-Aided Multi-Sensor Integrated Navigation Systems". In: *Micromachines* (2015).

[83] F. Lu and E. Milios. "Robot pose estimation in unknown environments by matching 2D range scans". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994.

[84] B. Ochoa and S. Belongie. "Covariance Propagation for Guided Matching". In: *3rd Workshop on Statistical Methods in Multi-Image and Video Processing, SMVP*. 2006.

[85] D. Simon. *Optimal State Estimation: Kalman, $\mathcal{H}_\infty$, and Nonlinear Approaches*. 1^{st} ed. Wiley Interscience, 2006.

[86] J. Solà Ortega. "Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach". PhD thesis. Institut National Polytechnique de Toulouse, 2007.

[87] E. B. Olson. "Robust and Efficient Robotic Mapping". PhD thesis. Massachusetts Institute of Technology, 2008.

[88] C. M. Bishop. *Pattern Recognition and Machine Learning*. 1^{st} ed. Springer, 2006.

[89] A. J. Cooper. "A Comparison of Data Association Techniques for Simultaneous Localization and Mapping". MA thesis. Massachusetts Institute of Technology, 2005.

[90] T. Bailey. "Mobile Robot Localisation and Mapping in Extensive Outdoor Environments". PhD thesis. The University of Sydney, 2002.

[91] L. R. Kenari and M. R. Arvan. "Comparison of Nearest Neighbor and Probabilistic Data Association Methods for Non-linear Target Tracking Data Association". In: *Proceeding of the 2nd RSI/ISM International Conference on Robotics and Mechatronics*. 2006.

[92] *Advanced Techniques for Mobile Robotics (Robotics 2)*. Albert-Ludwigs-Universität Freiburg. 2011. URL: http://ais.informatik.uni-freiburg.de/teaching/ws11/robotics2 (visited on 04/26/2020).

[93] *The Relationship between the Mahalanobis Distance and the Chi-Squared Distribution*. Thill M., Notes on Machine Learning, Statistics & Programming. 2017. URL: https://markusthill.github.io/mahalanbis-chi-squared/ (visited on 04/23/2020).

[94] T. Genevois and T. Zielisnka. "A simple and efficient implementation of EKF-based SLAM relying on laser scanner in complex indoor environment". In: *Journal of Automation, Mobile Robotics & Intelligent Systems* (2014).

[95] S. J. Julier and J. K. Uhlmann. "A counter example to the theory of simultaneous localization and map building". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 4. 2001, pp. 4238–4243.

[96] M. W. M. G. Dissanayake et al. "A Solution to the Simultaneous Localisation and Map Building (SLAM) Problem". In: *IEEE Transactions on Robotics and Automation* (2001).

[97] *Simultaneous localization and mapping with the extended Kalman filter*. 2014. URL: http://www.joansola.eu/ (visited on 05/01/2020).

[98] P. M. Newman. "On the Structure and Solution of the Simultaneous Localisation and Map Building Problem". PhD thesis. The University of Sydney, 1999.

[99] A. Martinez and E. Fernández. *Learning ROS for Robotics Programming*. 1st ed. Packt Publishing, 2013.

[100] M. Quigley, B. Gerkey, and W.D. Smart. *Programming Robots with ROS*. 1st ed. O'Reilly Media, 2015.

[101] W.S. Newman. *A Systematic Approach to Learning Robot Programming with ROS*. 1st ed. CRC Press, 2018.

[102] *ROS.org: TurtleBot3*. URL: http://wiki.ros.org/turtlebot3 (visited on 06/03/2020).

[103] *Robots e-Manual TurtleBot3*. URL: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (visited on 06/03/2020).

[104] *ROS.org: messagefilter*. URL: http://wiki.ros.org/message_filters (visited on 06/03/2020).

Appendix A

Line Landmarks

A.1 Derivation of the Sensor Model

Consider the two situations in Figure A.1, which is actually just a slightly simplified version of Figure 5.3, but with some additional helpful constructions.

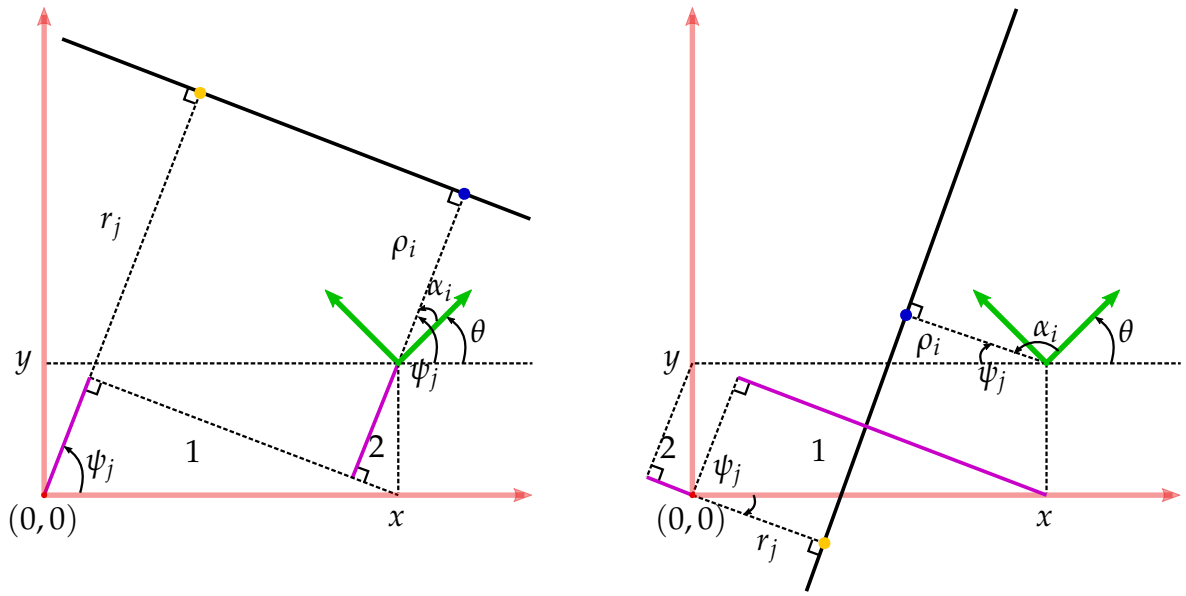


Figure A.1: Illustration of the robot's onboard sensor measuring the relative position of a line landmark. *Left:* $l_p \cap l_j = \varnothing$; *Right:* $l_p \cap l_j \neq \varnothing$

Let us start by looking to the left side of the figure. In the triangles labeled with 1 and 2, the sides coloured in magenta have the expressions $x\cos(\psi_j)$ and $y\sin(\psi_j)$, respectively. From the figure, it is now obvious that subtracting these two segments from r_j will yield ρ_i. From the figure again, we see that α_i is obtained very easily as well, by subtracting the heading of the robot from ψ_j. These will give the sensor model when no intersection occurs.

$$\begin{bmatrix} \rho_t^i \\ \alpha_t^i \end{bmatrix} = \begin{bmatrix} r_j - x\cos(\psi_j) - y\sin(\psi_j)) \\ \psi_j - \theta \end{bmatrix} \tag{A.1}$$

When there is indeed an intersection, the derivation is quite similar, although a bit less straightforward. Looking at the right side of Figure A.1, we see that ψ_j is a negative angle. To arrive at the sensor model expression through geometrical means in this case, we need therefore to use the positive $-\psi_j$. With this observation in mind, we again shift our attentions to the lines coloured in magenta in triangles 1 and 2, expressed as $x\cos(-\psi_j)$ and $y\sin(-\psi_j)$, respectively. We can get ρ_i by subtracting r_j and $y\sin(-\psi_j)$ from $x\cos(-\psi_j)$. The angular parameter α_i is obtaoned by noting that θ, α_i and $-\psi_j$ should sum to π.

$$\begin{bmatrix} \rho_t^i \\ \alpha_t^i \end{bmatrix} = \begin{bmatrix} -r_j + x\cos(\psi_j) + y\sin(\psi_j)) \\ \psi_j - \theta + \pi \end{bmatrix} \tag{A.2}$$

A.2 Line Segment Extraction for Landmark Visualization Purposes

The content of this section is entirely based on [81], and the relations given here are utilized in the project for plotting purposes once line landmark parameters are extracted. Let us denote by ζ_h^i the vector of coordinates $[x_h^i \;\; y_h^i]^\top$ and by $\boldsymbol{\lambda}$, the vector of the mean values of the Cartesian coordinates for all the points belonging to the i-th cluster of points. Thus, $\boldsymbol{\lambda} = [\bar{x}^i \;\; \bar{y}^i]^\top$. In order to extract segments from the already extracted infinite-line parameters, we employ Equation A.3:

$$\zeta_{h,R}^i = \mathbf{R}_{-\alpha_t^i}(\zeta_h^i - \boldsymbol{\lambda}) \tag{A.3}$$

where

$$\mathbf{R}_\alpha = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{A.4}$$

is the 2D-rotation matrix that rotates a vector counterclockwise by the angle α. Using Equation A.4 rotates all the points belonging to the segment $\mathcal{S}_i$ by the angle $-\alpha_t^i$ around the mean of these points, whereas Equation A.3 effectively moves the mean of the line segment to the frame's origin.

It is obvious that now the line of best fit for the cluster of $\zeta_{h,R}^i$ points is perfectly vertical in the local frame, and a finite-length segment can be elucidated by finding the endpoints $\mathbf{e}_{1,R}^i = [0 \;\; y_{min}^i]^\top$ and $\mathbf{e}_{2,R}^i = [0 \;\; y_{max}^i]^\top$ corresponding to the minimum and maximum $y_{h,R}^i$ values, respectively. We can then apply the reciprocal of Equation A.5 to get the segment endpoints in the initial configuration, as shown in Equation A.5:

$$\mathbf{e}_k^i = \mathbf{R}_{\alpha_t^i}\mathbf{e}_{k,R}^i + \boldsymbol{\lambda}, \;\; \text{where } k \in \{1, 2\} \tag{A.5}$$

Appendix B

Maximum Likelihood Association

The likelihood of associating the measurement $\mathbf{z}_t^i$ with the landmark $\mathbf{m}_j$ is given by the following equality, where $\boldsymbol{\nu}_i = \mathbf{z}_t^i - \hat{\mathbf{z}}_t^j$ is the n-dimensional innovation vector and $\mathbf{S}_t^j$ is the $n \times n$ innovation matrix:

$$p_i = \frac{1}{\sqrt{\det\left(2\pi \mathbf{S}_t^j\right)}} \exp\left(-\frac{1}{2}\boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i\right) \tag{B.1}$$

Taking the natural logarithm on both sides of Equation B.1, gives:

$$\begin{aligned} \ln p_i &= -\ln \sqrt{\det\left(2\pi \mathbf{S}_t^j\right)} - \frac{1}{2}\boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i \\ &= -\ln\left((2\pi)^{n/2} \sqrt{\det\left(\mathbf{S}_t^j\right)}\right) - \frac{1}{2}\boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i \\ &= -\ln\left((2\pi)^{n/2}\right) - \frac{1}{2}\ln\left(\det\left(\mathbf{S}_t^j\right)\right) - \frac{1}{2}\boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i \\ &= -\ln\left((2\pi)^{n/2}\right) - \frac{1}{2}\left(\ln\left(\det\left(\mathbf{S}_t^j\right)\right) + \boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i\right) \end{aligned} \tag{B.2}$$

The leftmost term on the right side of the equality in Equation B.2 is a constant for a fixed n and therefore does not influence the maximization of $\ln p_i$. Thus, we have that the maximum log-likelihood is given by:

$$\underset{j}{\operatorname{argmax}} \ln p_i = \underset{j}{\operatorname{argmax}} -\frac{1}{2}\left(\ln\left(\det\left(\mathbf{S}_t^j\right)\right) + \boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i\right) \tag{B.3}$$

We can transform this optimization problem in a minimization by multiplying both sides by -2. This is how one arrives to the equivalent objective of identifying the minimum normalized log-likelihood N_i (the normalized distance):

$$\underset{j}{\operatorname{argmin}}\, N_i = \underset{j}{\operatorname{argmin}}\, \boldsymbol{\nu}_i^\top [\mathbf{S}_t^j]^{-1} \boldsymbol{\nu}_i + \ln\left(\det\left(\mathbf{S}_t^j\right)\right) \tag{B.4}$$

In deriving the proof shown in this appendix, the material discussed in [89, 90] has been utilized.

Appendix C

SLAM Covariance Augmentation

When a new confirmed landmark needs to be added to the map, the inverse measurement model should be employed. A landmark is generated from an observation in the sensor frame and the global estimated position of the robot.

$$\mathbf{m}_{N+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa) \tag{C.1}$$

The state vector $\mathbf{y}_t$ after the addition can be written as:

$$\mathbf{y}_t = \begin{bmatrix} \overline{\mathbf{y}}_t \\ \mathbf{m}_{N+1} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{y}}_t \\ \mathbf{f}(\mathbf{x}_t, \mathbf{w}_t, \kappa) \end{bmatrix} = \xi(\mathbf{x}_t, \mathbf{m}, \mathbf{w}_t, \kappa) \tag{C.2}$$

where $\overline{\mathbf{y}}_t = [\mathbf{x}_t \quad \mathbf{m}]^\top$ is the state after the Kalman prediction step. Introducing the notation $\mathbf{q}_t = [\mathbf{x}_t \quad \mathbf{m}, \quad \mathbf{x}_t]^\top$ writing now the first-order Taylor expansion of $\xi(\mathbf{x}_t, \mathbf{m}, \mathbf{w}_t, \kappa)$ at the linearization point $(\overline{\mu}_{t,\mathbf{x}}, \overline{\mu}_{t,\mathbf{m}}, \mathbf{z}_t^i)$, which is the mean of $\mathbf{q}_t$, will yield the covariance mapping:

$$\mathbf{\Sigma}_t = \Xi_t \begin{bmatrix} \overline{\mathbf{\Sigma}}_{t,\mathbf{xx}} & \overline{\mathbf{\Sigma}}_{t,\mathbf{xm}} & \mathbf{0}_{3\times 2} \\ \overline{\mathbf{\Sigma}}_{t,\mathbf{mx}} & \overline{\mathbf{\Sigma}}_{t,\mathbf{mm}} & \mathbf{0}_{N\times 2} \\ \mathbf{0}_{2\times 3} & \mathbf{0}_{2\times N} & \mathbf{R}_t^i \end{bmatrix} \Xi_t^\top \tag{C.3}$$

Matrix $\mathbf{\Sigma}_t$ is the covariance of the augmented state $\mathbf{y}_t$ and Ξ_t is the Jacobian:

$$\Xi_t = \frac{\partial \xi(\mathbf{x}_t, \mathbf{m}, \mathbf{w}_t, \kappa)}{\partial \mathbf{q}_t}\bigg|_{\overline{\mu}_{t,\mathbf{x}},\, \overline{\mu}_{t,\mathbf{m}},\, \mathbf{z}_t^i} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times N} & \mathbf{0}_{N\times 2} \\ \mathbf{0}_{N\times 3} & \mathbf{I}_{N\times N} & \mathbf{0}_{N\times 2} \\ \mathbf{F}_{t,\mathbf{x}} & \mathbf{0}_{2\times N} & \mathbf{F}_{t,\mathbf{w}} \end{bmatrix} \tag{C.4}$$

Performing the multiplications leads to the sought expression for the augmented state covariance matrix, suitable for efficient implementation:

$$\mathbf{\Sigma}_t = \begin{bmatrix} \overline{\mathbf{\Sigma}}_{t,\mathbf{xx}} & \overline{\mathbf{\Sigma}}_{t,\mathbf{xm}} & (\mathbf{F}_{t,\mathbf{x}}\overline{\mathbf{\Sigma}}_{t,\mathbf{xx}})^\top \\ \overline{\mathbf{\Sigma}}_{t,\mathbf{mx}} & \overline{\mathbf{\Sigma}}_{t,\mathbf{mm}} & (\mathbf{F}_{t,\mathbf{x}}\overline{\mathbf{\Sigma}}_{t,\mathbf{xm}})^\top \\ \mathbf{F}_{t,\mathbf{x}}\overline{\mathbf{\Sigma}}_{t,\mathbf{xx}} & \mathbf{F}_{t,\mathbf{x}}\overline{\mathbf{\Sigma}}_{t,\mathbf{xm}} & \mathbf{F}_{t,\mathbf{x}}\overline{\mathbf{\Sigma}}_{t,\mathbf{xx}}\mathbf{F}_{t,\mathbf{x}}^\top + \mathbf{F}_{t,\mathbf{w}}\mathbf{R}_t^i\mathbf{F}_{t,\mathbf{w}}^\top \end{bmatrix} \tag{C.5}$$

We used [90] as the main reference in writing this appendix.