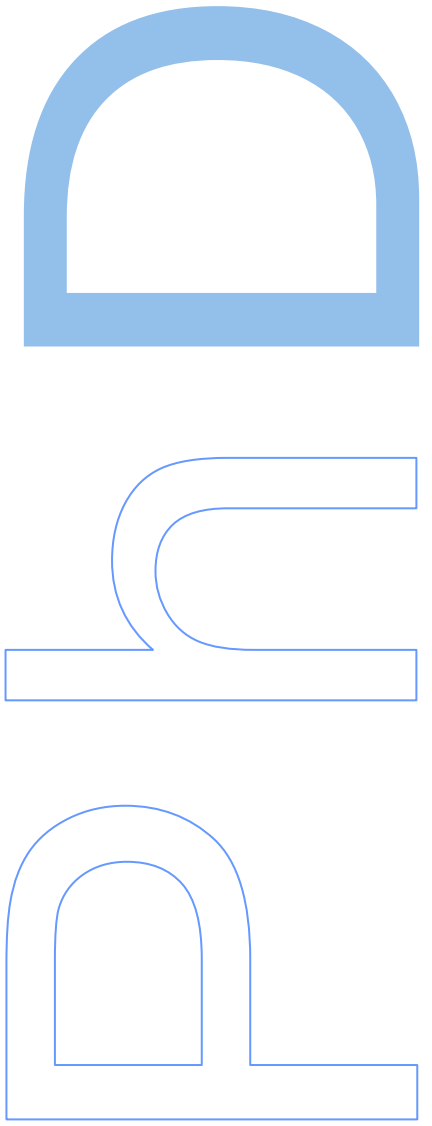
Online range-based SLAM and active vision for robotic systems

Rômulo Teixeira Rodrigues
Programa Doutoral em Matemática Aplicada
Departamento de Matemática
2022

Orientador
António Pedro Aguiar, Professor Catedrático,
Faculdade de Engenharia, Universidade do Porto

Coorientador
António Pascoal, Professor Associado,
Instituto Superior Técnico, Universidade de Lisboa



UNIVERSIDADE DO PORTO

DOCTORAL THESIS

Online range-based SLAM and active vision for robotic systems

Author:
Rômulo T. RODRIGUES

Supervisor:
A. Pedro AGUIAR

Co-supervisor:
António PASCOAL

A thesis submitted in fulfilment of the requirements
for the degree of PhD in Applied Mathematics

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

May 19, 2022

Acknowledgements

I would like to express my deepest appreciation to my supervisors Prof. A. Pedro Aguiar and Prof. António Pascoal for guiding me through this enriching experience. The insightful discussions and advice will follow me in my professional career.

I am also grateful to Dr. Pedro Miraldo and Prof. Dimos V. Dimarogonas for receiving me at KTH and the innumerous discussions regarding the active vision problem.

I also had great pleasure of working with Dr. Nikolaos Tsiogkas (KU Leuven) in the SLAM problem and Pedro Roque (KTH) and Dr. André Mateus (ISR) in the depth estimation experiments.

Thanks should also go to all the SYSTEC Team, in particular to Paulo Lopes, which has always backed me up in administrative matters. I also wish to thank Doctoral program PDMA-NORTE-08-5369-FSE-000061 for financially supporting this work.

I also would like to thank my C2SR colleagues and friends for their companionship in all these years and the ones yet to come.

Last but not least I am extremely grateful for my family (on both sides of the Atlantic) whose unconditional love and support was paramount to my daily well-being.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

PhD in Applied Mathematics

Online range-based SLAM and active vision for robotic systems

by Rômulo T. RODRIGUES

Online Simultaneous Localization and Mapping (SLAM) addresses the problem of concurrently building a map of the environment and estimating the current pose of the robot within this map. The standard solution for online SLAM represents the world in a discrete map known as occupancy-grid. However, in addition to not scaling well for large environments, this approach also suffers from accuracy loss due to cell/obstacle misalignment. The first part of this thesis aims at improving online SLAM by representing the world in a continuous manner, using B-splines. Two efficient SLAM techniques that tackle the aforementioned limitations of occupancy-grid maps are presented, namely B-spline curve SLAM and B-spline surface SLAM. The former describes the map in a compact manner, fitting well in long-term autonomous operations where storage is a concern. The latter keeps the same advantages as occupancy-grid approaches, while improving the map consistency by avoiding measurement discretization. Simulations, public data set evaluations and experiments with a real-life robot show that B-spline Surface SLAM outperforms online SLAM methods in the literature, and performs similarly as methods relying on offline optimizations.

The second part of this thesis focuses on depth estimation for single camera systems, aiming at actively controlling the camera to improve the depth estimation with formal guarantees of convergence. Current techniques providing theoretical guarantees require the actuation to be such that a projected point appears in the origin of the image plane. This does not scale well in multiple point scenarios, as this restriction can only be met for one point at a time. This thesis proposes actuation policies that manage to improve depth estimation relaxing this requirement. Results on the visual servoing problem show that adding an active convergence phase increases the chance of success.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

Programa Doutoral em Matemática Aplicada

SLAM *online* e visão ativa para sistemas robóticos

por Rômulo T. RODRIGUES

Localização e Mapeamento Simultâneos (SLAM) *online* aborda a estimação de um mapa do ambiente, bem como a posiçãp e orientação do robô, em simultâneo. Soluções convencionais representam o mundo através de um mapa discreto. No entanto, além de não escalarem bem para ambientes de grandes dimensões, estas soluções sofrem de imprecisões, devido a desalinhamentos entre as células do mapa discreto e os obstáculos no mundo real. A primeira parte desta tese tem como objectivo melhorar os métodos de SLAM *online*, explorando mapas contínuos com curvas *B-spline*. A fim de evitar os problemas mencionados dos métodos atuais, são propostas duas técnicas eficientes, nomeadamente SLAM de curvas *B-spline* e SLAM de superfícies *B-spline*. A primeira descreve o mapa de forma compacta, adequando-se a operações de longa duração com memória de armazenamento limitada. A segunda técnica, comparada com métodos discretos, permite obter um mapa mais consistente, ao evitar discretizar as medidas do sensor. Simulação, dados públicos e experiências com um robô demonstram que o método proposto de SLAM de superfícies *B-spline* atinge uma precisão superior à de outros métodos *online* na literatura, e apresenta um desempenho semelhante ao de métodos *offline*.

A segunda parte desta tese explora a estimação de profundidade para sistemas visuais mono-câmara. O objetivo consiste em controlar a câmara ativamente de forma a melhorar a estimação, com garantias de convergência. Técnicas convergentes atuais exigem que a atuação seja tal que um ponto observado seja projetado na origem do plano da imagem. Estes métodos não se adequam a casos com múltiplos pontos, pois esta condição pode ser aplicada a apenas um ponto de cada vez. Nesta tese são propostas regras de controlo que permitem relaxar essa restrição. Resultados demonstram que o problema de servovisão é beneficiado ao acrescentar uma fase activa de convergência.

Contents

List of Figures

List of Tables

List of Algorithms

Symbols

$\{M\}$	Map (inertial) frame	
$\{B\}$	Body-fixed frame	
$\{C\}$	Camera-fixed frame	
$\mathbf{p} \in \mathbb{R}^2$	Position of the vehicle in $\{M\}$	m
$\psi \in \mathbb{R}$	Orientation of the vehicle in $\{M\}$	rad
$\boldsymbol{\xi} \in \mathbb{R}^3$	Pose of the vehicle $\{M\}$	m, rad
$\boldsymbol{u} = [v_x, v_y, w_z] \in \mathbb{R}^3$	Control input of the vehicle (velocity)	m/s, rad/s
$\mathbf{v} = [v_x, v_y, v_z] \in \mathbb{R}^3$	Linear velocity of the camera	m/s
$\mathbf{w} = [w_x, w_y, w_z] \in \mathbb{R}^3$	Angular velocity of the camera	rad/s
$\mathbf{b}^d$	B-spline function of degree d	
$\boldsymbol{\phi}$	B-spline tensor	
$\mathbf{c}$	control points	
f	focal length of the camera	
u_0, v_0	camera optical center	
ρ_u, ρ_v	pixel scaling factor	

Glossary

SLAM	Simultaneous Localization and Mapping
GPS	Global Positioning System
IMU	Inertial Measurement Unit
2D	two-dimensional
3D	three-dimensional
LiDAR	Light Detection and Ranging
TSDF	Truncated Signed Distance Function
EKF	Extended Kalman Filter
IDC	Iterative Dual Correspondence
PF	Particle Filter
RBPF	Rao-Blackwellized Particle Filtering
SfM	Structure from Motion
NLS	Nonlinear Least Squares
ADNN	Average Distance to the Nearest Neighbor
w.r.t.	with respect to

Chapter 1

Introduction

This chapter provides a brief overview of this thesis: the importance of the research topic, the considered open challenge, the investigated hypotheses, and the accomplishments. Section 1.1 motivates the research towards autonomous mobile robots and perception. Sections 1.2 and 1.3 provide a short introduction to the perception problem known as Simultaneous localization and Mapping (SLAM) and 3D depth estimation, respectively. Section 1.4 presents the contributions and publications developed throughout this thesis. Finally, Section 1.5 addresses the structure of the remainder of this document.

1.1 Motivation

The robotics market has two major segments: industrial robots and autonomous mobile robots. Industrial robots are fast, reliable, and accurate. They come in different configurations, e.g., articulated and cartesian, being well suited for different industrial tasks such as welding and assembling. Driven by the high demands of production lines all over the world, industrial robots have been dominating the largest portion of the robotics market share for the last decades. However, a change is likely to take place soon. The technological advisory company ABI Research forecasts that the number of shipments of autonomous mobile robots will increase from less than half million in 2020 to about 6 million in 2030 [1]. According to the same report, these shipments represent a revenue of 202 billion euros for the autonomous mobile robot sector, compared to 35 billion euros for the industrial robot segment. So what exactly is an autonomous mobile robot?

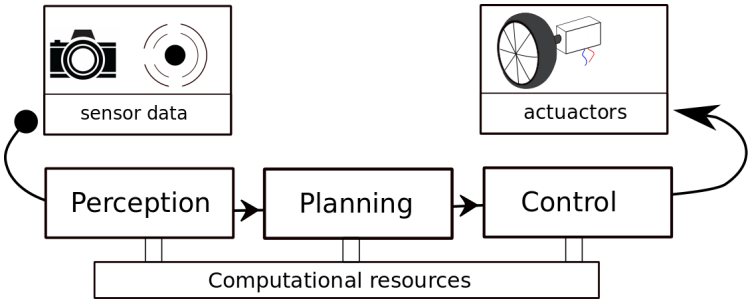


FIGURE 1.1: The main components in the perception, planning, and control architecture. The workflow is from the left (sensors) to the right (actuators).

Autonomous mobile robots

Autonomous mobile robots, also known as autonomous vehicles or sensor-controlled robots, are robotics systems capable of travelling through the environment to achieve a specific goal in an automated or semi-automated manner. Some impactful industrial and civilian applications for autonomous vehicles include logistics in warehouses, inspection of plantation fields, and disinfection of hospitals. Autonomous mobile robots are equipped with on-board sensors, processing power, and actuators. These hardware components may vary in nature and quality and have to be chosen accordingly to the desired application and budget. Regarding the software architecture, the basic functionalities of sensor-controlled robots are typically split into three modules: perception, planning, and control [2]. This software architecture is illustrated in Fig. 1.1, with the information flow going from the top left (sensors) to the top right (actuators)

Sensors provide observations of the robot and environment. On the other end, actuators provide means to interact with the world. The perception block, also known as state estimation, processes the sensory data for building a meaningful estimation of the state of the world and the robot. The planning task deliberates the sequence of actions that the vehicle has to follow to achieve a spatial goal, e.g., a list of waypoints, a path, or a trajectory. The control task ensures that the vehicle follows the motion planning sequence by computing the actuation inputs, e.g., velocity, acceleration or torques. The perception block provides feedback for both planning and control modules. For this reason, a poor perception typically compromises the performance of the remaining tasks. Before diving deeper into state estimation, which is the focus of this thesis, the interplay between these three blocks is analyzed for a well-known application: robotic vacuum cleaners.

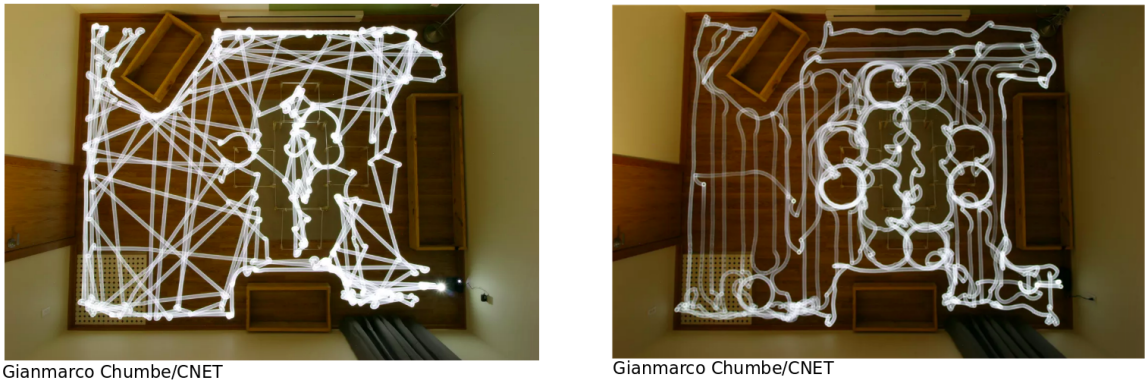


FIGURE 1.2: Comparing vacuum cleaner robots in the same environment: (a) shows the trajectory of a robot that does not build a model of the environment. (b) shows the trajectory of a robot that builds a map using an on-board range sensor. Source: [3]

Domestic robots

The first generation of vacuum cleaner robots is equipped with a limited and primitive set of sensors such as collision and cliff detectors. The perception task reasons whether the robot has collided with an obstacle or it is close to a staircase. The planning and control are basic: the robot either moves forward or turns in a different direction if moving forward is not a safe or possible option. This leads to poor coverage, since the robot is not able to acknowledge whether it has previously visited a region or not. More modern (and expensive) vacuum cleaner robots are equipped with range and image sensors. Using the data collected by these sensors, modern perception algorithms are able to build a map of the surrounding environment and localise the robot within the map. Therefore, it is possible to plan an optimal coverage trajectory using for example a lawnmower pattern. More sophisticated behaviours, such as evading obstacles, docking for automatically recharging, and deliberately avoid certain regions, are also possible. An experiment comparing the performance of different vacuum cleaner robots in an industry-standard testing room is presented in [3]. Figure 1.2a illustrates the trajectory of a robot without a range sensor, while Fig. 1.2b shows the trajectory of a robot equipped with a LiDAR range sensor. The former robot takes more than an hour to clean the room, while the latter required less than 25 minutes. Building and using a map of the environment not only boosts the coverage performance, but also reduces the consumed time and energy.

The perception of most robots equipped with a range sensor - including the one exemplified here - is based on a technique known as range-based SLAM and it is the first focus

of this thesis. The second focus of this thesis, 3D depth estimation, aims at recovering the 3D structure of the environment and it is paramount for vision-based robots.

1.2 Research Problem: SLAM

Simultaneous localization and Mapping, or simply SLAM, is a state estimation technique for concurrently estimating the pose of a mobile sensor (localization task) and building a model of its surrounding environment (mapping task). SLAM has been widely adopted for deploying sensor-controlled robots in scenarios where an external referencing system works poorly or is not available at all. For example, global positioning systems (GPS) do not work well in indoor environments like factories, hospitals, and houses.

The challenge in SLAM arises from the fact that localization and mapping are tightly coupled. The pose of the robot is estimated using the map. In turn, the map is built using the estimated pose of the robot. As a consequence, if either localization or mapping degrades, the other also quickly deteriorates. Since SLAM provides feedback to a number of tasks, including planning and control, a poor SLAM solution may lead to undesired behaviour, such as mission failure or equipment damage.

The SLAM problem has received considerable attention over the last decades, leading to theoretical and computational breakthroughs. Different techniques using a variety of sensors, such as sonars, LiDARs, and cameras, have been developed for ground [4, 5], aerial [6], and marine vehicles [7]. Robustness has been achieved by relying on a two-stage architecture, namely the *front-end* and the *back-end* stages [8]. The front-end stage, also known as online SLAM, is responsible for processing the raw sensor data and providing state estimation at least as fast as the sensor rate. Then, at lower rates, the back-end optimizes the overall state estimation using graph-based optimization techniques. The graph to be optimized at the back-end stage is fed with the pose estimated at the front-end stage. Sources such as wheel encoders and Inertial Measurements Units (IMU) potentially enrich the graph informativeness, leading to better results. An important difference between the two stages is that the back-end works offline. Thus, it may improve past estimations using smoothing and backward estimation techniques. However, it does not deliver real-time feedback, which is important for tasks such as motion control.

The work by Cadena and colleagues [8] presents a solid overview of the SLAM problem, covering both past and recent strategies. In particular, they discuss several SLAM

challenges which are yet to be solved, such as long-term autonomy (scalability and robustness), metric maps, semantic maps, active SLAM, theoretical tools, sensors, and machine learning. The reader is referred to the aforementioned work for an introductory discussion on each of these open challenges. This thesis targets the issues related to metric map models. Before diving deeper into this problem, the scope of the work is delimited.

Scope

The first focus of this thesis is to **propose alternative metric mapping strategies** that are well-suited for the **front-end stage of a range-based SLAM**. In other words, the proposed methods aim at overcoming the problems faced by current map models within the context of SLAM. The output of the methods presented here can be used to build a graph which is fed into a back-end approach such as [9]. However, back-end optimization is not within the scope of this work.

Open challenge: metric maps

The IEEE Standard for robot data representation [10] classifies 2D maps in two major categories: topological and metric. Topological maps are often represented by edges and nodes of a graph that describe a connectivity relationship. A metric map encodes the geometry of the environment as perceived by the sensors and can be further categorised into three groups: feature-based, geometric, and discrete maps. The current predominant paradigm in SLAM belongs to the latter group and is known as discrete occupancy grid map, shortened here as discrete grid map or occupancy grid map. An occupancy grid map splits the world into cells of the same size. Each cell covers a region of the environment and its value denotes the probability of that region being occupied. Figure 1.3 shows the grayscale image representation of two occupancy grid maps. The darker a pixel is, the more likely it is to correspond to occupied space. Loosely speaking, black pixels represent obstacles, white pixels free area, and grey pixels unexplored (or inaccessible) regions for which it is equally likely to be free or occupied area.

The occupancy grid map was first proposed by Moravec and Elfes [11] in the late 1980's for robot navigation using sonar data. However, it was only in the last decade that occupancy grid maps have become widely adopted by the SLAM community. GMapping [4], Hector-SLAM [5], and Google Cartographer [12] are examples of popular SLAM strategies that have occupancy grid maps at their front-end stage. The main advantage of

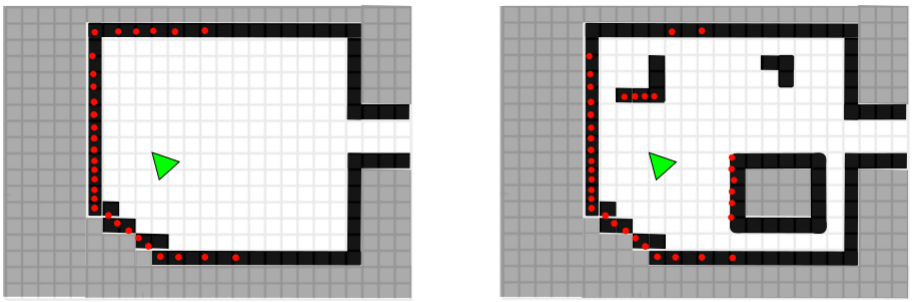


FIGURE 1.3: Discrete occupancy grid maps: (a) shows the model for an empty room and (b) illustrates the same room populated with obstacles. The darker a cell is, the more likely it is to correspond to occupied space. The green arrow represents the robot and the red dots represent valid scan measurements.

discrete grid maps is the efficiency in updating and evaluating the occupancy values of the cells. Both tasks are important for building a map and localising the robot within it. Despite its acceptance, occupancy grid-based SLAM has two major drawbacks:

1. **Memory consumption:** A considerable amount of memory is wasted in representing sparse environments and non-visible areas.

2. **Discrete resolution:** Even though measurements have a high resolution, localization accuracy is limited by the discrete (lower resolution) nature of the map.

The memory consumption problem is tightly linked to the scalability issue raised in [8]. In order to take advantage of modern range sensors, occupancy grid maps can have high resolution. The resolution of a cell-based map increases as the size of its cells decreases. However, since the occupancy grid map divides the world into equally-sized cells, large areas without meaningful information still consume a considerable amount of storage memory. Thus, discrete grid maps do not scale well for large environments. This is shown in Fig. 1.3a, which represents an empty room. Figure 1.3b illustrates the same room populated with obstacles. Both maps require the same amount of memory despite having a different amount of obstacles. Furthermore, note that areas outside the room and in the interior of obstacles also need to be represented, consuming storage memory.

The second problem concerns the discrete nature of the map. Range sensors provide high resolution measurements, which are discretized into cell resolution. This leads to approximation errors due to misalignment between a measurement and its corresponding cell in the map. For example, in Fig. 1.3, the discrepancy between an obstacle as registered in the map (black pixels) and the actual sensor measurements (red dots) is particularly large for the bottom left wall of the room. The difference between the map

in memory and the measurements becomes critical when performing localization. This is because the standard localization technique consists in computing the pose that best aligns the scan measurements and the map (scan-to-map alignment). An interesting aspect which is further explained in Chapter 4 is that scan-to-map alignment is typically done via gradient-descent methods. However, the limited precision of discrete maps hinders the computation of derivatives (gradients). A work-around is to obtain a continuous map with sub-cell resolution by interpolating the discrete grid map, e.g., bi-linear interpolation [5] or bi-cubic interpolation [12].

Hypotheses

In terms of accuracy, the performance of discrete grid SLAM could be improved in the front-end by increasing cell resolution. However, in practice this is only possible up to an extent due to limited on-board memory. Therefore, most recent research efforts have been drifting either towards alternatives for metric maps or towards back-end optimizations, such as robust loop-closure detection.

The hypothesis considered here is that it is possible to **improve the performance of SLAM at the front-end stage** by using **a map that effectively aligns with the high resolution nature** of sensor measurements. The hypothesis claims that the two-step process performed in classic occupancy grid SLAM - discretization followed by interpolation - may gradually lead to information loss and SLAM deterioration. For this reason, special attention is given to continuous functions which may have the potential to be viable alternatives for discrete grid maps. In particular, we investigate the use of B-spline functions. We believe that SLAM may take advantage of the inherent properties of B-spline functions for enhancing the estimation of the map of the environment and the pose of the robot. For this purpose, tailored suited tools for mapping and localization using B-spline have to be developed.

The first strategy, presented in Chapter 5, explores B-spline curves (see Fig 1.4a). B-spline curves are compact in terms of memory, yet are able to represent a rich variety of geometric shapes. Therefore, this approach tackles the memory consumption problem and avoids discretization.

The second strategy, presented in Chapter 6, resorts to B-spline surfaces (see Fig 1.4b). A B-spline surface requires a grid of control points, but no discretization is evolved in the process. Consequently, the surface is able to store with high precision the coordinates in

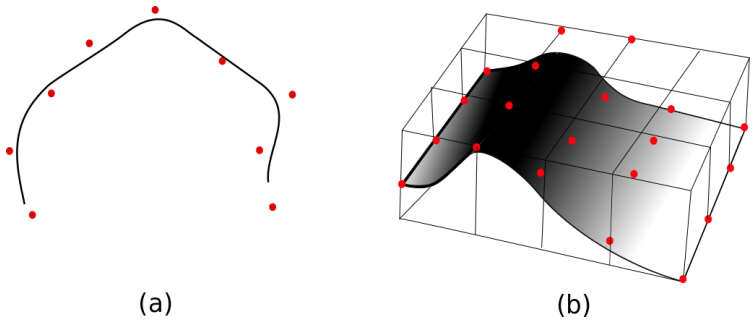


FIGURE 1.4: B-spline: (a) illustrates a 2D B-spline curve and (b) shows the 3D view of a B-spline surface. The control points, shown in red dots, play an important role in defining the shape of both the B-spline curve and surface.

which a measurement is observed. Therefore, B-spline surface maps have the potential to tackle the discrete resolution problem of discrete grid maps. In theory, it does not offer an improvement in terms of memory consumption.

1.3 Research Problem: 3D Depth Estimation

For vision controlled robots recovering the 3D structure of the scene is key for accomplishing tasks such as visual servoing and visual SLAM. The recovering process is required because projecting objects in the 2D image plane of a camera leads to an unknown scale, called here as depth. The problem has been investigated in the last decades by the computer vision and the robotics community, and well-established solutions are currently available. For instance, triangulation is the gold standard for objects registered with a calibrated stereo pair, i.e., two calibrated cameras for which the relative pose (baseline) of one camera with respect to the other is known, [13]. However, for monocular systems the problem is not trivial and additional information about the environment is required. According to the additional information exploited, strategies for recovering the 3D structure can be broadly classified into two categories: model-based and model-free.

Model-based approaches rely on some previous knowledge about the geometry (or semantics) of the scene, such as the dimensions of the object being observed or its 3D model, e.g., [14]. On the other hand, model-free methods do not require previous knowledge of the scene. Instead, they make use of multiple view-points with significant overlapping area and the relation between these frames. This is the case for a robotic system with an onboard camera, where the relation between two or more camera frames can be obtained from proprioceptive sensors, such as IMU and wheel/joint encoders, or extereoceptive sensors, such as GPS and LiDAR. The relation between frames can be described by 1)

the relative pose (translation and orientation) between them or 2) the dynamics of the camera to move from one view-point to the other. Model-free strategies that rely on the relative pose between frames are called *geometric* methods, while the ones that explored the dynamics of the camera are named *incremental* or *filtering* methods.

Geometric methods are correlated with the research problem known as structure-from-motion (SfM). The term SfM has its roots in the computer vision community (see [15]), in particular in the field of photogrammetry that studies physical properties of the environment based on images. In general, these methods have two stages. In the first stage, the goal is to estimate the depth of the 3D points from an ordered or unordered sequence of image frames for which the relative pose is know - otherwise the structure of the scene can only be recovered up to an unknown scale. This is achieved via n-view triangulation using linear least-squares or direct linear transform (DLT) [16]. The second stage strives for map consistency via offline batch optimization techniques, e.g., bundle adjustment [17]. The goal is to minimize the re-projection error, i.e., the error between the actual measurements and the expected feature positions obtained by projecting the estimated 3D points back in the image plane (see [13] for more detail). This is typically formulated as an optimization problem and solved using non-linear least square solvers such as Levenberg–Marquardt. Offline methods have had success, but 1) triangulation suffers from small baseline displacements, and 2) it lacks formal guarantees of convergence since bundle adjustment require a good initial guess to avoid local minima.

Filtering or *incremental-based* methods (e.g., [18]) explicitly consider the dynamics of 3D points tracked across a sequence of continuously acquired images. In contrast to geometric methods, incremental strategies are typically performed online and have low storage memory requirements due to the iterative nature of online filters, such as non-linear estimators [19], Luenberger observers [20], and Kalman Filters [21]. Furthermore, when compared to geometric methods, filtering techniques are more robust against small baseline displacement and pure camera rotation because neither situations lead to a singularity but to a lack of excitement. Last but not least, as it is shown in the aforementioned incremental works, it is possible to derive guarantees of convergence of the filter given that a particular set of constraints is satisfied. These constraints may be satisfied by chance as a result of the motion of the camera or enforced by controlling the motion of the camera. This is the key difference between passive and active vision.

In *passive* vision, the camera motion is not optimized to map the 3D environment. That

is to say, the convergence of the estimator is not considered in the control loop. In contrast, *active* vision techniques couple control with perception to assist the environment's reconstruction. For example, the authors in [22] include the goal of reconstructing 3D points, cylinders, straight lines, and spheres in the control policy of the camera. More recent, the active strategy in [23] consists in choosing the control actions that maximize an observability index for an observed feature, which is given by a persistency of excitation condition. The aforementioned strategy has been extended for the active estimation of 3D planes (in [24]) and 3D straight lines (see [25]).

Scope

The second focus of this thesis thesis is **active incremental depth estimation**, which tightly couples the perception and the control loop for improving the estimation of the 3D structure of the scene. In particular, we investigate the requirements for the depth estimation of 3D points with theoretical guarantees of convergence.

Open challenges

Active depth estimation for *multiple* points with *formal guarantees of convergence* is still an open problem. In the last few years, the active approach described in [24] has gained popularity. It consists in choosing the control actions that maximize an observability index for an observed feature, which is given by a persistency of excitation condition. The control policy is such that the estimated feature moves to the center of the image plane. Since only a single point may occupy the origin of the image frame at a time, this imposes a limitation for visual servoing and SLAM applications, where one typically wants to estimate multiple points at the same time. This issue is observed in [26], where the active estimator proposed in [23] is employed in a visual servoing application. The authors aimed at maximizing the persistency of excitation condition for the tracked points, but could only achieve local convergence guarantees.

Hypothesis

Chapter 8 investigates the control actions that ensure global convergence of a point relaxing the constraints that the point must be driven to the origin of the image frame. This theoretical result is then applied in Chapter 9 in the single point and multiple points visual servoing task. Most applications with multiple points will have conflicting objectives,

and it is unlikely that the theoretical constraints required for global convergence can be imposed for all points simultaneously during a given task. The hypothesis of this thesis is that, if global guarantees can be enforced for multiple points during a transient period, the solution will likely approach a local region of convergence, yielding better results in tasks such as visual servoing.

1.4 Contributions and Publications

1.4.1 Contributions

The core contribution is the development of two continuous metric map models applied to the mapping problem:

1. **B-spline curve map**: The B-spline curve map is a deterministic and compact continuous representation of the environment designed for long-term operation in vast areas. It is an alternative for geometric maps. Results shows that the memory required is considerable lower compared to discrete grid maps - up to 10 times in some scenarios.

2. **B-spline surface map**: The B-spline surface map is a probabilistic and accurate continuous representation of the environment which does not require discretizing measurements. Results show that B-spline surface maps are more accurate than occupancy grid maps for the same storage memory requirements. This is because B-spline maps are less punished by misaligment between the measurements and grid. The B-spline surface map can be sampled at any resolution for obtaining a discrete grid map. Thus, it is easy to integrate with existing motion planning and control software developed for discrete grid map.

Both B-spline curve and surface maps can be employed in tasks such as planning and collaborative robotics. However, within the context of sensor-controlled robots, the primary use of a map is for SLAM. For this reason, tailored-suit localization methods are developed for both B-spline maps. The following contributions are also presented here:

3. **B-spline curve SLAM**: localization is obtained by tracking the curves in the map and minimising the alignment error between the mapped curves and the actual measurements. Experimental results show that it is possible to obtain an accurate localization with a low memory consumption map.

4. **B-spline surface SLAM**: localization is obtained by matching the sensor measurements against regions of the surface that are likely to correspond to obstacles. Simulation and experimental results show that B-spline SLAM outperforms stand front-end approaches that rely on a discrete grid map. Since the map is able to retain better the position of measurements, it allows for a large basin of convergence when doing gradient-descent during localization. As a consequence, for the same localization accuracy, B-spline surface SLAM ends up consuming less memory than an occupancy-grid map. Surprisingly, online B-spline SLAM, without any back-end optimization, was also able to compete with discrete grid SLAM using back-end approaches, outperforming some of them in real experiments.

The methods developed in this thesis were designed taking into account that the SLAM front-end must work at least as fast as the sensor rate speed (online SLAM). We had to develop a tailor-made B-spline library that addresses our computational constraints. The Python implementation of the B-spline library and the online B-spline surface SLAM are available on a public repository:

5. **B-spline library and open-source SLAM code**: The theoretical results presented in this thesis are backed up by a publicly available package with examples, including ROS integration. A computational efficient B-spline library which can benefit other applications is made available. Link to the repository: https://github.com/C2SR/spline-slam.

The problem of depth estimation with theoretical guarantees of convergence has been relaxed by allowing the point to be outside of the origin of the image frame:

6. **Active depth estimation**: The proposed depth estimation framework provides theoretical guarantees for the estimation of points that are not located in the origin of the image frame. Applications for a single point and multiple points are presented.

1.4.2 Publications

The research developed throughout this thesis was published in well-renowned peer-reviewed robotic journal and conferences.

Journals

[SUB22] R. T. Rodrigues, P. Miraldo, and A. Pedro Aguiar, "On the Guarantees of Incremental Depth Estimation and its Coupling with Visual Servoing", The International Journal of Robotics Research (IJRR) (submitted) .

[RAL21] R. T. Rodrigues, N. Tsiogkas, A. Pascoal, and A. P. Aguiar, "Online range-based SLAM using B-spline surfaces", in IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 1958-1965, April 2021.

[RAL18] R. T. Rodrigues, M. Basiri, A. P. Aguiar and P. Miraldo, "Low-Level Active Visual Navigation: Increasing Robustness of Vision-Based Localization Using Potential Fields," in IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 2079-2086, July 2018.

Conferences

[OCEANS21] M. Reis, G. Andrade, F. Neves, P. Silva, R. T. Rodrigues, and A. P. Aguiar, "A ROS implementation of the situational awareness and maneuvering systems for an autonomous marine vessel". San Diego, CA, Sept. 2021 (Accepted).

[IROS20] R. T. Rodrigues, N. Tsiogkas, A. P. Aguiar, and A. Pascoal, "B-spline Surfaces for Range-Based Environment Mapping", in Proc. of IROS'20 - IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, CA, USA, Oct. 2020.

[ICRA20] R. T. Rodrigues, P. Miraldo, D. V. Dimarogonas and A. Pedro Aguiar, "Active Depth Estimation: Stability Analysis and its Applications," 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 2002-2008.

[IROS19] R. T. Rodrigues, P. Miraldo, D. V. Dimarogonas and A. P. Aguiar, "A Framework for Depth Estimation and Relative Localization of Ground Robots using Computer Vision," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 2019, pp. 3719-3724.

[AUV18] R. T. Rodrigues, A. P. Aguiar and A. Pascoal, "A coverage planner for AUVs using B-splines," 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV), Porto, Portugal, 2018, pp. 1-6.

[IROS18] R. T. Rodrigues, A. P. Aguiar and A. Pascoal, "A B-Spline Mapping Framework for Long-Term Autonomous Operations," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, 2018, pp. 3204-3209.

1.4.3 Robotics Competition

The B-spline library presented in Chapter 4 was employed in the navigation framework developed by the C2SR Atlantics team, which won the first place in the 2021 Njord - The Autonomous Ship Challenge[1], NTNU, Norway. The Atlantics team, formed by undergraduate and graduate student of the University of Porto, designed a navigation framework for an autonomous boat operating in a simulated environment - the 2021 competition was fully virtual. In the scope of the competition, the B-spline library was employed for motion planning, showing the potential of the package for other tasks not directly related to state estimation.

1.5 Structure

The remainder of this document is structured as follows:

- Chapter 2, *Mathematical Background*, introduces the notations, key operations, and algorithms that are employed in this document.
- Chapter 3, *Robotic Systems*, presents the class of robotic systems for which the proposed SLAM was designed for.
- Chapter 4, *B-Spline library*, introduces the main theoretical concepts of B-splines and the key ideas behind the computational efficient B-spline library implemented in Python3.
- Chapter 5, *Literature Review*, discusses the previous work related to range-based SLAM.
- Chapter 6, *B-Spline Curve SLAM*, presents the first strategy devised in this thesis: a continuous geometric SLAM based on B-spline curves.
- Chapter 7, *B-Spline Surface SLAM*, presents the second strategy: a probabilistic B-spline surface SLAM.
- Chapter 8, *Depth Estimation (theory)*, discusses active depth estimation and presents the proposed framework which relaxes current state-of-the-art constraints.

[1] https://www.njordchallenge.com/

- Chapter 9, *Depth Estimation (applications)*, applies the active filter presented in the previous chapter in single and multiple points applications.
- Chapter 10, *Conclusions*, addresses the final remarks and potential research directions for future work.

Chapter 2

Mathematical Background

This chapter presents the basic mathematical notation and theoretical background that are transversal to the remainder of this document and provides a quick refresh on key operations and state estimation algorithms. It is assumed that the reader has a basic background in linear algebra and is familiar with operations such as dot (scalar) product, matrix multiplication, inversion, and determinants. If this is not the case, consider reading first a linear algebra introductory book, e.g., [27].

2.1 Basic notations and definitions

The basic notations and definitions adopted in this document are as follow.

Real set: $\mathbb{R}$ denotes the set of real numbers, $\mathbb{R}^n$ the set of n-dimensional vectors, and $\mathbb{R}^{n\times m}$ is the set of $n \times m$-dimensional matrices.

Scalars, vectors, matrices: Scalar values are written in lower-case letter. Let $x \in \mathbb{R}$ be a scalar variable, then $|x|$ represents its absolute value. The floor function is denoted as $\lfloor x \rfloor$ and it returns the largest integer less than or equal to x. Vectors are typed in lower-case bold letter. Unless otherwise stated, vectors are columns. The vector $\boldsymbol{x} = [x_1, ..., x_n]^T \in \mathbb{R}^n$ has n rows and the entry x_i lies at the *i-th* row. The upper trailing T stands for the transpose of a vector. A vector can be described in a shorter notation as $\boldsymbol{x} = [x_i]_{i=1}^n$. The norm of $\boldsymbol{x}$, denoted by $\|\boldsymbol{x}\|$, considered here is the class of p-norms, given by

$$\|\boldsymbol{x}\|_p = (|x_1|^p + ... + |x_n|^p)^{1/p}, \quad 1 \leq p \leq \infty$$

and

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i|.$$

In particular, the notation $\|\boldsymbol{x}\|$ stands for the Euclidean norm:

$$\|\boldsymbol{x}\| = \|\boldsymbol{x}\|_2 = (|x_1|^2 + ... + |x_n|^2)^{1/2} = (\boldsymbol{x}^T\boldsymbol{x})^{1/2}.$$

The vectors **1** and **0** contains all their entries equal to 1 and 0, respectively.

Matrices are typed in upper-case letters. The matrix $A \in \mathbb{R}^{n\times m}$ has n rows and m columns. The entry a_{ij} sits at the *i-th* row and *j-th* column. A shorter notation for the matrix A is $A = [a_{ij}]_{i=1,j=1}^{n,m}$. The transpose of a matrix, represented by A^T, contains the elements of A reflected over its main diagonal. Transposing a matrix multiplication reverses its order, i.e., $(AB)^T = (B^TA^T)$, where A and B are matrices of appropriate dimensions. A square matrix has the same number of rows and columns. A square matrix S is said to be symmetric if it is equal to its transpose, i.e., $S = S^T$.

Definition 2.1. A symmetric matrix $S \in \mathbb{R}^{n\times n}$ is *positive definite* if and only if

$$\boldsymbol{x}^TS\boldsymbol{x} > 0, \quad \forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq \mathbf{0}.$$

This fact is stated as $S \succ 0$.

Definition 2.2. A symmetric matrix $S \in \mathbb{R}^{n\times n}$ is *positive semi-definite* if and only if

$$\boldsymbol{x}^TS\boldsymbol{x} \geq 0, \quad \forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq \mathbf{0}.$$

This fact is stated as $S \succeq 0$.

The identity matrix $I \in \mathbb{R}^{n\times n}$ contains 1's in the main diagonal and the other elements are 0. The inverse of a square matrix A, denoted as A^{-1}, is unique and respects the property $A^{-1}A = AA^{-1} = I$, where I is an identity matrix with proper dimension. A matrix is said to be singular if it is not invertible, and non-singular otherwise. The existence of an inverse may be verified by the determinant of a matrix – if $det(A) = 0$, then A is singular[1].

The vectorization operator, defined as $\text{vec}(\cdot)$, applies a linear transformation that stacks the columns of a matrix A on top of each other, yielding a single column-vector:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nm} \end{bmatrix}, \quad \text{vec}(A) = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{nm} \end{bmatrix}.$$

[1]Due to numerically instability, it is first evaluated whether $|det(A)|$ is greater than a threshold before computing the inverse of matrix A.

The Kronecker product of two matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{p\times q}$ is denoted as $A \otimes B \in \mathbb{R}^{np\times mq}$, and it is defined as

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nm}B \end{bmatrix}. \tag{2.1}$$

The transposition is distributive over the Kronecker product, that is, $(A \otimes B)^T = (A^T \otimes B^T)$. Let A, B, and C be matrices of appropriate dimensions, one has that

$$\text{vec}(ACB) = (B^T \otimes A)\,\text{vec}(C).$$

The *slicing* operator allows to extract a sub-matrix from a matrix. Let A be a matrix, then the sub-matrix $A_{2:5,:}$ is composed of the rows 2, 3, and 4 and all the columns of matrix A.

Scalars, vectors, and matrices may represent quantities that are functions of time t, for example, $x(t), \boldsymbol{x}(t)$, or $X(t)$. Whenever clear from the context, the notation is simplified by stating the dependency on time with leading subscript, for example, $x_t, \boldsymbol{x}_t$, or X_t.

Functions: The notation $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ means that the input of a function f is a n-dimensional vector and its output is a m-dimensional vector. Whenever clear from the context, the arguments of a function are omitted, for example, f is equivalent to $f(\boldsymbol{x})$. The domain of the function f, denoted as $\mathbf{dom} f$ specifies the subset of $\mathbb{R}^n$ for which f is defined.

The function f is said to be continuous at a point $\boldsymbol{x}$ if $f(\boldsymbol{x}_k) \to f(\boldsymbol{x})$ as $\boldsymbol{x}_k \to \boldsymbol{x}$, that is, if given $\varepsilon > 0$ there is $\delta > 0$ such that

$$\|\boldsymbol{x} - \boldsymbol{y}\| < \delta \implies \|f(\boldsymbol{x}) - f(\boldsymbol{y})\| < \varepsilon.$$

The derivative of the continuous function $f(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_1(\boldsymbol{x}_m)]^T$ with respect to (w.r.t.) $\boldsymbol{x} = [x_1, \ldots, x_2]^T$ is known as the Jacobian matrix, and it is defined as

$$\frac{df(\boldsymbol{x})}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\boldsymbol{x})}{\partial x_n} \end{bmatrix}.$$

The Jacobian matrix describes how changes in the input impact the output. For example, if $\frac{\partial f_1(\boldsymbol{x})}{\partial x_1}$ is large (low), it means that changes on x_1 have a large (low) effect on

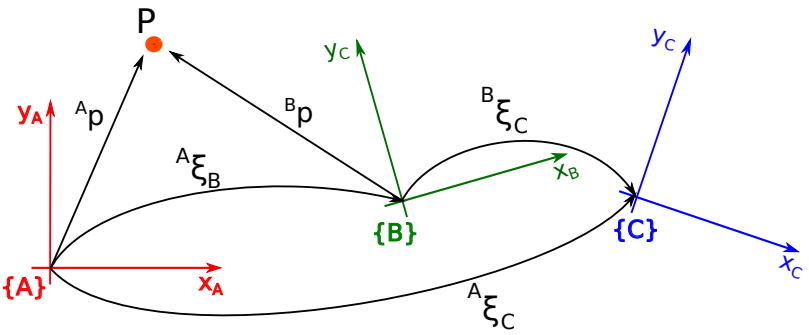


FIGURE 2.1: Coordination frames and relative poses

$f_1(\boldsymbol{x})$.

Coordination frames: A coordinate frame, also known as Cartesian coordinate system, is a set of orthogonal unitary axes that intersect in a point (called the origin of the coordination frame). Figure 2.1 shows three coordinates frames: $\{A\}$, $\{B\}$, and $\{C\}$. The term pose, which has been already introduced in the previous chapter, is the position and orientation of a coordinate frame (target) with respect to another coordinate frame (reference). The notation ${}^A\boldsymbol{\xi}_B$ denotes the pose of coordinate frame $\{B\}$ described in $\{A\}$. There are two important operators: composition and its inverse. Composition ($\oplus$) allows to compound relative poses, for example,

$$ {}^A\boldsymbol{\xi}_C = {}^A\boldsymbol{\xi}_B \oplus {}^B\boldsymbol{\xi}_C \tag{2.2} $$

is the pose of $\{C\}$ described in $\{A\}$ obtained via composition. The symbol $\ominus$ denotes the inverse operator and it defines the following equality:

$$ \ominus {}^A\boldsymbol{\xi}_B = {}^B\boldsymbol{\xi}_A \tag{2.3} $$

A typical manner to represent 2D poses is the vector $\boldsymbol{\xi} = [x, y, \psi]^T$, where (x, y) and ψ are a translational and rotational component, respectively. However, compounding poses in the vector representation is complex [28] and, for this reason, homogeneous transformation are often employed:

$$ \boldsymbol{\xi}(x, y, \psi) \equiv \begin{bmatrix} \cos\psi & \sin\psi & x \\ -\sin\psi & \cos\psi & y \\ 0 & 0 & 1 \end{bmatrix} \in SE(2), \tag{2.4} $$

where $SE(2)$ stands for special Euclidean group of dimension 2. Then, composition can

be computed as a product of matrices and the inverse operator by inverting the homogeneous transformation matrix, which is always non-singular.

The frame in which a vector is described in indicated by a leading superscript, e.g., ${}^{A}\boldsymbol{p}$ is the vector $\boldsymbol{p}$ described in $\{A\}$. A vector can be transformed from one frame to the other using the homogeneous transformation matrix as follows:

$$ {}^{A}\boldsymbol{p} = \begin{bmatrix} \cos {}^{A}\psi_B & \sin {}^{A}\psi_B & {}^{A}x_B \\ -\sin {}^{A}\psi_B & \cos {}^{A}\psi_B & {}^{A}y_B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{B}p_x \\ {}^{B}p_y \\ 1 \end{bmatrix} = R({}^{A}\psi_B){}^{A}\boldsymbol{p} + \begin{bmatrix} {}^{A}x_B \\ {}^{A}y_B \end{bmatrix}, \tag{2.5} $$

where $R({}^{A}\psi_B) \in SO(2)$ is a rotation matrix.

Probability: A probabilistic model describes an uncertain phenomenon using three components: a sample space, an event space, and a probabilistic law. The sample space is the set of all possible outcomes of an experiment. The event space is the subset of the sample space for whose the probability can be measured. The probability law or probability function assigns a probability that describes the likelihood of an event to happen. Let $\mathcal{X}$ be a sample space, $\mathcal{A}$ an event space, i.e., $\mathcal{A} \subseteq \mathcal{X}$, and $x, y \in E$ be measurable events. A valid probability law $p : \mathcal{X} \to \mathbb{R}$ must respect Kolmogorov's axioms [29]:

1. $p(x) \geq 0, \quad \forall x \in \mathcal{A}$;
2. $p(\mathcal{X}) = 1$.
3. $p(x \cup y) = p(x) + p(y)$, if $x \cap y = \emptyset$.

Definition 2.3. (Conditional probability) Given two events x and y, the conditional probability of x given y, denoted as $p(x|y)$, describes the likelihood of event x given the information described by y. For $p(y) > 0$, the conditional probability is defined as:

$$ p(x|y) = \frac{p(x,y)}{p(y)}. \tag{2.6} $$

Definition 2.4. (Total Probability Law) Let Y be set of all possible outcomes of an experiment, then

$$ \begin{aligned} p(x) &= \sum_{y \in Y} p(x|y)p(y) \qquad \text{(Discrete)} \\ p(x) &= \int p(x|y)p(y)dy \qquad \text{(Continuous)}. \end{aligned} \tag{2.7} $$

Definition 2.5. (Bayes rule) The Bayes rule is defined as

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{2.8}$$

It can be verified using Definition 2.3. The Bayes rule describes how to update a probability given evidence (observations, for example). The probabilities are typically named as follows:

- $p(x|y)$: posterior - probability of event x obtained after incorporating the evidence;
- $p(y|x)$: likelihood - probability of event y given event x happens;
- $p(x)$: prior - probability of event x before incorporating evidence;
- $p(y)$: marginal - probability of event y.

Definition 2.6. (Markov Assumption) The Markov assumes that the conditional probability of an event does not depend on past outcomes but only on the most recent one, that is,

$$p(x_t|x_{t-1}, x_{t-2}, \ldots, x_0) = p(x_t|x_{t-1}), \tag{2.9}$$

where x_t stands for the outcome at time t.

Definition 2.7. (Odds) The odds of an event x, denoted as $odds(x) \in \mathbb{R}_+$ is the ratio of the probability of the event happening over the probability of not happening, that is,

$$odds(x) = \frac{p(x)}{1 - p(x)}. \tag{2.10}$$

Note that the odds is a positive scalar number. The greater the odds, the more likely is the event to happen. Given $odds(x)$ the probability $p(x)$ can be easily obtained from (2.10).

Definition 2.8. (Multivariate Normal Distribution) Let $\mathbf{x} \in \mathbb{R}^n$ be a continuous random variable that follows a multivariate Gaussian distribution. This is represented as $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ and Σ are the mean and the covariance matrix of the distribution, respectively. If Σ is positive definite, the probability density function of a multivariable Gaussian distribution is

$$f(\mathbf{x}) = \frac{\exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}))}{\sqrt{(2\pi)^n |\Sigma|}} \tag{2.11}$$

2.2 Estimators

2.2.1 Extended Kalman Filter

The Kalman filter is a well established tool for state estimation and multiple sensor data fusion. The classical Kalman Filter is a linear estimator, which is optimal if 1) the model is linear and known, 2) the process and noise covariance follow a Gaussian distribution and are known, and 3) the noise is uncorrelated. These assumptions often do not hold in robotics, and more critically, models are typically non-linear.

The Extended Kalman Filter (EKF) is able to deal with non-linear models but the optimalities guarantees do not hold and the filter converges locally. Nonetheless, the EKF is the golden standard within the robotics community, being widely deployed in robotics navigation. It is employed in this thesis for prediction and data fusion. For a quick introduction, consider the state variable at time t to be denoted as $\boldsymbol{x}_t \in \mathbb{R}^n$. Similarly, define the input variable $\boldsymbol{u}_t \in \mathbb{R}^m$ and the input noise $\boldsymbol{q}_t \in \mathbb{R}^m$. The noise is assumed to be uncorrelated and Gaussian. The dynamics of the system is defined as

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{q}_t), \qquad \boldsymbol{q}_t \sim \mathcal{N}(\boldsymbol{0}, Q_t). \tag{2.12}$$

The EKF keeps track of the state variable x_t using two Gaussian distributions: a prior and a posterior. The prior distribution, denoted as $\mathcal{N}(\check{\boldsymbol{x}}_{t+1}, \check{\Sigma}_{t+1})$, is a projection (prediction) of the state (mean and covariance) at time $t+1$ computed at time t. The prior is obtained before incorporating the measurements. The posterior distribution, given by $\mathcal{N}(\hat{\boldsymbol{x}}_t, \hat{\Sigma}_t)$, provides the best estimate of the state (mean and covariance) at time t computed using the measurements obtained at time t.

The *prediction* step computes the mean and the covariance of the prior distribution of the state variable. For that, the filter propagates the state variable from time t into $t+1$ by exciting the dynamic model with the input variables, that is,

$$\check{\boldsymbol{x}}_{t+1} = f(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \boldsymbol{0}), \tag{2.13}$$

$$\check{\Sigma}_{t+1} = \frac{\partial f}{\partial \boldsymbol{x}} \hat{\Sigma}_t \frac{\partial f}{\partial \boldsymbol{x}}^T + \frac{\partial f}{\partial \boldsymbol{u}} Q_t \frac{\partial f}{\partial \boldsymbol{u}}^T. \tag{2.14}$$

In the prediction step the state variables are not (directly or indirectly) observed. The uncertainty associated to prior distribution grows, that is, the distribution becomes more flat.

The *correction* step incorporates the measurements into the model to compute a posterior distribution (mean and covariance). The measurement model

$$z_t = h(\boldsymbol{x}_t, \boldsymbol{v}_t), \qquad \boldsymbol{v}_t \sim \mathcal{N}(\mathbf{0}, V_t), \tag{2.15}$$

where $\mathbf{z}_t \in \mathcal{R}^p$ are measurements, maps the state variable into the measurement space. The measurements are perturbed by uncorrelated Gaussian noise with covariance V_t. The correction step considers the error between the *actual* measurement and the *expected* measurement using the measurement model. The difference between these two quantities (called innovation) is weighted by the Kalman gain (K) to update the posterior distribution in the following manner:

$$K_t = \check{\Sigma}_t \frac{\partial h}{\partial \boldsymbol{x}}^T (\frac{\partial h}{\partial \boldsymbol{x}} \check{\Sigma}_t \frac{\partial h}{\partial \boldsymbol{x}}^T + \frac{\partial h}{\partial \boldsymbol{z}} V_t \frac{\partial h}{\partial \boldsymbol{z}}^T)^{-1} \tag{2.16}$$

$$\hat{\boldsymbol{x}}_t = \check{\boldsymbol{x}}_t + K_t(z_t - h(\check{\boldsymbol{x}}_t, 0)) \tag{2.17}$$

$$\hat{\Sigma}_t = (I - K_t \frac{\partial h}{\partial \boldsymbol{x}}) \check{\Sigma}_t \tag{2.18}$$

2.2.2 Linear least squares

The linear least squares is a popular method in different fields of science for linear regression (data fitting). In essence, it estimates the parameters of a linear model using observed data. For a quick introduction, consider the model

$$y = f(\boldsymbol{\theta}, \mathbf{x}) = \theta_1 x_1 + \cdots + \theta_p x_p = \boldsymbol{x}^T \boldsymbol{\theta} \tag{2.19}$$

where $y \in \mathbb{R}$ and $\boldsymbol{\theta}, \boldsymbol{x} \in \mathcal{R}^p$. The vector $\boldsymbol{\theta}$ contains the parameters to be estimated. Given a set of observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the residual error vector is denoted as $\mathbf{r} \in \mathbb{R}^n$, where its *i-th* elements is

$$r_i = y_i - \boldsymbol{x}_i^T \boldsymbol{\theta} \tag{2.20}$$

The least square formulation minimizes the quadratic error of the residue, that is,

$$\boldsymbol{\theta}^* = \arg\min_{\theta} \sum_i (y_i - \boldsymbol{x}_i^T \boldsymbol{\theta})^2. \tag{2.21}$$

The residual error to be minimized is the difference between the value predicted by the model and the values that were observed. The solution for the least square problem

can be found by computing the derivative of (2.21) with respect to $\boldsymbol{\theta}$ and equating to zero:

$$2\sum_i (y_i - \boldsymbol{x}_i^T \boldsymbol{\theta})\boldsymbol{x}_i^T = 0 \tag{2.22}$$

$$\boldsymbol{\theta}\sum_i \boldsymbol{x}_i \boldsymbol{x}_i^T = \sum_i \boldsymbol{x}_i^T y_i, \tag{2.23}$$

from which the closed-form least square solution is obtained:

$$\boldsymbol{\theta}^* = (\sum_i \boldsymbol{x}_i \boldsymbol{x}_i^T)^{-1} \sum_i \boldsymbol{x}_i^T y_i. \tag{2.24}$$

There must be at least p independent observations such that the matrix in (2.24) is non-singular. Typically, the number of observations is much greater than the number of parameters of the model, i.e., $n >> p$.

2.2.3 Non-linear least squares

The non-linear least square is an iterative method to estimate the parameters of a non-linear model. An initial solution must be provided and there is no guarantee of convergence for the global optimum value if the model is not convex. Nonetheless, it is a tool often employed to solve different SLAM formulations. Consider the non-linear function $y = f(\boldsymbol{\theta}, \mathbf{x}) : \mathbb{R}^p \times \mathbb{R}^m \rightarrow \mathbb{R}$, where $\mathbf{x} \in \mathbb{R}^m$ and $\boldsymbol{\theta} \in \mathbb{R}^p$. The problem formulation is similar to the one presented in the previous section:

$$\boldsymbol{\theta}^* = \arg\min_{\theta} \sum_i (y_i - f(\boldsymbol{\theta}, \mathbf{x}_i))^2. \tag{2.25}$$

where $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is a set of measurements. Finding a solution for this problem is non-trivial. There are different numerical methods that find a solution for the non-linear least square problem: Gauss-Newton, Levenberg-Marquardt (Damped Least Square), Newton, and so on. The Gauss-Newton method, which approximates the non-linear function by the first order Taylor approximation, is demonstrated next. The idea is computing the first derivative of the approximated cost function and equating to zero to obtain a gradient. The method is repeated iteratively until it converges (or a maximum number of iterations has been made).

Gauss-Newton algorithm: Let $\check{\boldsymbol{\theta}}$ be an initial guess for the parameter vector, such that the solution can be written as $\boldsymbol{\theta}^* = \check{\boldsymbol{\theta}} + \Delta\boldsymbol{\theta}$, where $\Delta\boldsymbol{\theta} \approx \mathbf{0}$. First, re-state the problem in

(2.25) using the first order Taylor expansion of f around $\check{\boldsymbol{\theta}}$:

$$\min_{\Delta\theta} \sum_i (y_i - f(\check{\boldsymbol{\theta}}, \mathbf{x}_i) - \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)^T}{\partial \boldsymbol{\theta}} \Delta\boldsymbol{\theta})^2. \tag{2.26}$$

Next, for the Gauss-Newton algorithm, take the derivative of the cost function w.r.t. to $\Delta\boldsymbol{\theta}$ and equate to zero:

$$2\sum_i (y_i - f(\check{\boldsymbol{\theta}}, \mathbf{x}_i) - \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)^T}{\partial \boldsymbol{\theta}} \Delta\boldsymbol{\theta}) \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)^T}{\partial \boldsymbol{\theta}} = 0. \tag{2.27}$$

The terms in the previous equation can be manipulated to obtain the update step:

$$\Delta\boldsymbol{\theta}^* = -(\sum_i \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)}{\partial \boldsymbol{\theta}} \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)^T}{\partial \boldsymbol{\theta}})^{-1} \sum_i \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)}{\partial \boldsymbol{\theta}} (y_i - f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)). \tag{2.28}$$

The solution obtained in the current iteration is $\check{\boldsymbol{\theta}} + \Delta\boldsymbol{\theta}^*$. The fitting error in (2.25) is evaluated with the new candidate. While the cost does not converge, one iterates (2.28) using the new solution as an initial guess.

2.2.4 Non-linear weighted least squares

The non-linear weighted least squares formulation allows to specify a weight for each residual error. Thus, it is possible to give more importance to a specific observation than other, e.g., the uncertainty associated to an observation is smaller due to sensor specification. Using the definitions from Sec. 2.2.3, the non-linear weighted least squares formulation is as follows:

$$\boldsymbol{\theta}^* = \arg\min_{\theta} \sum_i [w_i (y_i - f(\boldsymbol{\theta}, \mathbf{x}_i))]^2, \tag{2.29}$$

where $\mathbf{w} = [w_1, \ldots, w_n]^T \in \mathbb{R}^n$ is the weight vector. Using the Gauss-Newton method presented in the previous section, the step update step is given by

$$\Delta\boldsymbol{\theta}^* = -(\sum_i w_i^2 \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)}{\partial \boldsymbol{\theta}} \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)^T}{\partial \boldsymbol{\theta}})^{-1} \sum_i w_i^2 \frac{\partial f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)}{\partial \boldsymbol{\theta}} (y_i - f(\check{\boldsymbol{\theta}}, \mathbf{x}_i)). \tag{2.30}$$

Chapter 3

Robotics Systems

This chapter presents the robotic tools employed in this work. Section 3.1 introduces the mathematical model of the robot and the range sensor. Section 3.2 presents the camera model. An overview of the supporting navigation filter is discussed in Sec. 3.3. Section 3.4 introduces the Turtlebot3, the robotic platform employed to evaluate the proposed SLAM framework in the C2SR facilities, at FEUP. Section 3.5 addresses the software tools that provided the means to simulate as well as deploy the code in experimental datasets and in the real robot.

3.1 Plaftorm model

In this section we present the mathematical model of a non-holonomic vehicle equipped with a range sensor. The reader is introduced to concepts such as state, input, and sensor measurements.

Dynamic System

Consider a map coordinate frame $\{M\}$ attached to the origin of the map, and a body fixed coordinate frame $\{B\}$ attached to the center of mass of a wheeled ground vehicle, as shown in Fig. 3.1. The map frame is also known as inertial frame. Let the pose of the vehicle be $\boldsymbol{\xi} = [\boldsymbol{p}^T, \psi]^T$, where $\boldsymbol{p} = [x, y]^T$ is the position of the vehicle described in the inertial frame ,and ψ is its orientation.

The input vector of the vehicle is $\boldsymbol{u} = [\boldsymbol{v}^T, w_z]^T$, where $\boldsymbol{v} = [v_x, v_y]^T$ comprises the forward (v_x) and lateral (v_y) velocities of the vehicle described in the body frame, and w_z

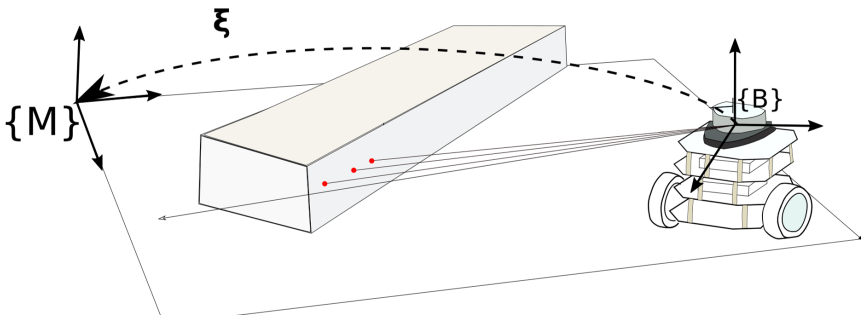


FIGURE 3.1: A robot equipped with a range sensor detects occupied space (red dots) at discrete intervals. The pose of the robot $\boldsymbol{\xi}$ is unknown and needs to be estimated online.

is its angular rate. The motion of the robot is given by the nonlinear differential system

$$\dot{\boldsymbol{\xi}} = f(\boldsymbol{\xi}, \boldsymbol{u}) \Leftrightarrow \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} R(\psi)\boldsymbol{v} \\ w_z \end{bmatrix}, \tag{3.1}$$

where $R(\psi)$ is a rotation matrix from $\{B\}$ to $\{M\}$ and it is as defined in Sec. 2.1. In particular, for a non-holonomic ground vehicle that translates either forward or backward, that is, the vehicle does not move sideways, we have that $v_y = 0$.

Range sensor model

As illustrated in Fig. 3.1, the robot is equipped with a 2D range sensor. For the sake of simplicity, assume that the sensor lies at the center of mass of the vehicle. If that was not the case, an additional rigid body transformation from the sensor frame to the body frame would have to be considered. The sensor provides l range measurements of the environment $(r_i)_{i=1}^{l}$ at discrete angle intervals $(\alpha_i)_{i=1}^{l}$ w.r.t. the x-axis of $\{B\}$.

The measurements (r_i, α_i) in polar coordinates are transformed to Cartesian coordinates via the transform

$$\boldsymbol{z}(r_i, \alpha_i) \equiv \boldsymbol{z}_i = r_i \begin{bmatrix} \cos\alpha_i \\ \sin\alpha_i \end{bmatrix}. \tag{3.2}$$

Finally, let $\mathbf{z}_i$ be a measurement described in body frame ($\{B\}$) and $\boldsymbol{\xi}$ be the pose of the robot when the observation was made. Then, the function $T(\boldsymbol{\xi}, \boldsymbol{z}_i)$ transforms the measurement to the map frame:

$$T(\boldsymbol{\xi}, \boldsymbol{z}_i) \equiv T_i(\boldsymbol{\xi}) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \boldsymbol{z} + \begin{bmatrix} x \\ y \end{bmatrix} = R(\psi)\boldsymbol{z} + \boldsymbol{p}, \tag{3.3}$$

3.2 Camera model

The pinhole camera was a pioneering invention in the history of photography. It consists of a dark chamber with a single tiny aperture - the pinhole - through which the light rays travel. . On the image plane Π[1], fixed at the inner rear surface of the camera, an inverted image is formed as illustrated in Fig. 3.2a. The light rays converge to the pinhole, which is also called center of projection of the optical center, or just optical center. The optical axis contains the optical center and is perpendicular to image plane. Pinhole cameras have a major flaw: images are quite dim, since only a small portion of light is capable of transverse the pinhole. Naturally, the larger the hole, the brighter the image. However, this is achieved at a high cost - the pinhole camera model does not hold for large holes.

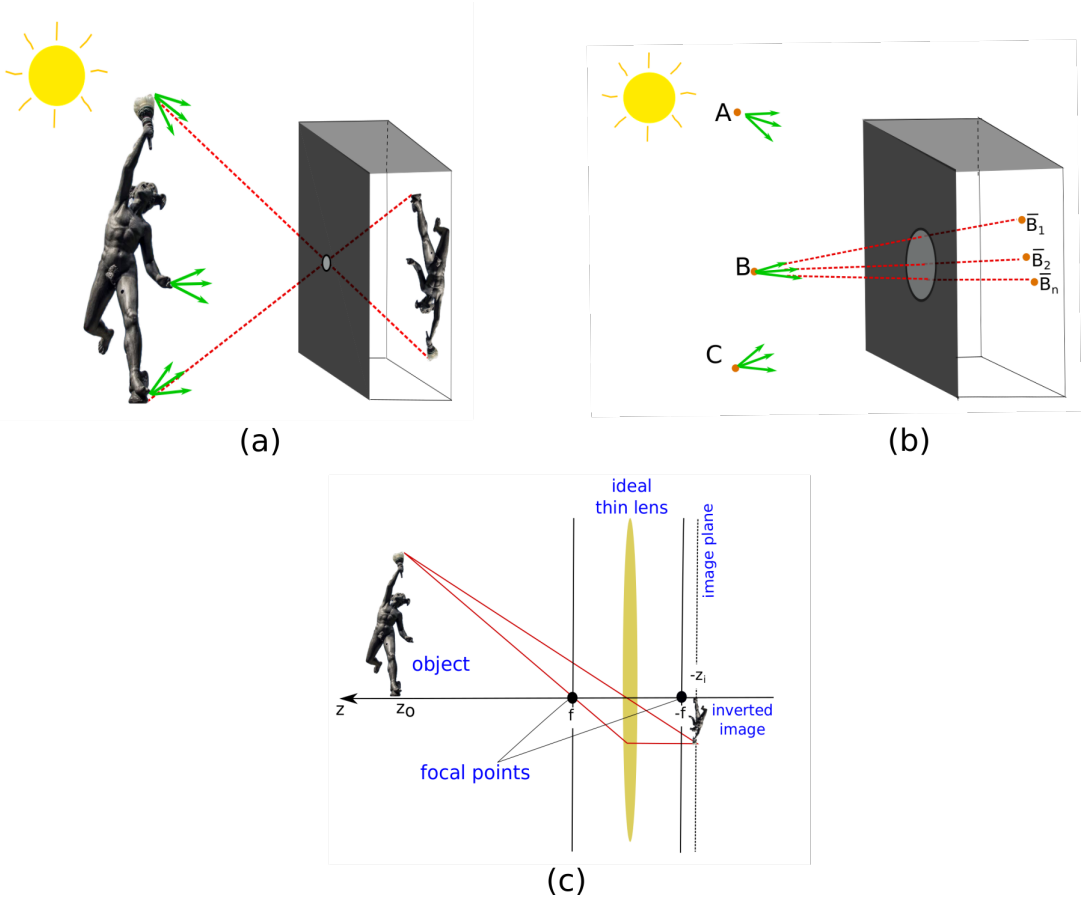


FIGURE 3.2: (a) image formation process in a pinhole camera; (b) a large pinhole contain multiple projections of the same points; (c) ideal thin lens projection model. Green arrows represent light reflected by the object and red dashed lines represent the light captured by the camera.

[1]The image plane is the surface where the image is projected.

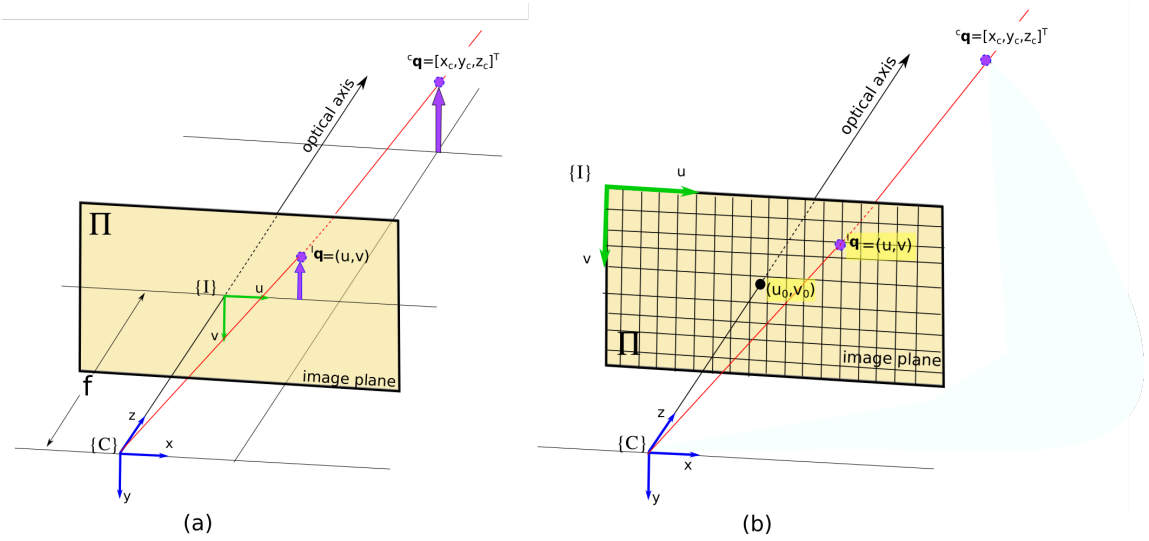


FIGURE 3.3: (a) central projection model and (b) general central projection model.

As depicts Fig. 3.2, a point in the world becomes associated with multiple points in the image plane (and vice-versa!). The result is an unfocused image.

Camera lens tackle the lack of brightness by gathering more light while still achieving an inverted focused image. Let Fig. 3.2 illustrates the ideal thin lens case, according to the lens law:

$$\frac{1}{z_i} + \frac{1}{z_0} = \frac{1}{f} \tag{3.4}$$

where z_0 is the distance from the lens to the object, z_i the distance from the lens to the image plane, and f is the focal length. For a given object at $z = z_0$ and fixed focal length, the lens law gives the position where the image plane must lay for obtaining a sharp image. Inspecting the lens law equation, one concludes that for a focused image as $z_0 \to \infty$, $z_i \to f$. Then, for objects at infinity, i.e., $z_0 \gg f$, the lens of a focal length f can be approximate to a pinhole a distance f from the image plane.

The reader is now ready to be formally introduced to the *simplified central projection model* shown in Fig. 3.3a. First, consider the following coordinate frames:

- $\{C\}$: 3D frame fixed to the optical center. Its z-axis coincides with the optical axis;
- $\{I\}$: 2D frame fixed to the image plane. Its u and v axes aligned with the x and y axes of the camera frame, respectively.

Without loss of generality, the image plane is fixed at $z_i = f$ ahead of $\{C\}$. This time, a non-inverted image is formed, preserving the orientation of objects. The perspective projection model that maps a world point described in the camera frame ${}^C\mathbf{q} = [x_c, y_c, z_c]^T$

to a point in the image plane ${}^{I}\mathbf{q} = [u, v]^T$ is

$$u = f\frac{x_c}{z_c}, \qquad v = f\frac{y_c}{z_c} \tag{3.5}$$

The simplified model introduced takes for granted (1) the image plane is a discrete grid of light sensitive elements and its coordinates are measured in picture elements (pixels); (2) the pixels are not necessarily square, that is, there are independent scale factors ρ_u and ρ_v for the pixelization of coordinates u and v; and (3) the pixels coordinates are non-negative. By convention, the top-right pixel corresponds to the element $(0, 0)$ and it is assumed as the origin of the image coordinate system $\{I\}$. The point (u_0, v_0) corresponds to the camera optical center coordinate. Fig. 3.3b addresses these notes graphically. The three aforementioned considerations correspond to a translation and a rescaling of (3.5):

$$u = \frac{f}{\rho_u}\frac{x_c}{z_c} + u_0, \qquad v = \frac{f}{\rho_v}\frac{y_c}{z_c} + v_0. \tag{3.6}$$

The parameters f, ρ_u, ρ_v, u_0, and v_0 are called intrisic parameters of the camera and they are obtained in a process known as camera calibration, e.g., see the seminal work by Zhang [30]. The normalized homogeneous coordinates in the camera frame are denoted as $\mathbf{s} = [x, y] \equiv [x_c/z_c, y_c/z_c]$ and for a calibrated camera one may obtain as:

$$x = \frac{f}{\rho_u}(u - u_0), \qquad y = \frac{f}{\rho_v}(v - v_0). \tag{3.7}$$

3.3 Navigation filter

Simultaneous localization and mapping and 3D depth estimation require data collected by on-board sensors, which are often classified in two groups: *proprioceptive* and *exteroceptive*. Proprioceptive sensors gather observations about the internal state of the robot, while exteroceptive sensors collect measurements of the environment where a robot operates.

Wheel encoders and IMU are popular examples of proprioceptive sensors. These sensors provide measurements about the motion of the vehicle, e.g. linear velocity, angular rates, and accelerations. The pose of the robot can be estimated via a kinematic or dynamic model of the system by integrating the sensor data - this is known as dead-reckoning. Naturally, sensors measurements are plagued by noise and subject to errors. Thus, by integrating noisy and erroneous data, dead-reckoning is prone to accumulate significant

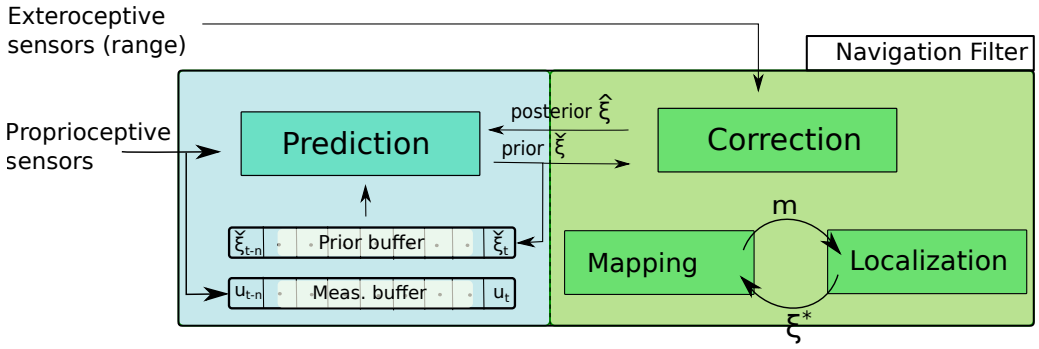


FIGURE 3.4: Navigation filter: proprioceptive sensors (e.g., IMU and encoders) feed the prediction module, which updates the pose prior. The SLAM algorithms presented in this thesis takes as input range data and the pose prior for computing the map and the pose of the robot within the map.

error over time causing pose estimation to drift from the actual pose. However, proprioceptive sensors typically work at a fast rate and are useful sources of local information when a more reliable source is not available.

Camera and LiDAR are representative examples of exteroceptive sensors. If the scene is known *a priori*, the robot may use the sensor data to reason about its most likely location in the environment. In SLAM, the environment is unknown and the robot must concurrently build a map and estimate the location in the map that best explains its current observations. Compared to dead-reckoning, SLAM is less susceptible to pose estimation drifts since localization takes place in a fixed-reference frame - the map frame. In 3D depth estimation, the structure of the environment is also unknown but instead of the absolute pose, one is interested in the relative pose displacement between the frames that features have been observed.

The navigation framework proposed in this thesis fuses odometry data provided by proprioceptive sensors with the SLAM estimation. The filter, shown in Fig. 3.4, has a **prediction** and **correction** scheme. The prediction step estimates the pose of the robot at high rates by propagating the odometry data and control inputs using the system dynamics presented in (Sec. 3.1) - this estimate is called prior. The correction step computes the posterior estimate by fusing the prior with the output of one of the proposed SLAM algorithms. The simple, yet important feature of the navigation filter is dealing with asynchronous data and algorithms. The prediction module holds a buffer for proprioceptive data and another buffer for prior estimations. At time t, the correction module queries the prediction module for the prior at time $t - T$, which corresponds to the timestamp of the extereoceptive data. The correction modules computes and shares with the prediction

module the posterior corresponding to time $t - T$. Finally, it is possible to improve the prior at time t by using the latest posterior from time $t - T$ and proprioceptive data from $t - T, \ldots, t$, which is stored in the buffer.

Implementation

The navigation filter presented in Fig. 3.4 is implemented as a discrete EKF filter. The reader is referred to Sec. 2.2.1 for a brief introduction on EKF. The motion model is assumed to be as in (3.1), but perturbed by additive Gaussian white noise, that is,

$$\boldsymbol{\xi}_{t+1} = f(\boldsymbol{\xi}_t, \boldsymbol{u}_t, \mathbf{q}_t) = \boldsymbol{\xi}_t + \begin{bmatrix} R(\psi)\boldsymbol{v} \\ w_z \end{bmatrix} \Delta t + \mathbf{q}_t, \tag{3.8}$$

where Δt is the sampling time and $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, Q_t)$ is the process noise. The prediction step computes $f(\hat{\boldsymbol{\xi}}_t, \boldsymbol{u}_t, \mathbf{0})$ and propagates the covariance:

$$\begin{aligned} \check{\boldsymbol{\xi}}_{t+1} &= \begin{bmatrix} \hat{x}_t + \Delta t\, v_{x,t} \cos(\hat{\psi}_t) - \Delta t\, v_{y,t} \sin(\hat{\psi}_t) \\ \hat{y}_t + \Delta t\, v_{x,t} \sin(\hat{\psi}_t) + \Delta t\, v_{y,t} \cos(\hat{\psi}_t) \\ \hat{\psi}_t + \Delta t w_{z,t} \end{bmatrix} \\ \check{\Sigma}_{t+1} &= \frac{\partial f}{\partial \boldsymbol{\xi}} \hat{\Sigma}_t \frac{\partial f}{\partial \boldsymbol{\xi}}^T + \frac{\partial f}{\partial \boldsymbol{u}} Q_t \frac{\partial f}{\partial \boldsymbol{u}}^T, \end{aligned} \tag{3.9}$$

where $\check{\Sigma}_t$ and $\hat{\Sigma}_t$ are the prior and posterior state covariance matrices, respectively. The discrete model considers that the vehicle first translates and then rotates. Similarly to [28], it is assumed that the displacement between two consecutive time steps is small enough for the order of the applied displacements not to matter. The Jacobian matrices evaluated at $\hat{\boldsymbol{\xi}}_t$ are as stated below:

$$\frac{\partial f}{\partial \boldsymbol{\xi}} = \begin{bmatrix} 1 & 0 & -\Delta t\, v_{x,t} \sin(\hat{\psi}_t) - \Delta t\, v_{y,t} \cos(\hat{\psi}_t) \\ 0 & 1 & \Delta t\, v_{x,t} \cos(\hat{\psi}_t) - \Delta t\, v_{y,t} \sin(\hat{\psi}_t) \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \frac{\partial f}{\partial \boldsymbol{u}} = \begin{bmatrix} \Delta t \cos(\hat{\psi}_t) & -\Delta t \sin(\hat{\psi}_t) & 0 \\ \Delta t \sin(\hat{\psi}_t) & \Delta t \cos(\hat{\psi}_t) & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \tag{3.10}$$

The correction step incorporates the pose estimation $\boldsymbol{\xi}_t^*$ provided by a SLAM algorithm. The observation model can be written as

$$h(\check{\boldsymbol{\xi}}_t, \mathbf{v}_t) = \check{\boldsymbol{\xi}}_t + \mathbf{v}_t, \tag{3.11}$$

where $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, V_t)$ is the observation noise. The correction step of the navigation filter is obtained as follows:

$$K_t = \check{\Sigma}_t \frac{\partial h}{\partial \xi}^T (\frac{\partial h}{\partial \xi} \check{\Sigma}_t \frac{\partial h}{\partial \xi}^T + V_t)^{-1} = \check{\Sigma}_t (\check{\Sigma}_t + V_t)^{-1} \tag{3.12}$$

$$\hat{x}_t = \check{x}_t + K_t(\xi_t^* - h(\check{x}_t, \mathbf{0})) \tag{3.13}$$

$$\hat{\Sigma}_t = (I - K_t \frac{\partial h}{\partial \xi}) \check{\Sigma}_t = (I - K_t) \check{\Sigma}_t, \tag{3.14}$$

where the Jacobian matrix $\frac{\partial h}{\partial \xi}$ is an identity matrix of appropriate dimensions.

For initializating the filter, it is considered that the vehicle starts at the origin of the map frame, i.e., $\xi_0 = \mathbf{0}$, and that the state covariance $\hat{\Sigma}_0$ is close to zero.

3.4 Platform and hardware

The experimental tests in the C2SR/SYSTEC facilities were performed using a Turtlebot3 Burger by ROBOTIS (see Fig. 3.5). The Turtlebot3 is a research platform family that has a similar morphology and on-board sensory as a broad-range of non-holonomic vehicles employed in civil and industrial applications, such as automated carts operating in warehouses and domestic robots for indoor service robotics. In particular, the Turtlebot3 Burger is equipped with a 360 Laser Distance Sensor LDS-01. Two Dynamixel motors equipped with encoders provide actuation to the vehicle, which can move forward/backward and rotate clockwise/anti-clockwise. A Raspberry Pi 3 Model B is responsible for on-board computations.

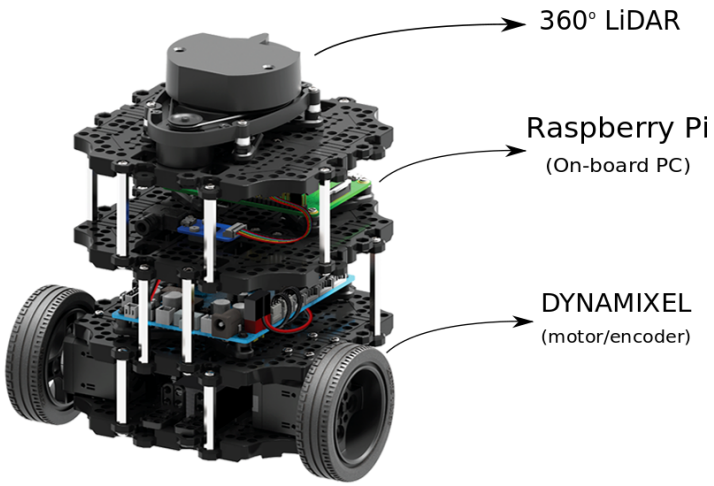


FIGURE 3.5: Turtlebot 3 Burger by ROBOTIS. Sensory-wise the robot is equipped with an IMU, wheel enconders, and a LiDAR sensor. Dynamixel motors are responsible for actuation. A Raspberry Pi 3 provides on-board computational power.

LDS-01 Range sensor

The main features of the 2D LiDAR sensor LDS-01 can be found in Table 3.1[1]. The sensor provides 360 measurements of the environment at 5 Hz frequency. All 360 degrees of the field of view are covered, meaning that the sensor angular resolution is 1 degree. The minimum and maximum distance registered by the sensor (range distance) is 0.12 m and 3.5 m, respectively.

The accuracy of the sensor is determined by the similarity between measured and actual distance to an obstacle. For quantifying the accuracy, it is required to accurately measure 1) the location of the origin of the sensor and 2) the distance between the origin of the sensor and an observed object. Neither of these quantities is trivial to obtain, and, for this reason, an approximation of the specifications provided by the manufacturer was used. It is assumed that the accuracy follows a normal distribution, and the (non-linear) standard deviation is approximated by the following linear model:

$$\sigma_a(r) = 0.05r. \tag{3.15}$$

The notation $\sigma_a(r)$ stands for the standard deviation of the accuracy as a function of the measured range. The variable r is the measured range distance. The slope coefficient 0.05 was obtained from Table 3.1, assuming that it also applies for $r < 0.5$ m.

Consider that the sensor takes multiple measurements of the same object at a fixed distance. The precision is determined by how repeatable these multiple measurements are. Note that precision does not imply accuracy - the measurements may be repeatable but inaccurate. In fact, to quantify the precision we do not need to know the real distance between the object and the sensor. Therefore, we performed experiments at the C2SR

TABLE 3.1: Performance specifications of range sensor LDS-01. These values were provided provided by ROBOTIS[1].

Property	Specifications
Distance range	0.12 ∼3.5 m
Range accuracy (0.12 ∼0.5 m)	± 0.015 m
Range accuracy (0.5 ∼3.5 m)	± 5 %
Range precision (0.12 ∼0.5 m)	± 0.01 m
Range precision (0.5 ∼3.5 m)	± 3.5 %
Angular view	360^o
Angular resolution	1^o

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

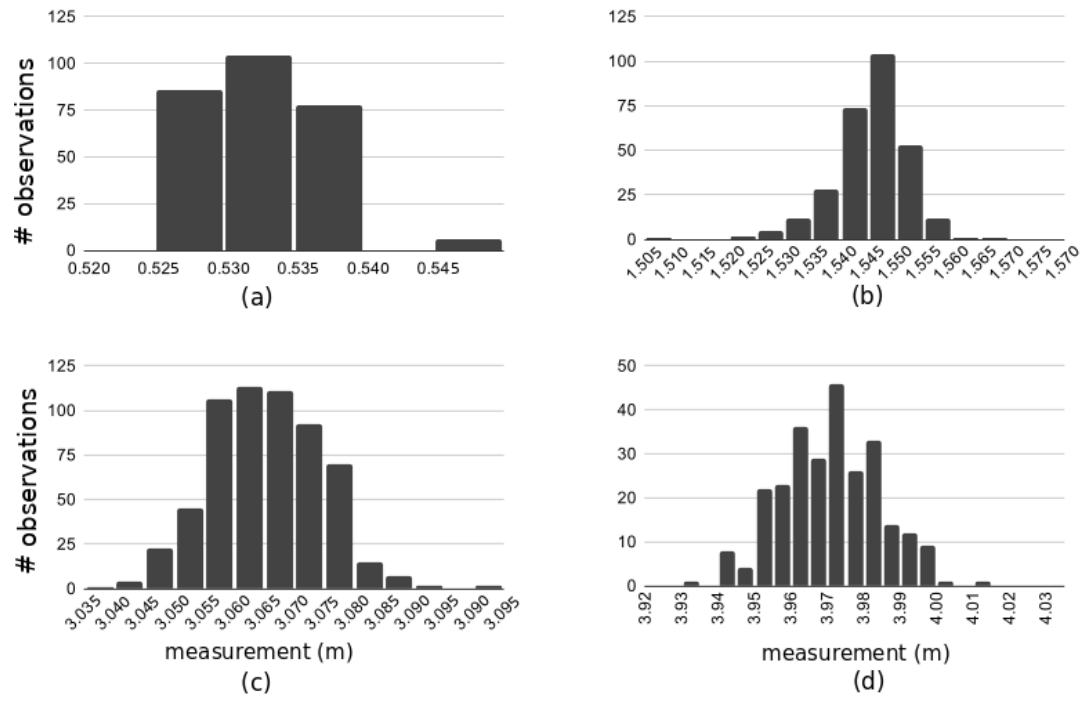


FIGURE 3.6: Measuring precision of the range sensor. The histograms in (a), (b), (c), and (d) were built using range measurements obtained to an object placed at a distance of approximately 0.53 m, 1.55 m, 3.06 m, and 3.98 m, respectively.

facilities for assessing the sensor precision and the normality of the distribution. The experiment consisted in placing the sensor data fixed distance to a wall and capture several measurements (from 300 to 500 readings). This process was repeated for different range distances, from 0.3 m to 4 m. The histograms corresponding to the sensor being at approximately 0.53 m, 1.55 m, 3.06 m and 3.98 m are shown in Fig. 3.6. For building the histograms, the measurements were classified in 0.005 m wide bins, which is half of the precision described by the manufacturer. One may conclude that the distribution of the measurements roughly follows a normal distribution. In a more careful analysis, the set of measurements collected at 0.05 m and 3.98 m failed the Shapiro Wilk normality test [31] with an alpha level of 0.05. Nonetheless, on a later stage, it will be assumed that the distribution is approximately normal. The spread of the distribution grows proportionally to the distance between the sensor and the obstacle, in accordance with the manufacturer specifications.

Assuming that the distribution is approximately normal, we computed the standard deviation for each set of distances. The result is shown in Fig. 3.7. The standard deviation can be approximated by a linear model:

$$\sigma_p(r) = 0.0026r + 0.00253, \tag{3.16}$$

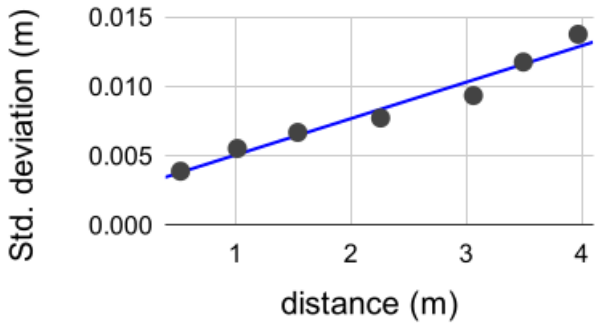


FIGURE 3.7: Standard deviation of the precision of the LiDAR LDS-01 as a function of the measured distance. A blue line represents the linear model that best fits the points in a least square sense.

where $\sigma_p(r)$ is the standard deviation of the precision of the range sensor. During the experiments, it was observed that the material reflecting a beam has impact on the standard deviation and normality of the distribution. For the tests discussed in this section, it was measured the distance to wall and furniture (office alike environment). Metal and translucent materials may yield less predictable results.

The standard deviation of the sensor was obtained by combining the precision and the accuracy distributions. The standard deviation associated to a measurement ($\sigma_s(r)$) is given by

$$\sigma_s(r)^2 = \sigma_a(r)^2 + \sigma_p(r)^2. \tag{3.17}$$

3.5 Software

Python 3

The code was implemented in the programming language Python 3, which is a high-level object-oriented programming language that runs through an interpreter. The main motivations for choosing Python were its fast prototyping and powerful visualisation tools.

At the time of writing of this thesis, the three most popular range-based SLAM packages are written in C++. In contrast to Python, C++ is pre-compiled, providing better performance in terms of memory efficiency and speed. However, C++ is not well suited for prototyping, a major requirement when choosing the programming language for this work. In the evaluated setup, the proposed SLAM package achieves online performance (SLAM runs at least as fast as the sensor rate) thanks to the tailored-suited Python libraries developed for B-spline SLAM. Nonetheless, a C or C++ implementation would improve

the computational performance and allows for further improvements, i.e., pose-graph optimization.

Robot Operating System

Robot Operating System (ROS) [32] is a middleware that aims at encouraging collaborative robotics software development by interfacing communication between processes in a transparent fashion. Helpful features include device drivers, message formats, graphs, sensor visualizers and more. Thereby, roboticists may focus on higher level problems. Research community plays a major role in ROS development and popularisation. For example, state-of-the-art SLAM solutions such as Google Cartographer [12], Hector-SLAM [5], and GMapping [4] are available for free. This encourages code (re)-using, discussions, and further improvements.

ROS adopts the **publisher-subscriber** architecture. Its organization relies on four basic elements: nodes, topics, messages, and services.

- **Nodes:** A node can be a publisher, subscriber or, more commonly, both. A publisher advertises a topic, and sends messages in this topic. A subscriber subscribes to a topic and receives messages in this topic. Most nodes receive input messages, run routines and publish the outcome;
- **Topics:** Topics can be regarded as communication pipelines that ”carry” messages. Only one publisher can send messages in a topic, but several subscribers can listen to these messages. Each topic has a message type and a name (unique identifier);
- **Messages:** Messages contain the meaningful data on a well defined format. Subscribers need to know the format upon subscribing to their topic;
- **Services:** While messages are continuously published under a topic for possibly multiple subscribers, services are usually a one-shot action for a specific task or query.

In ROS, there are typically several nodes running on different processes, and messages are exchanges using a TCP/IP connections (TCPROS). Therefore, the trade-off for adding a communication layer such as ROS is the introduction of limitations that can be important on critical real-time applications. For this reason the SLAM package developed in this thesis is ROS-agnostic, meaning that, it is possible to use the algorithms without

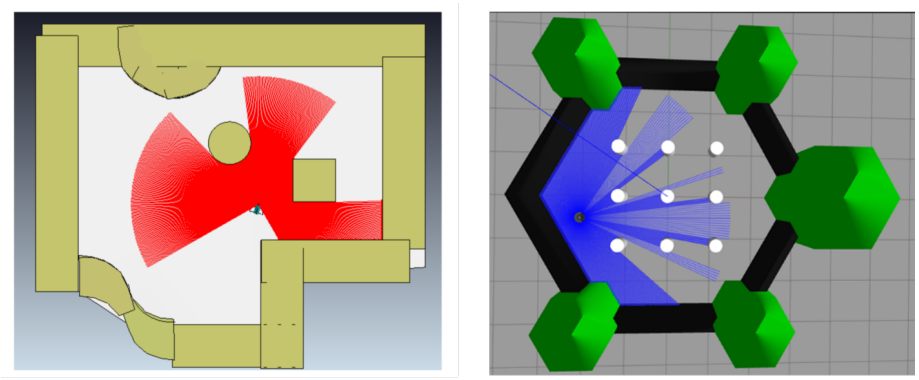


FIGURE 3.8: Simulators: (a) V-REP scene with a robot equipped with a range sensor and (b) Turtlebot3 World scenario in Gazebo.

installing ROS. The library includes a ROS-wrapper, which is a thin layer that supports ROS compatibility, for those who wish to integrate it in a ROS environment.

Virtual environments

In general, implementation of algorithms is plagued by many sources of errors, from conceptual to rounding errors due numerical precision. Therefore, at an early development stage, running field tests can be time-consuming, frustrating, and potentially hazardous. Withing this context, virtual simulators are a suitable option for coding, debugging, assessing performance and improving algorithms before deploying on a real vehicle. It permits performing multiple tests in well controlled environments in a relatively short time when compared to field tests. Last but not least, it is possible to obtain the actual pose of the vehicle (also referred as ground truth) from the simulator, which allows evaluating the performance of a SLAM strategy.

Different simulators are available under free software licenses, like V-REP by Coppelia Robotics, Gazebo by OSRF, and Morse by LAAS-CNRS Open. In the first stage of this thesis, V-REP was employed due to its clear and user-friendly graphical user interface (GUI), ROS interface, and long-term support from Coppelia Robotics. Figure 3.8a shows V-REP scene designed for our tests. After the acquisition of the Turtlebot3 Burger, Gazebo Simulator was preferred motivated by the fact that the Turtlebot3 Burger is fully integrated with Gazebo and ROS (see Fig. 3.8b). As a consequence, it is possible to develop, debug, and evaluate the algorithms in simulations, and then transfer them to the real platform, with minimal modifications.

Chapter 4

B-splines

This chapter discusses B-spline functions, B-spline curves, and B-spline surfaces. Section 4.1 addresses the theoretical concepts behind B-spline functions and shapes. Then, Section 4.2 presents the algorithms and techniques implemented in our tailor-made Python3 B-spline library. It is compared the performance of the proposed package with other Python3 libraries publicly available. Results show that for the tasks required in our SLAM framework, the proposed library is considerable faster than alternative packages.

4.1 B-spline theory

B-spline basis function

A B-spline is a vector-valued function $\boldsymbol{b}^d(\tau|\boldsymbol{t}) : \mathbb{R} \rightarrow \mathbb{R}^m$ that spans a polynomial space of degree d. The variable τ is called the parametric variable and the knot vector $\boldsymbol{t} = [t_i]_{i=1}^{m+d+1}$, with $t_i \leq t_{i+1}, \forall i$ is said to support the B-spline function $\boldsymbol{b}^d(\tau|\boldsymbol{t}) = [b_1^d, \ldots, b_m^d]^T$. An interesting result from the De Boor's recursive algorithm [33] is that high order B-splines can be defined by recursion as

$$b_i^r(\tau|\boldsymbol{t}) = \frac{\tau - t_i}{t_{i+r} - t_i} b_i^{r-1}(\tau|\boldsymbol{t}) + \frac{t_{i+r+1} - \tau}{t_{i+r+1} - t_{i+1}} b_{i+1}^{r-1}(\tau|\boldsymbol{t}), \quad i = 1, \ldots, m \tag{4.1}$$

where, in particular, the zero-*th* order spline is

$$b_i^0(\tau|\boldsymbol{t}) = \begin{cases} 1 & , t_i \leq \tau < t_{i+1} \\ 0 & , \text{otherwise} \end{cases}. \tag{4.2}$$

From this point forward, whenever the knot vector and the degree of the B-spline are clear from the context, the notations $\boldsymbol{b}^d(\tau|\boldsymbol{t})$, $\boldsymbol{b}^d(\tau)$, and $\boldsymbol{b}(\tau)$ are employed interchangeable. The same applies for the notation of B-spline coefficients $b_i^d(\tau|\boldsymbol{t})$, $b_i^d(\tau)$, and $b_i(\tau)$.

Several B-spline properties can be derived from the recursive definition stated in (4.1) and (4.2). In particular, the SLAM algorithms proposed in this thesis make extensive use of four B-spline properties. These are the local knot, the local support, the convex combination, the continuity, and the differentiation properties.

Property 1. (*Local knot*) The B-spline coefficient $b_\mu^d(\tau)$ depends only on the knots $\{t_i\}_{i=\mu}^{\mu+d+1}$.

Figure 4.1 illustrates the local knot property. From the De Boor's recursive algorithm, at the lowest level of the recursion, the term $b_\mu^d(\tau)$ is a combination of $d+1$ zero-*th* order spline coefficients, specifically $b_\mu^0, \dots b_{\mu+d}^0$. From (4.2), for $\tau \in [t_\mu, t_{\mu+d+1})$, one and only of these zero-*th* order coefficients is non-null. If $\tau \notin [t_\mu, t_{\mu+d+1})$, then $b_\mu^0, \dots b_{\mu+d}^0$ are zero and, consequently, $b_\mu^d(\tau) = 0$. Therefore, it is said that a B-spline coefficient depends locally on the knot vector. Mathematically, the local knot property can be stated as

$$b_\mu^d(\tau) \begin{cases} \neq 0 & \text{, if } t_\mu \leq \tau < t_{\mu+d+1} \\ = 0 & \text{, otherwise} \end{cases}.$$

Property 2. (*Local support*) For $\tau \in [t_\mu, t_{\mu+1})$, the B-spline function $\boldsymbol{b}^d(\tau)$ has at most $d+1$ non-zeros coefficients. These are the coefficients $b_{\mu-d}^d, \dots, b_\mu^d$.

For the local support property, consider Fig. 4.2. It shows in different colours several cubic B-spline coefficients ($d = 3$). Evaluating a cubic B-spline function at a specific value τ yields only four non-zero coefficients. Typically the number of B-spline coefficients is much larger than the degree of the B-spline, i.e., $d \ll m$. Hence, by exploiting the local support property, instead of computing m coefficients, only $d+1$ have to be considered.

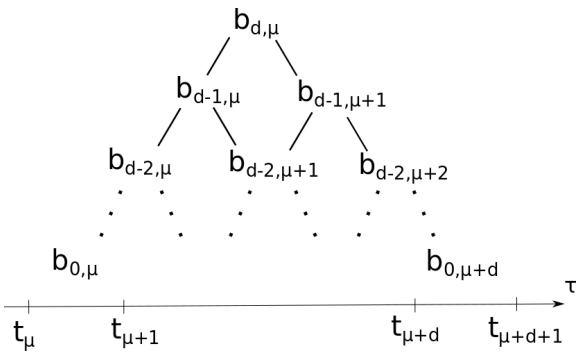


FIGURE 4.1: *Local support*: Visualization of the De Boor's recursive algorithm applied to $b_\mu^d(\tau)$. The coefficient $b_\mu^d(\tau)$ is non-zero if and only if $\tau \in [t_\mu, t_{\mu+d+1})$.

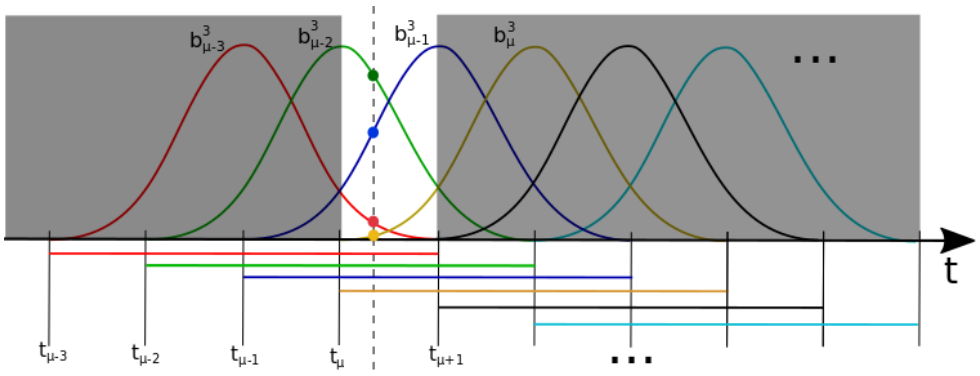


FIGURE 4.2: A cubic B-spline function: colored circles indicate the non-zero coefficients of the B-spline function at the evaluation point specified by the vertical dashed line. For a cubic B-spline ($d = 3$), only four coefficients are non-zero ($d + 1$). This is called the *local support* property and it is employed for speeding up computations.

This increases considerable the computational efficiency for evaluating a B-spline function.

Property 3. (*Convex combination*) For $\tau \in [t_\mu, t_{\mu+1})$, the non-null coefficients of the B-spline function $\boldsymbol{b}^d(\tau)$ are positive and add up to 1, i.e., $\sum_{i=\mu-d}^{\mu} b_i^d(\tau) = 1$.

Property 4. (*Continuity*) Suppose that the knot t_μ occurs κ times among the knots $(t_i)_{i=\mu-d}^{\mu+d}$, with κ some integer bounded by $1 \leq \kappa \leq d+1$. Then, the spline function $\boldsymbol{b}^d(\tau)$ has continuous derivatives up to order $d - \kappa$ at the knot t_μ.

Property 5. (*Differentiation*) Assume that the derivative of a B-spline function $\boldsymbol{b}^d(\tau)$ w.r.t. τ exists. Then, the B-spline derivative is denoted as $\frac{d\boldsymbol{b}(\tau)}{d\tau}$ and its *i-th* coefficient is given by

$$\frac{db_i^d(\tau)}{d\tau} = d\left(\frac{b_i^{d-1}(\tau)}{t_{i+d} - t_i} - \frac{b_{i+1}^{d-1}(\tau)}{t_{i+d+1} - t_{i+1}}\right). \tag{4.3}$$

The B-spline algorithms derived here employ uniformly spaced non-clamped knot vectors, which means that the interval between any two consecutive knots is the same and the initial/final knots are not repeated. Thus, from the continuity property, one may conclude that B-spline functions have derivatives up to order $d - 1$ at any point.

For more properties the reader is referred to [33, 34].

B-spline curve

A B-spline curve $\boldsymbol{s}^d(\tau|\boldsymbol{t}) : \mathbb{R} \to \mathbb{R}^p$ is a linear combination of B-spline basis functions according to

$$\boldsymbol{s}^d(\tau|\boldsymbol{t}) = \sum_{i=1}^{m} \boldsymbol{c}_i b_i^d(\tau|\boldsymbol{t}),$$

where $\boldsymbol{t} \in \mathbb{R}^{m+d+1}$ is the knot vector and $\boldsymbol{c}_i \in \mathbb{R}^p$ is called a control point. The notation $\boldsymbol{s}^d(\tau|\boldsymbol{t})$, $\boldsymbol{s}^d(\tau)$, and $\boldsymbol{s}(\tau)$ are used interchangeable when the supporting knot vector and the degree are clear from the context. The B-spline curve is a convex combination of the control points, see Property 3. Thus, the control points define the shape of the curve and the curve is always with the convex hull defined by the control points.

Consider the control point matrix $C = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m] \in \mathbb{R}^{p \times m}$, then the B-spline curve can be written in matrix form as

$$\boldsymbol{s}(\tau) = C\boldsymbol{b}(\tau). \tag{4.4}$$

For $d > 1$ and an uniformly spaced knot vector, the derivative of the B-spline curve w.r.t. τ exists (see Property 4) and it is described by the following equation:

$$\frac{d\boldsymbol{s}(\tau)}{d\tau} = C\frac{d\boldsymbol{b}(\tau)}{d\tau}.$$

Proposition 4.1. *The computational cost to evaluate a B-spline curve is $O(d+1)$, that is, constant w.r.t. to the number of control points.*

Proof. From Property 2 (Local Support) we have that for $\tau \in [t_\mu, t_{\mu+1})$ only $d+1$ coefficients of the B-spline $\mathbf{b}^d(\tau)$ are non-null. In fact, (4.4) can be stated as

$$\boldsymbol{s}^d(\tau|\boldsymbol{t}) = \sum_{i=\mu-d}^{\mu} \boldsymbol{c}_i b_i^d(\tau|\boldsymbol{t}), \quad \forall \tau \in [t_\mu, t_{\mu+1})$$

Consequently, $\boldsymbol{s}^d(\tau)$ is a convex (see Property 3) combination of the $d+1$ control points $c_{\mu-d}, \ldots c_\mu$. Therefore, regardless of its length and number of control points, by considering only the non-null B-spline coefficients, the computational complex to evaluate a B-spline curve is $O(d+1)$. □

B-spline surface

A B-spline surface is a scalar-valued function $s^d(\boldsymbol{\tau}|\boldsymbol{t}_x, \boldsymbol{t}_y) : \mathbb{R}^2 \to \mathbb{R}$ defined by the tensor product of two B-spline functions, $\boldsymbol{b}_x^d(\tau_x|\boldsymbol{t}_x)$ and $\boldsymbol{b}_y^d(\tau_y|\boldsymbol{t}_y)$, and the control points $\boldsymbol{c}_{ij}$ as follows:

$$s^d(\boldsymbol{\tau}|\boldsymbol{t}_x, \boldsymbol{t}_y) = \sum_{i=1}^{m_x}\sum_{j=1}^{m_y} \boldsymbol{c}_{ij} b_{y,j}^d(\tau_y) b_{x,i}^d(\tau_x), \tag{4.5}$$

where $\boldsymbol{\tau} = [\tau_x, \tau_y]^T \in \mathbb{R}^2$. The notation $s^d(\boldsymbol{\tau}|\boldsymbol{t}_x, \boldsymbol{t}_y)$, $s^d(\boldsymbol{\tau})$, and $s(\boldsymbol{\tau})$ are used interchangeable when the supporting knot vector and the degree are clear from the context. It is assumed that both B-spline functions $\boldsymbol{b}_x$ and $\boldsymbol{b}_y$ have the same degree, d. For this reason,

the degree of the B-spline surface is d^2. At first glance, the notation of B-spline curve and surface may look similar. It is highlighted that a B-spline curve $\boldsymbol{s}(\tau)$ take the input from $\mathbb{R}$ to $\mathbb{R}^p$. The curve is typed in bold font since it is a vector and its parametric variable is written in non-bold font because it is a scalar. Meanwhile, a B-spline surface $s(\boldsymbol{\tau})$ maps from $\mathbb{R}^2$ to $\mathbb{R}$. The surface is typed in non-bold font because it is a scalar value and its parameter is typed in bold font since it represents a vector.

Let $C \in \mathbb{R}^{m_x \times m_y}$ be a real matrix with entries $\boldsymbol{c}_{ij}$. Then, a B-spline surface can be written in matrix form as

$$s(\boldsymbol{\tau}) = \boldsymbol{b}_x(\tau_x)^T C \boldsymbol{b}_y(\tau_y). \tag{4.6}$$

Applying the vectorization operator in (4.6), yields

$$\begin{aligned} s(\boldsymbol{\tau}) &= \operatorname{vec}\left(\boldsymbol{b}_x(\tau_x)^T C \boldsymbol{b}_y(\tau_y)\right) \\ &= \left(\boldsymbol{b}_y(\tau_y)^T \otimes \boldsymbol{b}_x(\tau_x)^T\right) \operatorname{vec}(C) \\ &= \boldsymbol{\phi}(\boldsymbol{\tau})^T \boldsymbol{c}, \end{aligned}$$

where $\boldsymbol{\phi}(\boldsymbol{\tau}) = \boldsymbol{b}_y(\tau_y) \otimes \boldsymbol{b}_x(\tau_x) \in \mathbb{R}^{m_x m_y}$ and $\boldsymbol{c} = \operatorname{vec}(C) \in \mathbb{R}^{m_x m_y}$. Transposing the previous equation for the B-spline surface, yields

$$s(\boldsymbol{\tau}) = \boldsymbol{c}^T \boldsymbol{\phi}(\boldsymbol{\tau}). \tag{4.7}$$

For a uniformly spaced knot vector, the spline function is continuous up to degree d (see Property 4). In this case, the first derivative of the surface with respect to the parametric variable exists and can be written in compact form as

$$\frac{d\boldsymbol{s}(\boldsymbol{\tau})}{d\boldsymbol{\tau}} = \boldsymbol{c}^T \frac{d\boldsymbol{\phi}(\boldsymbol{\tau})}{d\boldsymbol{\tau}} = \boldsymbol{c}^T \left[\frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau})}{\partial \tau_x} \;\middle|\; \frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau})}{\partial \tau_y} \right], \tag{4.8}$$

where the partial derivatives of the spline tensor are defined as

$$\begin{aligned} \frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau})}{\partial \tau_x} &= \boldsymbol{b}_y(\tau_y) \otimes \frac{d\boldsymbol{b}_x(\tau_x)}{d\tau_x} \\ \frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau})}{\partial \tau_y} &= \frac{d\boldsymbol{b}_y(\tau_y)}{d\tau_y} \otimes \boldsymbol{b}_x(\tau_x) \end{aligned} \tag{4.9}$$

Proposition 4.2. *The computational cost to evaluate a B-spline curve is $O(d^2 + 2d + 1)$, that is, constant w.r.t. to the number of control points.*

The proof for this proposition is similar to the one presented in Proposition 4.1 and it is stated here for the sake of completeness.

Proof. From Property 2 (Local Support) we have that for $\tau \in [t_{\mu_x}, t_{\mu_x+1}) \times [t_{\mu_y}, t_{\mu_y+1})$ only $d+1$ coefficients of the B-spline $\mathbf{b}_x^d(\tau)$ and $\mathbf{b}_y^d(\tau)$ are non-null and, consequently, (4.5) is equivalent to

$$s^d(\boldsymbol{\tau}|\boldsymbol{t}_x, \boldsymbol{t}_y) = \sum_{i=\mu_x}^{\mu_x+1} \sum_{j=\mu_y}^{\mu_y+1} \boldsymbol{c}_{ij} b_{y,j}^d(\tau_y) b_{x,i}^d(\tau_x), \quad \forall \tau \in [t_{\mu_x}, t_{\mu_x+1}) \times [t_{\mu_y}, t_{\mu_y+1})$$

In fact, $\boldsymbol{s}^d(\tau)$ is a combination of the $(d+1)^2$ control points. Therefore, regardless of its length and number of control points, by considering only the non-null B-spline coefficients, the computational complexity to evaluate a B-spline curve is $O(d^2 + 2d + 1)$. □

4.2 B-spline sparse library

To deal with the computational constraints imposed by online SLAM, a sparse B-spline library custom-made to our needs was implemented in the programming language Python3. The key-concepts considered for computational efficiency were:

1. using a non-clamped uniformly spaced knot vector;
2. avoid the recursive relation described by the De Boor's algorithm (4.1);
3. to take into account only the non-null B-spline coefficients and corresponding control points;

The advantage of using a non-clamped uniformly spaced knot vector are mainly three. First, it allows to easily modify the size of the map. Although a wide range of applications use clamped knot vectors, they are not interesting in SLAM. The reason is that in clamped knot vectors the final and initial knots are repeated and, therefore, extending the map would require rebuilding the frontier of the map where the knots are clamped. The second advantage is that storing a uniformly space knot vector is compact. Let m be the number of control points, an uniformly spaced knot vector is well defined by three values: the initial knot (t_1), the final knot (t_{m+d+1}), and the knot interval (Δ). Finally, and most important, when the knot interval is constant the B-spline functions are shifted version of one another. For example, in Fig. 4.3, $\mathbf{b}(\tau_2)$ and $\mathbf{b}(\tau_3)$ can be obtained by shifting the coefficient of $\mathbf{b}(\tau_1)$. This can be easily verified using the recursive relation described in (4.1). Let $\tau_1 \in [t_\mu, t_{\mu+1})$ and $\tau_2 = \tau_1 + \Delta$, where Δ is the constant knot interval. To obtain *i-th* coefficient of $\mathbf{b}(\tau_2)$ evalute the expression

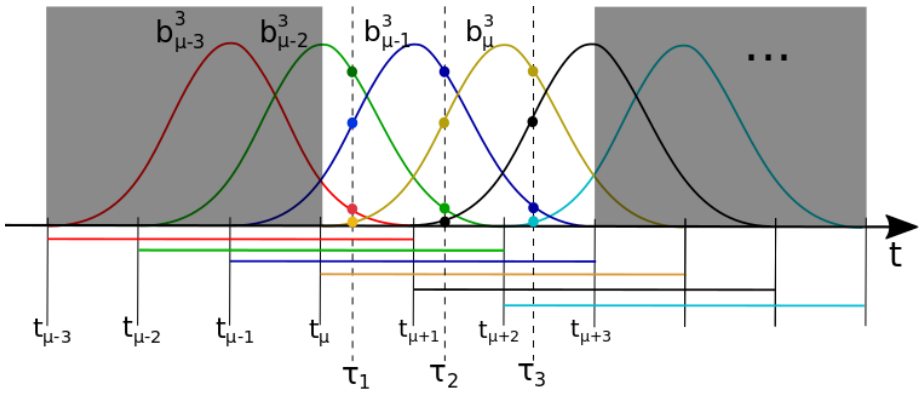


FIGURE 4.3: Evaluating a cubic B-spline function supported by an uniform knot vector with constant knot interval Δ. Since $\tau_2 = \tau_1 + \Delta$ and $\tau_3 = \tau_2 + \Delta$, the non-null B-spline coefficients of $\mathbf{b}(\tau_2)$ and $\mathbf{b}(\tau_3)$ can be obtained by shifting the coefficient of $\mathbf{b}(\tau_1)$.

$$
\begin{aligned}
b_i^r(\tau_2) &= \frac{\tau_2 - t_i}{t_{i+r} - t_i} b_i^{r-1}(\tau_2) + \frac{t_{i+r+1} - \tau_2}{t_{i+r+1} - t_{i+1}} b_{i+1}^{r-1}(\tau_2). \\
&= \frac{\tau_1 + \Delta - t_i}{t_{i+r} - t_i} b_i^{r-1}(\tau_2) + \frac{t_{i+r+1} - (\tau_1 + \Delta)}{t_{i+r+1} - t_{i+1}} b_{i+1}^{r-1}(\tau_2),
\end{aligned}
$$

$$
b_i^r(\tau_2) = \frac{\tau_1 - (t_i - \Delta)}{(t_{i+r} - \Delta) - (t_i - \Delta)} b_i^{r-1}(\tau_2) + \frac{(t_{i+r+1} - \Delta) - \tau_1}{(t_{i+r+1} - \Delta) - (t_{i+1} - \Delta)} b_{i+1}^{r-1}(\tau_2).
$$

Since the knot interval is constant, $t_{k-1} = t_k - \Delta, \forall k$ holds, and the previous equation can be rewritten as

$$
b_i^r(\tau_2) = \frac{\tau_1 - t_{i-1}}{t_{i+r-1} - t_{i-1}} b_i^{r-1}(\tau_2) + \frac{t_{i+r} - \tau_1}{t_{i+r} - t_i} b_{i+1}^{r-1}(\tau_2). \tag{4.10}
$$

Now, compare the previous equation with the $(i-1)$-*th* coefficient of $\mathbf{b}(\tau_1)$, that is,

$$
b_{i-1}^r(\tau_1) = \frac{\tau_1 - t_{i-1}}{t_{i+r-1} - t_{i-1}} b_{i-1}^{r-1}(\tau_1) + \frac{t_{i+r} - \tau_1}{t_{i+r} - t_i} b_i^{r-1}(\tau_1). \tag{4.11}
$$

The terms in (4.10) multiplying $b_i^{r-1}(\tau_2)$ and $b_{i+1}^{r-1}(\tau_2)$ are the same terms multiplying $b_{i-1}^{r-1}(\tau_1)$ and $b_i^{r-1}(\tau_1)$ in (4.11). Applying in a recursive manner, one concludes that

$$
b_i^r(\tau_2) = \frac{\tau_1 - t_{i-1}}{t_{i+r-1} - t_{i-1}} b_{i-1}^{r-1}(\tau_1) + \frac{t_{i+r} - \tau_1}{t_{i+r} - t_i} b_i^{r-1}(\tau_1) = b_{i-1}^r(\tau_1).
$$

The coefficients of $\mathbf{b}(\tau_2)$ and $\mathbf{b}(\tau_1)$ are related by a single shift in the knot vector.

The recursive relation given by the De Boor's algorithm is useful to define the basic B-spline properties. Moreover, it allows to compute the coefficients of a B-spline function of any degree supported by any valid knot vector. However, such a a generic formulation comes at a cost: computing coefficients in a recursive manner is computationally expensive. This is because many of the terms computed are actually multiplied by zero,

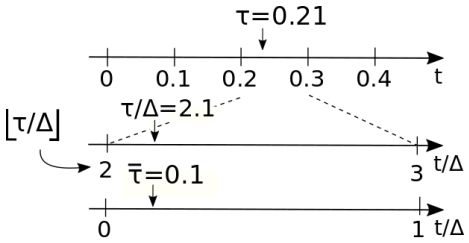


FIGURE 4.4: Normalising parametric variable $\tau = 0.21$ defined on a uniform knot interval with $\Delta = 0.1$. The result $\bar{\tau} = 0.1$ is obtained via (4.12).

which is returned at the bottom of the recursion by a null zero-*th* order coefficient. For this reason, an alternative method was sought.

In fact, there is an efficient manner to compute the coefficients when $\tau \in [0, 1)$ and the constant knot interval is unitary, i.e., $\Delta = 1$. This is addressed in the literature as a normalised uniform B-spline [35]. To take advantage of this, it is first necessary to map a non-normalised knot into a normalised knot[1]. The normalization process is only possible because the knot interval is constant. Let the constant knot interval be Δ, for some $\tau \in \boldsymbol{t}$, its normalised value is denoted as $\bar{\tau} \in [0, 1)$ and it is computed as

$$\bar{\tau}(\tau) = \frac{\tau}{\Delta} - \left\lfloor \frac{\tau}{\Delta} \right\rfloor. \tag{4.12}$$

This process is shown in Fig. 4.4 for $\Delta = 0.1$ and $\tau = 0.21$, which yields $\bar{\tau} = 0.1$. Evaluating a B-spline function at τ is similar to evaluate a normalised B-spline function at $\bar{\tau}(\tau)$. For example, for a cubic order B-spline function ($d = 3$), the non-null coefficients at $\tau \in [t_\mu, t_{\mu+1})$ can be obtained via the polynomial form for a normalised B-spline function:

$$\begin{aligned} b^3_\mu(\tau) &= \frac{1}{6}(-\bar{\tau}^3 + 3\bar{\tau}^2 - 3\bar{\tau} + 1) \\ b^3_{\mu-1}(\tau) &= \frac{1}{6}(3\bar{\tau}^3 - 6\bar{\tau}^2 + 4) \\ b^3_{\mu-2}(\tau) &= \frac{1}{6}(-3\bar{\tau}^3 + 3\bar{\tau}^2 + 3\bar{\tau} + 1) \\ b^3_{\mu-3}(\tau) &= \frac{1}{6}\bar{\tau}^3 \end{aligned} \tag{4.13}$$

For more polynomial forms of normalised B-spline functions the reader is referred to [35].

The derivative of $\boldsymbol{b}(\tau)$ can be simplified by substituting the constant knot interval Δ and $\tau = \bar{\tau}$ in (4.3), that is,

$$\frac{db^d_i(\tau)}{d\tau} = \frac{1}{\Delta}[b^{d-1}_i(\bar{\tau}) - b^{d-1}_{i+1}(\bar{\tau})].$$

[1] A normalised knot is always in the interval $[0, 1)$

Algorithm 1: Computing B-spline coefficients

signature: $\boldsymbol{b}_\mu, \boldsymbol{db}_\mu, \mu \leftarrow$ compute_spline(τ)
Input: τ
Output: $\boldsymbol{b}_\mu$, $\boldsymbol{db}_\mu$, and μ
Internal parameters: d, Δ, and t_1.
1: Normalise τ to the interval $[0, 1)$: $\bar{\tau} \leftarrow$ (4.12)
2: Compute $d+1$ non-null B-spline coefficients $[b_i]_{i=\mu-d}^{\mu}$: $\boldsymbol{b}_\mu \leftarrow$ (4.13)
3: Compute $d+1$ non-null B-spline derivative coefficients $[db_i]_{i=\mu-d}^{\mu}$: $\boldsymbol{db}_\mu \leftarrow$ (4.14)
4: Find μ such that $\tau \in [t_\mu, t_{\mu+1})$: $\mu \leftarrow$ (4.15)

Applying the polynomial form for a normalised uniform quadratic B-spline function in the previous equation, yields

$$\begin{aligned}
\frac{db_\mu^3(\tau)}{d\tau} &= \frac{1}{6\Delta}(-2\bar{\tau}^2 + 6\bar{\tau} - 3) \\
\frac{db_{\mu-1}^3(\tau)}{d\tau} &= \frac{1}{2\Delta}(3\bar{\tau}^2 - 4\bar{\tau}) \\
\frac{db_{\mu-2}^3(\tau)}{d\tau} &= \frac{1}{2\Delta}(-3\bar{\tau}^2 + 2\bar{\tau} + 1) \\
\frac{db_{\mu-3}^3(\tau)}{d\tau} &= \frac{1}{2\Delta}\bar{\tau}^2.
\end{aligned} \tag{4.14}$$

Now that only the non-null terms are computed, the algorithm must keep track of the index $\mu - d, \ldots, \mu$ that these coefficients refers to. For $\tau \in [t_\mu, t_{\mu+1})$, μ is given by

$$\mu = \left\lfloor \frac{\tau - t_1}{\Delta} \right\rfloor + 1, \tag{4.15}$$

where t_1 is the initial knot.

Algorithm 1 shows the steps for computing a B-spline function in an efficient manner. The input is the non-normalised scalar parametric value τ and the output are the non-null coefficients of the B-spline function ($\boldsymbol{b}_\mu$) and its derivative ($\boldsymbol{db}_\mu$), and the index (μ). The index allows later to find the control points associated to the non-null coefficients.

The B-spline tensor is obtained using the previous result. The tensor defined in (4.5) has $(d+1)^2$ non-null coefficients. These are computed as follows:

$$\phi_k(\bar{\boldsymbol{\tau}}) = b_{y,j}(\bar{\tau}_y) b_{x,i}(\bar{\tau}_x), \qquad \begin{aligned} k &= (j-1)(d+1) + i \\ i &= \mu_x - d, \ldots, \mu_x \\ j &= \mu_y - d, \ldots, \mu_y \end{aligned} \tag{4.16}$$

where for a cubic B-spline basis function $\mathbf{b}_x = [b_{x,i}]_{i=\mu-d}^{\mu}$ and $\mathbf{b}_y = [b_{y,i}]_{i=\mu-d}^{\mu}$ are computed as in (4.13). The pairs ($\bar{\tau}_x$, $\bar{\tau}_y$) and (μ_x, μ_y) are obtained using (4.12) and (4.15),

Algorithm 2: Computing B-spline tensor coefficients

signature: $\boldsymbol{\phi}_{\mu}, d\boldsymbol{\phi}_{\mu,x}, d\boldsymbol{\phi}_{\mu,y}, \mu_x, \mu_y \leftarrow$ compute_tensor_spline($\boldsymbol{\tau}$)
Input: $\boldsymbol{\tau} = [\tau_x, \tau_y]$
Output: $\boldsymbol{\phi}_{\mu}, d\boldsymbol{\phi}_{\mu,x}, d\boldsymbol{\phi}_{\mu,y}, \mu_x$, and μ_y
Internal parameters: d
1: Compute the B-spline along the x-axis: $\boldsymbol{b}_{\mu_x}, \mu_x \leftarrow$ compute_spline(τ_x)
2: Compute the B-spline along the y-axis: $\boldsymbol{b}_{\mu_y}, \mu_y \leftarrow$ compute_spline(τ_y)
3: Compute $(d+1)^2$ non-null coefficients of the B-spline tensor: $\boldsymbol{\phi}_{\mu} \leftarrow$ (4.16)
4: Compute $(d+1)^2$ non-null coefficients of B-spline tensor partial derivatives: $d\boldsymbol{\phi}_{\mu,x}, d\boldsymbol{\phi}_{\mu,y} \leftarrow$ (4.17)

respectively. Similarly, the derivative of a B-spline tensor is given by

$$\begin{cases} \frac{\partial \phi_k(\bar{\boldsymbol{\tau}})}{\partial \tau_x} = b_{y,j}(\bar{\tau}_y)\frac{db_{x,i}(\bar{\tau}_x)}{d\tau_x} \\ \frac{\partial \phi_k(\bar{\boldsymbol{\tau}})}{\partial \tau_y} = \frac{db_{y,j}(\bar{\tau}_y)}{d\tau_x} b_{x,i}(\bar{\tau}_x) \end{cases}, \qquad \begin{aligned} k &= (j-1)(d+1)+i \\ i &= \mu_x - d, \ldots, \mu_x \\ j &= \mu_y - d, \ldots, \mu_y \end{aligned} \tag{4.17}$$

where the B-spline derivatives $[\frac{db_{x,i}}{d\tau_x}]_{i=\mu_x-d}^{\mu_x}$ and $[\frac{db_{y,j}}{d\tau_y}]_{j=\mu_y-d}^{\mu_y}$ are as in (4.14). These results are gathered in Algorithm 2, which shows the steps for computing a B-spline. The input is the non-normalised parametric variable $\boldsymbol{\tau} \in \mathbb{R}^2$. The output are the non-null B-spline tensor coefficients ($\boldsymbol{\phi}$), its partial derivatives ($d\boldsymbol{\phi}_{\mu,x}$, $d\boldsymbol{\phi}_{\mu,y}$), and their corresponding indices (μ_x, μ_y).

B-spline curve

Using the notation as in Algorithm 1, evaluating a B-spline curve boils down to computing

$$\boldsymbol{s}(\tau) = C_{\mu}\boldsymbol{b}_{\mu}(\tau),$$

where, for a cubic B-spline basic function, $C_{\mu} = [c_i]_{i=\mu-3}^{\mu}$ has four control points. The B-spline derivative is given by

$$\frac{d\boldsymbol{s}(\tau)}{d\tau} = C_{\mu}d\boldsymbol{b}_{\mu}(\tau).$$

B-spline surface

Similarly, resorting to Algorithm 2, a B-spline surface is

$$s(\boldsymbol{\tau}) = \boldsymbol{c_{\mu}}^T\boldsymbol{\phi}_{\mu}(\boldsymbol{\tau})$$

where, for a surface with cubic basis functions, $\boldsymbol{c}_{\boldsymbol{\mu}} = [c_{(j-1)4+i}]_{i=\mu_x-3,j=\mu_y-3}^{\mu_x,\mu_y}$ gathers 16 control points. The partial derivative of the B-spline surfaces are

$$\frac{d\boldsymbol{s}(\boldsymbol{\tau})}{d\boldsymbol{\tau}} = \boldsymbol{c}_{\boldsymbol{\mu}}{}^{T} \left[d\boldsymbol{\phi}_{\mu,x}(\boldsymbol{\tau}) \quad \middle| \quad d\boldsymbol{\phi}_{\mu,y}(\boldsymbol{\tau}) \right].$$

4.2.1 Computational performance comparison

A lightweight B-spline library that meets our computational needs was implemented in the programming language Python3. The package, named as *spline-sparse*, was designed having in mind that only a few B-spline coefficients are non-zero (as summarised in Algorithm 1 and 2). For evaluation purposes, the performance of the developed library is compared with other B-spline Python3 packages, namely *NURBS-Python*, *bspline*, and *Splipy*:

- **NURBS-Python**: This is an object-oriented B-spline and Non-Uniform Rational Basis Spline (NURBS) library developed by Onur R. Bingol. It provides functionalities for B-spline curve, surface, and volume. It has a friendly user-interface and support for 3D visualization. Available at: https://nurbs-python.readthedocs.io/

- **bspline**: This is a B-spline library that implements the B-spline basis functions using the De Boor's algorithm. The library, designed by John T. Foster and Juha Jeronen, provides Matlab integration. Available at: https://pypi.org/project/bspline/

- **Splipy**: This library focuses on B-spline curves, surfaces, and volumes. It claims to be an alternative for classical CAD tools when fine-grained control over the geometric shape is required. It provides interesting built-in features such as curve and surface fitting. Available at: https://pypi.org/project/Splipy/

These four libraries were evaluated on a laptop equipped an Intel Core i5-3317U 1.7GHz. Three tests were considered, which aims at assessing the temporal cost to evaluate a B-spline curve and its derivative, a B-spline surface and its derivative, and the impact of increasing the length of the support knot vector. For each combination of task and library, the same experiment is repeated 100 times (to diminish the impact of outlier time measurements). For all the tests, the degree of the B-spline basis and the knot interval were fixed at $d = 3$ and $\Delta = 0.05$, respectively.

The first results, presented in Fig. 4.5, show the cost to evaluate B-spline geometric shapes as the number of evaluation points increases. For both B-spline curve (Fig. 4.5a)

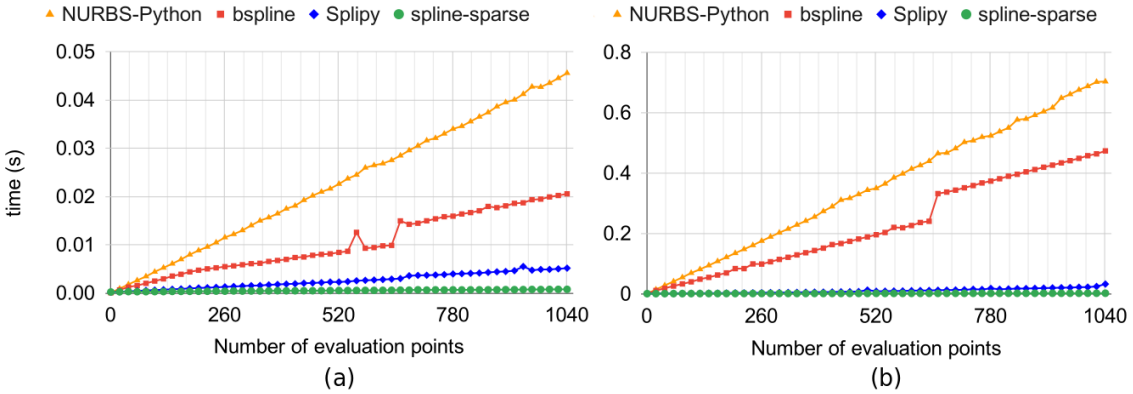


FIGURE 4.5: Assessing the computational performance of four B-spline libraries for a mapping alike task. The cost to evaluate a B-spline curve and a B-spline surface as the number of points increases are shown in (a) and (b), respectively.

and B-spline surface (Fig. 4.5b) the proposed library (spline-sparse) and Splipy scored the best results, that is, the smallest time to perform the task. In our SLAM algorithms, evaluating a B-spline curve or surface at multiple points is required during the mapping task - it is a one time operation per observation reported by the sensor. Bear in mind that modern range sensors provide several observations of the environment at a high rate. For example, in this thesis it is analyzed datasets from range sensors that provide 180 (at 5 Hz), 360 (at 5 Hz), and 1040 (at 40 Hz) measurements per scan reading. The numerical results for evaluating a B-spline curve and a surface at these specific number of points are shown in Table 4.1. The table highlights in bold the values for which a combination of B-spline library and sensor (number of points per measurements and rate) would be accepted for online mapping, that is, the operation works at least as fast as the sensor rate. For example, consider a sensor that works at 5 Hz and captures 180 measurements per scan reading. Then, processing a single scan reading shall take no longer than 200 ms. The proposed library was the only one that succeed in all the scenarios considered. Spline-sparse was specially faster at evaluating a B-spline surface.

TABLE 4.1: Cost to query the value of a curve and a surface at 180, 360, and 1040 points. Values that respect the time constraints imposed by the sensor rate are typed in bold.

	B-spline curve			**B-spline surface**		
Number of points	180	360	1040	180	360	1040
Max acceptable time (ms)	200	200	25	200	200	25
NURBS-Python (ms)	**7.95**	**15.64**	45.58	**121.13**	241.33	702.80
bspline (ms)	**4.37**	**6.53**	20.54	**128.85**	214.73	492.80
Splipy (ms)	**0.94**	**1.68**	**5.14**	**2.39**	**5.19**	32.23
spline-sparse (ms)	**0.33**	**0.43**	**0.76**	**0.91**	**1.05**	**1.89**

TABLE 4.2: Computational time required for evaluating a curve and a surface as well as their first order derivatives. Values that respect time constraints imposed by the sensor rate are typed in bold.

	B-spline curve and its derivative			**B-spline surface and its derivative**		
Number of points	180	360	1040	180	360	1040
Max acceptable time (ms)	40	40	5	40	40	5
NURBS-Python (ms)	**11.68**	**24.16**	67.12	137.60	274.51	797.05
bspline (ms)	**69.09**	135.99	473.21	442.80	729.85	2043.94
Splipy (ms)	**1.08**	**1.87**	13.13	**6.48**	**13.64**	77.40
spline-sparse (ms)	**0.52**	**0.61**	**1.02**	**1.32**	**1.56**	**2.84**

The second set of experiments analyses the time to compute B-spline curves and surfaces as well as their first order derivatives. These computations are required during the localization tasks. However, since localization typically resorts to iterative methods, these operations are done multiple times per scan measurement. Table 4.2 shows the time that each library takes to complete the task for 180, 360, and 1040 points. The minimum acceptable time assumes that a localization algorithm runs up to five iterations per scan. As as consequence, it evaluates the B-spline curve (or surface) and its first order derivative up to five times. The acceptable time for online operations are highlighted in bold. The proposed library is from 5 to 100 times faster than alternative Python3 libraries. Also, it is the only one capable of respecting the time constraints imposed by all the sensor considered evaluated by our SLAM algorithms. The behaviour of the computational time as the number of points increases is shown in Fig. 4.6. The graph only shows the results for the Splipy and spline-sparse libraries because the others were significantly slower. In the B-spline curve scenario, it is unknown the reason for the sudden increase in the computational time of Splipy from 3.2 ms (640 points) to 8.3 ms (660 points). Regardless of

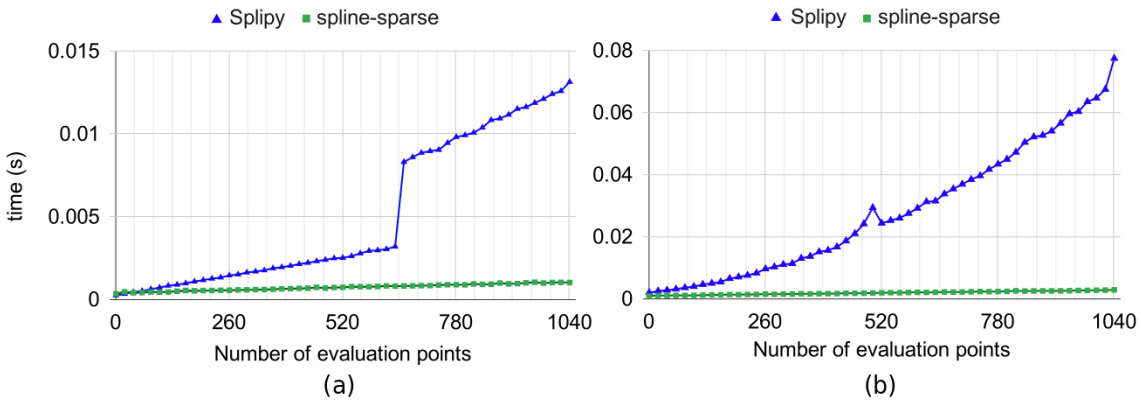


FIGURE 4.6: Assessing the computational performance of two B-spline libraries for a localization alike task. (a) shows the temporal cost to evaluate a B-spline curve and its derivative as the number of point increases. Similarly, (b) shows the cost to evaluate a B-spline surface and its first order derivative.

that, the big picture shows that when evaluating a few points, Splipy and spline-sparse have similar computational time. As the number of point increases, the temporal cost to complete the task increases faster for Splipy.

The number of control points of the B-spline curves and surfaces employed in this work is considerable large than the degree of their B-spline basis. From Property 2, local support, the number of non-null coefficients does not depends on the number of control points, but rather on the degree of the B-spline. For instance, by fixing the B-spline degree at $d = 3$, it is know that there are up to 4 non-null coefficients for a B-spline curve and 16 non-null coefficients for a B-spline surface. Computing the curve or the surface through (4.1) and (4.5) is not efficient because most of the coefficients are null. Figure 4.7 shows the time required to evaluate a B-spline curve and its derivative as the number of control points increases. As a refresher, this is an operation carried out during the localization and it is the most computational demanding task demanded by our SLAM framework. The results for the NURBS-Python and bspline libraries are not plotted because as observed in the previous tests these libraries are considerable slower. For the spline-sparse library the computational time is constant as the length of the B-spline basis increases. This supports our claim that the cost is constant with respect to the number of control points. In contrast, for the Splipy library the temporal cost increases as the length of the control points increases. By analysing their code, it is possible to confirm that it performs several multiplications between a control points and a null coefficient.

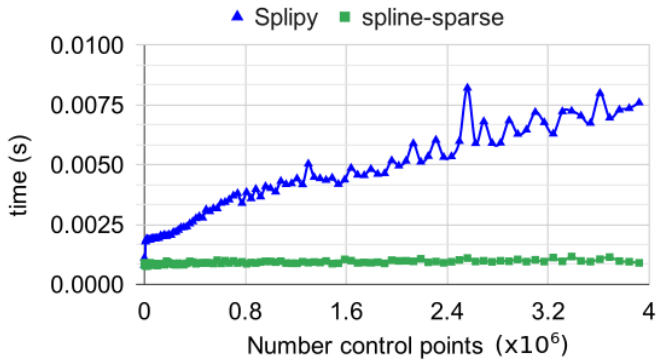


FIGURE 4.7: Computational time required for evaluating a B-spline surface and its derivative as the number of control points increases. The number of control considered to plot this graph corresponds to cubic B-spline surface maps of 1 m^2 to 100 m^2 built with a constant knot interval $\Delta = 0.05$.

Final remarks

The design and implementation of the spline-sparse library aimed at computational performance and it was tailored-made for our needs. While the other libraries evaluated excel in other features, such as user-friendly interfaces, generic approach, and 3D visualization, they do not satisfy the computational constraints imposed by online SLAM. For instance, NURBS-python, bspline and Splipy store the entire B-spline function vector $\mathbf{b}(\tau) \in \mathbb{R}^m$. However, as discussed before, only the coefficients $b_{\mu-d}(\tau), \ldots, b_{\mu}(\tau)$ for $\tau \in [t_{\mu}, t_{\mu+1})$ are non-null. As a consequence, since $m >> d$, the aforementioned libraries perform several B-spline coefficients and control points multiplication which are meaningless. NURBS-python and bspline implement the De Boor's algorithm for computing the B-spline coefficients. This is considerable slower than our polynomial approach. It is not clear the method implemented on Splipy. Implementation-wise, we strive to take advantage of the characteristics inherent to Python3 to speed up computations, e.g., avoiding for-$loop$'s.

Part I

Range-based SLAM

Chapter 5

A Review on Simultaneous Localization and Mapping

The problem of Simultaneous Localization and Mapping has been an active research topic for approximately 30 years. The goal of this chapter is to present some of the solutions developed by the community, taking into consideration the historical importance but also focusing on methods that are relevant for the particular problems discussed in this document. The chapter is organized as follows: Sec. 5.1 introduces the SLAM problem; Sec. 5.2 presents different types of maps; Sec. 5.3 covers key SLAM techniques; and Sec. 5.5 highlights the limitations which are addressed in this thesis.

5.1 Formulation

The state in SLAM encodes the map and the pose of the robot. Estimating the map is known as mapping, while estimating the pose of the robot is named localization. The key elements presented in a SLAM formulation are

- $\boldsymbol{\xi}_{0:t+1} = \{\boldsymbol{\xi}_0, \ldots, \boldsymbol{\xi}_{t+1}\}$: pose of the robot from time $0, \ldots, t+1$;
- $\boldsymbol{\xi}_{0:t} = \{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_t\}$: odometry (or control input) of the vehicle from time $0, \ldots, t$;
- $z_{0:t} = \{z_0, \ldots, z_t\}$: measurements of the environment from time $0, \ldots, t$;
- $\boldsymbol{m}$: map of the environment as perceived by the robot.

The relationships between these quantities are represented in Fig. 5.1. The vehicle observes the measurement $\mathbf{z}_i$ at pose $\boldsymbol{\xi}_i$. The control $\mathbf{u}_i$ excites the vehicle, which evolves to

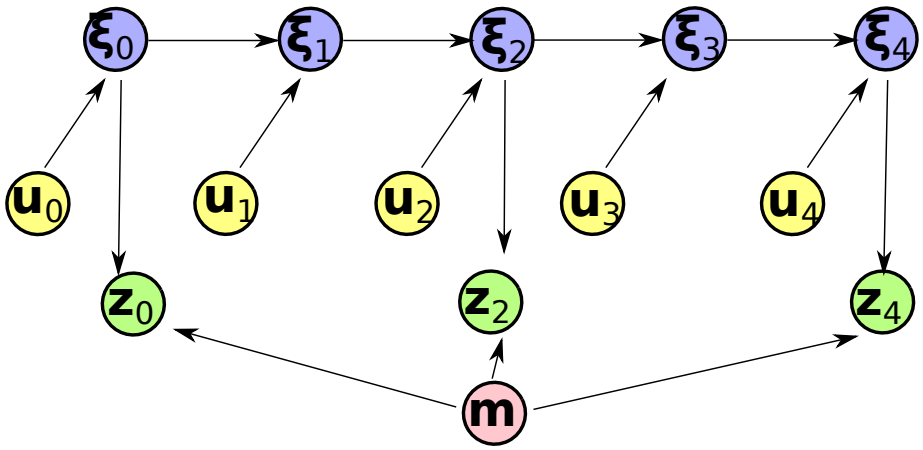


FIGURE 5.1: Graphical model of the SLAM problem: the goal is to estimate the pose of the robot ($\boldsymbol{\xi}_i$) and the map of the environment ($\boldsymbol{m}$). The control input $\boldsymbol{u}_i$ and measurements $\boldsymbol{z}_i$ are known. The causality of a relationship is indicated by the direction of the edge.

pose $\boldsymbol{\xi}_{i+1}$. The pose of the vehicle and the map are not directly observed by the robot and must be estimated using the odometry (or control input) and the measurements provided by an exteroceptive sensor. Note that odometry data is often provided at a higher rate than environment observations[1].

Sensor measurements are subject to different source of errors, e.g., sensor noise, erroneous data, wrong data association, and so on. For this reason, SLAM is often stated as a probabilistic problem. Two probabilistic formulations are presented next: full SLAM and online SLAM.

Problem 1. **(Full SLAM)** The full SLAM problem computes the probabilistic distribution of the map and the pose of the robot as

$$p(\boldsymbol{\xi}_{1:t}, \boldsymbol{m}_t | \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}, \boldsymbol{\xi}_0). \tag{5.1}$$

The probability distribution in (5.1) is not causal: the probability of a state is computed by resorting to data from the future.

Problem 2. **(Online SLAM)** The online SLAM problem computes the probabilistic distribution of the map and the pose of the robot at time t considering the previous pose and measurements, that is,

$$p(\boldsymbol{\xi}_t, \boldsymbol{m}_t | \boldsymbol{\xi}_{0:t-1}, \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}). \tag{5.2}$$

In essence, in contrast to full SLAM, online SLAM is causal.

[1]Odometry and environment measurements are not well synchronized as Fig. 5.1 may suggest. The navigation filter presented in Sec. 3.3 addresses this synchronization issue.

Solving the joint probability in (5.2) is decoupled in a localisation and a mapping problem. For the localization it is assumed that the map is known, that is,

$$p(\boldsymbol{\xi}_t|\boldsymbol{\xi}_{0:t-1}, \boldsymbol{m}_t, \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}). \tag{5.3}$$

On the other hand, the mapping problem assumes that the localization is known:

$$p(\boldsymbol{m}_t|\boldsymbol{\xi}_{0:t}, \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}). \tag{5.4}$$

As discussed in Chapter 1, most sophisticated SLAM frameworks rely on a two stage architecture called front-end and back-end stages. The front-end runs an online SLAM algorithm, while the back-end runs an offline SLAM algorithm. The latter is similar to the full SLAM problem in (5.1), but instead of continuously estimating the probability distribution for all data, it solves the problem for a finite set of correlated poses and measurements.

There are a variety of mapping and localization techniques which can be combined to deliver a specific SLAM strategy. The literature review focused on front-end range-based SLAM, from classical and well-established solutions to more recent approaches that have shown promising results. Section 5.2 presents the map models as well as mapping strategies to build them. Then, Sec. 5.3 focuses on SLAM frameworks, discussing the localization techniques associated to each of them.

5.2 Mapping

A map is an important asset for any mobile robot as it is required for localization, motion planning, human interaction, task execution and monitoring, etc. Over the years several mapping techniques have been proposed by the robotics community. In an effort to standardise the mapping data representation, the IEEE standard for robot maps [36] classified models into two categories, namely, topological and metric maps. A topological map is a high-level representation of the world, which is most often encoded in a graph. Nodes describe places or objects in the world, while edges describe the topological relationship between those nodes. An edge connecting two nodes usually denotes adjacency of these two nodes in the real world. A topological map provides vital information for high level navigation, but it does not preserve the scale or shape of the environment. Metric maps, on the other hand, encode the metric distance between elements in their model. This is

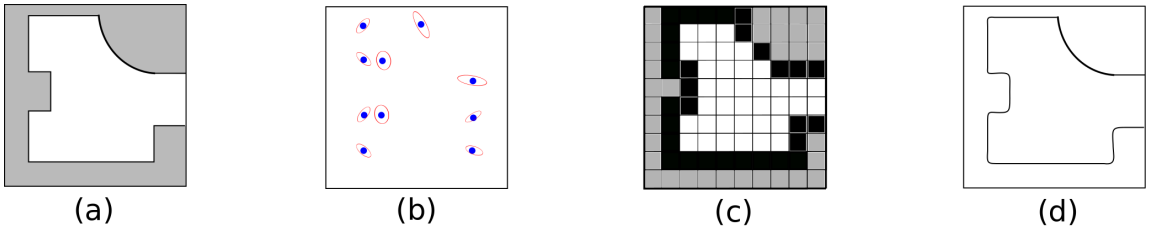


FIGURE 5.2: Different representations of the environment presented in (a): landmark-based map (b), discrete grid map (c), and continous map (d).

convenient for SLAM, since the underlying idea in localization is computing the displacement between the map and sensor readings. As illustrated in Fig. 5.2, metric maps can be further categorized into landmark, discrete grid, and geometric representations [37]. The last method is categorised here in a more broad group, which it is called continuous maps.

Landmark maps

Landmark maps store artificial or non-artificial features detected by the sensors of the robot. The first group contains artifacts introduced by humans, such as reflectors or artificial tags. Non-artificial landmarks can be seen as critical points extracted from raw measurements, such as corners or pillars. The Victoria park dataset [38] is a well-known dataset for evaluating landmark based SLAM. It contains LiDAR readings acquired while driving in an outdoor scenario populated with trees.

A landmark is described by a short descriptor, which encodes its pose and a few other parameters to identify it. Also, in general, landmarks are sparsely distributed in the environment. Consequently, a range sensor may collect a considerable amount of data but only a small subset is stored in the map (Fig. 5.2b). Since landmark maps are a compact representation of the environment, the model scales well for large environments and model-related data exchange among robots is relatively efficient. However, proper landmark-based localization requires a densely covered environment. The same applies for the execution of other tasks, including motion planning and exploration.

Discrete maps

[**Occupancy-grid**] Discrete grid representation consists in decomposing the world into small cells (Fig. 5.2c). The most well-known paradigm in this category are the discrete occupancy-grid maps. Early work on occupancy grid maps was done by Moravec and

Elfes [39] for robot motion planning and navigation in cluttered environments. The key idea is assigning to each cell an occupancy probability, which is reinforced as free or occupied by multiple sensor readings. Mathematically, let the occupancy probability of the *i-th* cell of a map be $p(\mathbf{m}_i)$, then it can be shown that assuming that the pose of the robot ($\boldsymbol{\xi}_t$) is known, the recursive approach holds:

$$\log \text{odds}(m_i = 1 | z_{1:t}, \boldsymbol{\xi}_{1:t}) = \log \text{odds}(m_i = 1 | z_{1:t-1}, \boldsymbol{\xi}_{1:t}) + \log \text{odds}(m_i = 1 | z_t, \boldsymbol{\xi}_t). \quad (5.5)$$

The previous equation reads as *the occupancy probability of the i-th cell after incorporating the measurements up to time t (posterior) is equal to the occupancy probability up to the previous time (prior) updated by the evidence*. The cells that the sensor has reported as occupied have a positive evidence, increasing the occupancy. On the other hand, the cells reported as free have a negative evidence, decreasing the occupancy. The occupied space is directly provided by the sensor, while the free space has to be inferred from the data. The occupied space corresponds to the points that the sensor beam was reflected by an obstacle. The free space is computed via ray rasterisation, e.g., Bresenham's line algorithm [40]. In essence, for each sensor beam it is assumed that the cells that lie between the robot and an obstacle (or the maximum range if no obstacle is detected) correspond to free space. The cell size can be chosen to be arbitrarily small, which allows for an accurate description of the environment at the expense of storage memory. The map is stored in an indexed matrix and as a consequence it is fast to evaluate or update a cell state - both are $O(1)$. Most likely, these are the reasons that led the SLAM community to favor occupancy grid maps, e.g., [4, 5, 12]. The major drawbacks of this method are: 1) it discretises high resolution measurements, 2) it does not account for the fact that nearby regions have similar occupancy state, 3) misalignment between cell and obstacles can deteriorate, and 4) it does not scale well to large environments.

[**TSDF**] For the particular task of SLAM, it has been observed that non-brute force localization techniques operating on discrete occupancy grid maps have a small region of convergence [41]. To alleviate this problem, the use of truncated signed distance function (TSDF) maps has been investigated. TSDFs maps have been exploited in the vision community for quite some time [42]. Nonetheless, TSDF maps for range data is a relatively recent topic with interesting results, see the work in [41, 43] (discussed in the next section). Instead of storing the occupancy probability, the cells in a TSDF map represent the

distance to the nearest object in the map. Therefore, it has two advantages over occupancy maps: 1) signed distance functions are differentiable and 2) it provides sub-cell accuracy. However, it is not a probabilistic map: a dynamic obstacle is added to the TSDF map after a single reading, while in occupancy-grid maps the obstacle must be observed multiple times before being added to the map.

[**Miscellaneous**] There are many other flavours of discrete maps, such as the exact and adaptive cell decomposition maps [40]. In exact cell decomposition maps boundaries are extracted from critical points, defined by the geometry of the environment. The free configuration space is represent as nodes in a graph. Therefore it can capture large areas in a compact representation. Also, it is able to represent the connectivity of the environment and transversable areas. An adaptive cell decomposition map (quadtree) is an hybrid between the fixed cell and exact cell decomposition. Free space is represented by large rectangles, while occupied cells are decomposed in smaller ones up to a pre-defined resolution. These maps are more commonly employed in motion planning and other tasks where speed and accuracy are not as critical as in SLAM.

Continuous maps

In continuous or geometric representations obstacles are annotated individually using continuous functions or geometric primitives, e.g., lines or polygons. These features have a continuous range of values, i.e., floating point resolution [44]. Using the closed-world assumption, regions of the map that do not contain any annotation correspond to obstacle-free areas. Crowley [45] proposed one of the first successful geometric models, based on line segments. Diedrich et al. [46] proposed a representation that uses polygonal curves and addressed a matching methodology to find previously mapped curves. Vásquez-Martín et al. [47] modelled the environment using points, linear segments, and curved lines. The computed map is accurate, but it cannot handle the inherent uncertainty that arises from noisy sensor data. A compact, and yet accurate representation using wireframes is shown in [48]. The method performs well while the assumption that the environment can be represented by line segments holds, e.g., straight walls. To cope with map uncertainty, their solution relies on a particle filter that assigns a particle to each wireframe candidate. The main drawbacks in most geometric approaches are merging the geometric primitives and the cost to evaluate whether a point in the space belongs to the occupied or free space. *Gaussian Processes* (GPs) maps [49, 50] tackle the aforementioned limitations

at the expense of computational complexity. Yuan et al. [51] presented for the first time an online GP-mapping. This promising method was shown to be able to map regions at rates close to 10 Hz, which is fast enough for some mapping applications.

5.3 SLAM

Landmark

Theoretical tools that deal with the probabilistic uncertainty associated with landmarks have been developed over the years, making landmark-based SLAM a mature technique by now [8]. See for instance the seminal book by Thrun et al. [40] that covers landmark-based SLAM algorithms using the two techniques discussed here: Extended Kalman Filters (EKF) and Rao-Blackwellized particle filters (RBPF).

[EKF] The landmark based EKF-SLAM was a pioneering solution proposed by Smith et al. [52] in the early 1990's. The underlying idea is representing the prior and the posterior distributions of the SLAM state using Gaussian distributions. The algorithms has two stages: prediction (time-update) and correction (measurement update). The prediction usually runs at a higher rate than the correction step. Based on a motion model (kinematics or dynamics of the vehicle), the prediction updates the pose of the platform using odometry data (IMU, wheel enconderes, etc) or simply assuming a constant velocity

Algorithm 3: Landmark based EKF SLAM

Input: $\hat{\boldsymbol{q}}_{t-1}$, $\hat{\boldsymbol{P}}_{t-1}$ $\boldsymbol{u}_{t-1}$, and $\boldsymbol{z}$
Output: $\hat{\boldsymbol{q}}_t, \hat{\boldsymbol{P}}_t$

/* Prediction step: */
1: $\check{\boldsymbol{\xi}}_t = f(\hat{\boldsymbol{\xi}}_{t-1}, \boldsymbol{u}_{t-1}, \boldsymbol{0})$
2: $\check{\Sigma}_t = \frac{\partial f}{\partial \boldsymbol{\xi}} \hat{\Sigma}_t \frac{\partial f}{\partial \boldsymbol{\xi}}^T + \frac{\partial f}{\partial \boldsymbol{u}} Q_t \frac{\partial f}{\partial \boldsymbol{u}}^T)$
/* Correction step: */
for $z_i \in \mathbf{m}_{t-1}$ **do**
  /* Landmark is already in the map: */
  3: $H_{2i:2i+2,:} = \frac{\partial h_i}{\partial \boldsymbol{q}}(\check{\boldsymbol{q}}_t, \boldsymbol{0})$
  4: $\tilde{r}_i = \mathbf{z}_i - h_i(\check{\mathbf{q}}_t, \boldsymbol{0})$
5: $K = \check{P}_t H^T (H \check{P}_t H^T + V_t)$
6: $\hat{\boldsymbol{q}}_t = \check{\mathbf{q}}_t + K\tilde{\mathbf{r}}$
7: $\hat{P}_t = (I - KH)\check{P}_t$
/* Initialize landmark in the map */
for $z_i \notin \mathbf{m}_{t-1}$ **do**
  8: $\hat{\mathbf{m}}_t, \hat{\mathbf{P}}_t \leftarrow AddLandmarktoMap(\hat{\mathbf{m}}_t, \hat{\mathbf{P}}_t, \boldsymbol{z}_i)$

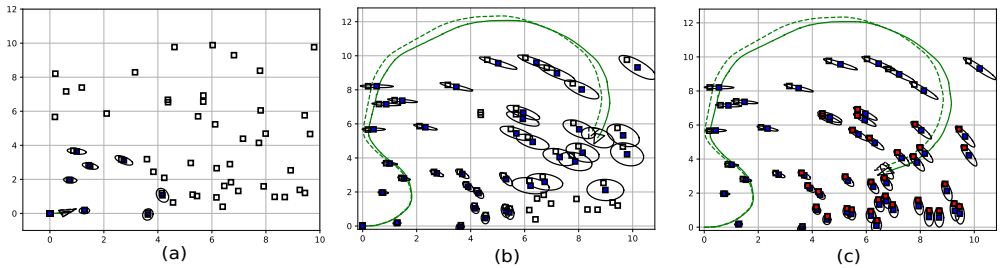


FIGURE 5.3: Landmark based EKF-SLAM. The images show (a) the beginning of the trajectory, (b) before revisiting initial landmarks, and (c) initial landmarks are revisited. The true and estimated pose (trajectory) vehicle are indicated by a intermittent and solid triangle (line). The ellipse around the vehicle and landmarks corresponds to 99.7% confidence interval. Code available at https://github.com/C2SR/pyarena

model. The uncertainty of the pose of the robot grows, but the uncertainty of initialized landmarks remains constant. In the correction step, measurements of the environment are incorporated by means of an observation model. Observing the environment decreases the uncertainty associated to the robot pose and landmark locations. It also allows to initialize new landmarks that were not observed before.

An online EKF-SLAM is presented in Algorithm 3. Let n be the number of landmarks in the map and the state be denoted by $\mathbf{q} \in \mathbb{R}^{3+2n}$, such that

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\xi}^T & | & \boldsymbol{m}^T \end{bmatrix}^T = \begin{bmatrix} x & y & \phi & | & \mathbf{m}_1 & \dots & \mathbf{m}_n \end{bmatrix}^T, \tag{5.6}$$

where $\mathbf{m}_i = [m_{i,x}, m_{i,y}]^T \in \mathbb{R}^2$ are the coordinates of a 2D landmark described in the map frame. The prediction (steps 1-2) computes the prior distribution $\mathcal{N}(\check{\boldsymbol{q}}_t, \check{\boldsymbol{P}}_t)$ using the motion model presented in Sec. 3.3. The correction (steps 3-7, Algorithm 3) updates the state (map and pose) via the measurements to obtain the posterior distribution $\mathcal{N}(\hat{\boldsymbol{q}}_t, \hat{\boldsymbol{P}}_t)$. Assuming that the Cartesian coordinates of landmarks are observable, the observation model takes the form

$$h_i(\boldsymbol{q}, \mathbf{v}_t) = R(\psi)^T(\mathbf{m}_i - \mathbf{p}) + \mathbf{v}_t, \tag{5.7}$$

where $\mathbf{v}_t$ is Gaussian white noise. The main advantages of EKF-SLAM is that it is simple to understand/implement and the results are good if neither the motion model nor the observation model violates the Gaussian assumptions. Loop-closure is explicitly performed every time the robot re-observes a mapped feature. This is shown in the example illustrated in Fig. 5.3. The uncertainty associated with the first observed landmark is low. As the vehicle navigates, pose uncertainty slowly grows and consequently landmarks initialized along the trajectory have a large uncertainty. Whenever the vehicle revisits the

landmarks which were observed in the beginning of the trajectory, the overall map uncertainty decreases. Loop-closure drastically reduces the uncertainty of the pose of the vehicle and landmarks because the relative localization of landmarks are tightly coupled via the covariance matrix.

The main drawbacks are that 1) the computational effort to incorporate an observation grows quadratically in the number of landmarks, 2) the method requires landmark detection and data association (a major research topic in itself), and 3) the Gaussian model assumptions are often violated in the real world. The data association step is responsible for extracting landmarks from the raw laser data and uniquely identifying each feature in the map. Poor data association, e.g., mismatching, can be catastrophic. Moreover, bad data association and the non-linearities of the motion model may push the Gaussian assumptions to the limit leading to poor and overconfident state estimation. To keep the dimension of the filter compact, Dissanayake and colleagues [53] proposed removing landmarks based on the information content. Later, Guivant and Nebot presented a computationally efficient online algorithm in [54]. Efficiency increases by using a submap approach and limiting the maximum number of landmarks, favoring the ones that provide maximum information. However, it would not be until the introduction of particle filters that SLAM would jump from a few hundreds to thousands of landmarks.

[RBPF] Murphy [55] was one the first authors to apply Rao-Blackwellized particle filter to the solution of the SLAM problem. A few years later, Montemerlo et al. presented FastSLAM [56], an online strategy able to deal with thousands of landmarks revolutionizing landmark-based SLAM. In FastSLAM, a particle (or sample) represents a hypothesis on the path of the robot - leading to the term multi-belief filter. It applies the concepts of particle filter and, similar to EKF-SLAM, it has a prediction and a correction step. A particle (or sample) represents a hypothesis of the pose of the robot. Each particle has a set of independent Extended Kalman Filter associate to each landmark initialized in the map. Therefore, for K particles and M landmarks, there are KM Kalman filters. The strategy can be split into pose estimation and landmark location estimation. Pose estimation requires first propagating the particles by sampling the probabilistic motion model, a process called sampling from the proposal distribution. Later FastSLAM 2.0 [57], would also take into account measurements for computing the proposal distribution improving the accuracy of the solution. The correction step assigns importance weights to particles

according to the likelihood of the observations reported by the sensor. High weights indicates particles that are likely to explain the state of the system correctly. A resampling policy applies the survival of the fitness principle: particle with high weights are likely to survive and low weight particles are likely to be removed. For landmark location estimation, each particle keeps track of the position of the landmarks. However, in contrast to EKF-SLAM, only observed landmarks are impacted by measurements. The posterior of non-observed landmarks remains unchanged. Moreover, for computationally efficiency, a tree-based structure decreases the complexity from quadratic (EKF) to logarithm in the number of landmarks. The key advantages of FastSLAM are that it 1) relaxes the Gaussian assumption, 2) deals better with non-linearities, and 3) is computationally more efficient than previous approaches. Loop closure is implicitly performed by particle weighting and re-sampling. The key to avoid the particle depletion problem are increasing the number of particles, using accurate proposal distributions, and drawing and resampling particles in an effective way. Since each particle carries an individual map, the maximum number of particles is limited by the computational power available.

Occupancy grid

In landmark-based SLAM, the environment must be populated with a reasonable number of landmarks, which have to be extracted from the sensor data and correctly matched against features in the map at each laser scan. In contrast, occupancy grid-based SLAM is a dense method that uses all the measurements from the range sensor.

[Scan-to-scan] The popularization of occupancy-grid based SLAM took place after the first wave of landmark based SLAM. Nevertheless, occupancy grid maps have been around since 1985, when Moravec and Elfes [39] proposed them as method for registering dense range measurements. Paramount for its usage in SLAM, was the emergence of scan-to-scan alignment methods. Mathematically, scan-to-scan can be stated as a maximum likelihood problem:

$$\boldsymbol{\xi}_t^* = \arg\max_{\boldsymbol{\xi}_t} p(\boldsymbol{\xi}_t | \hat{\boldsymbol{\xi}}_{t-1}, \boldsymbol{z}_{t-1:t}). \tag{5.8}$$

The goal is to find the pose that most likely describes the current scan reading ($\boldsymbol{z}_t$) given the previous pose ($\hat{\boldsymbol{\xi}}_{t-1}$) - note that a map itself is not employed (but can be built!). The IDC (and derived algorithms) is one of the most influential work in scan-matching for range based SLAM. Originally presented by Lu and Milios [58], IDC stands for Iterative Dual Correspondence method. It performs a two stage search, where each stage finds the

relative pose $\boldsymbol{\xi}_{t,t-1}$ from time t to time $t-1$ that minimizes the least square error

$$\boldsymbol{\xi}^*_{t,t-1} = \arg\min_{\boldsymbol{\xi}_t} \sum_i \|\mathbf{z}_{t-1,j} - T(\boldsymbol{\xi}_{t,t-1}, \mathbf{z}_{t,i})\|^2, \tag{5.9}$$

where $\mathbf{z}_{t-1,j}$ is the point j from the the previous scan reading that is matched against point i in the current iteration according to a rule and the function $T(\cdot)$ is defined as in (3.3). The first search (closest-point rule) minimizes the translation error by matching the points that are the closest and minimizing their distance (similar to the computer vision Iterative Closest Point algorithm by Besl and McKay [59]). The second stage refines the previous estimate to obtain the rotation by matching range points that are within the same distance in their respective reference frame (matching-range point rule). The two stages are performed iteratively until convergence. The output of IDC is the relative pose. For localization, the relative pose can be integrated throughout the frames to obtain an absolute pose w.r.t. the initial frame, i.e., map frame.

[Scan-to-map] Hähnel et al. [60] presented a SLAM method inspired in IDC. In the context of map building and localization via scan matching, the key differences are two: 1) the authors built a local occupancy grid map with the most recent T measurements, denoted here as $\boldsymbol{m}_{t-1,T}$ and 2) the motion model is incorporated to the problem formulation. This is achieved by maximizing the marginal likelihood

$$\boldsymbol{\xi}^*_t = \arg\max_{\boldsymbol{\xi}_t} p(\boldsymbol{\xi}_t|\boldsymbol{m}_{t-1,T}, \boldsymbol{z}_t) p(\boldsymbol{\xi}_t|\hat{\boldsymbol{\xi}}_{t-1}, \boldsymbol{u}_{t-1}). \tag{5.10}$$

The first term on the right-side considers the consistency of the local grid map with the measurements, while the second term accounts for the consistency of the new estimated pose at time t with the previous estimated pose and control action (motion model). After computing the new pose, the local map is updated as in [11]. Algorithm 4 describes the steps for a vanilla implementation of three occupancy-grid based SLAM. In the preprocessing stage (steps 1-2) wrong measurements (e.g., below minimum range) are discarded and the valid polar measurements are transformed to Cartesian coordinates. During localization (step 3) a scan-to-map technique, such as maximum likelihood, finds the pose that best align the current measurements against a local (or global) map. After that, mapping takes place (step 4-6): free cells are computed via ray rasterization, detected occupied and free cells are transformed to the map frame, and the corresponding map cells are updated.

Locally, maximum likelihood using a local discrete occupancy grid map yields good

Algorithm 4: Discrete occupancy-grid SLAM

Input: $\hat{\boldsymbol{\xi}}_{t-1}$ and $\boldsymbol{z}$
Output: $\boldsymbol{\xi}_t^*, \hat{\boldsymbol{m}}_t$

```
/* Pre-processing:                                                    */
```
1: Remove spurious measurements
2: Transform meas. from polar to Cartesian coord.: $\boldsymbol{z}_i \leftarrow$ (3.2)
```
/* Localization:                                                      */
```
3: Solve scan-to-map alignment to estimate pose of the robot:
    3a: maximum likelihood (single-belief): $\boldsymbol{\xi}_t^* \leftarrow$ (5.10)
    3b: full posterior (multi-belief): $\boldsymbol{\xi}_t^* \leftarrow$ (5.11)
    3c: non-linear least squares (single-belief): $\boldsymbol{\xi}_t^* \leftarrow$ (5.12)
```
/* Mapping:                                                           */
```
for $\boldsymbol{z}_i \in \mathbf{z}$ **do**
    4: Compute free space using a ray rasterization technique
    5: Transform free and occupied space to inertial frame using estimated pose of the robot: (3.3)
    6: Update cell occupancy: $\mathbf{m}_t \leftarrow$ (5.5)

results, but small drifts still make it hard to deal with large loop closure. In the search for long-term SLAM, researchers turned to multi-belief filters and offline strategies.

[Scan-to-map PF] Thrun et al. [61] explored particle filters using a local map representation similar to [60]. Instead of maximum the marginal likelihood, the authors maximize the full posterior distribution:

$$\text{Bel}(\boldsymbol{\xi}_t) = p(\boldsymbol{\xi}_t | \boldsymbol{m}_{t-1}, \boldsymbol{z}_t, \boldsymbol{u}_{1:t}),$$

where *Bel* stands for the belief of the robot about a particular event. Applying Bayes Law (Definition 2.5), Markov Assumption (Definition 2.6), and Total Probability Law (Definition 2.4), it is possible to rewrite the full posterior as

$$\text{Bel}(\boldsymbol{\xi}_t) = \eta p(\boldsymbol{z}_t | \hat{\boldsymbol{\xi}}_t, \boldsymbol{m}_{t-1}) \int p(\boldsymbol{\xi}_t | \hat{\boldsymbol{\xi}}_{t-1}, \boldsymbol{u}_{t-1}) \text{Bel}(\hat{\boldsymbol{\xi}}_{t-1}) d\hat{\boldsymbol{\xi}}_{t-1}, \tag{5.11}$$

where η is a normalizing factor, $p(\boldsymbol{\xi}_t | \hat{\boldsymbol{\xi}}_{t-1}, \boldsymbol{u}_{t-1})$ is the motion model derived from the kinematics of the vehicle, and $p(\boldsymbol{z}_t | \boldsymbol{\xi}_t, \boldsymbol{m}_{t-1})$ is the observation model. The latter is computed under the assumption that it is not likely that occupied space in previous measurements are detected as free space in the current scan reading. The posterior distribution is approximated using samples. Then, for each sample, the solution for (5.11) is computed via gradient ascent. The results are better than single-belief filters. Nonetheless, degradation of the localization becomes critical in long trajectories with loop-closure. This is

addressed using an offline method that corrects a batch of pose estimations when the disparity between the maximum a posterior and maximum likelihood estimates is non-null.

[Scan-to-map RPBF] An online grid-based RBPF is presented in [62]. A scan-matching routine transforms sequences of range measurements into accurate odometry measurements. The range-based odometry has variance lower than pure wheel encoder odometry and it is employed for generating more accurate proposal distribution for every particle. The lower the variance of the proposed distribution, fewer samples and less resampling steps are required, increasing the robustness and speed of the algorithm. In [63], instead of having a fixed proposal distribution for all the particles, the authors propose running a scan-matching process for each particle. While it enhances the proposal distribution for an individual particle, the odometry information is not properly considered. The authors also use an adaptive resampling that estimates the performance of a set of particles, which allows to reduce the number of resampling operations. The GMapping algorithm by Grisetti et al. [4], an efficient and mature grid-based RBPF, became a popular SLAM solution and it is still widely employed by the robotic community. It is an extension of [63] that considers the odometry information. The enhanced proposal distribution draws new particles more accurately, decreasing the number of particles required and improving the results.

[Scan-to-map NLS] Kolbrecher and colleagues developed Hector-SLAM [5], yet another popular method within the robotics community. In fact, Hector-SLAM and the method proposed in Chapter 7 (B-spline Surface SLAM) have a similar structure. In Hector-SLAM, a navigation filter incorporates IMU data at high rates to generate prediction of the pose of the robot. For the localization step, it is assumed that an occupied cell has the value 1 (normalized). The localization is formulated as a non-linear least square problem:

$$\boldsymbol{\xi}_t^* = \arg\min_{\boldsymbol{\xi}_t} \sum_i \|1 - m_c(T(\boldsymbol{\xi}_t, \mathbf{z}_i))\|^2, \tag{5.12}$$

where $m_s(\cdot)$ is a function that computes the value of the map at a given point using bi-linear interpolation. In other words, a continuous map and its gradient is obtained from the discrete map by bi-linear filtering. The localization problem is solved solved using a Gauss-Newton method. Since bi-linear interpolation is a non-smooth linear approximation of the map gradient there is no guarantees of local quadratic convergence. Multi-resolution map representation mitigate local minima issues. The estimation computed

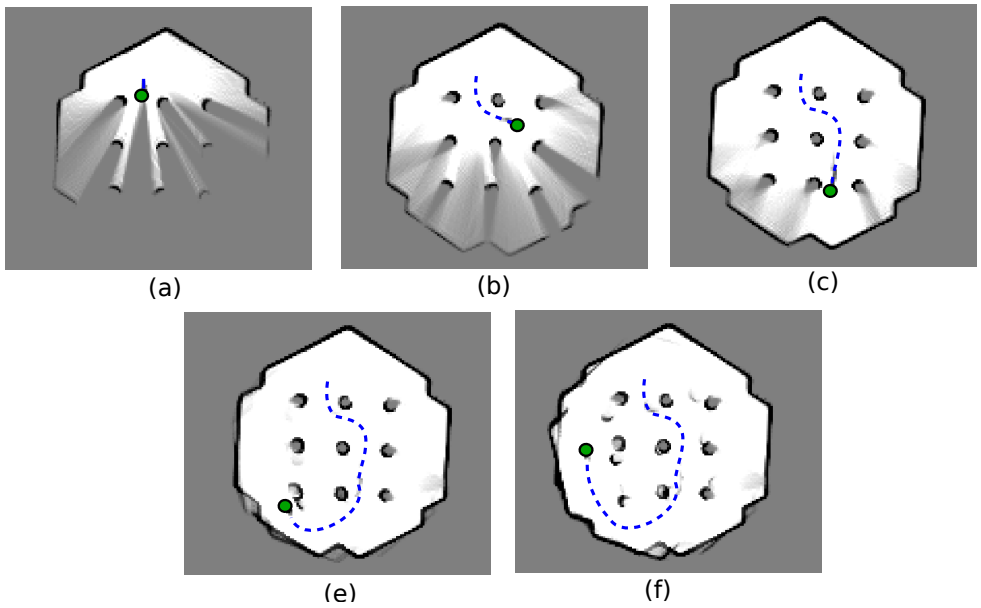


FIGURE 5.4: Occupancy-grid based SLAM. From top left to bottom right, the images shows a vanilla implementation of the NLS SLAM using bi-linear interpolation. The vehicle is depicted by a green circle and the trajectory by a dashed line. The code and dataset for reproducing the results are available at: https://github.com/C2SR/pyarena

using a low resolution map is used as hot start for a higher resolution map. The algorithm is able to work at sensor rate speed and the results using a sensor with high scan rates and low noise are accurate. Figure 5.4 shows the results for a vanilla implementation of the NLS SLAM using bi-linear interpolation that follows the steps described in Algorithm 4. The range data obtained using a turtlebot 3 burger in Gazebo simulator. The vehicle starts at the top of the map. The method performs well when the vehicle goes under low rotations. However, due to the low sensor rate (5 Hz), it fails the robot perform sharp curves (see distortion in Fig. 5.4f).

More recently, Hess and colleagues proposed Google Cartographer [12] - the gold standard in 2D range based SLAM. This method explicitly detects loop closure through a branch-and-bound approach and performs offline pose optimisation. The front-end is very similar to Hector-SLAM, however, continuous maps are obtained using bi-cubic interpolation. As a consequence, the occupied cells of the normalized discrete map which should have maximum value at 1 - as in (5.12) - are likely to have values higher than the norm.

Fossel et al. [41] presented the first TSDF-based 2D range based SLAM. The localization technique is very similar to the one presented in Hector-SLAM (Gauss-Newton), however the map is represented by a signed distance function - instead of representing the

occupancy probability, the map captures the distance to the closest obstacle. Since signed distance functions are continuous, no interpolation is required for solving the localization problem. The authors report results for the front-end using as metric the root-mean square deviation from the ground truth. In simulations, the results are up to 270% better than Hector-SLAM, while in real data the improvement is 14%. In [43], Daun and colleagues, extend the previous work to account for loop-closure using the branch and bound technique developed in [12] and incorporating offline optimization. Their offline TSDF-based SLAM outperforms Google Cartographer (offline occupancy-grid based SLAM). The authors credits the improvement due to the nature of the map, which is able to represent the environment more accurately.

Continuous

A few geometric strategies were presented in the very beginning of range based SLAM. Then, in the period 1990-2010 landmark and occupancy-grid techniques were dominant. However, in the last decades, as range data becomes more dense and accurate, geometric SLAM has drawn considerable attention from the research community due to the fact that it allows representing the data in a continuous manner.

[Lines] Crowley [45] proposed representing the map and estimating the pose of the robot using lines extracted from the range data. The authors proposed representing the lines by a set of minimal and redundant parameters. The minimal set comprises the midpoint of the segment, orientation, half-length and associated uncertainty, while the redundant set includes the distance to the origin, the perpendicular intercept, and end-points. Line matching is performed by comparing three metrics: orientation, alignment, and distance of the center points to the sum of the half length of each line. The longest line segment that conforms to the three metrics is used for estimated the pose of the robot using an algorithm inspired in the Kalman filter update equations. The position is estimated using the perpendicular distance between matched segments, and the orientation via the difference of the angle between the observed segment and the model in the map. The map is enlarged and updated by recomputing the minimal and redundant parameters.

More recently, Holý [64] presented a line-based algorithm for computing the pose displacement between two consecutive sensor readings. Lines are extracted from a sensor reading in a two-step algorithm. First, using the split-and-merge algorithm, points are classified in different lines. Then, via RANSAC, the method attempts to detect more lines

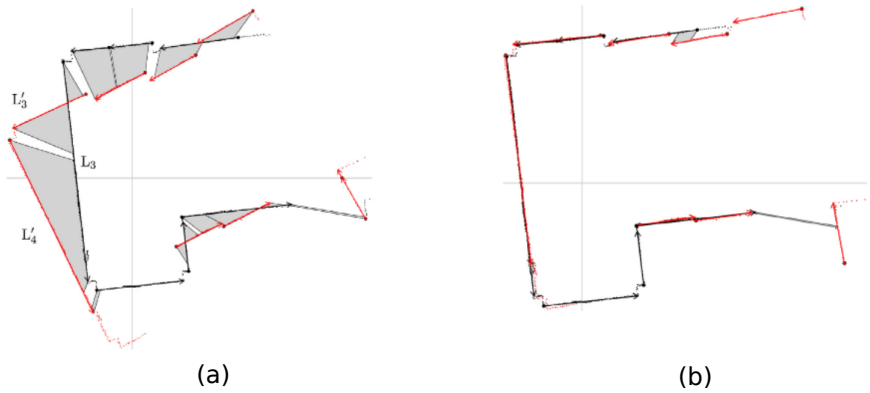


FIGURE 5.5: Line-based scan matching: two consecutive laser scans depicted in black and red lines are shown (a) before and (b) after the alignment process. The gray area corresponds to the cost function to be minimized. Source: [64].

on the remaining points which split-and-merge failed to match against a line candidate. The alignment error function is a continuous function of the lines (gray region in Fig. 5.5a) and gradient descent is applied for obtaining the transform that minimizes the alignment error (see Fig. 5.5b). The minimization takes place in the sensor space and, therefore, the method is more robust against wrong line associations. The author does not cover the problem of map merging and updating.

[Polynomials] The work by Pedraza et al. [65] addressed the SLAM problem using B-spline curves. Given a point cloud, the first step is clustering the points that belongs to the same curve, which is accomplished using the distance and orientation between consecutive points. After fitting a B-spline curve to each cluster, comes the matching step: finding the curve in the map that corresponds to the one observed in the current reading. The authors addressed matching by computing the distance between the control points the curves. This is critical because two similar B-spline curves can be represented with different control points. The SLAM algorithm is an EKF that keeps tracks of the control points of the B-spline curves and the pose of the robot. Later, Liu et al. [66] extended the spline-SLAM framework to take the covariance error of each control point into account improving the results. Zhao et al. [67] proposed using implicit functions for solving the SLAM problem. Like B-sspline, implicit functions allow representing a wide variety of shapes. The authors formulated SLAM as an energy minimization problem, where the optimisation vector describes the pose of the robot and the changeable parameters of the implicit functions that model the environment. They show that using implicit functions to represent geometric features outperforms using pre-fixed geometric shapes such as lines

and ellipses.

[Gaussian Process] Li et al. [68] proposed GP-SLAM, a Gaussian Process Simultaneous Localization and Mapping technique. The map is a GP model which takes the points from the scan reading as training data during the regression stage. The space is split into sub-regions and points are clustered in each of these regions. The normal directions for each region is computed using Principal Components Analysis (PCA) and the angle between the normal direction and the map axes. Then, using this data, the GP model is trained using classical GP-regression. For the localization task, testing points are extracted from the GP-model and compared with the observed data. However, instead of minimizing the square distance between two points, the method finds the pose that minimizes the difference between the eigenvalues of the covariance matrix associated to the testing and observed points.

5.4 Datasets and metrics

The previous section presented several SLAM strategies which were designed by considering different mapping strategies (landmarks, occupancy grid, signal distance functions, etc) and localization algorithms (Extended Kalman Filter, maximum likelihood, non-linear least square, etc). For comparison, these different SLAM frameworks must be evaluated using the same metrics and datasets. A metric provides a quantitative or qualitative performance indicator of a particular method, while a dataset allows to compare (via some metric) different methods operating on the same data. Since SLAM is a chicken-egg problem, measuring the performance of localization indirectly provides an insight on the performance of the mapping and vice-versa.

5.4.1 Metrics

The most straightforward metric is a qualitative one: visually inspecting the estimated map with the expected map. The latter map can be provided by a human operator who is familiar with the testing environment or by the blue-print of the building where the data was recorded. For instance, by comparing the SLAM result reported in Fig. 5.4 with the actual environment shown in Fig. 3.8, it is possible to conclude that the estimated map exhibits *some* unexpected artifacts.

Yagfarov et al. [69] compared different SLAM methods using precise ground truth for the map. In particular, the authors compared Gmapping, Hector-SLAM, and Google Cartographer, which they claimed to be the most popular range based SLAM techniques available in ROS in 2018. The data for comparison was collected with a ground vehicle equipped with a LiDAR sensor (range measurements) and wheel encoders (odometry). The vehicle navigates in an office room and the data comprises four difference cases regarding the dynamic of the vehicle: slow forward speed and smooth rotations, fast forward speed and smooth curves, fast forward speed and sharp curves, and no loop-closure. For comparing the map accuracy, the authors employed the average distance to the nearest neighbor (ADNN), which consists in computing the normalized sum of the distance between the nearest occupied cell in the map to the expected occupied cell in the ground truth model. The smallest errors in the different scenarios are scored by the Gmapping (fast ride, sharp rotations) and Google Cartographer (remaining scenarios).

A candidate for quantitative metric would be measuring the absolute pose estimation error. The issue is that obtaining the ground truth for absolute pose is complicated in the most simple scenario where the robot transverses a few rooms and corridors. This is due to the fact that motion capture systems are only viable in a single chamber-alike environments (e.g., a room), high precision laser distance meters require line of sight, and measuring instruments such as metric tapes do not provide the accuracy required for a ground truth. Having that in mind, Kummerle et al. [70] proposed four quantitative metrics that do not require an inertial reference frame. To introduce the metric, consider the following quantities:

- $\boldsymbol{\xi}_i$, $\boldsymbol{\xi}_j$: ground truth poses at time t_i and t_j, respectively.
- $\hat{\boldsymbol{\xi}}_i$, $\hat{\boldsymbol{\xi}}_j$: estimated poses at time t_i and t_j, respectively.

As previously discussed, in the general case the ground truth poses cannot be easily recovered. However, it is possible to measure the relative ground truth pose, which is denoted here as $\boldsymbol{\xi}_{i,j} = \boldsymbol{\xi}_j \ominus \boldsymbol{\xi}_i$, where $\ominus$ is the inverse of the standard pose composition operator. The metrics proposed in [70] compare the relative pose estimated by a SLAM algorithm with the ground truth by individually considering the translational and rotational components:

$$\epsilon_{trans,2}(\boldsymbol{\xi}) = \frac{1}{N}\sum_{(i,j)} \|trans(\boldsymbol{\xi}_{i,j} \ominus \hat{\boldsymbol{\xi}}_{i,j})\|_2^2, \qquad \epsilon_{rot,2}(\boldsymbol{\xi}) = \frac{1}{N}\sum_{(i,j)} rot(\boldsymbol{\xi}_{i,j} \ominus \hat{\boldsymbol{\xi}}_{i,j})^2; \tag{5.13}$$

$$\epsilon_{trans,1}(\boldsymbol{\xi}) = \frac{1}{N}\sum_{(i,j)} \|trans(\boldsymbol{\xi}_{i,j} \ominus \hat{\boldsymbol{\xi}}_{i,j})\|_2, \qquad \epsilon_{rot,1}(\boldsymbol{\xi}) = \frac{1}{N}\sum_{(i,j)} |rot(\boldsymbol{\xi}_{i,j} \ominus \hat{\boldsymbol{\xi}}_{i,j})|. \tag{5.14}$$

The motivation for separating the components is clear: translation is measured in meters, while orientation is described in radians (or degrees). The authors states that the squared metric in (5.13) corresponds to the energy required to transform the estimated relative pose into the ground truth, while the absolute error (5.14) does not have a practical meaning but it has been historically well accepted by the community. In both cases, the metrics are related to local consistency.

5.4.2 Dataset

The Robotics Data Set Repository (Radish for short) [71] is a large collection of data publicly available thanks to the contribution of several research groups around the world. It contains logs of laser, sonar, and odometry data recorded using real and simulated robots. The corresponding maps generated by humans and robots are also available for qualitative assessment. For example, Figure 5.6 illustrates the map for the data recorded in MIT-CSAIL building dataset. The great advantage of the RADISH repository is that a fair ground truth for several relative poses were obtained by human operators with knowledge of the building [70]. As a matter of fact, there are several papers, e.g., [12, 72] that report results generated using the RADISH data and the metrics described in (5.13) and (5.14). The RADISH logs for which the relative pose measurements are available were recorded using a LiDAR sensor with 180 deg field of view (1 deg angular interval) operating at approximately 5 Hz. The odometry measurements (provided from wheel

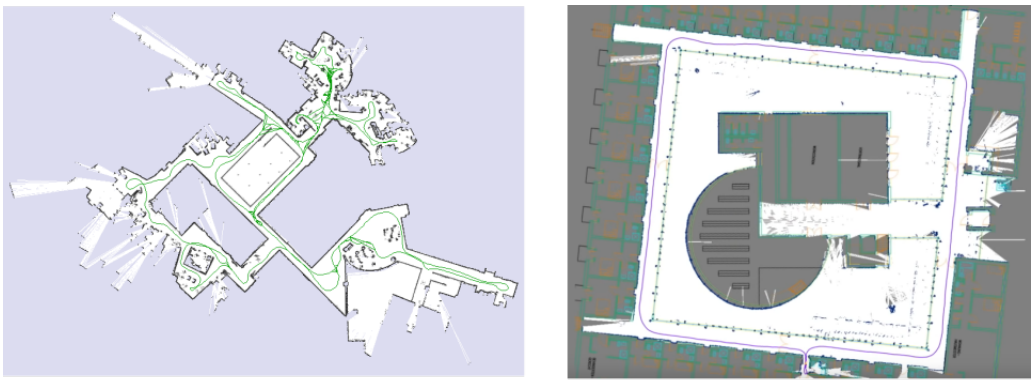


FIGURE 5.6: SLAM datasets: (a) map of the MIT-CSAIL building from the RADISH dataset [71]; (b) map with the blue-print of the Schloss Dagstuhl New Building from the TU-Darmstadt dataset [5].

encoders) are about the same rate. For these reasons, these logs are challenging for most front-end SLAM. In fact, to the author's knowledge, GMapping is the only pure front-end SLAM which is able to correctly process all these logs. However, these results, which are reported in [70], were obtained by pre-processing the scan measurements for enhancing the odometry before running the SLAM algorithm.

The TU-Darmstadt repository [73] contains data recorded using a Hokuyo UTM-30LX LiDAR, which operates at 40 Hz and it has a 270 deg field of view, 0.25 deg angular interval. There are three different scenarios, including the Schloss Dagstuhl building shown in Fig. 5.6. In common, the datasets focus on Urban Search and Rescue missions. However, in some scenarios the data was recorded using a robot, while in others using a handheld sensor kit. A ground truth is not available, so these logs are only good for qualitative comparison.

5.5 SLAM Limitations

The literature review presented in this chapter focuses mainly on map representations and localization techniques, which are the core of any SLAM framework. However, there are other active research topics, such as loop-closure detection and validation, robustness in the presence of dynamic obstacles, and pose-graph optimization. In fact, to the author's knowledge, there is no SLAM strategy that solves all the possible scenarios envisioned for sensor-controlled robots, and it is not the goal of this thesis to provide such a solution. Instead, special attention is given to the following SLAM limitations:

- **[Geometric SLAM] Representation of complex geometric shapes:** Lines have been the default geometric feature for representing geometric shapes in SLAM. The richness of B-splines has been explored in [65], but tracking control points instead of the curve itself is not appropriate due to the fact that control points are not observable and two set of different control points may yield the same curve.

- **[Occupancy-grid SLAM] Cell/obstacle misalignment**: The cells of a discrete grid-map are aligned with the coordinates of the map frame. On the other hand, obstacles in the simplest human-made environment will likely fail to be aligned with the cells of a discrete map, e.g., an empty square room with straight walls. Once a scan reading has been fully processed by the mapping algorithm, the raw data is discarded by the front-end stage. Thus, while it is possible to infer whether a cell (a region) is

occupied or not, it is not possible to recover the location within the cell that led to its occupancy. While TSDF-SLAM [43] has shown promising results to diminish the problem, it is sensible in the presence of outliers and noise due to the fact that it is not a probabilistic map.

- **[Occupancy-grid SLAM] Gradient based localization using discrete maps**: The front-end of current state-of-the-art range-based SLAM algorithms typically runs a scan-matching routine. Scan-matching can be addressed using brute-force approaches, e.g., [74]. However, in recent SLAM strategies, scan matching has been formulated as an optimization problem and solved using gradient descent methods. The problem is that when storing the map in a discrete grid, interpolation or smoothing techniques must be employed, e.g., non-smooth linear approximation [5] or bi-cubic interpolation [12]. In a nutshell, occupancy grid-based SLAM converts measurements into discrete resolution for efficient storage. Later, scan-matching requires the computation of derivatives and sub-cell accuracy which is achieved by interpolating the discrete grid. Such a process potentially leads to loss of information and degraded performance in the local consistency of SLAM.

The first problem is addressed by in Chapter 6 (B-spline curve SLAM), while the second and third issues are addressed in Chapter 7 (B-spline Surface SLAM).

Chapter 6

B-spline Curve SLAM

This chapter presents a geometric 2D B-spline curve SLAM that represents sparse environments in a compact fashion. Sec. 6.1 highlights the trade-offs between the proposed strategy and literature methods. B-spline curve SLAM has three modules, which are presented in Sec. 6.2 (pre-processing), Sec. 6.3 (mapping), and Sec. 6.4 (localization). Simulated results are presented in Sec. 6.5.

6.1 Introduction

Geometric approaches for the SLAM problem are memory-wise more efficient than discrete maps because geometric primitives are able to represent obstacles in the environment in a compact manner. For example, a long corridor can be represented by a few parameters using lines, while representing the same scene with an occupancy grid map requires multiple grid cells. However, it is often the case that the environment does not boil down to circles, ellipses or polylines alike geometries.

In the proposed B-spline curve SLAM algorithm the map is a collection of B-spline curves. Each curve represents an obstacle, which may have a rich geometric shape. For example, the curve $\mathbf{s}_i(\tau|\mathbf{t}_i)$ corresponds to the *i-th* object in the map, and it is supported by a non-clamped uniform knot vector $\mathbf{t}_i$. It will be convenient to write the knot vector as $\boldsymbol{t}(n, I, \Delta)$, where n is the number of knots, $\Delta \in \mathbb{R}^+$ is the uniform step between consecutive knots, and $I \in \mathbb{Z}$ is an offset. A particular knot can be recovered as

$$t_i = (i-1)\Delta + I\Delta, \quad \text{for } i = 1, \ldots, n.$$

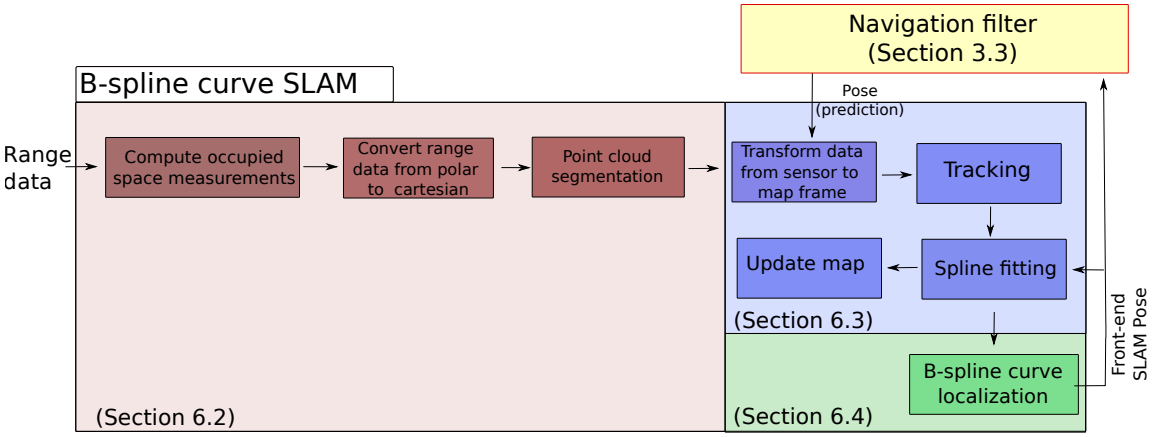


FIGURE 6.1: Workflow of the B-spline SLAM algorithm. From left to right: preprocessing (red box), mapping (blue box), and localization (green box). Mapping provides an initial guess to localization. Later, localization provides an optimized estimation of the pose of the robot and B-spline parameters which allows to improve the map.

This representation allows us to store the knot vector using only three parameters. Moreover, it is simple to increase the size of the knot vector as the robot explores the world - one has to update the offset (I) and/or the number of knots (n).

The main challenge in the proposed method is acknowledging that two B-spline curves from different sensor readings represent the same artifact in the map. This is achieved by tracking a curve across consecutive frames. For this, it is assumed that the dynamics of the vehicle are slow when compared to the dynamics of the sensor. As a consequence, the observation of an obstacle in two consecutive readings are close by.

The workflow of B-spline curve SLAM is depicted in Fig. 6.1. The first step, the preprocessing stage (red box) is addressed in Sec. 6.2. This module is responsible for removing wrong measurements from the raw sensor data, transforming data from polar to Cartesian coordinates, and most importantly, segmenting them into clusters. In the mapping stage (blue box), Sec. 6.3, the clusters are transformed to the map frame using the best available estimate - the prior provided by the navigation framework. The tracking task searches for clusters that are nearby in consecutive sensor readings. If tracking succeeds, it implies that the current cluster corresponds to a B-spline curve which has already been initialized in the map. Regardless of the tracking result, B-spline fitting is applied to each cluster. However, the curve obtained from a cluster associated to a mapped curve needs to be fused (map update). The mapping and the localization stages (green box) happens interleaved in time. As presented in Sec. 6.4, the key idea in localization is aligning the B-spline curves obtained in the current sensor readings with the previously mapped curves.

The correspondence between curves are obtained via the tracking task. In addition to estimating the pose of the robot, localization also optimizes parameters of the B-spline curve. These optimized results are fed back into the mapping stage to obtain a more accurate B-spline map. Finally, localization also provides pose corrections for the navigation filter.

6.2 Pre-processing

Occupied space measurements

A range sensor provides l range measurements of the environment $\boldsymbol{r} = (r_i)_{i=1}^{l}$ at discrete angle intervals $\boldsymbol{\alpha} = (\alpha_i)_{i=1}^{l}$ w.r.t. the x-axis of $\{B\}$ - see red dots in Fig. 3.1. Let r_{min} and r_{max} be the maximum and minimum range of the sensor. A valid range measurement that corresponds to an obstacle is within the interval $[r_{\text{min}}, r_{\text{max}}]$, that is,

$$(\boldsymbol{r}^{occ}, \boldsymbol{\alpha}^{occ}) = \{(r_i, \alpha_i) | \quad r_{\text{min}} \leq r_i \leq r_{\text{max}}, \quad \forall i = 1, \ldots, l\}, \tag{6.1}$$

where $\boldsymbol{r}^{occ}, \boldsymbol{\alpha}^{occ} \in \mathbb{R}^{l_{occ}}$. The number of range measurements (l) is fixed, while the number of valid occupied measurements (l_{occ}) may vary due to the absence (or proximity) of an obstacle in a particular direction.

Polar to Cartesian coordinates

The valid occupied measurements are transformed to Cartesian coordinates as in (3.2), that is,

$$\boldsymbol{z}(r_i, \alpha_i) \equiv \boldsymbol{z}_i = r_i \begin{bmatrix} \cos \alpha_i \\ \sin \alpha_i \end{bmatrix}, \tag{6.2}$$

where (r_i, α_i) is a tuple that describes a range beam and its direction, and $\boldsymbol{z}_i$ are the Cartesian coordinates of a measurement described in the sensor frame.

Point cloud segmentation

Next, the point cloud is divided into clusters or blobs. Points that belong to the same blob represent the same continuous geometric feature in the scene, e.g., a corridor. Inspired by [65], we use two metrics to segment the point cloud: the absolute distance between two

consecutive points and the relative distance between three consecutive points:

$$\|z_i - z_{i-1}\| \leq \gamma_{abs} \tag{6.3}$$

$$\|z_i - z_{i-1}\| \leq \gamma_{rel}\|z_{i-1} - z_{i-2}\| \tag{6.4}$$

where $\gamma_{abs}, \gamma_{rel} \in \mathbb{R}^+$. The parameter γ_{abs} enforces that two consecutive points are not far apart, while γ_{rel} imposes the maximum relative distance between three points. In addition, the parameter $\gamma_{min} \in \mathbb{N}$ is introduced. It describes the minimum number of elements a cluster must have to be considered a valid set.

Implementation note

The point cloud segmentation algorithm was refined after experiments using simulated data. The final version, which has three phases, is shown in Algorithm 5. The first phase consists in identifying that the candidate point and the preceding point belongs to the same blob. For that, the absolute distance between the two consecutive points must be below the threshold given by γ_{abs} (instructions 1.1 to 1.3). In the second stage, once a blob

Algorithm 5: Point cloud segmentation

Input: Point cloud: $z_1, \ldots, z_{l_{occ}}$
Output: Segmented point cloud: $\mathcal{Z}_1, \ldots, \mathcal{Z}_k$
Initialization: $k \leftarrow 0$, $LABELLING \leftarrow False$
Parameters: $\gamma_{abs}, \gamma_{rel}, \gamma_{\min}$

```
for i = 2, ..., l_occ do
    if LABELLING is False then
        /* Searching for a blob based on absolute distance */
        if ||z_i − z_{i−1}|| ≤ γ_abs then
            1.1: Z_{k+1}.append(z_{i−1})
            1.2: Z_{k+1}.append(z_i)
            1.3: LABELLING := True
    /* Add point to blob based on relative distance */
    else if ||z_i − z_{i−1}|| ≤ γ_rel ||z_{i−1} − z_{i−2}|| then
        2: Z_{k+1}.append(z_i)
    /* Removing blobs that do not have enough points */
    else
        3: LABELLING ← False
        if |Z_{k+1}| < γ_min then
            4: Z_k ← {}
        else
            5: k ← k + 1
```

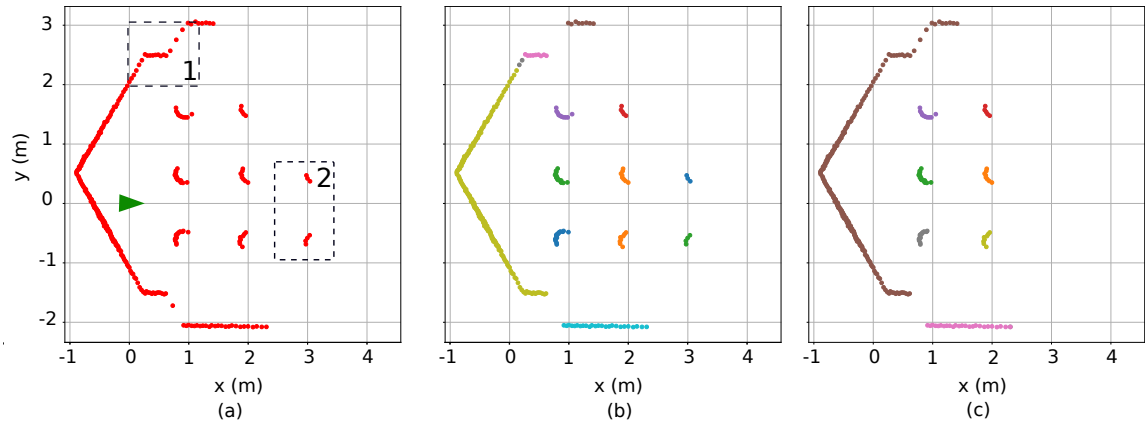


FIGURE 6.2: Point cloud segmentation: (a) vehicle is represented by a green triangle and sensor reading by red circles; (b) point cloud segmentation based on the absolute distance; (c) point cloud segmentation using both relative and absolute metrics, discarding blobs with a few number of elements. The clusters in (b) and (c) are illustrated using different colors

candidate has been detected in the previous step, the following points that respect the relative distance metric are included in the same blob (instruction 2). Note that at this second stage only the relative metric in (6.4) must be respected. The third step acknowledges that a blob has finished, which happens when the relative metric check fails (instruction 3). Clusters with too few points are excluded (instruction 4).

The relative distance metric is more relaxed than the absolute distance and it deals better with the nature of the measurements. As illustrated in Fig. 6.2a, objects close to the robot are more densely sampled, while obstacles further away are more sparsely sampled. The point cloud segmentation presented in Fig. 6.2b was computed using the absolute distance metric in (6.3) with $\gamma_{abs} = 0.1\ m$. Notice that the top left side of the sensor reading (annotation "1") is segmented into several distinct blobs, while some of them do not fit in any blob at all. Figure 6.2c show the result using Algorithm 5, which considers the relative metric and it is more likely to classify points that are far from the sensor in the correct blob. Note that blobs with few points (annotation "2") are discarded.

6.3 B-spline Curve Mapping

In this section the mapping task is brought to light. To present the mapping strategy, the pose of the vehicle $\boldsymbol{\xi}$ is assumed to be known and the point cloud has been segmented into blobs. Said that, the mapping strategy breaks down into four sub-tasks:

1. Transforming measurements to map frame: transforms the segmented blobs from the sensor to the map frame.

2. Data association: tracks a cluster between consecutive readings;

3. Spline interpolation: Interpolates the discrete data using a B-spline curve.

4. Map update: Updates the prior map, which includes extending and removing curve segments.

The mapping steps are performed individually for each set of points $\mathcal{Z}_1, \ldots, \mathcal{Z}_k$. However, for the sake of simplicity, the algorithm is exposed for a single blob that contains the first m points, that is, the set $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$.

Transformation from sensor to map frame

A Cartesian point $\mathbf{z}_i$ described in the body frame $\{B\}$ is transformed to the map frame $\{M\}$ by applying the rigid body transform

$$\boldsymbol{q}_i \equiv T(\boldsymbol{\xi}, \boldsymbol{z}_i) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \boldsymbol{z}_i + \begin{bmatrix} x \\ y \end{bmatrix} = R(\psi)\boldsymbol{z}_i + \boldsymbol{p}, \tag{6.5}$$

where $\boldsymbol{q} \in \mathbb{R}^2$ is the notation for a measurement point described in the map frame.

Data association

From the assumption that the sensor sampling rate is sufficiently high with respect to the dynamics of the vehicle, we induce that between consecutive readings the snapshot of the environment is almost similar. Based on the aforementioned assumption, tracking is devised by comparing the distance between the centroids of the clusters at instant t and $t-1$. The centroid of a cluster at time t is computed as

$$\bar{\boldsymbol{q}}(t) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{q}_i, \tag{6.6}$$

where m is the number of points detected in the blob considered here and may vary from cluster to cluster. The tracking is said to succeed at time instant t if there is a cluster detected at a previous time such that

$$m = \arg\min_i \|\bar{\boldsymbol{q}}(t) - \bar{\boldsymbol{q}}_i(t-1)\| \leq \gamma_{track}, \tag{6.7}$$

where $\bar{\boldsymbol{q}}_i(t-1)$ are the centroids detected in the previous sensor readings and $\gamma_{track} \in \mathbb{R}^+$ is the acceptable distance between two clusters in consecutive sensor readings. If tracking succeeds, it means that the centroid $\bar{\boldsymbol{q}}_m(t-1)$ observed in the previous reading corresponds to the centroid $\bar{\boldsymbol{q}}(t)$ of the current reading. The subscript m stands for a map index - it is assumed that the points represented by the centroid $\bar{\boldsymbol{q}}_m(t-1)$ and $\bar{\boldsymbol{q}}(t)$ are described in the B-spline map by the curve $\mathbf{s}_m$.

B-spline curve fitting

The data points in each cluster are approximated by a B-spline curve. Also known as spline approximation, spline interpolation can be categorized into local or global methods. Local approximation methods such as piecewise linear interpolant and cubic Hermite interpolant preserve well the shape of the data. In contrast, global approximation methods relax the constraint that the curve must pass at each point being interpolated, yielding smoother curves. The reason why we use global interpolation is three-fold: *1)* the number of control points and knot vectors per sensor reading is smaller, *2)* it naturally filters out the sensor noise, and *3)* the map update method requires that the same geometric feature yields similar knot vectors across consecutive readings, which can not be directly obtained using local interpolants due the discrete nature of the input data.

The interpolation problem consists in computing the control points of the curve $\boldsymbol{s}_t(\tau|\boldsymbol{t}_t) = C_t\boldsymbol{b}(\tau)^T$, which best fit the m data points of the cluster. The problem can be formulated as

$$\min_{C_t} \sum_{i=1}^{m} \|C_t\boldsymbol{b}(\tau_i)^T - \boldsymbol{q}_i\|^2. \tag{6.8}$$

To solve this problem, an association $(\boldsymbol{q}_i, \tau_i)$ that describes the position of $\boldsymbol{q}_i$ along the curve $\boldsymbol{s}_t(\tau|\boldsymbol{t}_t)$ has to be derived. This is obtained using the cord length parameterization:

$$\begin{aligned} \tau_i &= \tau_{i-1} + \|\boldsymbol{q}_i - \boldsymbol{q}_{i-1}\|_2, \qquad i = 2, \ldots, m \\ \tau_1 &= \begin{cases} \bar{\tau}_t & : \text{tracking} \\ 0 & : \text{no tracking} \end{cases} \end{aligned} \tag{6.9}$$

where $\bar{\tau}_t$ is an estimation of the offset between the first point in the current cluster and the corresponding point in the mapped curve for which tracking has succeeded. A cheap (in a computationally sense) estimation is available from the previous iteration: $\bar{\tau}_t = \tau_{t-1,1}$.

The reader is now ready to be introduced to the non-clamped uniform knot vector $\boldsymbol{t}_t(n_t, I_t, \Delta)$ that supports the B-spline $\boldsymbol{s}_t(\tau|\boldsymbol{t}_t)$:

- Δ: knot interval, which is pre-specified.
- $I_t = \lfloor \frac{\tau_1}{\Delta} \rfloor - d$: offset of the first point in the current cluster with respect to the origin of the mapped curve.
- $n_t = \lceil \frac{\tau_m}{\Delta} \rceil - \lfloor \frac{\tau_1}{\Delta} \rfloor + 2D + 1$: number of knots that supports the patch of the curve corresponding to the current sensor observation.

The solution for (6.8) that minimizes the error in a least square sense is $C_t = PB^{\dagger} \in \mathbb{R}^{2\times n_t - d}$, where

$$P = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m] \in \mathbb{R}^{2\times m},$$
$$B^T = \begin{pmatrix} b_1(\tau_1) & \ldots & b_{n_t-d-1}(\tau_1) \\ \vdots & \ddots & \vdots \\ b_1(\tau_m) & \ldots & b_{n_t-d-1}(\tau_m) \end{pmatrix} \in \mathbb{R}^{m\times n_t - d}, \tag{6.10}$$

and $B^{\dagger} = B(B^T B)^{-1}$ is the right pseudo-inverse of B^T.

If the cluster being considered does not correspond to any cluster in the previous sensor reading, i.e., tracking has failed, then $\bar{\tau}_t = 0$ and the interpolation task ends here.

Tracking

If track has succeed, the additional constraint $\boldsymbol{s}_t(\tau) = \boldsymbol{s}_m(\tau), \forall \tau \in \cap(\boldsymbol{t}_m, \boldsymbol{t}_t)$ must also hold. This constraint ensures that both curves yield the same value when evaluated in the region that both overlap. As illustrated in Fig. 6.3, this is accomplished by introducing the offset $\bar{\tau}_t$ that minimizes the curve alignment error defined as

$$\frac{1}{2}\sum_{i=1}^{r} \|\boldsymbol{s}_m(\tau_i|\boldsymbol{t}_m) - \boldsymbol{s}_t(\tau_i|\boldsymbol{t}_t)\|^2, \tag{6.11}$$

for $\tau \in \cap(\boldsymbol{t}_m, \boldsymbol{t}_t)$. Notice that r can take any value, as samples of the continuous curve rather than the data points observed in the sensor reading are employed. Simulations show that a small r suffices. Assume that an initial guess for $\bar{\tau}_t$, which is available from

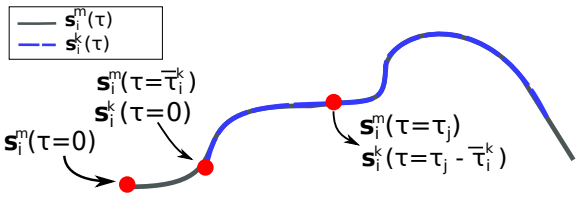


FIGURE 6.3: Two overlapping B-splines curves that describe the same geometric feature.

the previous iteration, feeds (6.9). The problem can be posed as finding $\Delta\bar{\tau}_t$ that minimizes the error

$$\frac{1}{2}\sum_{i=1}^{r}\|\boldsymbol{s}_m(\tau_i) - \boldsymbol{s}_t(\tau_i + \Delta\bar{\tau}_t)\|^2.$$

The first order Taylor expansion of $\boldsymbol{s}_t(\tau_i + \Delta\bar{\tau}_t)$ yields

$$\frac{1}{2}\sum_{i=1}^{r}\|\boldsymbol{s}_m(\tau_i) - \boldsymbol{s}_t(\tau_i) - \frac{\boldsymbol{s}_t(\tau_i)}{d\tau}\Delta\bar{\tau}_t\|^2 \tag{6.12}$$

The solution for $\Delta\bar{\tau}_t$ is computed using the Gauss-Newton method by taking the derivative of (6.12) with respect to $\Delta\bar{\tau}_t$ and setting it to zero, yielding

$$\Delta\bar{\tau}_t = -H^{-1}\sum_{i=1}^{r}[\frac{\boldsymbol{s}_t(\tau_i)}{d\tau}]^T[\boldsymbol{s}_m(\tau_i) - \boldsymbol{s}_t(\tau_i)] \tag{6.13}$$

$$H = \sum_{i=1}^{r}[\frac{\boldsymbol{s}_t(\tau_i)}{d\tau}]^T[\frac{\boldsymbol{s}_t(\tau_i)}{d\tau}], \tag{6.14}$$

where $H \in \mathbb{R}^{2\times 2}$ is typically full rank for a few samples and cheap to invert. Finally, the interpolation problem (6.8) is solved once again to re-compute $\boldsymbol{s}_t$, $\boldsymbol{t}_t$, and $\boldsymbol{C}_t$, this time letting $\tau_1 = \bar{\tau}_t + \Delta\bar{\tau}_t$.

Map update

The final step consists in storing or updating the detected curve, that is, the control points and the knot vector. If tracking has failed, the new curve $\boldsymbol{s}_m^{new} = \boldsymbol{s}_t$ is initialized in the map by storing

$$\boldsymbol{C}_m = \boldsymbol{C}_t \quad I_m = I_t \quad n_m = n_t \quad \boldsymbol{\alpha} = \mathbf{1}, \tag{6.15}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{n_m - d}$ is the weight associated to the control points of $\boldsymbol{s}_m$. As patches of the curve that correspond to the same control points are re-observed, the weight associated to these control points increase up to γ_{weight}. The threshold is employed to filter outlier associated to a static object in the map.

If tracking has succeeded, then the updated curve is a combination of the mapped curve $\boldsymbol{s}_m$ and its corresponding most recently interpolated curve $\boldsymbol{s}_t$. For merging both curves, first we introduce the weighted sum theorem for B-spline curves.

Theorem 6.1. *(Weighted sum) Consider $\boldsymbol{s}_a(\tau|\boldsymbol{t}_a)$ and $\boldsymbol{s}_b(\tau|\boldsymbol{t}_b)$, two B-spline curves of degree d, where $\boldsymbol{t}_a(n_a, I_a, \Delta)$ and $\boldsymbol{t}_b(n_b, I_b, \Delta)$. The control points of the weighted sum curve, denoted as $\boldsymbol{s}_{ws}(\tau|\boldsymbol{t}_{ws})$, are the weighted sum of the control points of $\boldsymbol{s}_b$ and $\boldsymbol{s}_a$ associated with the B-spline $B(\tau|\boldsymbol{t}_{ws})$.*

Proof. The knot vector $\boldsymbol{t}_{ws}(n_{ws}, I_{ws}, \Delta) = \cap(\boldsymbol{t}_a, \boldsymbol{t}_b)$ is defined by $I_{ws} = d + \max(I_a, I_b)$ and $n_{ws} = \min(n_a + I_a, n_b + I_b) - I_{ws} - d$. Given the weights $\alpha_a, \alpha_b \in \mathbb{R}$, it follows that

$$\begin{aligned}
\boldsymbol{s}_{ws}(\tau) &= \alpha_a \sum_{i=I_{ws}-I_a+1}^{I_{ws}-I_a+n_{ws}} \boldsymbol{c}_{a,i} B_i(\tau|\boldsymbol{t}_a) + \alpha_b \sum_{i=I_{ws}-I_b+1}^{I_{ws}-I_b+n_{ws}} \boldsymbol{c}_{b,i} B_i(\tau|\boldsymbol{t}_b) \\
&= \sum_i (\alpha_a \boldsymbol{c}_{a,i+(I_{ws}-I_a)} + \alpha_b \boldsymbol{c}_{b,i+(I_{ws}-I_b)}) B_i(\tau|\boldsymbol{t}_{ws}) \\
&= \sum_i \boldsymbol{c}_{ws} B_i(\tau)
\end{aligned}$$

□

For the region in which the curves overlap, the weighted sum curve $\boldsymbol{s}_{ws}(\tau|\boldsymbol{t}_{ws})$ is computed using Theorem 6.1. The control points of $\boldsymbol{s}_{ws}$ are given by

$$\boldsymbol{c}_{ws,i} = \frac{(\alpha_{i+I_{ws}-I_m} - 1)\boldsymbol{c}_{m,i+I_{ws}-I_m} + \boldsymbol{c}_{t,i+I_{ws}-I_t}}{\alpha_{i+I_{ws}-I_m}}$$

for $j = 1, \ldots, n_{ws} - d - 1$. The updated curve $\boldsymbol{s}_m^{new}$ is defined by the knot vector:

$$I_m^{new} = \min(I_m, I_t) \qquad n_m^{new} = n_m + n_t - n_{ws} - 2D, \tag{6.16}$$

and its control points are as described in Table 6.1, where

$$\begin{aligned}
u_1 &= I_{ws} - \min(I_m, I_t) \\
u_2 &= u_1 + n_{ws} - d - 1 \\
u_3 &= u_2 + \max(n_m + I_m, n_t + I_t) - \min(n_m + I_m, n_t + I_t) + d \\
&= \max(n_m + I_m, n_t + I_t) - \min(I_m, I_t) - d - 1 \\
u_4 &= -u_2 + I_{ws} + n_{ws} - d - 1 = \min(I_m, I_t)
\end{aligned}$$

TABLE 6.1: Control point updating scheme for merging curves

$\boldsymbol{c}_{m,i}^{new}$	$\alpha_{m,i}^{new}$	Condition	Interval
$\boldsymbol{c}_{m,i}$	$\alpha_{m,i}$	if $I_m \leq I_t$	$1 \leq i \leq u_1$
$\boldsymbol{c}_{t,i}$	1	otherwise	
$\boldsymbol{c}_{ws,i-u_1}$	$\min(\gamma_{weight}, \alpha_{m,i-u_1+I_{ws}-I_m} + 1)$	-	$u_1 + 1 \leq i \leq u_2$
$\boldsymbol{c}_{m,i+u_4-I_m}$	$\alpha_{m,i}$	if $n_m + I_m \geq n_t + I_t$	$u_2 + 1 \leq i \leq u_3$
$\boldsymbol{c}_{t,i+u_4-I_t}$	1	otherwise	

Algorithm 6: Map building algorithm

Input: Prior pose $\check{\boldsymbol{\xi}}$, mapped curves $\mathbf{s}_m$, and segmented point cloud $\mathcal{Z}_1, \ldots, \mathcal{Z}_k$
Output: Mapped curves $\boldsymbol{s}_t^m$
Initialization: $\boldsymbol{\xi}$, $\bar{\tau}_i^k$, and $\boldsymbol{s}_t^m$
Parameters: d, Δ, γ_{track} γ_{weight}

for *each cluster* **do**
- 1: Compute the centroid (6.6)
- 2: Track cluster between readings: (6.7)
- 3: Solve the fitting problem (6.8)
- **if** *track succeeded* **then**
  - 4: Solve the curve alignment problem (6.11),(6.13)
  - 5: Merge interpolated curve and mapped curve (6.16) and Table 6.1.
- **else**
  - 6 Initialize curve in the map (6.15)

Implementation

Algorithm 6 describes the key steps to build the B-spline curve map. From step 1-3 a cluster of discrete points is transformed in a B-spline curve. Tracking requires two additional operations: computes the displacement between the current and the mapped curves (step 4) and merge them (step 5). However, if a curve does not match against a mapped curve, then there is nothing to be merged, but only the need to initialialize the curve in the map (step 6). Fig. 6.4 shows the the segment point-cloud and the resulting B-spline curve map with $d = 3$. The curves are a smoothed version of the point cloud. If the B-spline degree is fixed, the smoothness of the curve is directly attached to the knot interval.

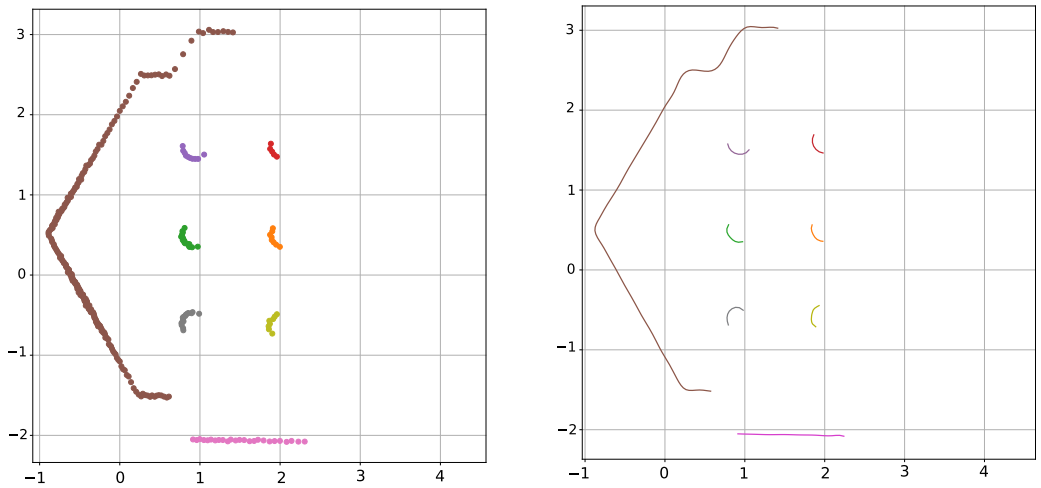


FIGURE 6.4: Transforming a segmented point cloud (left image) on a B-spline curve map (right image).

6.4 B-spline Curve SLAM

This section presents the B-spline curve SLAM algorithm. The algorithm pipeline is similar to the mapping framework presented in Sec. 6.3. However, this time, it is not assumed that the pose of the vehicle is known, but it is estimated instead. In a nutshell, the proposed solution finds the pose that best aligns the mapped curve and its observation in the current sensor reading. For every tracked curve, rather than solving (6.11), the following optimization problem is considered:

$$\min_{\Delta\boldsymbol{\xi},\Delta\bar{\boldsymbol{\tau}}_t} \frac{1}{2}\sum_{j=1}^{k}\sum_{i=1}^{r} \|\boldsymbol{s}_{m,j}(\tau_i) - T(\Delta\boldsymbol{\xi}, \boldsymbol{s}_{t,j}(\tau_i + \Delta\bar{\tau}_{t,j}))\|^2,$$

where $\Delta\bar{\boldsymbol{\tau}}_t = [\Delta\bar{\tau}_{t,1}, \ldots, , \Delta\bar{\tau}_{t,k}] \in \mathbb{R}^k$ is the knot interval displacement vector for the k tracked curves. Linearizing the cost function using first order Taylor expansion about $[\Delta\boldsymbol{\xi}^T, \Delta\bar{\boldsymbol{\tau}}_t^T]^T = [\mathbf{0}^T, \mathbf{0}^T]^T$ yields

$$\frac{1}{2}\sum_{j=1}^{k}\sum_{i=1}^{r} \|\boldsymbol{s}_{m,j}(\tau_i) - \boldsymbol{s}_{t,j}(\tau_i) - \frac{\partial T(\mathbf{0}, \boldsymbol{s}_{t,j}(\tau_i))}{\partial\boldsymbol{\xi}}\Delta\boldsymbol{\xi}_i - \frac{\partial T(\mathbf{0}, \boldsymbol{s}_{t,j}(\tau_i))}{\partial\tau}\Delta\bar{\tau}_{t,j}\|^2,$$

where

$$\frac{\partial T(\mathbf{0}, \boldsymbol{s}_{t,j}(\tau_i))}{\partial\boldsymbol{\xi}} = \begin{bmatrix} 1 & 0 & -\boldsymbol{s}_{t,y} \\ 0 & 1 & \boldsymbol{s}_{t,x} \end{bmatrix} \in \mathbb{R}^{2\times 3}$$
$$\frac{\partial T(\mathbf{0}, \boldsymbol{s}_{t,j}(\tau_i))}{\partial\tau} = \frac{d\boldsymbol{s}_{t,j}(\tau_i)}{d\tau} \in \mathbb{R}^{2\times 1}.$$

The solution can be found by matching the partial derivative w.r.t. $\Delta\boldsymbol{\xi}$ and $\Delta\bar{\boldsymbol{\tau}}_{t,j}$ to zero:

$$\begin{bmatrix} \Delta\boldsymbol{\xi} \\ \Delta\bar{\tau}_{t,j} \end{bmatrix} = H^{-1}\sum_{j=1}^{k}\sum_{i=1}^{r} \left[\begin{array}{c|ccc} \frac{\partial T(\mathbf{0},\boldsymbol{s}_{t,j}(\tau_i))}{\partial\boldsymbol{\xi}} & \frac{d\boldsymbol{s}_{t,1}(\tau_i)}{d_1\tau}, & \ldots, & \frac{d\boldsymbol{s}_{t,k}(\tau_i)}{d_k\tau} \end{array}\right]^T (\boldsymbol{s}_{m,j}(\tau_i) - \boldsymbol{s}_{t,j}(\tau_i)), \tag{6.17}$$

where

$$H = \sum_{j=1}^{k}\sum_{i=1}^{r} \left[\begin{array}{c|ccc} \frac{\partial T(\mathbf{0},\boldsymbol{s}_{t,j}(\tau_i))}{\partial\boldsymbol{\xi}} & \frac{d\boldsymbol{s}_{t,1}(\tau_i)}{d_1\tau}, & \ldots, & \frac{d\boldsymbol{s}_{t,k}(\tau_i)}{d_k\tau} \end{array}\right]^T \left[\begin{array}{c|ccc} \frac{\partial T(\mathbf{0},\boldsymbol{s}_{t,j}(\tau_i))}{\partial\boldsymbol{\xi}} & \frac{d\boldsymbol{s}_{t,1}(\tau_i)}{d_1\tau}, & \ldots, & \frac{d\boldsymbol{s}_{t,k}(\tau_i)}{d_k\tau} \end{array}\right] \tag{6.18}$$

$$\frac{d\boldsymbol{s}_{t,j}}{d_w\tau} = \begin{cases} \frac{d\boldsymbol{s}_{t,j}}{d\tau} & : j = w \\ \mathbf{0} & : j \neq w \end{cases} \tag{6.19}$$

Discussion

The parameters d, Δ, γ_{rel}, γ_{abs}, γ_{min},γ_{track}, and γ_{weight} play a major a role on the performance and robustness of the proposed framework. The spline degree d must be at least 2, to ensure that the spline curve has continuous first order derivative, a constraint imposed by (6.13). The uniform knot step Δ determines the map resolution. A large value smooths out the sensor noise and sharp geometric shapes, while small values preserves better the shape of the point cloud, but also leads to more oscillations. Furthermore, due to the nature of interpolation, Δ cannot be arbitrarily small. By choosing γ_{abs} smaller than Δ, one ensures that $\bar{B}$ in (6.10) has a pseudo-inverse and the interpolation problem (6.8) can be solved. As the value of Δ increases, γ_{rel} becomes crucial to correctly separate the data into clusters. A minimum number of elements per cluster is guaranteed by the threshold γ_{min}. The parameter γ_{track} relies on the assumption that the robot does not move between two consecutive sensor readings. A small value of γ_{track} increases the false negative tracking rejection, while a large value increases the false positive tracking success. The last parameter, γ_{weight}, determines the dynamic of the control points in the presence of noise and changes in the environment. After several simulations, the following parameter setup is suggested: $d = 3$, $\gamma_{rel} = 3$, $\gamma_{abs} = .7\Delta$, $\gamma_{min} = 4(d+1)$, $\gamma_{track} = \Delta$, $\gamma_{weight} = 50$, and Δ is free.

For better convergence, different map resolutions are employed. This is a well-known technique in laser-based SLAM [5] and visual-SLAM [75]. In B-spline maps, the resolution is determined by the knot spacing Δ. First, the estimation (localization and mapping) is computed for the coarsest map, that is, the one corresponding to the largest knot spacing. Then, the pose estimation obtained previously is used in the next resolution as an initial guess.

6.5 Results

The proposed solution was evaluated using the V-REP simulator of Coppelia Robotics. Figure 6.5 shows the simulator environment and the proposed B-spline map using $\Delta = 0.2$ m. In a 64-bit size architecture, storing a B-spline curve requires 9 bytes per control point (2 floats and 1 uint8_t) and 12 bytes per knot vector (1 float and 2 ints). An occupancy grid map usually requires 1 byte (uint8_t) per cell. In particular, for the scenario shown

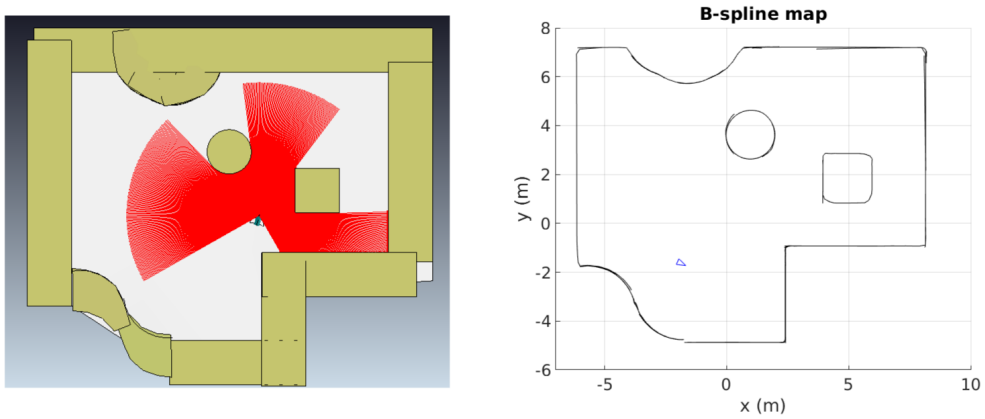


FIGURE 6.5: Qualitative evaluation of the proposed mapping framework in an environment with different geometric shapes. (left) shows the V-REP simulation scenario and (right) the B-spline map presented in this work

in Fig. 6.5, the B-spline map is almost 10 times more compact than a similar 0.05 m grid map.

The performance of the spline-based SLAM algorithm is assessed in two different scenarios, as illustrated in Fig. 6.6. In the first scenario, the vehicle follows a wall, while translating horizontally. In the second scenario, the vehicle rotates around itself, while mapping the environment. For both scenarios, the map represented by the B-spline curves is similar to the ground truth. In each scenario, three map resolutions are employed: $\Delta_1 = 0.4$ m, $\Delta_2 = 0.2$ m, $\Delta_3 = 0.1$ m. Also, as previously mentioned in Sec. 6.4, a multi-resolution approach that uses $\Delta_1, \Delta_2, \Delta_3$ in pyramid manner is evaluated. The results for the first scenario are shown in Fig. 6.7. In general, as expected, higher resolution

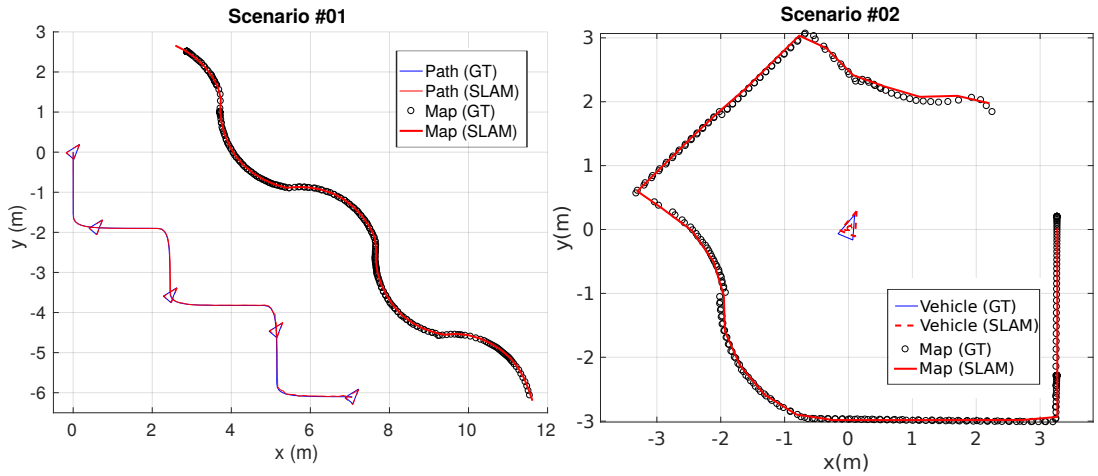


FIGURE 6.6: Missions performed to evaluate the performance of the proposed B-spline SLAM. (left) vehicle moves in the horizontal plane and (right) vehicle rotates 360^o while standing still. GT stands for ground truth.

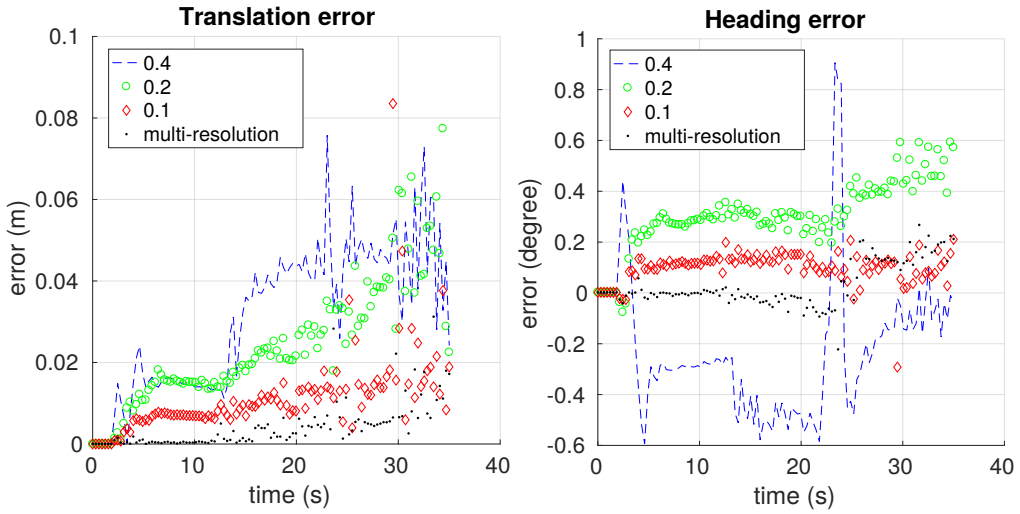


FIGURE 6.7: Estimation error for different knot steps (.4 ,.2, .1) m for the first scenario shown in Fig. 6.6.

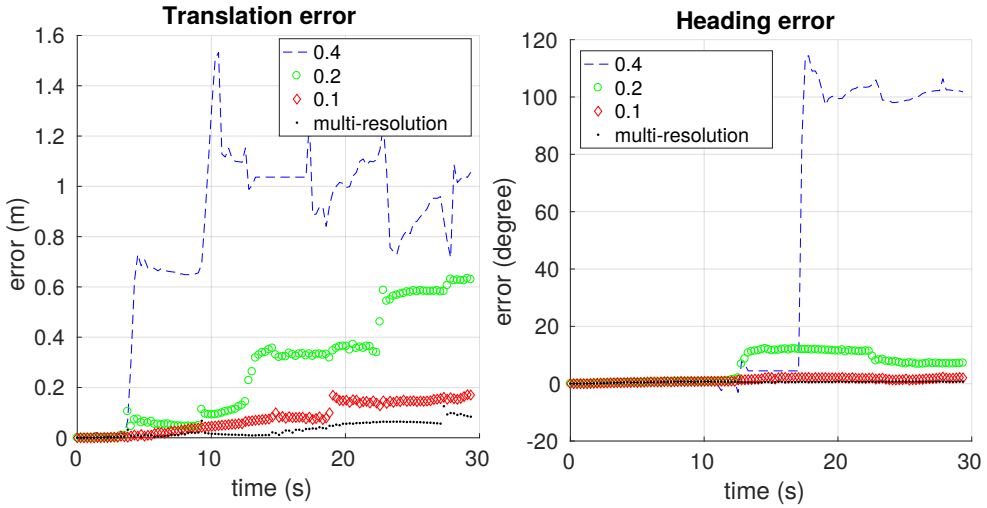


FIGURE 6.8: Estimation error using different knot steps (.4, .2, .1) m for the second scenario shown in Fig. 6.6.

maps lead to lower translation and orientation estimation errors because the map is able to capture better the actual shape of the obstacles. However, using the solution of lower resolution maps as a hot start still significantly improves the final solution. In the multi-resolution trial, the final translation error was 0.2% of the total length of the trajectory. Similar conclusions are drawn from the second scenario, shown in Fig. 6.8. Although the lowest resolution map ($\Delta_1 = 0.4$) failed to estimate the orientation, the multi-resolution approach converges to a good estimation. The orientation error after a 360^{o} spin is less then 1^{o}.

6.5.1 Remarks

B-spline curve SLAM is a memory efficient approach for the SLAM problem. The method requires clustering measurements, fitting a B-spline curve to each cluster, matching the current curve against a previously mapped one, and finding the pose that best aligns B-spline curves registered in the map and in the current reading. The most challenging parts are clustering and tracking, which are coupled. In particular, a tracking mismatch is equivalent to a wrong loop-closure and may quickly lead to map and localization degradation. As a consequence, while method works well in sparse environment with small and large objects, it is prone to fail in cluttered environments (like offices) where small cluster are either discarded or erroneously matched with nearby objects. In the next chapter, the need for clustering and tracking is removed by representing the map with a B-spline surface instead of curves.

Chapter 7

B-spline Surface SLAM

This chapter presents an online B-spline surface SLAM algorithm for range based measurements. Section 7.1 highlights the motivation for an alternative method for discrete occupancy-grid maps and the advantages of B-spline surface mapping for the SLAM problem. The proposed method is split in three modules: pre-processing (Sec. 7.2), mapping (Sec. 7.3), and localization (Sec. 7.4). The performance of the B-spline SLAM algorithm is discussed in Sec. 7.5. For that, the proposed algorithm is compared with other strategies in both well controlled scenarios using simulated data and more challenging environments using public datasets.

7.1 Introduction

The proposed B-spline Surface SLAM algorithm aims at overcoming two limitations of discrete occupancy-grid based SLAM: 1) misalignment between discrete cell and obstacles and 2) the pre-interpolation step for integration with gradient-descent methods. As shown in Fig. 7.1 the impact of an occupied measurement is greatest at the corresponding point (not cell) of the map, and smoothness is guaranteed by the inherent properties of B-splines. No interpolation is required because the map itself is continuous.

[**Computational complexity**] A valid concern is the computational complexity paid for a continuous map. The computational complexity of evaluating and updating a B-spline surface is constant and equal to $O(d^2)$, where d is the degree of the B-spline surface map (see Proposition 4.2). Thus, the proposed B-spline surface mapping is slightly more expensive than occupancy-grid map, which is $O(1)$. Nonetheless, the computational cost is low: d is typically lower or equal than 3 and updating the B-spline surface is constant

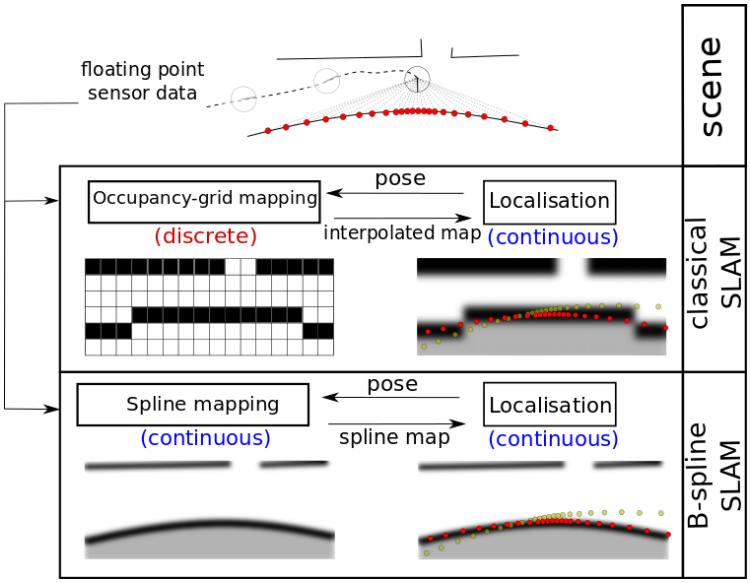


FIGURE 7.1: Comparing classical (occupancy grid) SLAM and B-spline surface SLAM. The top of the image shows a mobile robot equipped with a range sensor. Measurements are shown in yellow (before scan-matching) and red dots (after scan-matching). The middle image illustrates the discrete map which is stored in occupancy-grid SLAM. Typically, for localization, a smoothed/continuous version of regions of the map is derived for computing scan-matching using gradient-based strategies. The bottom image shows the proposed B-spline SLAM strategy. The map is stored in a B-spline surface and localization operates directly on it. The B-spline surface map is less impacted by the misalignment between cells and hits, and potentially yields better scan-to-map alignment results.

with respect to the size of the map. B-spline surface localization is able to achieve similar computational complexity as an occupancy-grid based localization that relies on an interpolation method. For instance, the method in Google Cartographer [12] applies bi-cubic interpolation. Similar complexity is obtained using a cubic B-spline surface, that is, $d = 3$.

[**Workflow**] The pipeline of the B-spline surface SLAM algorithm is shown in Fig. 7.2. The algorithm has three modules. The pre-processing step (red box) is addressed in Sec. 7.2. It takes as input the raw polar range data from the sensor. It filters, infers free space, and transforms the measurements to Cartesian coordinates. The pose of the robot, which describes the transform from the sensor frame to the map frame, is provided by the navigation framework addressed in Sec. 3.3. The B-spline mapping (blue rectangle) is presented in Sec. 7.3. The core idea is representing the occupancy of the environment by a B-spline surface map. Traditional techniques for surface fitting, which rely on least square minimization for a batch of measurements, are ill-advised for the SLAM problem. The reasons for that are two-fold: 1) the computational cost to obtain the least square solution is prohibitive for online operation using current onboard computers, and 2) a typical

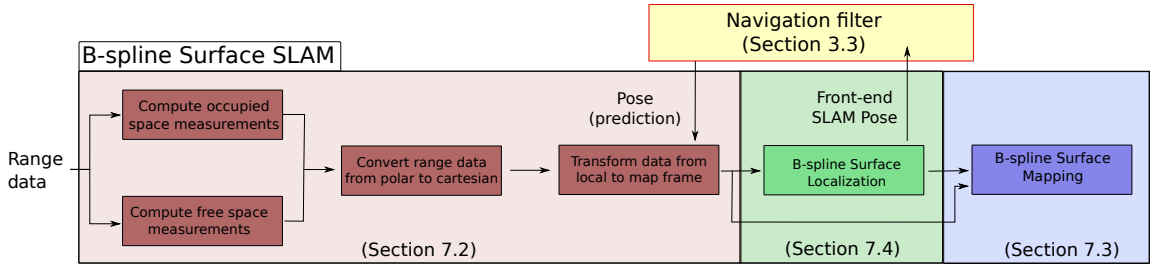


FIGURE 7.2: B-spline surface SLAM pipeline. From left to right: the raw range sensor is first pre-processed for obtaining the occupied/free points data (red rectangle); localization computes the pose that best explains the current measurements (green rectangle); and the B-spline surface map is updated (blue rectangle) using the best available pose estimate.

scan reading is not dense enough for obtaining a full rank matrix as required in the least square formulation. Thus, a lightweight technique is developed for updating the control points of a probabilistic B-spline surface in an recursive manner. Section 7.4 presents the localization method (green rectangle). The localization computes the displacement between the current sensor reading and the B-spline surface map using a tailor-made scan-matching algorithm. Although localization comes before mapping in the workflow of the algorithm, the mapping task is presented first for introducing the notation of the B-spline map.

7.2 Pre-processing

Compute occupied space measurements

A range sensor provides l range measurements of the environment $\boldsymbol{r} = (r_i)_{i=1}^{l}$ at discrete angle intervals $\boldsymbol{\alpha} = (\alpha_i)_{i=1}^{l}$ w.r.t. the x-axis of $\{B\}$ - see red dots in Fig. 7.3. Let $r_{\min}$ and $r_{\max}$ be the maximum and minimum range of the sensor. A valid range measurements that corresponds to an obstacle is within the interval $[r_{\min}, r_{\max}]$, that is,

$$(\boldsymbol{r}^{occ}, \boldsymbol{\alpha}^{occ}) = \{(r_i, \alpha_i) | \quad r_{\min} \leq r_i \leq r_{\max}, \quad \forall i = 1, \ldots, l\}, \tag{7.1}$$

where $\boldsymbol{r}^{occ}, \boldsymbol{\alpha}^{occ} \in \mathbb{R}^{l_{occ}}$, and l_{occ} is the number of valid occupied measurements. The superscript occ highlights that these measurements correspond to occupied space, i.e., obstacles.

Compute free space measurements

The free space virtual measurements correspond to samples between the end point of each beam and the robot or a beam that did not detect any obstacle up to $r_{\max}$ - see blue

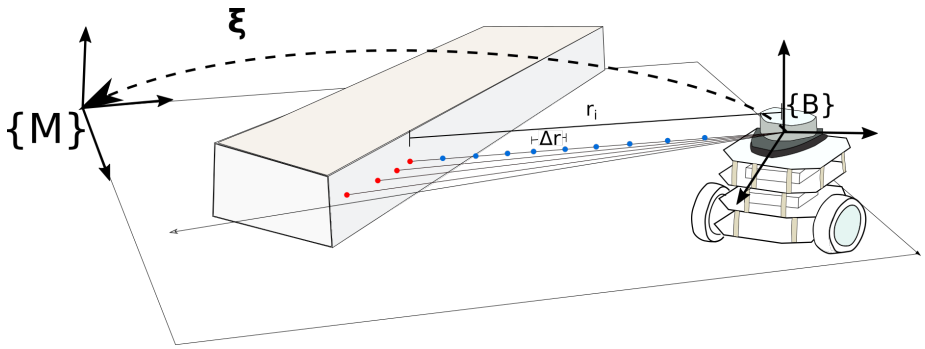


FIGURE 7.3: A robot equipped with a range sensor detects occupied space (red dots) at discrete intervals. The length of the i-th beam is denoted as r_i. The points between the robot and the obstacle are assumed as free space (blue dots) and obtained at sampling intervals of Δr along a beam.

dots in Fig. 7.3. These are called virtual measurements because free space is not explicitly detected by the sensor. The free space is computed as

$$(\boldsymbol{r}^{free}, \boldsymbol{\alpha}^{free}) = \{(n\Delta r, \alpha_i) | \, r_{\min} \leq n\Delta r \leq r_{\max}, \, \forall i = 1, \ldots, l, \, \forall n = 1, \ldots, \lfloor \min(\frac{r_i}{\Delta r}, \frac{r_{\max}}{\Delta r}) \rfloor \}, \tag{7.2}$$

where Δr is an appropriate sampling interval. It is assumed that there are l_{free} virtual measurements in a sensor reading. The value l_{free} is not a fixed number - it depends on the structure of the world. In general, the following inequalities hold:

- $l_{occ} \leq l$: some measurements may not be within the valid interval, e.g., obstacle is very far or very near;
- $l < l_{free}$: multiple free space measurements can be inferred from a single range beam.

Convert from polar to Cartesian coordinates

Both occupied and free space measurements are transformed to Cartesian coordinates applying the transform

$$z(r, \alpha) \equiv z = r \begin{bmatrix} \cos \alpha \\ \sin \alpha \end{bmatrix}, \tag{7.3}$$

where (r, α) is a tuple that describes a range and its direction, and z are the Cartesian coordinates of a measurement described in the sensor frame.

Transform from sensor to map frame

For updating the map and performing scan-to-map alignment[1], the measurements described in the sensor frame (z) have to be transformed to the map frame. For that, apply the transform

$$\boldsymbol{\tau}(\boldsymbol{\xi}) \equiv \boldsymbol{\tau} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} z + \begin{bmatrix} x \\ y \end{bmatrix} = R(\psi)z + \boldsymbol{p}, \tag{7.4}$$

where $\boldsymbol{\xi} = [\boldsymbol{p}, \psi]^T$ is the pose of the robot and $\boldsymbol{\tau}$ is a vector that describes the measurement z in the map frame. Before the localization step, the best pose estimate available is the prior (prediction), which is denoted as $\check{\boldsymbol{\xi}}$. During the mapping task, the best estimate is provided by the localization module and it is denoted as $\boldsymbol{\xi}^*$.

7.3 B-spline Surface Mapping

7.3.1 Probabilistic Mapping

Let m_i be a binary discrete random variable that represents the occupancy state at the position $\boldsymbol{\tau}_i = [\tau_{i,x}, \tau_{i,y}]$. The occupancy state is either free ($m_i = 0$) or occupied ($m_i = 1$). The probabilistic model derived in this section is similar to the discrete occupancy grid [40], except that there is no assumption that a measurement corresponds to a cell.

Given the measurements $z_{1:t}$ and the robot poses $\boldsymbol{\xi}_{1:t}$, the probability of $\boldsymbol{\tau}_i$ being occupied is given by

$$\begin{aligned}
p(m_i = 1|z_{1:t}, \boldsymbol{\xi}_{1:t}) &= \frac{p(z_t|m_i = 1, z_{1:t-1}, \boldsymbol{\xi}_{1:t})p(m_i = 1|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}{p(z_t|z_{1:t-1}, \boldsymbol{\xi}_{1:t})} && \textbf{Bayes rule} \\
&= \frac{p(z_t|m_i = 1, \boldsymbol{\xi}_t)p(m_i = 1|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}{p(z_t|z_{1:t-1}, \boldsymbol{\xi}_{1:t})} && \textbf{Markov Assumption} \\
&= \frac{p(m_i = 1|z_t, \boldsymbol{\xi}_t)p(z_t|\boldsymbol{\xi}_t)p(m_i = 1|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}{p(m_i = 1|\boldsymbol{\xi}_t)p(z_t|z_{1:t-1}, \boldsymbol{\xi}_{1:t})} && \textbf{Bayes rule} \\
&= \frac{p(m_i = 1|z_t, \boldsymbol{\xi}_t)p(z_t|\boldsymbol{\xi}_t)p(m_i = 1|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}{p(m_i = 1)p(z_t|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}. && \textbf{Markov Assumption}
\end{aligned}$$

The probability of $\boldsymbol{\tau}_i$ being free can be computed in a similar fashion, yielding

$$p(m_i = 0|z_{1:t}, \boldsymbol{\xi}_{1:t}) = \frac{p(m_i = 0|z_t, \boldsymbol{\xi}_t)p(z_t|\boldsymbol{\xi}_t)p(m_i = 0|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}{p(m_i = 0)p(z_t|z_{1:t-1}, \boldsymbol{\xi}_{1:t})}.$$

[1] Scan-to-map matching could be performed in the sensor space by transforming the map to sensor frame. Nonetheless, the same transform would still be required but in the opposite direction (inverse transform).

Computing the ratio of the two distributions:

$$\frac{p(m_i=1|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})}{p(m_i=0|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})}=\frac{p(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t)p(m_i=1|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})p(m_i=0)}{p(m_i=0|\boldsymbol{z}_t,\boldsymbol{\xi}_t)p(m_i=0|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})p(m_i=1)}.$$

Since m_i is a binary variable, it is clear that $p(m_i=0)=1-p(m_i=1)$. Substituting in the previous equation:

$$\frac{p(m_i=1|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})}{1-p(m_i=1|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})}=\frac{p(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t)}{1-p(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t)}\frac{p(m_i=1|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})}{1-p(m_i=1|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})}\frac{1-p(m_i=1)}{p(m_i=1)}$$

The previous ratio describes how much more likely it is to $\boldsymbol{\tau}_i$ being occupied than free. Using the odds notation, it can be written as

$$\text{odds }(m_i=1|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})=\frac{\text{odds }(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t)\text{odds }(m_i=1|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})}{\text{odds }(m_i=1)}.$$

Applying the logarithm in both sides

$$\begin{aligned}\text{log-odds}(m_i=1|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})=\text{log-odds}(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t)+\text{log-odds}(m_i=1|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})+\\ -\text{log-odds}(m_i=1).\end{aligned}$$

These terms corresponds to:

- log-odds$(m_i=1|\boldsymbol{z}_{1:t},\boldsymbol{\xi}_{1:t})\equiv$ log-odds$(\hat{m}_i=1)$: Posterior at time t;
- log-odds $(m_i=1|\boldsymbol{z}_{1:t-1},\boldsymbol{\xi}_{1:t})\equiv$ log-odds$(\check{m}_i=1)$: Prior at time t;
- log-odds $(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t)$: Inverse sensor model;
- log-odds $(m_i=1)$: Prior knowledge of the environment;

The posterior and the prior at time t consider all measurements up to time t and $t-1$, respectively. The prior knowledge of the environment describes the knowledge about the occupancy state before any observation has been made by the robot. The latter can be incorporated by the prior at time $t-1$, leading to the compact equation

$$\text{log-odds}(\hat{m}_i=1)=\text{log-odds}(\check{m}_i=1)+\text{log-odds}(m_i=1|\boldsymbol{z}_t,\boldsymbol{\xi}_t). \tag{7.5}$$

Thus, the posterior stated on the left hand side of the previous equation is obtained in a recursive manner by updating the prior with the information provided by the current observation, using the inverse sensor model.

7.3.2 B-spline surface mapping

The core idea of B-spline mapping is representing the log-odds probability map obtained in the previous section using a continuous B-spline surface. For that, first rewrite (7.5) as

$$\hat{s}(\boldsymbol{\tau}_i) = \check{s}(\boldsymbol{\tau}_i) + \text{log-odds}(m_i = 1 | z_t, \boldsymbol{\xi}_t),$$

where $\hat{s}(\boldsymbol{\tau}_i) = \boldsymbol{\phi}(\boldsymbol{\tau}_i)^T \hat{\boldsymbol{c}}$ and $\check{s}(\boldsymbol{\tau}_i) = \boldsymbol{\phi}(\boldsymbol{\tau}_i)^T \check{\boldsymbol{c}}$ are the posterior and prior B-spline surface map, respectively. The B-spline tensor is a function of the coordinates of the measurements, while the control points describe the shape of the surface.

A 3D and 2D view of a B-spline surface map is illustrated in Fig. 7.4. The darker an area is, the more likely it is to represent occupied space. The 2D view is the preferred visualisation scheme because the 3D view quickly becomes cumbersome for large maps. For an intuitive way to understand the B-spline map consider that the control points, shown in red dots, are stitched to the surface. As the control points are moved upward (downward), a local patch of the surface also moves in the same direction. Thus, the key idea formulated in the problem below is how to update the control points such that the map represents the occupancy of the environment as perceived by the onboard range sensor.

Problem 3. (*B-spline surface mapping problem*) Given a prior map $\check{s}(\boldsymbol{\tau})$ and a measurement at the map coordinates $\boldsymbol{\tau}_i$, compute the posterior map $\hat{s}(\boldsymbol{\tau})$ that drives the mapping error defined in (7.6) to the origin:

$$e_i(\hat{\boldsymbol{c}}) = \boldsymbol{\phi}(\boldsymbol{\tau}_i)^T \hat{\boldsymbol{c}} - \boldsymbol{\phi}(\boldsymbol{\tau}_i)^T \check{\boldsymbol{c}} - \text{log-odds}(m_i = 1 | z_t, \boldsymbol{\xi}_t). \tag{7.6}$$

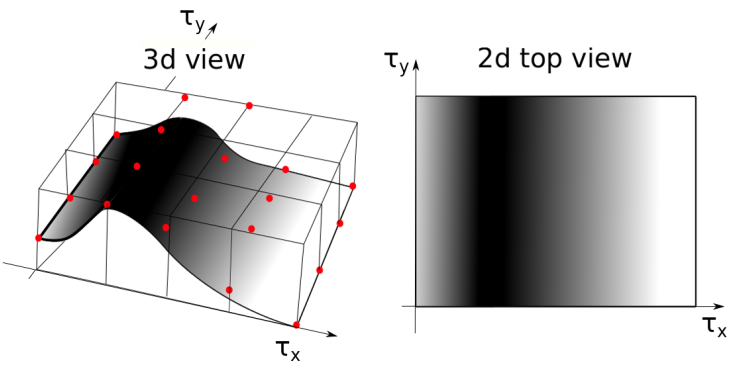


FIGURE 7.4: Visual representation of a B-spline surface map: (left) shows the 3D view of a B-spline surface and (right) illustrates a top view of the surface using a grayscale code to represent the value of $s(\boldsymbol{\tau})$. Control points are shown in red circles.

In other words, the tasks is computing the control points of the posterior map that updates the prior map using the inverse sensor model.

For solving Problem 3, the following cost function is employed:

$$J_i(\hat{\boldsymbol{c}}) = \frac{1}{2}\|e_i(\hat{\boldsymbol{c}})\|^2.$$

Solving the mapping problem is equivalent to finding the control points $\hat{\boldsymbol{c}}$ that minimize $J(\cdot)$. In order to do this, start with the classical gradient descent:

$$\hat{\boldsymbol{c}} = \check{\boldsymbol{c}} - \mu \left[\frac{\partial J_i}{\partial \hat{\boldsymbol{c}}}\right]^T, \tag{7.7}$$

where $\mu \in \mathbb{R}^+$ is the step size. The derivative of the cost function w.r.t. the posterior control points is given by

$$\frac{\partial J_i}{\partial \hat{\boldsymbol{c}}} = \frac{dJ_i}{de_i}\frac{\partial e_i}{\partial \hat{\boldsymbol{c}}} = e_i(\hat{\boldsymbol{c}})\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T. \tag{7.8}$$

Thus, by substituting (7.8) into the recursive equation (7.7), the gradient descent step is computed as

$$\hat{\boldsymbol{c}} = \check{\boldsymbol{c}} - \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)e(\hat{\boldsymbol{c}}_i). \tag{7.9}$$

A possible solution is to move forward with the gradient descent method. For that, one can take the prior as an initial guess for the posterior and perform iterations until the mapping error has converged to a minimum. However, iterative methods are computationally expensive and shall be avoided whenever possible. Next, it is shown that one may obtain a closed-form solution by exploiting the recursive nature of the mapping error and the properties of the B-spline tensor.

[Closed-form solution] Apply the definition of the mapping error (7.6) into the gradient descent solution (7.9) to obtain

$$\hat{\boldsymbol{c}} = \check{\boldsymbol{c}} - \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)[\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T\hat{\boldsymbol{c}} - \boldsymbol{\phi}(\boldsymbol{\tau}_i)^T\check{\boldsymbol{c}} - \text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t)]. \tag{7.10}$$

With some algebraic manipulation, the previous equation is re-arranged as

$$[I + \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T]\hat{\boldsymbol{c}} = [I + \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T]\check{\boldsymbol{c}} + \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t),$$

where I is an identity matrix with appropriate dimensions. The matrix $[I + \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T]$ is full rank and, therefore, invertible. Multiplying both sides of the equality by its inverse,

Algorithm 7: B-spline surface map algorithm.

Input: Pose $\boldsymbol{\xi}$, Range Scans $(r_i)_{i=0}^{l-1}$
1: Pre-process occupied space: (7.1)
2: Pre-process free space: (7.2)
3: Transform polar to Cartesian coordinates: (7.3)
4: Transform free and occupied space to global coordinate frame: (7.4)
5: Update the B-spline map: (7.14)
6: Clamp control points: (7.15)

yields

$$\hat{\boldsymbol{c}} = \check{\boldsymbol{c}} + [I + \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T]^{-1}\boldsymbol{\phi}(\boldsymbol{\tau}_i)\mu\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t). \tag{7.11}$$

Using the Sherman-Morrison formula [76], the following equality holds:

$$[I + \mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T]^{-1} = I - \mu\frac{\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T}{1 + \mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}. \tag{7.12}$$

Replacing the inverse matrix in (7.11):

$$\begin{aligned}
\hat{\boldsymbol{c}} &= \check{\boldsymbol{c}} + [I - \mu\frac{\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T}{1 + \mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}]\boldsymbol{\phi}(\boldsymbol{\tau}_i)\mu\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t) \\
&= \check{\boldsymbol{c}} + [\boldsymbol{\phi}(\boldsymbol{\tau}_i) - \frac{\mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)\boldsymbol{\phi}(\boldsymbol{\tau}_i)^T\boldsymbol{\phi}(\boldsymbol{\tau}_i)}{1 + \mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}]\mu\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t) \\
&= \check{\boldsymbol{c}} + \boldsymbol{\phi}(\boldsymbol{\tau}_i)[1 - \frac{\mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}{1 + \mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}]\mu\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t).
\end{aligned}$$

The control points of the posterior map are

$$\hat{\boldsymbol{c}} = \check{\boldsymbol{c}} + \frac{\mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)}{1 + \mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t) \tag{7.13}$$

The mapping error can be evaluated by applying the previous equation in (7.6):

$$e_i(\hat{\boldsymbol{c}}) = \boldsymbol{\phi}(\boldsymbol{\tau}_i)^T\frac{\mu\boldsymbol{\phi}(\boldsymbol{\tau}_i)}{1 + \mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t) - \text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t).$$

The goal is to drive the mapping error to zero, that is, $e_i(\hat{\boldsymbol{c}}) = 0$. This can be accomplished by selecting μ (which can be arbitrarily chosen) such that $\mu\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2 >> 1$. Finally, applying this result in (7.13), the map update equation upon reception of a measurement at $\boldsymbol{\tau}_i$ is obtained as

$$\hat{\boldsymbol{c}} = \check{\boldsymbol{c}} + \frac{\boldsymbol{\phi}(\boldsymbol{\tau}_i)}{\|\boldsymbol{\phi}(\boldsymbol{\tau}_i)\|^2}\text{log-odds}(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t). \tag{7.14}$$

Implementation

Algorithm 7 shows the main steps for implementing the proposed mapping strategy, including the pre-processing stage.

Clamping control points: In theory, the log-odds occupancy probability can grow (or decrease) indefinitely. In practice, this behaviour is not desired for three reasons. First, it leads to memory overflow in data representation. Second, it becomes relatively hard to change the state of a region after it has being observed many times as either free or occupied. For example, a closed door initially mapped as an obstacle may be opened and the map should be able to quickly react to that change in the environment. Finally, for localization purpose, one needs to infer with confidence that an area is indeed occupied using the log-odds map. However, if the log-odds probability can grow indefinitely, it means that one can always be more certain about its occupancy. The B-spline map is forced to be always within the values $[-c_{max}, c_{max}]$ by clamping its control points, that is,

$$\hat{\boldsymbol{c}} = \min(\max(\hat{\boldsymbol{c}}, -c_{max}), c_{max}). \tag{7.15}$$

This follows from Property 3 (Convex Hull), which states that a B-spline surface is a convex combination of its control points.

Inverse sensor model: The proposed mapping technique absorbs sensor measurements by updating the control points (red dots in Fig. 7.4) , which locally impacts the height of the surface. The value of the update is given by the inverse sensor model, that is, log-odds$(m_i = 1|\boldsymbol{z}_t, \boldsymbol{\xi}_t)$. It is an empirical value that reflects the probability of our hypothesis ($\boldsymbol{\tau}_i$ is occupied) given our data at time t (free or occupied measurement and robot pose). For an intuitive idea of the map update in (7.14), consider that the measurement $\boldsymbol{z}_t$ reports that $\boldsymbol{\tau}_i$ is occupied. Since the coefficients of the B-spline tensor $\boldsymbol{\phi}(\boldsymbol{\tau}_i)$ are non-negative, it follows that:

- $p(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) > 0.5 \rightarrow$ log-odds$(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) > 0$. By increasing the height of the control points, the occupancy state is reinforced;
- $p(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) = 0.5 \rightarrow$ log-odds$(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) = 0$. Control points update has no impact in the map;
- $p(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) < 0.5 \rightarrow$ log-odds$(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) < 0$. By decreasing the control points' height, the occupancy state is weakened.

If a measurement corresponds to occupied space, a value $p(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) > 0.5$ must be chosen. Conversely, if it corresponds to a free space measurement, a value $p(m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t) < 0.5$ must be chosen.

Memory storage requirements: Assume that we want to represent an area of 50 m $\times$ 50 m with a knot interval 10 cm. For a cubic B-spline surface map ($d = 3$), $(50/.1 + 3)(50/.1 + 3) = 253,009$ control points need to be stored (float data type). There is no need to store the knot vector because the knots are uniformly spaced (see Chapter 4). In the same scenario an occupancy grid requires $(50/.1)(50/.1) = 250,000$ cells (float data type). Thus, in this example, a B-spline surface map requires 1.2% more memory than a discrete occupancy grid with the same resolution. Now, increasing the resolution of the discrete occupancy-grid by 1 cm: $(50/.09)(50/.09) = 308,642$ floats. This is 20% more than the storage memory required by the B-spline map using 10 cm resolution. In the more generic case, let Δ be the resolution of the map and L be the size of the map, the relative difference in terms of storage between the two representations is given by

$$\frac{(L/\Delta + 3)(L/\Delta + 3)}{(L/\Delta)(L/\Delta)} - 1 = 6\frac{\Delta}{L} + 9(\frac{\Delta}{L})^2. \tag{7.16}$$

Since the resolution is typically much smaller than the size of the map, i.e., $\Delta \ll L$, the difference in storage memory becomes negligible in critical cases: small resolution, large map, or both.

7.4 B-Spline Surface Localization

The localization stack computes an estimate of the pose of the robot via scan-to-map alignment. First, the B-spline surface map is normalized such that $s(\boldsymbol{\tau}) \to 1$ as the log-odd likelihood occupancy at $\boldsymbol{\tau}$ converges to c_{max} and $s(\boldsymbol{\tau}) \to -1$ as it converges to $-c_{max}$:

$$\bar{s}(\boldsymbol{\tau}) = \frac{s(\boldsymbol{\tau})}{c_{max}},$$

where $\bar{s}(\boldsymbol{\tau})$ is the normalized B-spline map. Thus, given the pose of the robot, the error between the map and the i-th scan measurement (that corresponds to an obstacle) is

$$e_i(\boldsymbol{\xi}) = 1 - \bar{s}(\boldsymbol{\tau}_i(\boldsymbol{\xi})),$$

where $\boldsymbol{\tau}_i(\boldsymbol{\xi})$ is the map coordinate of the endpoint of a range beam using $\boldsymbol{\xi}$, as defined in (7.4). Now, it is defined the scan-to-map alignment cost function as [1]

$$J(\boldsymbol{\xi}) = \sum_{i=1}^{l_{occ}} e_i(\boldsymbol{\xi})\Omega_i e_i(\boldsymbol{\xi}) \tag{7.17}$$

where e_i is assumed to be a Gaussian error with covariance Ω_i^{-1}. The scan-to-map cost function describes how well a pose aligns the current scan measurements and the map. The goal in localization is to find the pose that minimizes the cost, that is,

$$\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi}} \sum_{i=1}^{l_{occ}} e_i(\boldsymbol{\xi})\Omega_i e_i(\boldsymbol{\xi})$$

This problem has no closed form solution, but it is possible to find a (locally optimal) solution using iterative local linearizations given that an initial guess is provided. Thus, the prior $\check{\boldsymbol{\xi}}$ is taken as an initial guess, and decompose the optimal pose that minimizes the cost in (7.17) as $\boldsymbol{\xi}^* = \check{\boldsymbol{\xi}} + \Delta\boldsymbol{\xi}$. The localization problem is reformulated as

$$\Delta\boldsymbol{\xi} = \arg\min_{\Delta\boldsymbol{\xi}} \sum_{i=1}^{l_{occ}} e_i(\check{\boldsymbol{\xi}} + \Delta\boldsymbol{\xi})\Omega_i e_i(\check{\boldsymbol{\xi}} + \Delta\boldsymbol{\xi}) \tag{7.18}$$

that is, one must compute the pose displacement that minimizes the scan-to-map alignment cost function. For solving this non-linear least square problem, it is assumed that the dynamic of the sensor is fast enough such that the displacement $\Delta\boldsymbol{\xi}$ is small. In this case, the non-linear function can be approximated using Taylor expansion around $\Delta\boldsymbol{\xi} = \mathbf{0}$, as

$$\bar{s}(\boldsymbol{\tau}_i(\check{\boldsymbol{\xi}} + \Delta\boldsymbol{\xi})) \approx \bar{s}(\boldsymbol{\tau}_i(\check{\boldsymbol{\xi}})) + \left.\frac{\partial \bar{s}(\boldsymbol{\tau}_i(\boldsymbol{\xi}))}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\check{\boldsymbol{\xi}}} \Delta\boldsymbol{\xi}. \tag{7.19}$$

Define the variables b_i and $\boldsymbol{h}_i$ as

$$b_i = 1 - \bar{s}(\boldsymbol{\tau}_i(\check{\boldsymbol{\xi}})) \tag{7.20}$$

$$\boldsymbol{h}_i^T = \left.\frac{\partial \bar{s}(\boldsymbol{\tau}_i(\boldsymbol{\xi}))}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\check{\boldsymbol{\xi}}} \tag{7.21}$$

Approximating the non-linear function in (7.18) by (7.19) and substituting (7.21)(7.20), yields

$$\min_{\Delta\boldsymbol{\xi}} \sum_{i=1}^{l_{occ}} (b_i - \boldsymbol{h}_i^T\Delta\boldsymbol{\xi})\Omega_i(b_i - \boldsymbol{h}_i^T\Delta\boldsymbol{\xi})$$

[1] For simplicity of presentation, a quadratic loss function is adopted. The proposed approach supports other functions like for example the robust loss functions Cauchy, German-McClure, and Welsch [77].

The least square solution can be obtained by derivating with respect to $\Delta\boldsymbol{\xi}$ and equating to zero:

$$\sum_{i=1}^{l_{occ}} \boldsymbol{h}_i \Omega_i (b_i - \boldsymbol{h}_i^T \Delta\boldsymbol{\xi}) = 0,$$

$$\sum_{i=1}^{l_{occ}} \Omega_i \boldsymbol{h}_i \boldsymbol{h}_i^T \Delta\boldsymbol{\xi} = -\sum_{i=1}^{l_{occ}} \Omega_i \boldsymbol{h}_i b_i,$$

which give us the update:

$$\Delta\boldsymbol{\xi} = -(\sum_{i=1}^{l_{occ}} \Omega_i \boldsymbol{h}_i \boldsymbol{h}_i^T)^{-1} \sum_{i=1}^{l_{occ}} \Omega_i \boldsymbol{h}_i b_i. \tag{7.22}$$

For the sake of completeness, the derivation of the term $\boldsymbol{h}_i$ is described next. From the definition of a B-spline surface in (4.7), its derivative w.r.t. $\boldsymbol{\xi}$ is

$$\boldsymbol{h}_i = \boldsymbol{c}^T \frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau}_i(\boldsymbol{\xi}))}{\partial \boldsymbol{\xi}}\bigg|_{\boldsymbol{\xi}=\check{\boldsymbol{\xi}}} = \boldsymbol{c}^T \left[\frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau}_i)}{\partial \boldsymbol{\tau}_i} \frac{\partial \boldsymbol{\tau}_i}{\partial \boldsymbol{\xi}}\right]\bigg|_{\boldsymbol{\xi}=\check{\boldsymbol{\xi}}},$$

where $\frac{\partial \boldsymbol{\phi}(\boldsymbol{\tau}_i)}{\partial \boldsymbol{\tau}_i}$ is as defined in (4.8) and

$$\frac{\partial \boldsymbol{\tau}_i}{\partial \boldsymbol{\xi}} = \begin{bmatrix} 1 & 0 & -z_{i,x}\sin\psi - z_{i,y}\cos\psi \\ 0 & 1 & z_{i,x}\cos\psi - z_{i,y}\sin\psi \end{bmatrix}.$$

Implementation

The B-spline SLAM algorithm is described in Algorithm 8. When solving the optimization problem described in (7.18) via the Gauss-Newton method, an adaptive step λ is used. The stop criterion are 1) the maximum number of iterations and 2) the tolerance ΔJ_{tol}. The former parameter ensures that the solver finishes in an online-acceptable time. The latter parameter describes the maximum value required in improving the objective function before accepting a solution. Moreover, in a multi-map resolution approach is employed, similar to [5]. The resolution of a B-spline map is given by the knot interval: the smaller the interval, the higher is the resolution. A solution to the scan-matching problem is first computed for the lowest map resolution. Then, the obtained solution for the localization problem is used as a "hot start" for the next map resolution and so on (from the lowest to the highest resolution). This process increases the robustness of the localization stack against local minima.

Algorithm 8: B-spline SLAM algorithm

input : Prior pose $\check{\boldsymbol{\xi}}$, prior map $\check{\mathbf{c}}$, range scans $(r_i)_{i=0}^{l-1}$
output : Posterior pose $\hat{\boldsymbol{\xi}}$, posterior map $\hat{\mathbf{c}}$
initialize : $\Delta J \leftarrow \infty$, *#iterations* $\leftarrow 0$, $\lambda \leftarrow 1$
parameters: ΔJ_{tol}, *max_iterations*

1: Pre-process occupied space: (7.1)
2: Pre-process free space: (7.2)
3: Transform polar to Cartesian coordinates: (7.3)

// Localization
for *each map resolution (low to high)* **do**
  while $\Delta J > \Delta J_{tol}$ *and #iterations* $<$ *max_iterations* **do**
    4: Transform measurements from sensor to global frame via $\check{\boldsymbol{\xi}}$: (7.4)
    5: Compute scan-to-map alignment error using prior: $\{b_i\}_{i=1}^{n} \leftarrow$(7.20)
    6: Compute Jacobian: (7.21)
    7: Pose update: $\Delta\boldsymbol{\xi} \leftarrow$ (7.22)
    8: Cost improvement: $\Delta J \leftarrow J(\check{\boldsymbol{\xi}} + \Delta\boldsymbol{\xi}) - J(\check{\boldsymbol{\xi}})$
    if $\Delta J < 0$ **then**
      9.1: $\check{\boldsymbol{\xi}} \leftarrow \check{\boldsymbol{\xi}} + \lambda\Delta\boldsymbol{\xi}$
      9.2: $\lambda \leftarrow 1.5\lambda$
    else
      9.3: $\lambda \leftarrow .5\lambda$
    10: *#iterations* $\leftarrow$ *#iterations* + 1
11: Update posterior: $\hat{\boldsymbol{\xi}} \leftarrow \check{\boldsymbol{\xi}}$

// Mapping
for *each map resolution* **do**
  12: Update the B-spline map: (7.14)
  13: Clamp control points: (7.15)

7.5 Results

7.5.1 Mapping

In this section, the B-spline surface mapping algorithm is compared against other methods described in the literature review. The goal is to present and evaluate the trade-offs between B-spline maps and other techniques before moving forward to localization (and thus, SLAM). The comparison is performed using both simulated and real-world data. The simulations are used to discuss quantitative performance, while qualitative results are shown using a real robot. For a fair comparison, the proposed solution and a vanilla version of the occupancy-grid mapping algorithms were implemented in Python3. Both

are available in our repository[1]. The occupancy-grid implementation follows the Algorithm 7, but it updates discrete cells instead of points in the surface map. For free space detection in occupancy-grid mapping, Bresenham's line algorithm is the chosen ray rasterization method.

Simulations

The proposed method is compared against occupancy-grid based strategies in a controlled environment, using extensive computer simulations. The simulated sensor is equivalent to the LDS-01 range sensor, presented in Sec. 3.4 - 360 degrees, 1 degree angular resolution. Both noiseless and noisy range measurements are considered. The occupancy-grid map cell resolution and the B-spline map knot interval are set to 10 cm and, therefore, having about the same memory storage requirements. For obtaining a continuous map, bilinear and bicubic interpolations are applied to the resulting occupancy-grid maps, as proposed in [5] and [12], respectively. The maps are finally normalized into the floating interval -1 (free) to 1 (occupied).

In order to compare the mapping accuracy of the continuous maps, the scan to map-alignment cost function stated in a more generic form is employed:

$$\sum_{i=0}^{l_{occ}} [1 - M_{cont}(\boldsymbol{\tau}_i)]^2, \tag{7.23}$$

where M_{cont} is a continuous map. The metric, which is also referred here as the *mapping error*, translates as the alignment between the measurements detected as obstacles and the actual continuous map belief.

The first simulated environment evaluates the limitations of a continuous map obtained from an occupancy-grid map versus the inherent continuous B-spline map. The robot stands still in the center of a square room, while taking measurements. There were enough sensor readings to stabilize the map cells/control points, such that additional readings did not have any impact on the map. The resulting maps for a square room of side length 4 m can be seen in Fig. 7.5. The methods managed to accurately map the given environment. Then, to analyze the discretization error that rises in occupancy grid mapping, the side length of the square room is varied from 4 m to 4.18 m (one experiment per side length). Figure 7.6 shows the mapping error. It can be seen that the mapping error using the proposed approach is consistently lower than any of the other approaches.

[1] https://github.com/C2SR/spline_slam/

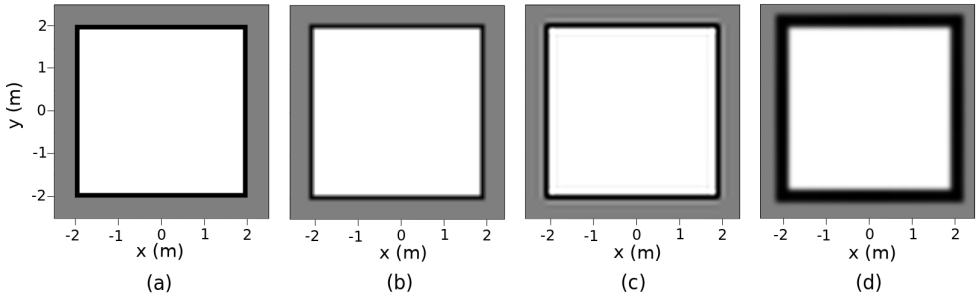


FIGURE 7.5: Mapping result for an artificially generated square room using: (a) occupancy-grid, (b) occupancy-grid + bilinear interpolation, (c) occupancy-grid + bicubic interpolation, and (d) proposed B-spline map. The robot is in the center of the room.

When the length of the square is 4 m the walls are aligned with the cell representation of the occupancy grid. Therefore, the occupancy grid based methods are able to achieve null error, which is slightly better than the spline map. Then, as the length increases, there is an increasing offset between the walls and the cells that represent them in the discrete map. The critical point occurs around half-cell resolution. The interpolation methods fail to capture the reality. In essence, the position within a cell which corresponds to a measurement is not registered by occupancy-grid map approaches. Meanwhile, the error in the B-spline map is considerable lower because when building the map, it is considering the exact place where the obstacle was detected.

The second simulated environment assesses the impact of noise in the range measurements provided by the sensor. This time, the robot stands still at the center of a circular space of 2 meters radius. The sensor range data is perturbed with zero mean Gaussian noise. The standard distribution of the noise varies from .025 m to .1 m, which corresponds from a quarter to one cell/knot interval length. The final map was built after 500

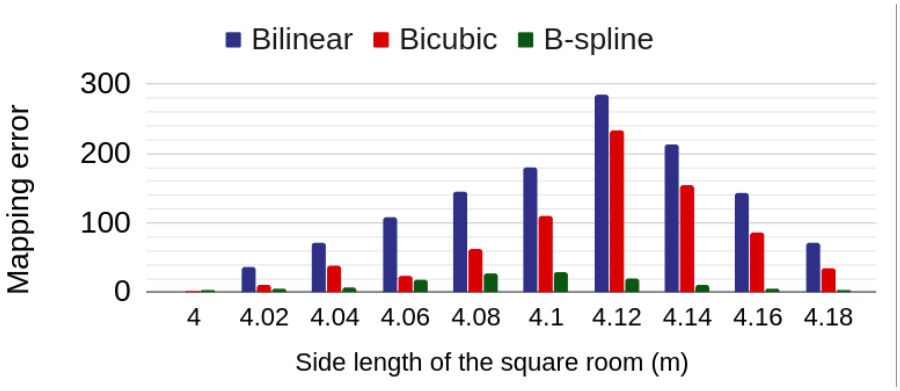


FIGURE 7.6: Mapping error using the room scenario shown in Fig. 7.5. Both the cell resolution and knot vector grid have a resolution of 0.1 m. Throughout the scenarios, the shortest distance from the robot to the wall varies from 2 m to 2.09 m.

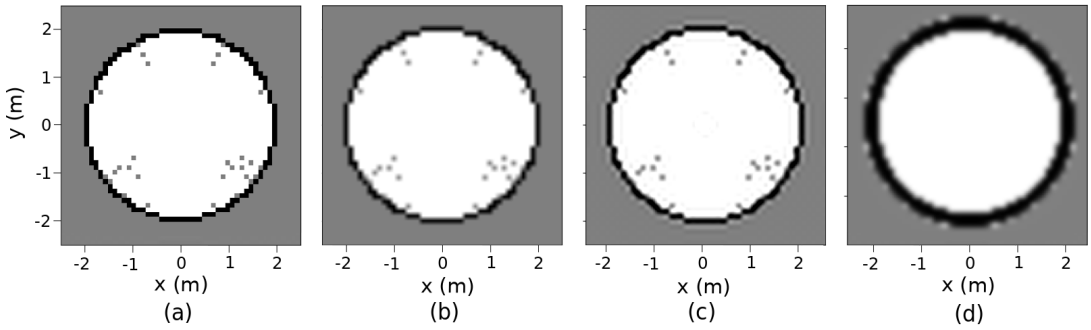


FIGURE 7.7: Mapping result for an artificially generated circular room using: (a) occupancy-grid, (b) occupancy-grid + bilinear interpolation, (c) occupancy-grid + bicubic interpolation, and (d) proposed B-spline map.

sensor readings. The resulting maps for the noiseless case are shown in Fig. 7.7. The occupancy based methods have regions of unknown space caused by their discrete nature and due to the ray-casting used, where sensor rays are not intersecting all the cells. The B-spline map does not present the gaps because a measurement has impact on nearby regions. The statistical analysis for the noisy case can be seen in Fig. 7.8. In total, 33 simulations for each set of map and standard deviation were performed. The proposed method behaves better in all the scenarios. A reason for that is because B-splines naturally filter the noise while building the map.

Real-data

The results using real-data were obtained using a Turtlebot3 (see Sec. 3.4). The robot was teleoperated through the corridors of the Faculty of Engineering of the University of Porto (FEUP), Building I, first floor, rooms I 119 and I 173. For generating the maps, first it is

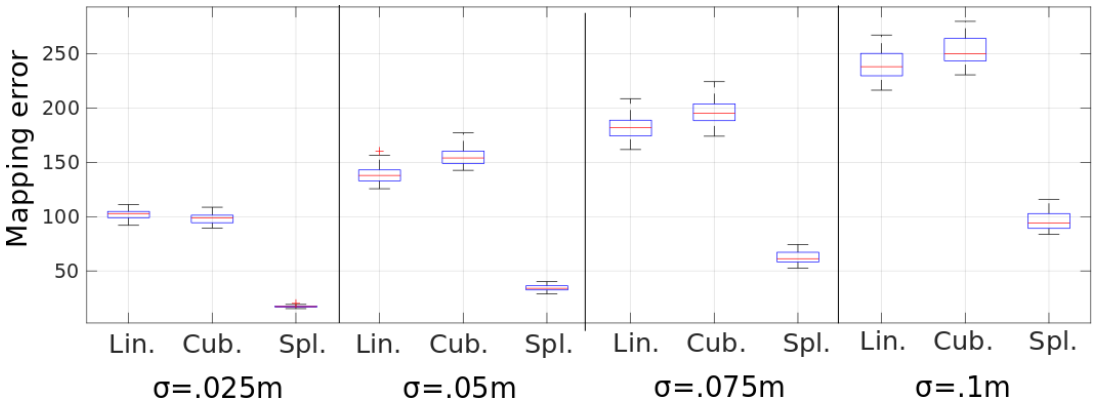


FIGURE 7.8: Statistical analysis using noisy sensor data for the scenario shown in Fig. 7.7. The noise follows a zero-mean Gaussian distribution. Different standard deviations were considered. *Lin.*, *Cub.*, and *Spl.* stands for the bilinear, the bicubic, and the B-spline map, respectively.

TABLE 7.1: Running-time for building the maps shown in Fig 7.9

	Occupancy-grid	Fast-GPOM	Spline-Map
Time (ms)	5.94	142.60	1.13

recorded a bag file with the encoder and sonar data. Then, the pose of the robot is estimated using GMapping [4, 72]. The estimated pose and LiDAR measurements were fed into different mapping techniques - all of them implemented in Python. The maps were generated using 5 cm resolution for both the discrete cells and knot spacing. The degree of the B-spline function is $d = 3$. Figure 7.9(a) shows the occupancy-grid map obtained using our vanilla implementation. Figure 7.9(b) illustrates the output of Fast-GPOM [78], a continuous Gaussian Process map. Figure 7.9(c) shows the map obtained using the proposed B-spline surface method. A few annotations are shown in red numbers. The three methods perform rather well. A careful inspection shows that the upper right section of the Fast-GPOM map is slight deformed, representing the region larger than it actually is. As highlighted in annotation 1, Fast-GPOM is not able to register well features with high curvature. Although this is a typical limitation of continuous maps, our method is able to capture sharp regions significantly better. The proposed B-spline map is also capable of handling small objects. For example, annotation 2 corresponds to the three legs of a large bench, each leg is 5 cm wide.

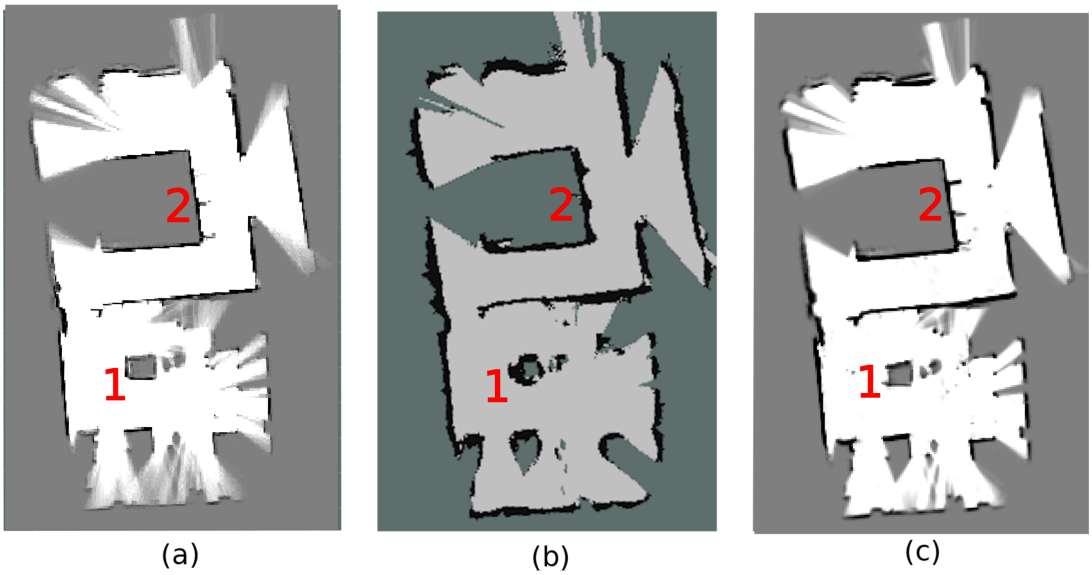


FIGURE 7.9: Experimental results using turtlebot/LiDAR setup in the corridors of FEUP. The results shown correspond to the output of occupancy-grid (a), Fast-GPOM (b), and the proposed B-spline map (c). The proposed approach is representing the environment in an accurate manner being able to capture even small obstacles. Annotations are depicted in red numbers.

The average time per iteration for building the maps is shown in Table I. The time were obtained by running the code in a notebook Intel Core i5-3317U 1.7GHz, 4 GB RAM. The results show that the B-spline map is the fastest one, followed by the occupancy-grid map. This was not expected, because an occupancy-grid map updates one cell per (occupied or free space) measurement, while the proposed strategy updates 16 control points ($d = 3$) per (occupied or free space) measurement. It is most likely that our B-spline map implementation is more optimized than our occupancy-grid map implementation. While both aforementioned methods have computational complexity $O(1)$ per measurement update, Gaussian Process maps have computational complexity $O(n^3)$ [79], where n is the size of the sub-matrix of the kernel to be updated. This explains the high average time per iteration.

7.5.2 SLAM Results

In this section the proposed algorithm is evaluated using public data sets that provide different sensors and odometry measurements. In our tests, the main characteristic that has an impact on our algorithm is the sensor rate. The higher the rate of the sensor, the better the assumption that the pose displacement between two readings is small holds, i.e., $\Delta\boldsymbol{\xi} \approx \mathbf{0}$. Therefore, when using low rate sensors more map resolutions are required to avoid getting trapped in a local minimum during localisation. The parameters of the proposed algorithm are the same for all experiments, except for the number of map resolutions, which is clearly stated. For visualisation, the B-spline surface is sampled at 0.05 m interval and displayed as a gray-scale image. The darker a pixel is, the more likely it is to correspond to an occupied area.

TU Darmstadt data set

(Map resolutions: 1, knot interval: 5 cm) The TU Darmstadt data set contains data recorded using an IMU and a Hokuyo UTM-30LX LIDAR (45 Hz). The resulting map for two scenarios computed by B-spline SLAM are shown in Fig. 7.10. The two sensors (IMU and LiDAR) were assembled in a handheld kit. For instance, the data from the Dagstuhl building (Fig. 7.10a) was acquired by a human walking through the environment carrying the sensors by hand. The sensor was subject to roll, pitch, and vertical oscillations breaking the planar movement constraint. The results presented in [5] (Hector-SLAM) correct the measurements by incorporating the IMU data to obtain a stabilised coordinate frame.

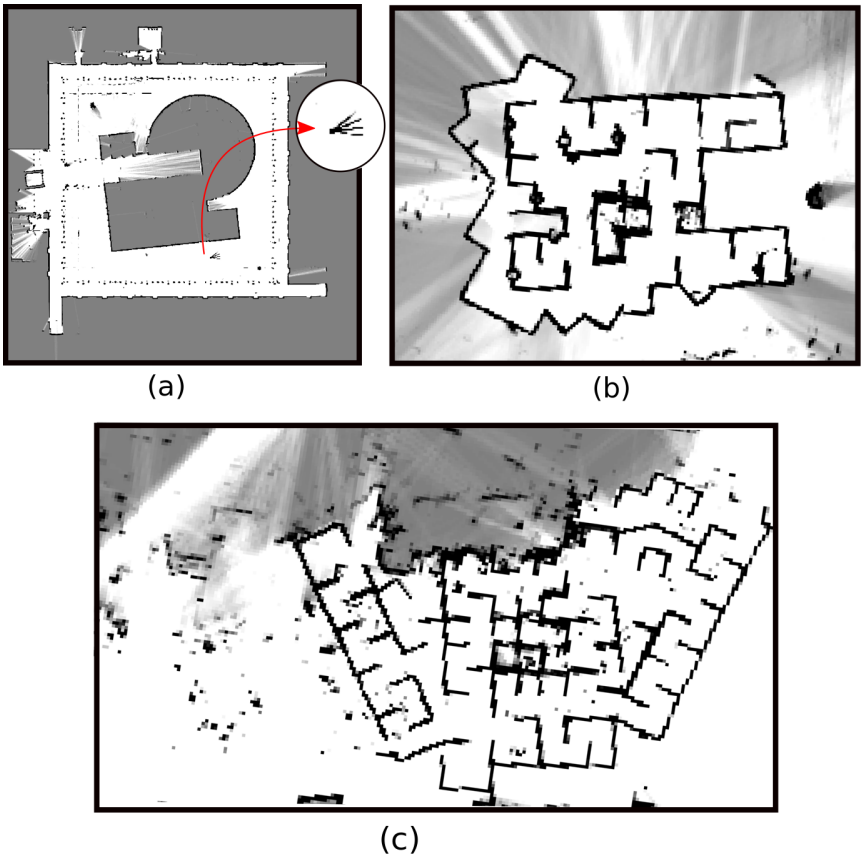


FIGURE 7.10: B-spline SLAM map for TU Darmstadt dataset: (a) Dagstuhl, (b) RoboCup 2010, and (c) RoboCup 2011 Rescue Arena.

In our SLAM framework the IMU data is not taken into account. Since no ground truth is provided, it is hard to assert whether a mapping structure presented in our solution but not in [5] represents an advantage or disadvantage. Given that both maps in Fig. 7.10 are coherent, one may conclude that our results are good in a qualitative sense.

Radish data set

(Map resolutions: 3, knot interval: 5, 12.5, 30 cm) The Robotics Data Set Repository (Radish) [71] contains odometry data and range scan measurements (approx. 5 Hz frequency). Figure 7.11 shows the qualitative results using the proposed algorithm in four of the five scenarios evaluated. For quantitative results, the metric proposed by Kummerle et al. [70] is used. The authors show that using global pose (i.e., a fixed reference frame) is sub-optimal for comparing SLAM algorithms. Instead, they propose comparing the translational and rotational errors between two relative poses (absolute and squared errors). For more about this metric, the reader is referred to Sec. 5.4.

Tables 7.2 and 7.3 show the comparison of our algorithm with others in the literature. Table 7.2 describes the properties inherent to each strategy. Table 7.3 quotes the results for RBPF with 50 particles (GMapping) [70], Cartographer [12], two-level optimisation [80], and TSDF [43]. The best results are highlighted in bold and a dash indicates no information was provided by the authors for the corresponding data set. The methods being compared use the odometry data available in the log files. Attempts were made to process the datasets using Hector-SLAM, but unfortunately it was failing to produce a correct map despite extensive tuning efforts. It is hypothesized that the high angular displacement between consecutive readings did not allow the correct operation of that SLAM method. On the other hand, B-spline SLAM fails only in the MIT Killian Court due to the long and narrow corridors which result to the infinite corridor problem. Other than this, it performs well. In the RBPF (50 part) algorithm there are 50 particles, and for each particle a scan matching on a discrete occupancy grid map is performed. The fact that the proposed algorithm is consistently better than RBPF shows the potential of B-spline based SLAM. Comparing with more recent methods that run back-end optimisation, the proposed strategy is still competitive: it outperforms Cartographer for the MIT CSAIL dataset and it is able to achieve either the smallest absolute or squared rotational error in the ACES, Intel, and Freiburg bld 69 data sets. It is likely that the accuracy of the proposed SLAM comes from the fact that the continuous B-spline maps are more accurate than discrete maps (as discussed in Sec. 7.5.1) and allow for a larger basin of convergence when performing scan-matching. There is still room to improve B-spline SLAM, since in contrast to these other methods neither loop-closure mechanism nor back-end optimisation is performed.

The implementation of the proposed algorithm can process the Radish data set 1.5 to 2 times faster than the sensor rate (evaluated on a computer with an Intel Core i5-3317U

TABLE 7.2: Main features of compared methods.

	Map	Belief	Loop-closure	Back-end
B-spline SLAM	B-spline	single	no	no
RBPF	discrete grid	multi	yes[1]	no
Cartographer	discrete grid	single	yes	yes
Two-level	landmarks	single	yes	yes
TSDF	TSDF	single	yes	yes

[1]Particles that do not explain the loop closure are likely to be removed.

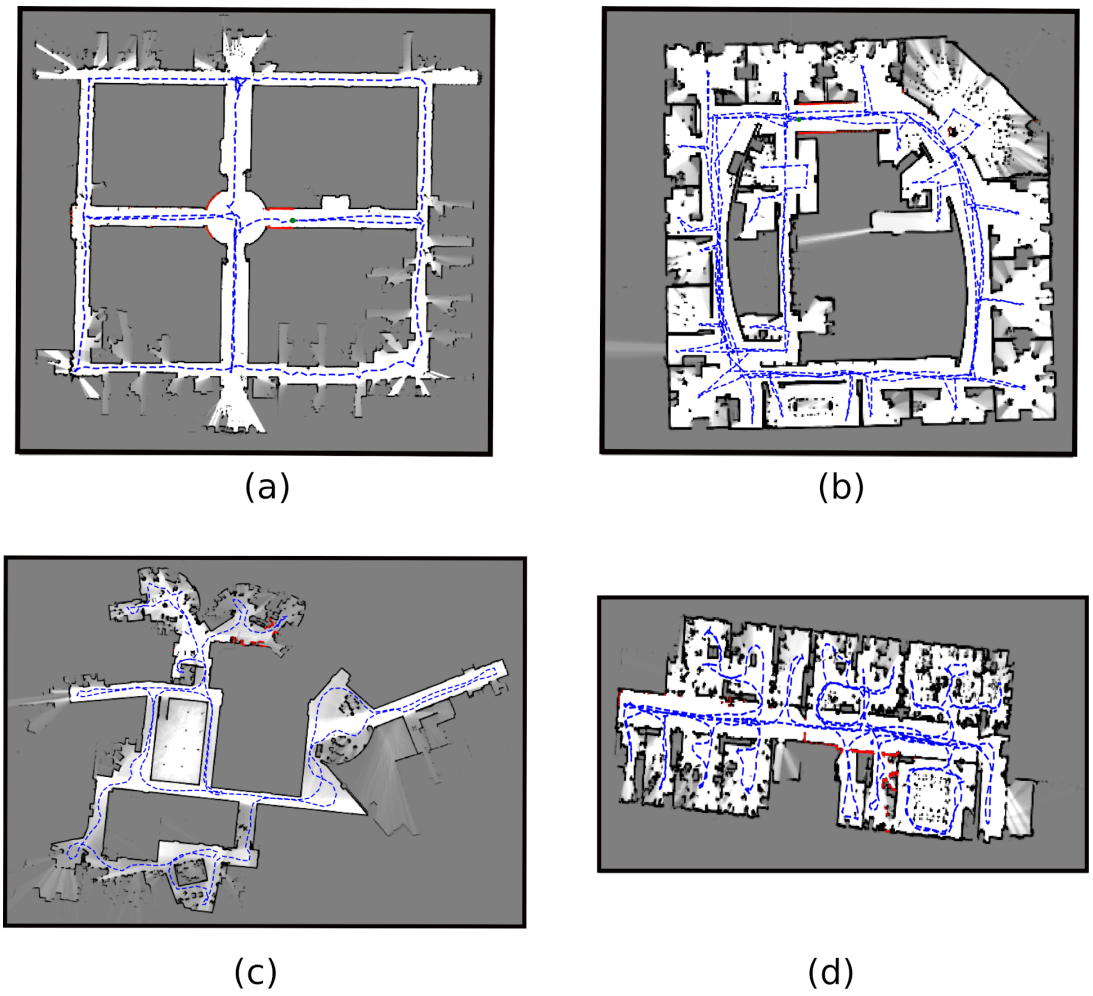


FIGURE 7.11: B-spline SLAM output using the Radish dataset [70]: (a) ACES Bulding, (b) Intel Research Lab, (c) MIT CSAIL building, and (d) Freiburg building 69. The blue dashed line is the path travelled by the robot.

1.7GHz). For the mapping task, updating or evaluating the map has computational complexity $O(1)$. Regardless of the size of the map, by exploiting the local property of B-spline, only 16 control points (for $d = 3$) have to be evaluated or updated. For the localisation task the computational cost is similar to the front-end of Cartographer or any other grid-based method that relies on bi-cubic interpolation. The complexity grows linearly with the number of iterations of the nonlinear least squares solver. In the worst case, the computational time is limited by the maximum number of iterations.

TABLE 7.3: Comparison using the metrics proposed in [70]. For each metric, best results are highlighted in bold. Dashes indicate that no information was provided by the authors for that data set.

	B-spline SLAM	RBPF (50 part)	Cartographer	Two-level opt.	TSDF
Aces					
Absolute translational (m)	0.0404 ± 0.0452	0.060 ± 0.049	0.0375 ± 0.0426	**0.0283 ± 0.0397**	-
Squared translational (m^2)	0.0036 ± 0.0127	0.006 ± 0.011	0.0032 ± 0.0285	**0.0027 ± 0.0100**	-
Absolute rotational (deg)	**0.340 ± 0.392**	1.2 ± 1.3	0.373 ± 0.469	0.351 ± 0.428	-
Squared rotational (deg^2)	**0.270 ± 1.657**	3.1 ± 7.0	0.359 ± 3.696	0.294 ± 1.253	-
Intel					
Absolute translational (m)	0.0262 ± 0.0281	0.070 ± 0.083	0.0229 ± 0.0239	**0.0150 ± 0.0204**	-
Squared translational (m^2)	0.0014 ± 0.0066	0.011 ± 0.034	0.0011± 0.0040	**0.0009± 0.0009**	-
Absolute rotational (deg)	0.445 ± 0.969	3.0 ± 5.3	0.453 ± 1.335	**0.390 ± 0.402**	-
Squared rotational (deg^2)	**1.137 ± 6.654**	36.7 ± 187.7	1.986 ± 23.988	1.629 ± 9.736	-
MIT Killian Court					
Absolute translational (m)	1.0379 ± 2.5719	0.122 ± 0.386[1]	0.0395 ± 0.0488	0.0367 ± 0.0473	**0.0276 ± 0.0235**
Squared translational (m^2)	7.6918 ± 24.8118	0.164 ± 0.814[1]	0.0039 ± 0.0144	0.0031± 0.0134	**0.0013 ± 0.0095**
Absolute rotational (deg)	0.779 ± 1.246	0.8 ± 0.8[1]	0.352 ± 0.353	0.294 ± 0.275	**0.2807 ± 0.2462**
Squared rotational (deg^2)	2.160 ± 5.652	0.9 ± 1.7[1]	0.248 ± 0.610	0.218 ± 0.439	**0.1394 ± 0.26865**
MIT CSAIL					
Absolute translational (m)	**0.0268 ± 0.0223**	0.049 ± 0.049[1]	0.0319 ± 0.0363	-	-
Squared translational (m^2)	**0.0012 ± 0.0041**	0.005 ± 0.013[1]	0.0023 ± 0.0099	-	-
Absolute rotational (deg)	**0.315 ± 0.274**	0.6 ± 1.2[1]	0.369 ± 0.365	-	-
Squared rotational (deg^2)	**0.175 ± 0.306**	1.9 ± 17.3[1]	0.270 ± 0.637	-	-
Freiburg bldg 69					
Absolute translational (m)	0.0410 ± 0.0315	0.061 ± 0.044[1]	0.0452 ± 0.0354	0.0421 ± 0.0349	**0.0382 ± 0.0292**
Squared translational (m^2)	0.0027 ± 0.0044	0.006 ± 0.020[1]	0.0033 ± 0.0055	0.0029 ± 0.0048	**0.0023 ± 0.0044**
Absolute rotational (deg)	**0.420 ± 0.472**	0.6 ± 0.6[1]	0.538 ± 0.718	0.483 ± 0.571	0.4245 ± 0.4610
Squared rotational (deg^2)	0.399 ± 1.310	0.7 ± 2.0[1]	0.804 ± 3.627	0.682 ± 1.533	**0.3926 ± 1.2308**

[1]Odometry was improved using a pre-processing scan-matching step (see [70]).

Part II

3D Depth Estimation

Chapter 8

Active Depth Estimation: Theory

This chapter presents an active incremental structure-from-motion framework for single camera systems. Section 8.1 introduces the problem addressed. Well-established solutions derived from geometric approaches are presented in Sec. 8.2. Afterwards, Sec. 8.3 discusses incremental methods for depth estimation that consider the dynamics of the camera. Finally, Sec. 8.4 presents the key results, namely the conditions that the perception and control loop must satisfy to ensure convergence of the estimator.

8.1 Depth Estimation

Depth estimation addresses the problem of recovering the 3D structure of the world from observations recorded by an image sensor. Triangulation is the gold standard for objects registered with a calibrated stereo pair, i.e., two calibrated cameras for which the relative pose (baseline) of one camera with respect to the other is known. On the other hand, depth estimation using a single camera is not a trivial problem, as projecting the world in the image frame leads to an unknown scaling factor. Consequently, information beyond the image frame itself is required. The same problem arises in the stereo camera case for far-away objects: if the baseline of the camera pair is significantly shorter than the distance to the observed objects, stereo vision degenerates to monocular vision.

This problem is illustrated in Fig. 8.1: cameras are bearing sensors, meaning that they capture angular information, and not distance. It is possible to recover the line (i.e., the direction) where a projected 3D point lies, but not its depth within this line. In fact, for the image feature ${}^{I}\mathbf{q}$ (represented by a small red solid circle in the image plane), any of the re-projections along the line (illustrated by hollow red circles) is a valid candidate for

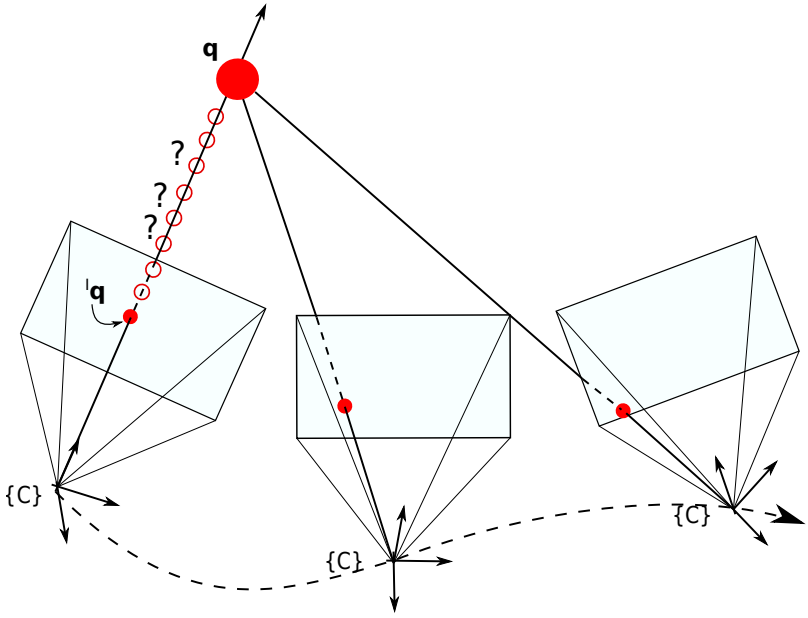


FIGURE 8.1: Depth estimation problem: using a single image frame, one may only recover the direction of a 3D point. Without previous knowledge of the structure of the world, different viewpoints are required for 3D reconstruction.

the observed 3D point. In essence, it is only possible to reconstruct the 3D point up to a scale.

It is possible to recover the 3D structure of an object from a single image frame if previous knowledge about the geometry of the points is known. One example is to use artificial tags with known dimensions (fiducial markers) [14]. This thesis investigates the generic case where the environment is unknown. In such circumstances, recovering the 3D structure of the world requires observing an object from two or more different view-points, and the relation between these view-points must be known or estimated. This is typically the case for a robotic system with an onboard camera: the camera sensor moves in the environment, capturing a temporally and spatially ordered sequence of images, and consecutive images are likely to contain significant overlapping area. The relation between two or more camera frames can be obtained from proprioceptive sensors, such as IMU and encoders, or exterceoptive sensors, such as GPS or LiDAR (for example, the SLAM covered in previous chapters).

Depth estimation is correlated with the computer vision topic structure-from-motion (SfM). However, SfM tackles the problem of estimating the relative poses of the camera and the 3D structure of the environment from a large collection of ordered or unordered images in a process which is typically done offline [81].

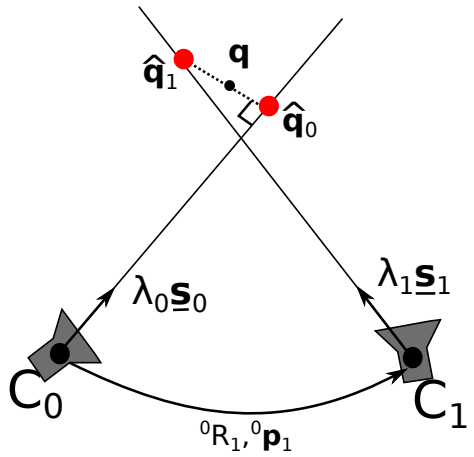


FIGURE 8.2: Linear triangulation for 2-view: the same camera observes a 3D point from two different view-points.

The following two sections cover geometric and incremental (filtering) methods for depth estimation. The former relies on basic geometric relations, while the latter accounts for the dynamics of the camera observing the world. Since no previous knowledge about the environment is given, both methods require two or more view-points.

8.2 Geometric Methods

Geometric-based techniques such as [16, 82] resort to triangulation for estimating the depth of the points detected in two or more different image frames, at different positions/orientations. The results obtained in [13, 83] are summarized next.

[2-view triangulation] The simplest geometric method consists in linearly triangulating point correspondences in two views. As illustrated in Fig. 8.2, consider a camera that captures a first image at $\{C_0\}$ and a second image at $\{C_1\}$. Both images capture a 3D point $\mathbf{q} = [x_c, y_c, z_c]^T$, described in the coordinate system $\{C_0\}$. $\mathbf{s}_i = [x_i, y_i]^T$ denotes the normalized projection of the 3D point $\mathbf{q}$ in the *i-th* image frame, obtained using the calibration parameters as in (3.7). Converting to homogeneous coordinates, the normalized projection vector becomes $\underline{\mathbf{s}}_i = [x_i, y_i, 1]^T$. Let ${}^0R_1$ and ${}^1\mathbf{p}_0$ be the rotation and translation of the camera from $\{C_0\}$ to $\{C_1\}$. Then, the estimated coordinates of the 3D point $\mathbf{q}$ described in $\{C_0\}$ are given by

$$\hat{\mathbf{q}}_0 = \lambda_0 \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}, \qquad \hat{\mathbf{q}}_1 = \lambda_1 {}^0R_1 \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} + {}^0\mathbf{p}_1, \tag{8.1}$$

where $\lambda_i \in \mathbb{R}$ is an unknown scaling factor. Due to measurement noise, it is unlikely that the rays described by $\underline{\mathbf{s}}_0$ and $\underline{\mathbf{s}}_1$ intercept. Nonetheless, one may compute the closest point to the interception, which lies in the center of the shortest possible line that is orthogonal to both rays. This line is defined by $\hat{\mathbf{q}}_0$ and $\hat{\mathbf{q}}_1$, which must satisfy the following constraints:

$$(\hat{\mathbf{q}}_0 - \hat{\mathbf{q}}_1)^T \underline{\mathbf{s}}_0 = 0 \text{ and } (\hat{\mathbf{q}}_0 - \hat{\mathbf{q}}_1)^T \underline{\mathbf{s}}_1 = 0 \tag{8.2}$$

These constraints can be stacked in matrix form, obtaining the linear system

$$\begin{bmatrix} \underline{\mathbf{s}}_0^T \underline{\mathbf{s}}_0 & -({}^0R_1 \underline{\mathbf{s}}_1)^T \underline{\mathbf{s}}_0 \\ \underline{\mathbf{s}}_0^T \underline{\mathbf{s}}_1 & -({}^0R_1 \underline{\mathbf{s}}_1)^T \underline{\mathbf{s}}_1 \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{p}_1{}^T \underline{\mathbf{s}}_0 \\ {}^0\mathbf{p}_1{}^T \underline{\mathbf{s}}_1 \end{bmatrix} \tag{8.3}$$

The solution is computed by inverting the matrix on the left side of the equation. Once the scales are recovered, the 3D point may be assumed to be the average of $\hat{\mathbf{q}}_0$ and $\hat{\mathbf{q}}_1$.

This method allows estimating the depth of a 3D point using two view-points, if the correspondence between points in different image frames is known. However, it does not work if the two view-points differ only in orientation (no translation). In this case, the left side of (8.3) becomes zero, and the only solution is the null vector, which does not correspond to the actual scale. Therefore, camera translation is required to solve the depth estimation problem.

[n-view triangulation] The described method may be extended to n view-points, if the point correspondences for a 3D point are known across the different view-points. In that case, the projection model of a 3D point $\mathbf{q}$ in the i-th camera frame using homogeneous coordinate is stated as follows:

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \propto \begin{bmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} {}^iR_0 & {}^i\mathbf{p}_0 \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = P_i \underline{\mathbf{q}}, \tag{8.4}$$

where the symbol $\propto$ indicates that the equality holds up to an unknown scale. The camera parameters $(\rho_u, \rho_v, u_0, v_0, f)$ are as described in Chapter 3. The matrix P_i, known as projection matrix, projects a point in the 3D world into the image plane. Let $\mathbf{p}_i^j$ be the *j-th*

column of the matrix P_i. Then, the cross product

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{p}_i^{1T}\underline{\mathbf{q}} \\ \mathbf{p}_i^{2T}\underline{\mathbf{q}} \\ \mathbf{p}_i^{3T}\underline{\mathbf{q}} \end{bmatrix} = \mathbf{0} \tag{8.5}$$

holds, because the two vectors are aligned. Thus, for each point, two constraints can be derived:

$$(\mathbf{p}_i^3 u_i^T - \mathbf{p}_i^1)^T \underline{\mathbf{q}} = 0, \qquad (\mathbf{p}_i^3 v_i^T - \mathbf{p}_i^2)^T \underline{\mathbf{q}} = 0 \tag{8.6}$$

The 3D point $\mathbf{q}$ can be recovered by assembling a system of $2n$ linear equations, where n is the number of observations. There are two ways of computing the solution. One can set the last coordinate of the homogeneous vector to 1 (as shown here), and obtain the solution using linear least squares. On the other hand, some authors, e.g., [16], assume the last coordinate of the homogeneous vector to be a variable, and obtain the solution via the *Direct Linear Transformation* method (DLT). The underlying idea is to compute the singular value decomposition of the matrix A in a system $A\mathbf{x} = 0$, that is, decompose the matrix as $A = USV^T$, where S is the matrix of singular values. The solution is the column of V associated with the smallest eigenvalue of S (the column that spans the null space of A). One can further refine the solution by forcing the eigenvalue to be 0 and recomputing the matrix A and its eigenvalue decomposition [13].

The disadvantage of triangulation methods is that the computational cost increases noticeably with more view-points. Furthermore, the solution is ill-conditioned when the baseline is relatively small relatively to the distance of the point being observed.

8.3 Incremental Depth Estimation

Filtering or incremental-based methods [84, 85] explicitly consider the dynamics of 3D points tracked across a sequence of continuously acquired images. While geometric-based techniques suffer from small baseline camera displacements, incremental-based methods take advantage of these small continuous motions of the cameras to continuously estimate the 3D structure of the scene and provide a robust estimation of the model uncertainties. Also, the computational cost for filtering methods is constant for multiple observations, while geometric strategies do not scale well.

Problem Definition

For a short introduction, consider a 6 Degrees of Freedom (DoF) camera moving freely in space, and the frame $\{C\}$ attached to the origin of the sensor. The camera observes a 3D point described in $\{C\}$ as $\mathbf{q} := [x_c, y_c, z_c]^T \in \mathbb{R}^3$ and the dynamics of $\mathbf{q}$ relative to the camera frame are

$$\dot{\mathbf{q}} = -\mathbf{w} \times \mathbf{q} - \mathbf{v} = \begin{cases} \dot{x}_c = y_c w_z - z_c w_y - v_x \\ \dot{y}_c = z_c w_x - x_c w_z - v_y \\ \dot{z}_c = x_c w_y - y_c w_x - v_z \end{cases}, \tag{8.7}$$

where $\mathbf{v} := [v_x, v_y, v_z]^T \in \mathbb{R}^3$ and $\mathbf{w} := [w_x, w_y, w_z]^T \in \mathbb{R}^3$ are the camera linear and angular velocities described in $\{C\}$ (assuming that the body and camera frames coincide). Both $\mathbf{v}$ and $\mathbf{w}$ are bounded by the actuation limits. Let $\mathbf{s} := [x, y]^T = [x_c/z_c, y_c/z_c]^T \in \mathbb{R}^2$ be the projection of $\mathbf{q}$ into the camera's normalized image plane[1].

Consider the change of variable $\chi = 1/z_c$. The time-derivative of the new variables is given by

$$\begin{cases} \dot{x} = \left(\dot{x}_c z_c - x_c \dot{z}_c\right) / z_c^2 \\ \dot{y} = \left(\dot{y}_c z_c - y_c \dot{z}_c\right) / z_c^2 \\ \dot{\chi} = -\dot{z}_c / z_c^2 \end{cases}. \tag{8.8}$$

Substituting (8.7) in (8.8) and writing in the new coordinate system, yields

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\chi} \end{bmatrix} = \begin{bmatrix} -\chi & 0 & x\chi & xy & -(1+x^2) & y \\ 0 & -\chi & y\chi & 1+y^2 & -xy & -x \\ 0 & 0 & \chi^2 & y\chi & -x\chi & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix}. \tag{8.9}$$

The dynamics of the system can be stated in a more compact form:

$$\begin{cases} \dot{\mathbf{s}} &= J_v \mathbf{v} \chi + J_w \mathbf{w} \\ \dot{\chi} &= J_q \mathbf{v} \chi^2 + J_l \mathbf{w} \chi \end{cases}, \tag{8.10}$$

[1] The normalized coordinates of a pixel in the image plane of a calibrated camera may be computed as in (3.7).

where

$$J_v = \begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \end{bmatrix}, \quad J_w = \begin{bmatrix} xy & -(1+x^2) & y \\ 1+y^2 & -xy & -x \end{bmatrix}, \\ J_q = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, \quad J_l = \begin{bmatrix} y & -x & 0 \end{bmatrix} \tag{8.11}$$

The linear and angular velocity of the camera are assumed to be known, as well as the projection of the point to the image plane $\mathbf{s}$ (also called measurable variable). The challenge lies in estimating the unknown depth described by χ (also denoted as unmeasurable variable). The difference between passive and active depth estimation is that, while the former uses the velocities $(\mathbf{v}, \mathbf{w})$ provided by some high level task, the later computes the velocities that guarantee convergence of the 3D depth estimation and/or improve the transient response of the estimator. Introducing the estimated variables $\hat{\mathbf{s}}$ and $\hat{\chi}$ and the respective estimation errors $\tilde{\mathbf{s}} = \mathbf{s} - \hat{\mathbf{s}}$ and $\tilde{\chi} = \chi - \hat{\chi}$, the problem tackled in this thesis is formally stated:

Problem 4. Consider the system described in (8.10) with control input ($\mathbf{v}$, $\mathbf{w}$) and known camera calibration parameters. Design observers ($\hat{\mathbf{s}}$,$\hat{\chi}$) and a control policy for ($\mathbf{v}$, $\mathbf{w}$), such that the state estimation errors $\tilde{\mathbf{s}}$ and $\tilde{\chi}$ converge to zero as time approaches infinity, i.e., $\tilde{\mathbf{s}} \to 0$ and $\tilde{\chi} \to 0$ as $t \to \infty$.

Estimator

For a calibrated camera, the normalized projection of a 3D point in the image plane is straightforward obtained from the pixel coordinates of the corresponding feature. For this reason, $\mathbf{s}$ is called a *measurable* variable. The challenge is to estimate the unknown depth described by χ – called here as an *unmeasurable* variable. The problem is addressed using an estimator inspired in [23], however the control input of the camera is not combined with the matrix J_v, J_w, J_q, and J_l:

$$\begin{cases} \dot{\hat{\mathbf{s}}} = J_v \mathbf{v} \hat{\chi} + J_w \mathbf{w} + k_s \tilde{\mathbf{s}} \\ \dot{\hat{\chi}} = J_q \mathbf{v} \hat{\chi}^2 + J_l \mathbf{w} \hat{\chi} + k_\chi (J_v \mathbf{v})^T \tilde{\mathbf{s}} \end{cases}, \tag{8.12}$$

where $k_s, k_\chi \in \mathbb{R}^+$ are the control gains. The corresponding estimation error dynamics is

$$\begin{cases} \dot{\tilde{\mathbf{s}}} = J_v \mathbf{v} \tilde{\chi} - k_s \tilde{\mathbf{s}} \\ \dot{\tilde{\chi}} = \tilde{\chi}(J_q \mathbf{v}(\chi + \hat{\chi}) + J_l \mathbf{w}) - k_\chi (J_v \mathbf{v})^T \tilde{\mathbf{s}} \end{cases} . \tag{8.13}$$

Since one may observe $\mathbf{s}$, the error $\tilde{\mathbf{s}}$ is measurable (known), while the depth estimation error $\tilde{\chi}$ is unmeasurable (unknown). Both variables are coupled by the term $J_v \mathbf{v}$, which is essential to the performance of the estimator. In the next section, we analyse the constraints that must be respected to enforce that the estimation error in (8.13) converges to the origin.

Active vision

In the last decade, some authors studied the use of active vision techniques, that is, including in the control loop the goals of 3D reconstruction. An observer similar to (8.12) is employed in an active framework in [86]. The authors show that the depth estimation error globally converges to the origin provided in the special case $\dot{\chi} = 0$, meaning that the parameter is unknown, but constant. Their optimal actuation policy computes the angular velocity that forces the point to move to and remain in the origin of the image plane, i.e., $\mathbf{s} = (0, 0)$, while the camera translates. In fact, the proposed optimal linear velocity is perpendicular to the camera optical axis, i.e., $v_z = 0$. This maximizes the rate of convergence dictated by the observability measurement $\sigma^2 = \|J_v \mathbf{v}\|_2^2$, which is a metric to be discussed in more detail in the next section. The framework was later used for environments containing cylinders, spheres (see [23]), 3D planes (in [24]), and 3D straight lines (see [87])

8.4 Convergence of the Estimator

In this section we present the stability analysis of the depth estimation filter while mapping a 3D point, i.e., Problem 4 defined in the previous section. The goal is to provide guarantees for the convergence of the unmeasurable depth for recovering the 3D structure of the world given by $\mathbf{q}^T = [\mathbf{s}, 1]/\chi$ using the actuation input of the camera. We will make use of the following assumption.

Assumption 1. The observed 3D point cannot lie behind the camera. Consequently, we restrict our analysis to the domain where χ is positive, that is, we assume $\chi \geq 0, \forall t$.

This assumption has an explicit physical meaning. In fact, cameras are not able to observe 3D points that are behind them. This would require a negative depth.

Lemma 8.1. *Consider the dynamic system* $\dot{\mathbf{x}}(t) = \phi(\mathbf{x}, \mathbf{u})g(\mathbf{x})$*, where* $\mathbf{u}$ *is the input of the system. If* $\dot{\mathbf{x}}(t)$ *converges to the origin as time approaches infinity, that is,* $\lim_{t\to\infty} \dot{\mathbf{x}} = \mathbf{0}$*, it must be the case that either* $\phi(\mathbf{x}, \mathbf{u}) \to 0$ *or* $g(\boldsymbol{x}) \to 0$*. By designing* $\phi(\mathbf{x}, \mathbf{u})$ *such that it is persistently exciting through all time, then one may conclude that* $g(\mathbf{x})$ *converges to zero. The signal* ϕ *is persistently exciting if the integral*

$$\int_{t_0}^{t} \phi(\mathbf{x}, \mathbf{u})^T \phi(\mathbf{x}, \mathbf{u}) d\tau \tag{8.14}$$

is positive definite $\forall t \geq t_0$*. Hence, the persistency of excitation (PE) condition holds if*

$$\phi(\mathbf{x}, \mathbf{u})^T \phi(\mathbf{x}, \mathbf{u}) > 0. \tag{8.15}$$

The proof for Lemma 8.1 can be found on [88]. The following result can now be stated.

Theorem 8.2. *Consider the estimator in* (8.12) *for the dynamic system* (8.10) *under Assumption* (1)*, and with* $\boldsymbol{v}$*,* $\boldsymbol{w}$*, and their time-derivatives bounded signals. The equilibrium point* $(\tilde{\mathbf{s}}, \tilde{\chi}) = \mathbf{0}$ *is stable and the estimation error converges to zero as* $t \to \infty$*, provided that* $\forall t \geq t_0$ *the following constraints hold simultaneously:*

1. $J_l \mathbf{w} \leq 0$;
2. $\begin{cases} J_q \mathbf{v} \leq 0 \text{ , if } \hat{\chi} > 0 \\ J_q \mathbf{v} = 0 \text{ , otherwise} \end{cases}$;
3. $\sigma^2 = (x v_z - v_x)^2 + (y v_z - v_y)^2 > 0$.

Proof. Consider the Lyapunov function candidate

$$V(\tilde{\mathbf{s}}, \tilde{\chi}) = \frac{1}{2}\|\tilde{\mathbf{s}}\|^2 + \frac{1}{2k_\chi}\tilde{\chi}^2, \tag{8.16}$$

with $k_\chi > 0$, and its time-derivative

$$\dot{V} = \tilde{\mathbf{s}}^T \dot{\tilde{\mathbf{s}}} + \frac{1}{k_\chi}\tilde{\chi}\dot{\tilde{\chi}}. \tag{8.17}$$

Substituting (8.13) in the previous equation:

$$\begin{aligned}\dot{V} &= \tilde{\mathbf{s}}^T(J_v\mathbf{v}\tilde{\chi} - k_s\tilde{\mathbf{s}}) + \\ &\quad + \frac{1}{k_\chi}\tilde{\chi}(J_q\mathbf{v}(\chi+\hat{\chi})\tilde{\chi} + J_l\mathbf{w}\tilde{\chi} - k_\chi(J_v\mathbf{v})^T\tilde{s}) \qquad (8.18)\\ &= -\tilde{\mathbf{s}}^T k_s\tilde{\mathbf{s}} + \frac{1}{k_\chi}\tilde{\chi}J_q\mathbf{v}(\chi+\hat{\chi})\tilde{\chi} + \frac{1}{k_\chi}\tilde{\chi}J_l\mathbf{w}\tilde{\chi}. \qquad (8.19)\end{aligned}$$

By combining Assumption 1 and the input constraints stated in Theorem 8.2, the following upper bounds can be derived: $J_q\mathbf{v}(\chi+\hat{\chi}) \leq 0$ and $J_l\mathbf{w} \leq 0$. This means that the three terms in the right-hand side of (8.19) are non-positive. Hence, $\dot{V} \leq 0$ and the equilibrium point $(\tilde{\mathbf{s}}, \tilde{\chi}) = \mathbf{0}$ is stable. We also conclude that $V(t) \leq V(t_0)$, and therefore, that the signals $\tilde{\mathbf{s}}$ and $\tilde{\chi}$ are bounded.

The critical case that precludes asserting asymptotically stability from (8.19) occurs when $J_q\mathbf{v} = 0$ and $J_l\mathbf{w} = 0$, and consequentially, $\dot{V} = -\tilde{\mathbf{s}}^T k_s\tilde{\mathbf{s}}$. This happens either because the feature lies in the origin of the image plane, or because $\mathbf{v} \in N(J_q)$ and $\mathbf{w} \in N(J_l)$, simultaneously. For $J_q\mathbf{v} = 0$ and $J_l\mathbf{w} = 0$, the second derivative of the Lyapunov candidate function is

$$\ddot{V} = -2\tilde{\mathbf{s}}^T k_s\dot{\tilde{\mathbf{s}}} = -2\tilde{\mathbf{s}}^T k_s(J_v\mathbf{v}\tilde{\chi} - k_s\tilde{\mathbf{s}}). \qquad (8.20)$$

As $\tilde{\mathbf{s}}$, $\tilde{\chi}$, and $\mathbf{v}$ (by definition) are bounded, the function $\ddot{V}$ is also bounded. Thus, $\dot{V}$ is uniformly continuous and from Barbalat's Lemma [89], we have that $\tilde{s} \to 0$ as $t \to \infty$. Now, for the asymptotic behaviour of $\tilde{\chi}$ when $J_q\mathbf{v} = 0$ and $J_l\mathbf{w} = 0$ from (8.13) we have, as $t \to \infty$,

$$\begin{cases} \lim_{t\to\infty}\dot{\tilde{\mathbf{s}}} = \lim_{t\to\infty} J_v\mathbf{v}\tilde{\chi} \\ \lim_{t\to\infty}\dot{\tilde{\chi}} = 0 \end{cases}. \qquad (8.21)$$

The second equation states that the depth estimation error becomes a constant, but not necessarily zero. To show that indeed it will converge to zero, we first show that $\dot{\tilde{\mathbf{s}}}$ is uniformly bounded because its time derivative given by

$$\begin{aligned}\ddot{\tilde{\mathbf{s}}} &= \dot{J}_v\mathbf{v}\tilde{\chi} + J_v\dot{\mathbf{v}}\tilde{\chi} + J_v\mathbf{v}\dot{\tilde{\chi}} - k_s\dot{\tilde{\mathbf{s}}} \qquad (8.22)\\ &= \dot{J}_v\mathbf{v}\tilde{\chi} + J_v\dot{\mathbf{v}}\tilde{\chi} - k_\chi J_v\mathbf{v}(J_v\mathbf{v})^T\tilde{\mathbf{s}} - k_s J_v\mathbf{v}\tilde{\chi} + k_s^2\tilde{\mathbf{s}} \qquad (8.23)\end{aligned}$$

is a function of bounded signals. Thus, since $\tilde{s}$ converges to the origin and $\dot{\tilde{\mathbf{s}}}$ is uniformly bounded, we conclude that $\dot{\tilde{\mathbf{s}}} \to 0$ as $t \to \infty$. Consequently, we have that

$$\lim_{t\to\infty} \dot{\tilde{\mathbf{s}}} = \lim_{t\to\infty} J_v \mathbf{v} \lim_{t\to\infty} \tilde{\chi} = 0. \tag{8.24}$$

One may apply Lemma 8.1 with $\phi(\cdot) = J_v\mathbf{v}$ and $g(\cdot) = \tilde{\chi}$. If the function $J_v\mathbf{v}$ is persistently exciting as in (8.14), then the depth estimation error converges to zero. The persistency of excitation (PE) condition holds if $(J_v\mathbf{v})^T J_v\mathbf{v} > 0$, which is the case from condition (3) in Theorem 8.2. Thus, one can now conclude that the equilibrium point $(\tilde{\mathbf{s}}^T, \tilde{\chi}) = \mathbf{0}$ is asymptotically stable. □ □

Based on Theorem 8.2 and its proof, we draw the following remarks.

Remark 8.3. The practical meaning of the constraints described in Theorem 8.2 are:

1. The persistency of excitation condition given by

$$\sigma^2 = (J_v\mathbf{v})^T J_v\mathbf{v} > 0 \tag{8.25}$$

   is related to the fact that the camera must translate for recovering the depth of a 3D point. This condition is also derived in [86]. The authors show that σ^2 plays the role of an observability index and increasing its value improves the rate of convergence of the estimator. Also, from (8.13), notice that $J_v\mathbf{v}$ is a cross-coupling term linking the dynamics of the estimation of the measurable and the unmeasurable variables. In fact, if $J_v\mathbf{v} = 0$, then the $\dot{\hat{\mathbf{s}}}$ and $\dot{\hat{\chi}}$ become independent from one another;

2. $J_q\mathbf{v} \leq 0 \implies v_z \leq 0$. By inspection of (8.9), $v_z < 0$ contributes to move $\mathbf{s}$ to the origin of the image frame. When $v_z = 0$, the motion of the camera must be perpendicular to the camera's principal axis (i.e. the camera's $z-$axis [13]); and

3. $J_l\mathbf{w} \leq 0 \implies yw_x - xw_y \leq 0$. The first two components of the angular velocity vector must be belong to a hyperplane defined by $\mathbf{s}$.

The three constraints cover the well-known trivial case where the camera performs a pure translation perpendicular to the optical axis. Mathematically, this scenario is expressed as $v_z = 0$, $\sigma^2 = v_x^2 + v_y^2 > 0$ and $\mathbf{w} = \mathbf{0}$. However, as the vehicle moves, the point may leave the finite image frame. Therefore, $\mathbf{v}$ and $\mathbf{w}$ can be such that the observed point remains within a region of interest of the image frame while still observing the required constraints for convergence.

Next, it is discussed a meaningful manner to select values for the gains k_s and k_χ, which are the only magic numbers in the estimator that have to be chosen by the designer.

8.4.1 Gain analysis

For choosing suitable gains (as described in the sequel) for the estimator in (8.12), the convergence rate for $\hat{\chi}$ is analyzed. For that purpose, consider the simplified behavior of the error when $J_q\mathbf{v} = 0$, that is,

$$\begin{cases} \dot{\tilde{\mathbf{s}}} = J_v\mathbf{v}\tilde{\chi} - k_s\tilde{\mathbf{s}} \\ \dot{\tilde{\chi}} = J_l\boldsymbol{w}\tilde{\chi} - k_\chi(J_v\mathbf{v})^T\tilde{s} \end{cases}. \tag{8.26}$$

This simplification, which corresponds to set v_z close to zero, allows us to remove the unknown depth χ from the gain analysis. Let the gain k_s can be chosen arbitrarily large and define $\Psi(\mathbf{s},\mathbf{v}) \overset{\text{def}}{=} J_l\mathbf{w}$ and $\Omega(\mathbf{s},\mathbf{v}) \overset{\text{def}}{=} J_v\mathbf{v}$. The second-order dynamics of the depth estimation error can be described as

$$\ddot{\tilde{\chi}} = \dot{\Psi}\tilde{\chi} + \Psi\dot{\tilde{\chi}} - k_\chi\dot{\Omega}^T\tilde{\mathbf{s}} - k_\chi\Omega^T\dot{\tilde{\mathbf{s}}} \tag{8.27}$$

$$= \dot{\Psi}\tilde{\chi} + \Psi\dot{\tilde{\chi}} - k_\chi\dot{\Omega}^T\tilde{\mathbf{s}} - k_\chi\Omega^T(\Omega\tilde{\chi} - k_s\tilde{\mathbf{s}}) \tag{8.28}$$

$$= (\dot{\Psi} - k_\chi\Omega^T\Omega)\tilde{\chi} + \Psi\dot{\tilde{\chi}} - k_\chi\Omega^T\Omega\tilde{\chi} + k_\chi(k_s\Omega^T - \dot{\Omega}^T)\tilde{\mathbf{s}}. \tag{8.29}$$

Considering that the velocity of the camera varies smoothly and that the coordinates of the points in the normalized image plane are small, it is possible to choose a k_s sufficiently large such that $k_s\|\Omega^T\tilde{\mathbf{s}}\| \gg \|\dot{\Omega}^T\tilde{\mathbf{s}}\|$. A lower bound for k_s can be obtained from the kinodynamics constraints of the camera and the calibration parameters of the camera. The former imposes limits on the velocity of the camera, while the later in the coordinates of the feature being tracked. Now, the previous equation can be simplified

$$\ddot{\tilde{\chi}} = (\dot{\Psi} - k_\chi\Omega^T\Omega)\tilde{\chi} + \Psi\dot{\tilde{\chi}} - k_\chi\Omega^T\Omega\tilde{\chi} + k_s(\Psi\tilde{\chi} - \dot{\tilde{\chi}}). \tag{8.30}$$

Re-arranging the terms of the previous equation, one may re-write it similar to a response of the Mass-Spring-Damper system:

$$\ddot{\tilde{\chi}} + (k_s - \Psi)\dot{\tilde{\chi}} + (k_\chi\Omega^T\Omega - k_s\Psi - \dot{\Psi})\tilde{\chi} = 0, \tag{8.31}$$

where the damping ratio is written as

$$\zeta = \frac{k_s - \Psi}{2\sqrt{k_\chi \Omega^T \Omega - k_s \Psi - \dot{\Psi}}}, \tag{8.32}$$

with natural frequency

$$\omega_n = k_\chi \Omega^T \Omega - k_s \Psi - \dot{\Psi}. \tag{8.33}$$

To obtain a close to critically damping factor (i.e., $\zeta = 1$):

$$k_\chi = \frac{(k_s - \Psi)^2}{4\Omega^T \Omega} + \frac{k_s \Psi}{\Omega^T \Omega} + \frac{\dot{\Psi}}{\Omega^T \Omega}. \tag{8.34}$$

Applying (8.34) in (8.33), we have that $\omega_n \propto (k_s - \Psi)^2$, which is always positive for a system that follows the constraints in Theorem 8.2 - Ψ is non-positive and k_s is positive. From this relation, we draw the following remark.

Remark 8.4. Consider the observer in (8.12) and the inputs as defined in Theorem 8.2. For k_s such that $k_s \|\Omega^T \tilde{\mathbf{s}}\| \gg \|\dot{\Omega}^T \tilde{\mathbf{s}}\|$ and k_χ as in (8.34), the depth estimation error converges to the origin with a close to a critically damped oscillation behaviour.

The results of Theorem 8.2 may be employed in a passive manner because it provides means to validate for some arbitrary control policy whether there are guarantees that depth estimation for any tracked point is actually converging. In the next section, we focus on the active case. The key challenge is how to derive a control policy that complies with the required constraints for convergence of the estimator and a high-level goal. The high-level task investigated here is visual servoing for a single point (Sec. 9.1) and multiple points (Sec. 9.2).

Chapter 9

Active Depth Estimation: Applications

This chapter applies the active depth estimation filter presented in Chapter 8 in more concrete applications where the camera has a limited field of view and conflicting objectives often prevent the actuation that best excites the estimator. More specifically, application for single point depth estimation is discussed in Sec. 9.1, while Sec. 9.2 covers applications for multiple points. Section 9.3 present the results and comparisons with literature methods for both single point and multiple points use-cases.

9.1 Single Point Applications

This section proposes an active depth estimation framework for a single 3D point. The key idea is exploring the theoretical stability guarantees derived in Sec. 8.4, while ensuring that the projected point does not leave the image space. In essence, the control input that respects Theorem 8.2 ensures that the depth estimation globally converges, but there is no guarantees that the feature coordinates remains within a bounded region of the image plane. This is a natural requirement that arises in practical applications due to the finite field of view of a camera. Therefore, first, define the visual servoing error for a single feature as follows:

$$\mathbf{e}(t) = \mathbf{s}(t) - \mathbf{s}_{des}(t), \tag{9.1}$$

where the continuous and smooth signal $\mathbf{s}_{des}(t)$ is the desired position of the feature in the image plane. The signal $\mathbf{s}_{des}$ is chosen such that the feature remains within the field of view of the camera during the depth estimation process.

Assume that the feedback control law $\boldsymbol{\pi}(t, \mathbf{s}, \mathbf{s}_{des})$ drives the tracking error to the origin, i.e., $\dot{\mathbf{s}} = \boldsymbol{\pi}, \forall t \geq t_0 \implies \mathbf{e} \to 0$ as $t \to \infty$. For example, if $\mathbf{s}_{des}$ is constant, then the proportional controller $\boldsymbol{\pi} = -k_p(\mathbf{s} - \mathbf{s}_{des})$, where $k_p \in \mathbb{R}^+$ ensures the desired behaviour.

The dynamic system is given by (8.10), which is repeated here for the sake of completeness:

$$\begin{cases} \dot{\mathbf{s}} &= J_v \mathbf{v} \chi + J_w \mathbf{w} \\ \dot{\chi} &= J_q \mathbf{v} \chi^2 + J_l \mathbf{w} \chi \end{cases}. \tag{9.2}$$

From inspection, $\dot{\mathbf{s}}$ depends on the unknown depth χ and the camera velocities. Since the depth is estimated online, it is only possible to shape the dynamics of $\dot{\mathbf{s}}$ up to an estimation error. That being said, the goal is to design a control law for $(\mathbf{v}, \mathbf{w})$ such that $\dot{\mathbf{s}}(\hat{\chi}, \mathbf{v}, \mathbf{w})$ tracks the signal $\boldsymbol{\pi}(t, \mathbf{s}, \mathbf{s}_{des})$, while:

i. imposing the constraints stated in Theorem 8.2, to assure that the stability property holds;

ii. improving the performance of the estimator, by maximizing σ^2 as defined in (8.25); and

iii. accounting for the kinodynamics constraints of the camera described by $\|\mathbf{v}\| \leq v_{\max}$ and $\|\mathbf{w}\| \leq w_{\max}$, where $v_{\max}$ and $w_{\max}$ are the maximum linear and angular speed of the camera, respectively.

Simultaneously tracking $\boldsymbol{\pi}$ and respecting all the forementioned constraints can lead to an infeasible problem. A workaround is proposed by introducing a scale factor $\lambda_\pi \in [0, 1]$ such that $\dot{\mathbf{s}}(\hat{\chi}, \mathbf{v}, \mathbf{w})$ is required to track the reference $\lambda_\pi \boldsymbol{\pi}$. As the depth converges, tracking the scaled vector $\lambda_\pi \boldsymbol{\pi}$ – rather than minimizing a norm error – ensures that the path of the feature in the image frame follows the assignment specified by $\boldsymbol{\pi}$. This allows us to design a path for the feature that does not visit the origin of the image frame. The problem is formulated next:

$$\begin{aligned} \underset{\mathbf{v}, \mathbf{w}, \lambda_\pi}{\text{maximize}} \quad & \lambda_\pi \\ \text{subject to} \quad & J_v \hat{\chi} \mathbf{v} + J_w \mathbf{w} = \lambda_\pi \boldsymbol{\pi} \\ & 0 \leq \lambda_\pi \leq 1 \\ & \text{constraints (i), (ii), and (iii)} \end{aligned}. \tag{9.3}$$

This problem is addressed in two configurations. The estimation strategy proposed in Section 9.1.1 does not implicitly impose the unknown depth to be constant. In contrast, Sec. 9.1.2 addresses the particular case that requires null depth rate (similar to [24]). Both cases take advantage of the following Theorem:

Theorem 9.1. *Consider the non-convex problem:*

$$\begin{aligned} \underset{\lambda_1,\lambda_2,\mathbf{v}_r}{\textit{maximize}} \quad & \lambda_1 \\ \textit{subject to} \quad & \lambda_1\mathbf{v}_1 + \lambda_2\mathbf{v}_2 = r\mathbf{v}_r \\ & \|\mathbf{v}_r\| = 1 \\ & 0 \leq \lambda_1 \leq 1 \\ & -b \leq \lambda_2 \leq b \end{aligned} \quad , \tag{9.4}$$

where $r, b \in \mathbb{R}^+, \mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^n$, $\|\mathbf{v}_1\| > 0$, *and* $\|\mathbf{v}_2\| = 1$. *The problem is always feasible if* $r \leq b$.

The reader is referred to Appendix A for the proof of Theorem 9.1 and a closed form solution for the problem in (9.4). The solution does not impose restrictions on the feature coordinates, except the origin of the image frame, i.e., $\mathbf{s} = [0, 0]^T$, which is a singularity.

9.1.1 Case: non-constant depth

The first scenario covers the generic case when the unknown depth of the 3D point may vary over time, i.e., $\dot{\chi} \neq 0$, for some $t \geq t_0$. By choosing a control policy for $\mathbf{v}$ and $\mathbf{w}$ that respects $J_q\mathbf{v} = 0$ and $J_w\mathbf{w} \leq 0$, one may take advantage of Theorem 9.1, while still respecting the requirements for global convergence stated in Theorem 8.2. In addition to that, the PE condition in (8.25) simplifies to $\sigma^2 = v_x^2 + v_y^2 = \|v_{(1:2)}\|^2$ and its maximum attainable value is limited by the kinodynamic constraint of the camera, $\sigma^2_{\max} = v_{\max}$. Under this specific scenario, problem in (9.3) can be formulated as

$$\begin{aligned} \underset{\mathbf{v},\mathbf{w},\lambda_\pi}{\text{maximize}} \quad & \lambda_\pi \\ \text{subject to} \quad & \dot{\mathbf{s}}(\hat{\chi}, \mathbf{v}, \mathbf{w}) = \lambda_\pi \boldsymbol{\pi} \\ & 0 < \lambda_\pi \leq 1 \\ & J_q\mathbf{v} = 0,\ J_l\mathbf{w} \leq 0 \\ & \|\mathbf{v}\| = v_{\max},\ \|w\| \leq w_{\max} \end{aligned} \tag{9.5}$$

and solved with the following proposition:

Proposition 1. Let the camera control input be

$$\mathbf{v} = v_{\max} \begin{bmatrix} \mathbf{v}_r \\ 0 \end{bmatrix} \text{ and } \mathbf{w} = \begin{bmatrix} S\boldsymbol{\lambda}_s / \|\mathbf{s}\| \\ 0 \end{bmatrix}. \tag{9.6}$$

and S and $\boldsymbol{\lambda}_s$ be defined as follow:

$$\begin{cases} S = \begin{bmatrix} -\frac{\mathbf{s}_\perp}{\|\mathbf{s}_\perp\|} & \frac{\mathbf{s}}{\|\mathbf{s}\|} \end{bmatrix} \\ \boldsymbol{\lambda}_s = [\lambda_{s_\perp}, \lambda_s]^T \end{cases}, \tag{9.7}$$

where $\lambda_{s_\perp} \in \mathbb{R}^+$, $\lambda_s \in \mathbb{R}$, and $\mathbf{s}_\perp = [-y, x]^T$ is a vector perpendicular to $\mathbf{s}$. In particular, take $\boldsymbol{\lambda}_s$ to be

$$\boldsymbol{\lambda}_s = \begin{cases} \lambda_w \|\mathbf{s}\| (J_{\bar{w}} S)^{-1} \boldsymbol{\pi} / \|\boldsymbol{\pi}\| & \text{if } (\|\boldsymbol{\pi}\| - \hat{\chi} v_{\max}) \mathbf{s}^T \boldsymbol{\pi} < 0 \\ \lambda_w \|\mathbf{s}\| [0, 1]^T & \text{otherwise} \end{cases}, \tag{9.8}$$

where $J_{\bar{w}}$ is defined as

$$J_{\bar{w}} = \begin{bmatrix} xy & -(1+x^2) \\ 1+y^2 & -xy \end{bmatrix}. \tag{9.9}$$

A sub-optimal solution[1] for the problem in (9.5) can be obtained by casting it in the form of the problem in (9.4), where the input variables are written as

$$\begin{cases} \mathbf{v}_1 = -\boldsymbol{\pi} \\ \mathbf{v}_2 = \begin{cases} \boldsymbol{\pi} / \|\boldsymbol{\pi}\| & \text{if } (\|\boldsymbol{\pi}\| - \hat{\chi} v_{\max}) \mathbf{s}^T \boldsymbol{\pi} < 0 \\ \mathbf{s}_\perp / \|\mathbf{s}_\perp\| & \text{otherwise} \end{cases} \\ r = \hat{\chi} v_{\max}, b = w_{\max} \end{cases} \tag{9.10}$$

and the outputs mapped into

$$\begin{cases} \lambda_\pi = \lambda_1^*, \lambda_w = \lambda_2^* \\ \mathbf{v}_r = \mathbf{v}_r^* \end{cases}. \tag{9.11}$$

Proof. First, we show that the control inputs are described as in (9.6). The constraint $J_q \mathbf{v} = 0$ implies that $v_z = 0$. Combining with $\|\mathbf{v}\| = v_{\max}$, the linear velocity vector can be stated as $\mathbf{v} = v_{\max}[\mathbf{v}_r^T, 0]^T$, where $\mathbf{v}_r \in \mathbb{R}^2$ is a unit vector. For the angular velocity, re-write the

[1] The sub-optimality of the solution is discussed in Remark 9.2, in the end of this section.

constraint $J_l\mathbf{w} \leq 0$ using the slack variable $\lambda_{s_\perp}$, such that

$$J_l\mathbf{w} \leq 0 \implies \begin{cases} J_l\mathbf{w} = \lambda_{s_\perp} \\ \lambda_{s_\perp} \leq 0 \end{cases}. \tag{9.12}$$

From $J_l\mathbf{w} = \lambda_{s_\perp}$ one concludes that $w_y = (y/x)w_x - (1/x)\lambda_{s_\perp}$. Applying this result into $J_w\mathbf{w}$:

$$J_w\mathbf{w} = J_w \begin{bmatrix} w_x \\ (y/x)w_x - (1/x)\lambda_{s_\perp} \\ w_z \end{bmatrix} \tag{9.13}$$

$$= \begin{bmatrix} -y/x & y \\ 1 & -x \end{bmatrix} \begin{bmatrix} w_x \\ w_z \end{bmatrix} + \begin{bmatrix} (1/x + x) \\ y \end{bmatrix} \lambda_{s_\perp}. \tag{9.14}$$

The column space of the first matrix on the right hand side of the previous equation can be generated assuming $w_z = 0$. Thus, the following equivalence holds:

$$J_l\mathbf{w} = \lambda_{s_\perp} \implies -\mathbf{s}_\perp^T\mathbf{w}_{(1:2)} = \lambda_{s_\perp}, \tag{9.15}$$

where $w_z = 0$ and $\mathbf{s}_\perp = [-y, x]^T$. In turn, under these assumptions, any feasible angular velocity can be described as

$$\mathbf{w}_{(1:2)} = -\frac{\mathbf{s}_\perp}{\|\mathbf{s}_\perp\|^2}\lambda_{s_\perp} + \frac{\mathbf{s}}{\|\mathbf{s}\|^2}\lambda_s \tag{9.16}$$

$$= \frac{1}{\|\mathbf{s}\|}\begin{bmatrix} -\frac{\mathbf{s}_\perp}{\|\mathbf{s}_\perp\|} & \frac{\mathbf{s}}{\|\mathbf{s}\|} \end{bmatrix} \begin{bmatrix} \lambda_{s_\perp} \\ \lambda_s \end{bmatrix} \tag{9.17}$$

$$= \frac{1}{\|\mathbf{s}\|} S\boldsymbol{\lambda}_s, \tag{9.18}$$

where S and $\boldsymbol{\lambda}_s$ are as defined in (9.7). With this setup the kinodynamics constraint $\|\mathbf{w}\| \leq w_{\max}$ is equivalent to $\|\boldsymbol{\lambda}_s\| \leq \|\mathbf{s}\|w_{\max}$:

$$\|\mathbf{w}\| = \|\mathbf{w}_{(1:2)}\| = \frac{1}{\|\mathbf{s}\|}\sqrt{\boldsymbol{\lambda}_s^T S^T S \boldsymbol{\lambda}_s} \tag{9.19}$$

$$= \frac{\|\boldsymbol{\lambda}_s\|}{\|\mathbf{s}\|} \leq w_{\max}. \tag{9.20}$$

This concludes the proof of (9.6) and (9.7).

Now, applying the control inputs into the first constraint of (9.5) and re-organizing the terms yields

$$\lambda_\pi(-\boldsymbol{\pi}) + J_w \begin{bmatrix} S\boldsymbol{\lambda}_s/\|\mathbf{s}\| \\ 0 \end{bmatrix} = -\hat{\chi} v_{\max} J_v \begin{bmatrix} \mathbf{v}_r \\ 0 \end{bmatrix} \tag{9.21}$$

$$\lambda_\pi(-\boldsymbol{\pi}) + \frac{1}{\|\mathbf{s}\|} J_{\bar{w}} S\boldsymbol{\lambda}_s = \hat{\chi} v_{\max} \mathbf{v}_r. \tag{9.22}$$

Let $\boldsymbol{\nu} = (1/\|\mathbf{s}\|) J_{\bar{w}} S\boldsymbol{\lambda}_s$ and notice that if $\|\boldsymbol{\pi}\| > \hat{\chi} v_{\max}$, $\boldsymbol{\lambda}_s$ must be such that $\boldsymbol{\pi}^T\boldsymbol{\nu} > 0$. On the contrary, if $\|\boldsymbol{\pi}\| < \hat{\chi} v_{\max}$, then one has to ensure $(-\boldsymbol{\pi})^T\boldsymbol{\nu} > 0$. Maximizing the dot product in both cases requires that $\boldsymbol{\nu} \| \boldsymbol{\pi}$ and, consequentially,

$$\boldsymbol{\lambda}_s \propto (J_{\bar{w}} S)^{-1} \boldsymbol{\pi}. \tag{9.23}$$

The matrix S is orthogonal and, therefore, full rank. The matrix $J_{\bar{w}}$ is also full rank: $\det(J_{\bar{w}}) = 1 + x^2 + y^2 \neq 0$. From the Sylvester rank inequality, we have:

$$\text{rank}(S) + \text{rank}(J_{\bar{w}}) - 2 \leq \text{rank}(J_{\bar{w}} S). \tag{9.24}$$

Since both S and $J_{\bar{w}}$ are 2×2 full rank matrices, one concludes that their product is always full rank (and invertible).

For feasibility, the first component of $\boldsymbol{\lambda}_s$ – corresponding to $\lambda_{s_\perp}$ – must be non-positive. Solving the right hand side of (9.23), $\lambda_{s_\perp}$ can be described up to a positive scalar as

$$\lambda_{s,\perp} \propto \begin{cases} \mathbf{s}^T\boldsymbol{\pi} & \text{if } \|\boldsymbol{\pi}\| > \hat{\chi} v_{\max} \\ -\mathbf{s}^T\boldsymbol{\pi} & \text{if } \|\boldsymbol{\pi}\| \leq \hat{\chi} v_{\max} \end{cases}. \tag{9.25}$$

If $\lambda_{s,\perp}$ is positive in either cases, it means that $\lambda_{s_\perp} = 0$ is the largest feasible value that maximizes the projection of $\boldsymbol{\nu}$ into $\boldsymbol{\pi}$ or $(-\boldsymbol{\pi})$. Putting the pieces together in a compact notation:

$$\boldsymbol{\lambda}_s = \begin{cases} \lambda_w \|\mathbf{s}\| (J_{\bar{w}} S)^{-1} \frac{\boldsymbol{\pi}}{\|\boldsymbol{\pi}\|} & \text{if } (\|\boldsymbol{\pi}\| - \hat{\chi} v_{\max}) \mathbf{s}^T\boldsymbol{\pi} < 0 \\ \lambda_w \|\mathbf{s}\| [0,1]^T & \text{otherwise} \end{cases}, \tag{9.26}$$

where $\lambda_w \in \mathbb{R}$. For the maximum feasible value of λ_w, one may compute the norm of the previous equation and compare it with (9.20). When $(\|\boldsymbol{\pi}\| - \hat{\chi} v_{\max}) \mathbf{s}^T\boldsymbol{\pi} > 0$, we have

$$\|\boldsymbol{\lambda}_s\| = \frac{\|\lambda_w\| \|\mathbf{s}\|}{\|\boldsymbol{\pi}\|} \|(J_{\bar{w}} S)^{-1} \boldsymbol{\pi}\| \leq \|\mathbf{s}\| w_{\max}. \tag{9.27}$$

The singular values of $(J_{\bar{w}}S)^{-1}$ are 1 and $1/(1+x^2+y^2)$. Since the maximum singular value is the unit, the upper bound $\|(J_{\bar{w}}S)^{-1}\boldsymbol{\pi}\| \leq \|\boldsymbol{\pi}\|$ holds. Applying it in (9.20), yields

$$\|\boldsymbol{\lambda}_s\| \leq \|\lambda_w\|\|\mathbf{s}\| \leq \|\mathbf{s}\|w_{\max} \tag{9.28}$$

$$\|\lambda_w\| \leq w_{\max}. \tag{9.29}$$

The same bound is obtained when $\boldsymbol{\lambda}_s = \lambda_w\|\mathbf{s}\|[0,1]^T$ in (9.26):

$$\|\lambda_w\|\mathbf{s}\|[0,1]^T\| \leq \|\mathbf{s}\|w_{\max} \Rightarrow \|\lambda_w\| \leq w_{\max} \tag{9.30}$$

Finally, substituting (9.26) in (9.22):

$$\hat{\chi}v_{\max}\mathbf{v}_r = \begin{cases} \lambda_\pi(-\boldsymbol{\pi}) + \lambda_w\frac{\boldsymbol{\pi}}{\|\boldsymbol{\pi}\|} & \text{if } (\|\boldsymbol{\pi}\| - \hat{\chi}v_{\max})\mathbf{s}^T\boldsymbol{\pi} > 0 \\ \lambda_\pi(-\boldsymbol{\pi}) + \lambda_w\frac{\mathbf{s}_\perp}{\|\mathbf{s}_\perp\|} & \text{otherwise.} \end{cases} \tag{9.31}$$

□ □

Remark 9.2. The sub-optimality of the proposed solution comes from the fact that the solution consists in projecting $\boldsymbol{\lambda}_s$ into $\boldsymbol{\pi}$ when $(\|\boldsymbol{\pi}\| - \hat{\chi}v_{\max})\mathbf{s}^T\boldsymbol{\pi} < 0$. The projection is done via the mapping $J_{\bar{w}}S$. The singular values of $J_{\bar{w}}S$ are 1 and $1+x^2+y^2$. Therefore, if $\mathbf{s} \neq [0,0]^T$, there can exist a $\boldsymbol{\lambda}_s$ that is not projected into $\boldsymbol{\pi}$, but the shear transformation performed by $J_{\bar{w}}S$ allows for a higher value of λ_π. Since in practical applications $1+x^2+y^2 \approx 1$, the solution obtained is not far from the optimal solution. The main advantage in the proposed approach is that it is possible to compute a direction for $\boldsymbol{\lambda}_s$ in a closed-form.

9.1.2 Case: $\mathbf{s} \neq \mathbf{0}$ and $\dot{\chi} = 0, \forall t \geq t_0$

Now, consider the specific scenario where the depth must be kept constant throughout the entire estimation process, that is, $\dot{\chi} = 0, \forall t \geq t_0$. This is the case for the estimator in [24] and it is shown here for comparison. However, note that this imposes motion constraints that are not required for non-constant depth. For an unknown χ in (8.10), setting $J_q\mathbf{v} = 0$ and $J_l\mathbf{w} = 0$ guarantees that $\dot{\chi} = 0$. Both aforementioned constraints are in accordance with Theorem 8.2. The problem, which is stated next:

$$\begin{aligned} \underset{\mathbf{v},\mathbf{w},\lambda_\pi}{\text{maximize}} \quad & \lambda_\pi \\ \text{subject to} \quad & \dot{\mathbf{s}}(\hat{\chi},\mathbf{v},\mathbf{w}) = \lambda_\pi \boldsymbol{\pi} \\ & 0 \leq \lambda_\pi \leq 1 \\ & J_q\mathbf{v} = 0,\, J_l\mathbf{w} = 0 \\ & \|\mathbf{v}\| = v_{\max},\, \|w\| \leq w_{\max} \end{aligned} \quad , \tag{9.32}$$

is a particular case of problem (9.5). According to the following corollary, an optimal solution can be obtained using Theorem 9.1.

Corollary 9.3. *Let the camera control input be described as*

$$\mathbf{v} = v_{\max}\begin{bmatrix}\mathbf{v}_r \\ 0\end{bmatrix} \text{ and } \mathbf{w} = \lambda_w \begin{bmatrix}\mathbf{s}/\|\mathbf{s}\| \\ 0\end{bmatrix}. \tag{9.33}$$

Then, the problem in (9.32) *is equivalent to the problem in* (9.4)*, where*

$$\begin{cases} \mathbf{v}_1 = -\boldsymbol{\pi},\, \mathbf{v}_2 = \mathbf{s}_\perp/\|\mathbf{s}_\perp\| \\ r = \hat{\chi} v_{\max},\, b = w_{\max} \end{cases} ; \tag{9.34}$$

and the outputs are mapped as:

$$\begin{cases} \lambda_\pi = \lambda_1^*,\, \lambda_w = \lambda_2^* \\ \mathbf{v}_r = \mathbf{v}_r^* \end{cases} . \tag{9.35}$$

The proof is similar to the one presented in Sec. 9.1.1 by imposing $\lambda_{s_\perp} = 0$, that is, no slackness. In this case, the solution is optimal because the shear mapping is not involved.

9.2 Multiple Points Applications

It is often the case in vision applications that one wants to estimate the depth of multiple points concurrently, examples include visual servoing and mapping. Motivated by this, this section addresses the coupled depth estimation and visual servoing problem for multiple points. The proposed active visual servoing method considers an estimate of the uncertainty of depth of the points. In essence, when the uncertainty is high, the constraints of Theorem 8.2 must hold. As the uncertainty decreases, the constraints are

relaxed in favor of the visual servoing goal. As a result, depth convergence is favored in the initial phase, and after that focus is given to the visual servoing task.

The notation and formulation of the multiple points visual servoing problem is addressed in Sec. 9.2.1. Then, Sec. 9.2.2 shows a metric for measuring the uncertainty associated with the estimated depth. Finally, Sec. 9.2.3 presents the proposed method for active visual servoing, which tackles the problem by considering a trade-off between convergence of the estimator and the visual servoing goal.

9.2.1 Formulation

For the sake of simplicity, consider with a slight abuse of notation $\mathbf{s} = [\mathbf{s}_1^T, \ldots, \mathbf{s}_n^T]^T \in \mathbb{R}^{2n}$ a vector of n image features and $\mathbf{s}_{des} = [\mathbf{s}_{1,des}^T, \ldots, \mathbf{s}_{n,des}^T]^T \in \mathbb{R}^{2n}$ a constant vector that describes the desired coordinates of the features. Borrowing some of the notation employed in Sec. 9.1, the visual servoing error of a vector of features and its dynamics are defined as

$$\mathbf{e} = \mathbf{s} - \mathbf{s}_{des}. \tag{9.36}$$

Computing the time derivative of (9.36) and applying (8.10), the dynamic of the visual servoing error can be described as

$$\dot{\mathbf{e}} = \dot{\mathbf{s}} - \dot{\mathbf{s}}_{des} = \begin{bmatrix} J_v(\mathbf{s}_1)\mathbf{v}\chi_1 + J_w(\mathbf{s}_1)\mathbf{w} \\ \vdots \\ J_v(\mathbf{s}_n)\mathbf{v}\chi_n + J_w(\mathbf{s}_n)\mathbf{w} \end{bmatrix} - \dot{\mathbf{s}}_{des} \tag{9.37}$$

$$= \begin{bmatrix} J_v(\mathbf{s}_1)\chi_1 \,,\, J_w(\mathbf{s}_1) \\ \vdots \\ J_v(\mathbf{s}_n)\chi_n \,,\, J_w(\mathbf{s}_n) \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} - \dot{\mathbf{s}}_{des} \tag{9.38}$$

$$= L_s(\mathbf{s}, \boldsymbol{\chi})\mathbf{u} - \dot{\mathbf{s}}_{des}, \tag{9.39}$$

where $\boldsymbol{\chi} = [\chi_1, \ldots, \chi_n]^T \in \mathbb{R}^n$, $\mathbf{u} = [\mathbf{v}^T, \mathbf{w}^T] \in \mathbb{R}^6$, and $L_s \in \mathbb{R}^{2n \times 6}$ is known as the interaction matrix . When $\dot{\mathbf{s}}_{des} = \mathbf{0}$ (typical case), the visual servoing error is forced to converge exponentially fast to the origin by modelling the error dynamics using the following control law:

$$\mathbf{u}^* = -\lambda_e L_s^{\dagger} \mathbf{e} \tag{9.40}$$

where $\lambda_e \in \mathbb{R}^+$ and $L_s^{\dagger} = (L_s^T L_s)^{-1} L_s^T$. This requires L_s to be full rank, which means that at least three features must be fed into the visual servo control loop.

In the problem discussed here, the structure of the world is unknown. In fact, this is the case in most realistic scenarios. Thus, the controller does not have access to L_s and resorts to an estimation of the interaction matrix given by $\hat{L}_s(\mathbf{s}, \hat{\boldsymbol{\chi}})$. The coupling between depth estimation and visual servoing arises from the fact that the controller attempts to drive the visual servoing error to the origin using the depth estimation generated by the observers. Therefore, if the depth does not converge, most likely, neither does the visual servoing error. In fact, as discussed in [90], the control law in (9.40) is only guaranteed to converge when the estimation error of the interaction matrix ($\hat{L}_s$) is small. An exception is when the depth at $\mathbf{s}_{des}$ is known a priori.

Next, we briefly introduce a method for measuring the uncertainty associated to the unknown depth. This metric is later employed in our method for imposing or relaxing the constraints that guarantee asymptotically stability of the estimator.

9.2.2 Measuring uncertainty

The uncertainty of the unmeasurable variable χ_i, denoted as $E_i(t)$, can be described by the following metric:

$$E_i(t) = \frac{1}{T}\int_{t-T}^{t} \tilde{\mathbf{s}}_i(\tau)^T \tilde{\mathbf{s}}_i(\tau) d\tau. \tag{9.41}$$

This is valid if during the time interval of length T the associated PE condition is non-zero and, therefore, the dynamics of $\tilde{\mathbf{s}}_i$ and $\tilde{\chi}_i$ are tightly coupled (see (8.13) and Remark 8.3). The interested reader is referred to [26], Appendix C.

9.2.3 Proposed method (Active)

For global depth convergence, each point being tracked must fulfill the properties stated in Theorem 8.2, while the camera concurrently performs the visual servoing task. This is potentially too restrictive. For instance, consider the case shown in Fig. 9.1(a) that illustrates $J_l(\mathbf{s}_i)\mathbf{w} \leq 0$ for $n = 3$ points. Each constraint defines a halfspace, represented in different colors. The admissible $\mathbf{w}_{(1:2)}$ lies in the intersection of the three constraints, which is a polyhedron (painted in a white wave pattern). As the number of points increase, the size of the polyhedron decreases and ultimately becomes a single point - the origin. In that case, only a specific set of problems which require no camera rotation can be accomplished. The strategy devised splits the visual servoing in three phases (each point may be in a particular phase of the task):

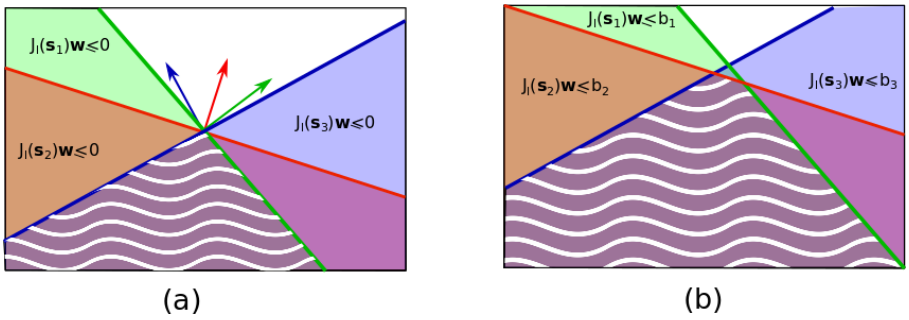


FIGURE 9.1: Constraint $J_l(\mathbf{s}_i)\mathbf{w} \leq 0$ (a) and its relaxed version (b) applied in a multiple points scenario. Each constraint is represented using a different color. In the shaded region, the corresponding constraint is satisfied. In the area painted in a white wave pattern, all the constraints are satisfied (intersection of the halfspaces).

- **Phase 1**: the convergence of the estimator is prioritized over the visual servoing goal. This is achieved by choosing a control input that enforces the constraints of Theorem 8.2.

- **Phase 2**: the more the measured depth uncertainty decreases, the more the constraints required for convergence of the estimator are relaxed (up to a threshold).

- **Phase 3**: visual servoing becomes the main goal, while the constraints for convergence are still imposed but relaxed according to a desired threshold.

The key idea is to remain in Phase 1 (constraints holds) until the 3D points has been properly estimated. The reason for that is that in Phase 2 the guarantees of convergence stated in the last two constraints of Phase 3 are relaxed. The relaxation for a specific point considers the uncertainty associated to its depth estimation. For that, let the constraints be re-stated as

$$\begin{cases} J_l(\mathbf{s}_i)\mathbf{w} \leq b_i,\ i = 1, \ldots, n \\ J_q\mathbf{v} \leq c \end{cases} . \tag{9.42}$$

where $b_i \in R$ and $c \in R$ will be soon introduced. Figure 9.1 shows that for some positive b_i and c this increases the set of feasible $\mathbf{w}$.

Taking into consideration the maximum angular velocity of the camera, define $b_{i,\max} = w_{\max} \max(|x_i|, |y_i|)$. Then, the coefficient b_i is described as follows

$$b_i = \begin{cases} 0, \text{ if } E_i \geq E_{ub} \\ b_{i,\max}\frac{(E_{ub}-E_i)}{(E_{ub}-E_{ub})}, \text{ if } E_{lb} \leq E_i \leq E_{ub} \\ b_{i,\max}, \text{ otherwise }, \end{cases} \tag{9.43}$$

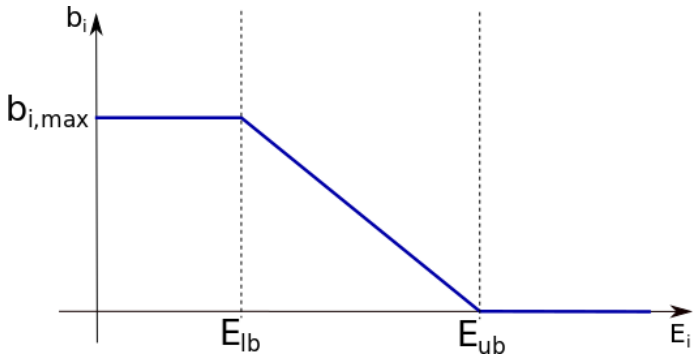


FIGURE 9.2: Relaxation of constraint $J_l(\mathbf{s}_i)\mathbf{w} \leq 0$ using the uncertainty measurement E_i.

where E_i is the measurement of the depth estimation uncertainty of the i-th feature given by (9.41) and $E_{ub}, E_{lb} \in \mathbb{R}^+$ defines an upper and lower bound of the uncertainty, respectively. A graphical interpretation is shown in Fig. 9.2. When the uncertainty is above the threshold E_{ub}, the constraint stated in Theorem 8.2 must be respected. As the uncertainty decreases, the constraints are partially relaxed. If the uncertainty drops below E_{lb}, the constraint is fully relaxed. The coefficient c is defined in a similar fashion:

$$c = \min(c_i), i = 1, \ldots, n; \tag{9.44}$$

$$c_i = \begin{cases} 0, \text{ if } E_i \geq E_{ub} \\ v_{\max}\frac{(E_{ub}-E_i)}{(E_{ub}-E_{ub})}, \text{ if } E_{lb} \leq E_i \leq E_{ub} \\ v_{\max}, \text{ otherwise .} \end{cases} \tag{9.45}$$

Finally, the coupled depth estimation and visual servoing problem are formulated as a convex optimization problem:

$$\begin{aligned} \underset{\mathbf{v},\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}\|\lambda_e \mathbf{e} + \hat{L}_s \mathbf{u}\|^2 \\ \text{subject to} \quad & J_l(\mathbf{s}_i)\mathbf{w} \leq b_i, \; i = 1, \ldots, n \\ & J_q \mathbf{v} \leq c \\ & \|\mathbf{v}\| \leq v_{\max} \\ & \|\mathbf{w}\| \leq w_{\max} \end{aligned} \tag{9.46}$$

where $v_{\max}$ and $w_{\max}$ are the maximum linear and angular velocity of the camera, respectively. If $\min\{\hat{\chi}_1, \ldots, \hat{\chi}_n\} < 0$, the second constraint must be replaced by $v_z = 0$. We highlight that such a situation should be avoided in a coding level, as $\chi_i = 0$ leads to a

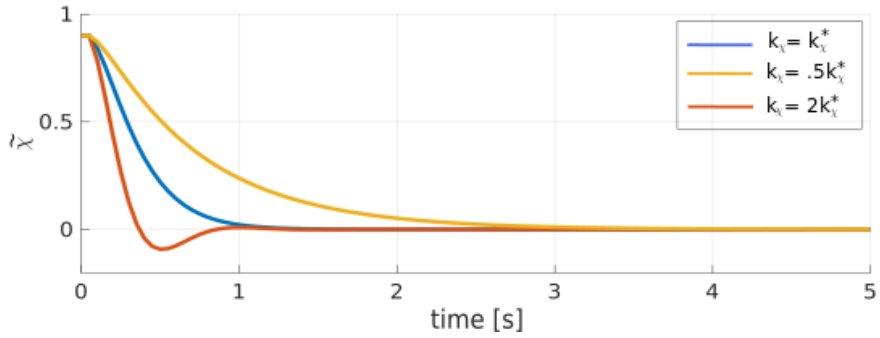


FIGURE 9.3: Evaluation of critically damped gain k_χ^* given by (8.34). The depth estimation results shown in this figure were obtained using the strategy described in Sec. 9.1.1.

singularity and $\chi_i < 0$ does not have a valid physical interpretation. The problem can be solved at frame rate speed using a numerical solver such as [91, 92].

9.3 Experiments

The theoretical results derived in this work are validated using a numerical simulator. Simulations run at 20 Hz with the following fixed parameters: $v_{\max} = 0.1$ m/s, $w_{\max} = 0.15$ rad/s, and $k_s = 10$. The latter is the estimator gain in (8.12). For choosing the gain k_χ, we first analyze the approximation discussed in Sec. 8.4.1 for the critically damped gain k_χ^*. In order for that, the strategy in Sec. 9.1.1 is evaluated using different values of k_χ. The results are shown in Fig. 9.3. Doubling the critical gain leads to overshooting. In contrast, the system response is considerable slower due to overdamping when decreasing k_χ^* by half. These results confirm that the approximation for computing the critically damped gain are valid. Thus, remaining simulations presented here employed the fixed value $k_\chi = k_\chi^*$. In particular, for the considered setup: $k_\chi = 2500$ and $\sigma_{\max}^2 = 0.01$ (maximum value for the PE condition).

9.3.1 Single point

In this section, three single points depth estimation strategies that ensure global convergence are compared:

- *Spica et al. 14*: Presented in [23, 86], for global convergence it requires the projection of the tracked 3D point to lie in the origin of the image plane and its corresponding depth to be constant, i.e., $\mathbf{s}_{des} = [0, 0]^T$ and $\dot{\chi} = 0$.;

- Sec. 9.1.1: The method proposed here ensures global convergence for any feature that does not lie in the origin of the image plane and does not impose its corresponding depth to be constant.

- Sec. 9.1.2: This is a particular case of the previous method. However, similar to *Spica et al. 14*, it keeps the unknown depth constant throughout the trajectory of the camera.

The method named here as *Spica et al. 14* requires $\mathbf{s}_{des} = [0,0]^T$, while the strategies proposed in this work require $\mathbf{s}_{des} \neq [0,0]^T$. Aiming at a fair comparison, the initial visual servoing error and the inverse depth estimation error are the same in the three cases. The simulations discussed here have the following initial configuration: $\|\mathbf{e}(t_0)\| = 0.2$ m and $\tilde{\chi}(t_0) = 0.9$ m^{-1} (with $\chi(t_0) = 1$ m^{-1} and $\hat{\chi}(t_0) = 0.1$ m^{-1}). The results are show in Fig. 9.4, where the literature method is represented in a red continuous line, the strategy in Sec. 9.1.1 in a dashed green line, and the method in Sec. 9.1.2 in a continuous blue line.

The behaviour of the depth estimation error is almost the same for the three methods - see Fig. 9.4a. In fact, the three strategies continuously hold the PE condition, given by σ^2, at its maximum value (Fig. 9.4c). The visual servoing error converges slower for the method described in Sec. 9.1.2. This is because the constraint $J_l\mathbf{w} = 0$ imposes severe limitations on the the angular velocity vector. *Spica et al. 14* guarantees asymptotically stability by driving the feature to the origin of the image frame, while the strategies proposed here ensure that the constraints described in Theorem 8.2 hold throughout the entire estimation process. In particular, the constraint associated to $J_l\mathbf{w}$ can be seen in Fig. 9.4d. For the method in Sec. 9.1.1, $J_l\mathbf{w}$ is smaller or equal to zero. For the method in Sec. 9.1.2, the constraint is always zero.

For the same scenario, Fig. 9.5 shows the ground truth and the depth estimation using the method in Sec. 9.1.1. In contrast to other continuous estimation strategies presented in the literature (including [23]), the method proposed in Sec. 9.1.1 ensures global convergence even thought the depth of the point with respect to the camera is not constant.

In our formulation, the desired feature coordinate $\mathbf{s}_{des}$ can be time-varying. Figure 9.6 shows a scenario where the goal is to have the projection of the feature moving in a circular pattern. More specifically, we define $\mathbf{s}_{des} = 0.1[\cos(2\pi/10t), \sin(2\pi/10t)]^T$. As shown in Fig. 9.6(a), the speed of convergence of the depth estimation error does not change when compared to the previous case (constant $\mathbf{s}_{des}$). Finally, Fig. 9.6(b) and (c) show that

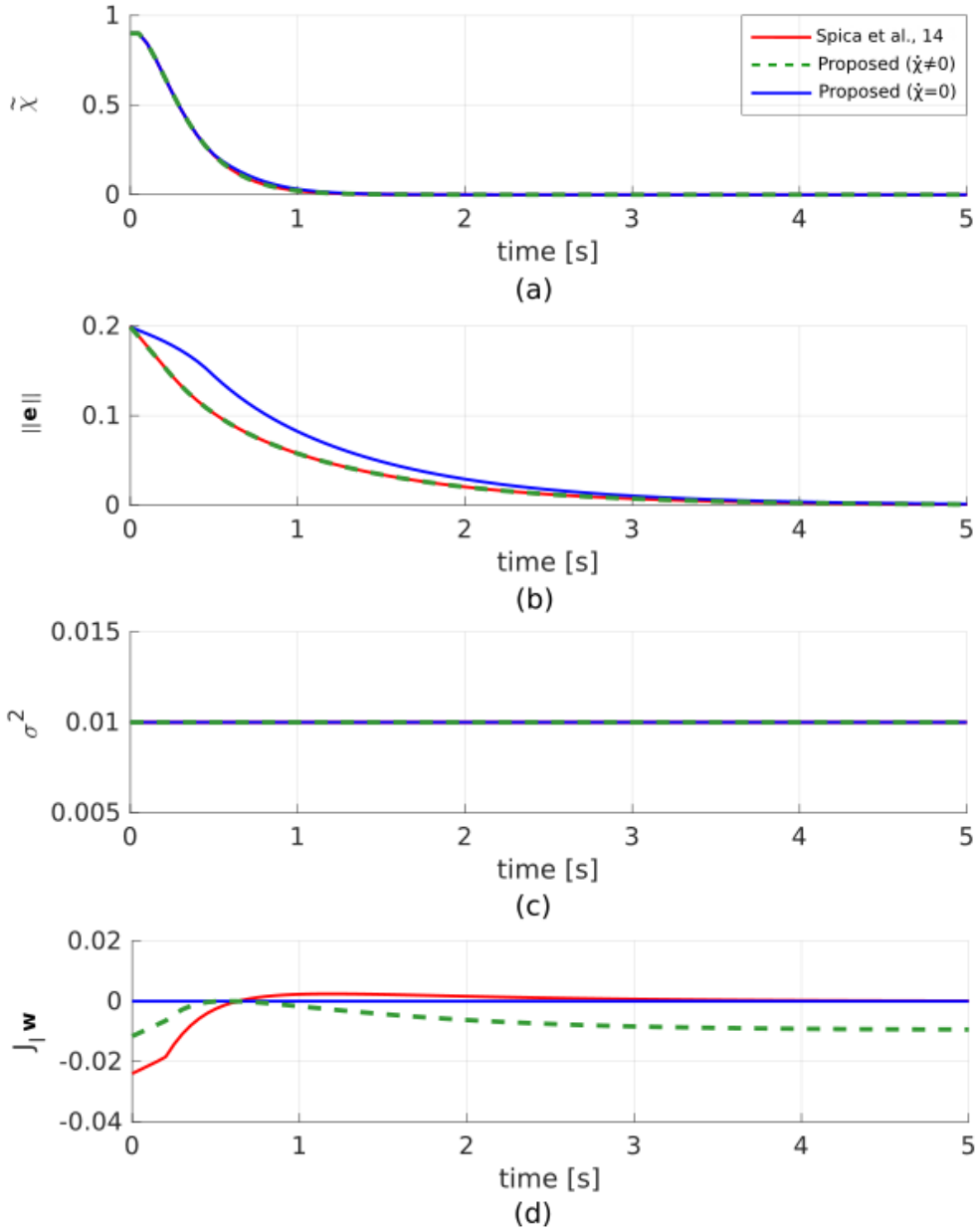


FIGURE 9.4: Comparison of the estimation strategies described in [23] (*Spica et al. 14*), Sec. 9.1.1 ($\dot{\chi} \neq 0$ relaxed), and Sec. 9.1.2 ($\dot{\chi} = 0$) . The initial inverse depth estimation error is $\tilde{\chi} = 0.9$ m^{-1} and the initial visual servoing error is $\|\mathbf{e}\| = 0.2$ m. From top to bottom, the results of (a) the inverse depth estimation error, (b) the visual servoing error, (c) persistence of excitation measurement σ^2, and (d) constraint $J_l\mathbf{w}$ described in Theorem 8.2.

while the depth estimation converges, the proposed control law is able to follow the time-varying signal $\mathbf{s}_{des}$.

In summary, the results for single point applications show that our method is able to guarantee global convergence without moving the feature to the origin of the image plane and the corresponding feature depth does not necessarily need to be constant throughout

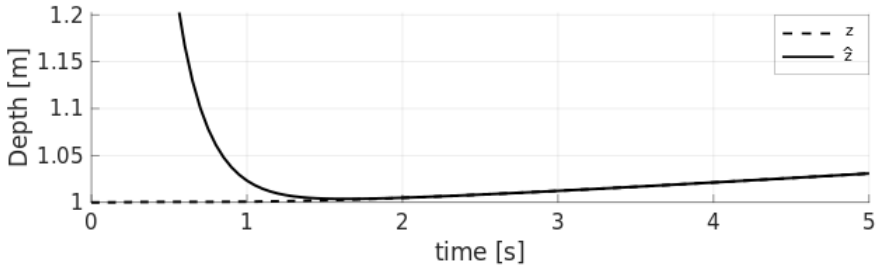


FIGURE 9.5: True depth ($z = 1/\chi$) and its estimation ($\hat{z} = 1/\hat{\chi}$) using the strategy described in Sec. 9.1.1 and the same setup as in Fig. 9.4

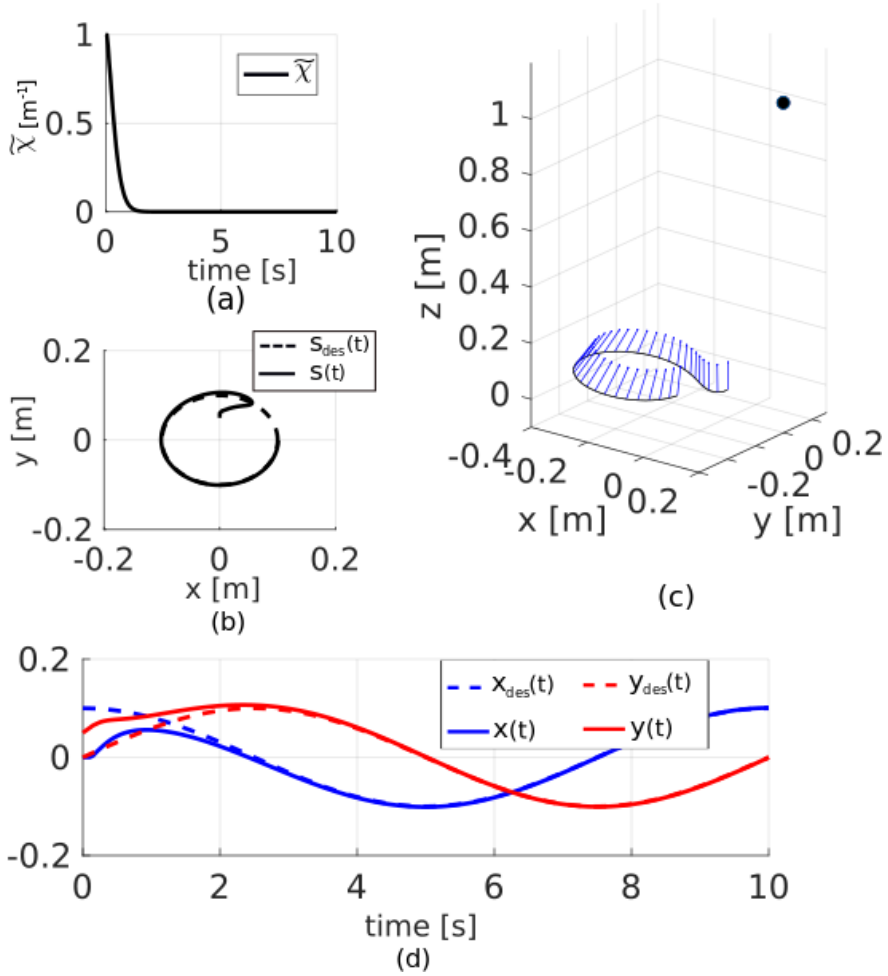


FIGURE 9.6: Assessing the performance of the proposed depth estimation framework when the desired feature coordinates ($\mathbf{s}_{des}(t)$) is time-varying. (a) shows the depth estimation error, (b) shows the desired and the current projection of the 3D point in the image plane, (c) illustrates the trajectory of the camera in a black line, the z–axis in a blue arrow, and the 3D point in black, and (d) the two previous signals over time per axis.

the estimation process. These new properties do not compromise the speed of convergence of the estimator when compared to other methods in the literature, namely [23, 86].

9.3.2 Multiple points

In this section we discuss the results for the concurrently multiple points depth estimation and visual servoing problem. The evaluation is carried out through an statistical analysis by simulating both the proposed (Sec. 9.2.3) and a passive method in several scenarios, which is briefly introduced next.

Baseline (passive method)

The passive approach for visual servoing is inspired in [90] using the kinodynamics constraints, that is, the solution for the following problem:

$$\begin{aligned} \underset{\mathbf{v},\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}\|\lambda_e \mathbf{e} + \hat{L}_s \mathbf{u}\|^2 \\ \text{subject to} \quad & \|\mathbf{v}\| \leq v_{\max} \\ & \|\mathbf{w}\| \leq w_{\max} \end{aligned} \quad . \tag{9.47}$$

Notice that this is similar to our formulation without the constraints inherited from Theorem 8.2 and their respective relaxations. While it would be interesting to compare our method with [26], their code is not public available. Their method aims at maximizing the persistency of excitation condition, which yields results better than the passive method. However, in contrast to our method, throughout the visual servoing task the guarantees of convergence of the estimator are not considered.

Experiments

- Initial pose of the camera $\mathbf{r}(t_0)$: the camera always starts at the origin of the world at $t = t_0$;
- Final pose of the camera $\mathbf{r}_{des}(T)$: A desired final pose is randomly selected using an uniform distribution. The translation is within 0.2 m and 1 m from the initial position and the camera is rotated up to 20^o in each axis. The desired final pose of the camera is employed for projecting the 3D points in the image frame, obtaining $\mathbf{s}_{des}$;
- Number of points (n): the number of tracked 3D points varies from 4 to 9;
- 3D points $\mathbf{q}_1, \ldots, \mathbf{q}_n$: Four points are fixed, mimicking the corners of a square tag of side length 0.2 m. These points are placed 1.5 m away from the $z-$axis of the

world frame. When $n > 4$, the remaining points are randomly generated. The x and y coordinates of the $n-4$ points are within the corresponding coordinates of the virtual square tag, while their depth is 1 m to 2 m away from the z-axis of the world frame;

- Visual servoing goal $\mathbf{s}_{des}$: The vector $\mathbf{s}_{des}$ is computed using the randomly generated final pose of the camera and 3D points; and
- Initial depth estimation: the initial guess for a point's depth is randomly chosen between 0.1 m to 3 m.

For the proposed method, the camera control inputs are computed according to (9.46), while for the passive strategy the actuation commands follow from (9.47). In both methods, the depth of each 3D point is estimated using the observer described in (8.12) with the same parameters as discussed in the previous section for single points. When decreasing the visual servoing error, the passive accounts only for the kinodynamics constraints. Therefore, convergence of the depth estimation filters may happen as a consequence of the camera's motion. Meanwhile the active method is concerned with the control inputs that simultaneously decrease the visual servoing error and ensure convergence of the depth estimators. As the uncertainty on the depth estimation decreases, the global convergence constraints are relaxed. We perform 1000 simulations for $n = 4, \ldots, 9$, running both the proposed and the passive methods for each scenario setup. In addition, to evaluate the temporal performance, we also run the visual servoing task assuming the depth is known a priori. In the ideal case, the control inputs are computed according to (9.47) using χ rather than $\hat{\chi}$. The duration of each task is 15 s, i.e., $T = 15$ s. The results obtained after 6000 simulations are summarized in Fig. 9.7. The passive strategy is shown in red and the active in blue.

The **visual servoing** task is said to have succeeded if the visual servoing error $\|\mathbf{e}\|$ is smaller than 0.01 m. Figure 9.7a shows the success rate of both strategies. The passive method is able to complete about $40\% - 50\%$, while the proposed method accomplished $85\% - 95\%$ of the scenes. Within the considered setup, this means that the proposed method doubles the chance to successfully accomplish the task. In none of our experiments there was a scenario where the passive worked, but the proposed did not. As the number of points being tracked increases, the success rate slowly decreases. In fact, as n increases and since we cannot ensure a 100% convergence for all depth estimation filters,

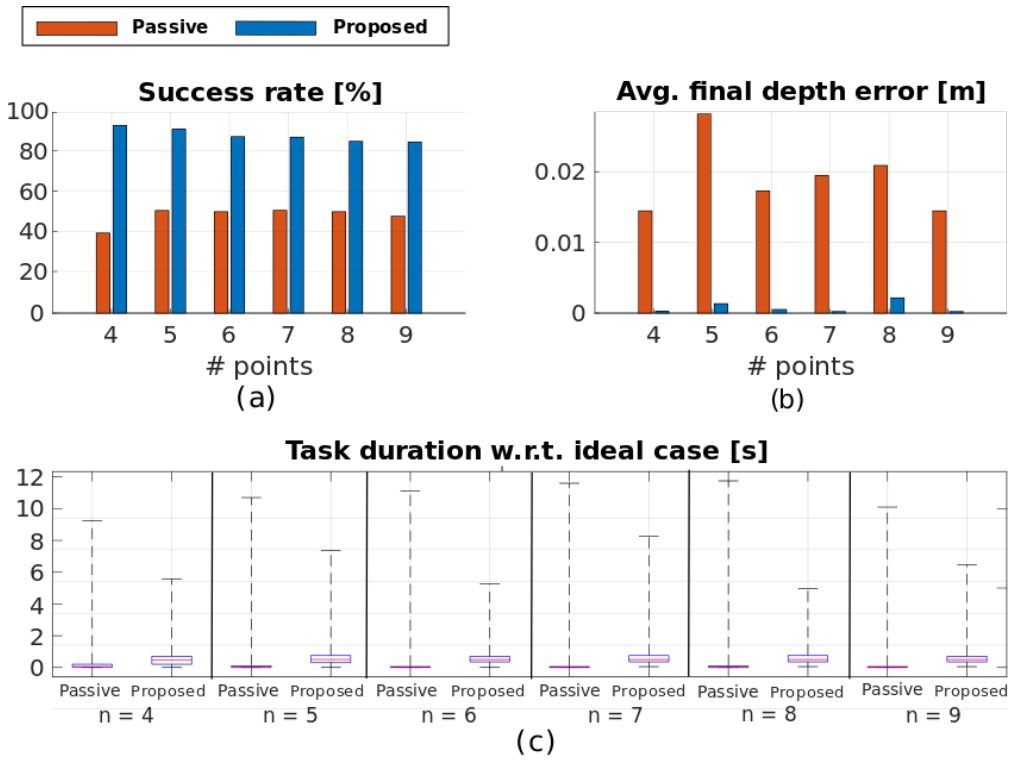


FIGURE 9.7: Coupled depth estimation and visual servoing for multiple points: 6000 simulations were performed using a passive and the proposed active strategy. Scenarios with 4 to 9 points were considered. (a) shows the success rate, (b) the average depth error, and (c) the time difference with respect to the ideal case (known depth).

it will also increase the probability of at least one filter not converging. As a consequence, the interaction matrix does not converge to its true value and the visual servoing error gets trapped in a local minima. The slight increase in the passive method from $n = 4$ to $n = 6$ is not expected. It may be better explained by an unexpected poor performance when the number of points was low. From the definition of the cost function in both strategies, once the visual servoing error reaches a local minima, the camera stops moving. In this case, the PE condition and the depth observer dynamics becomes zero.

By the end of the task, at $T = 15$ s, the depth estimation error in each scenario is computed as $\sum_{i=1}^{n} |1/\chi_i - 1/\hat{\chi}_i|$. Figure 9.7b shows the average error, which was computed considering only the cases that the passive strategy accomplished the visual servoing task. The proposed active method performed significantly better: its average final depth estimation errors were about 10 times ($n = 5, 6, 8$) to 100 times ($n = 4, 7, 9$) smaller than the ones presented by passive method. The results in terms of time are depicted in Fig. 9.7. The box plot addresses the additional time a strategy required to decrease the visual servoing error to $\|\mathbf{e}\| \leq 0.01$ m with respect to (w.r.t.) the ideal case - known depth. Once again, only the cases where the passive method succeeded are compared. The additional

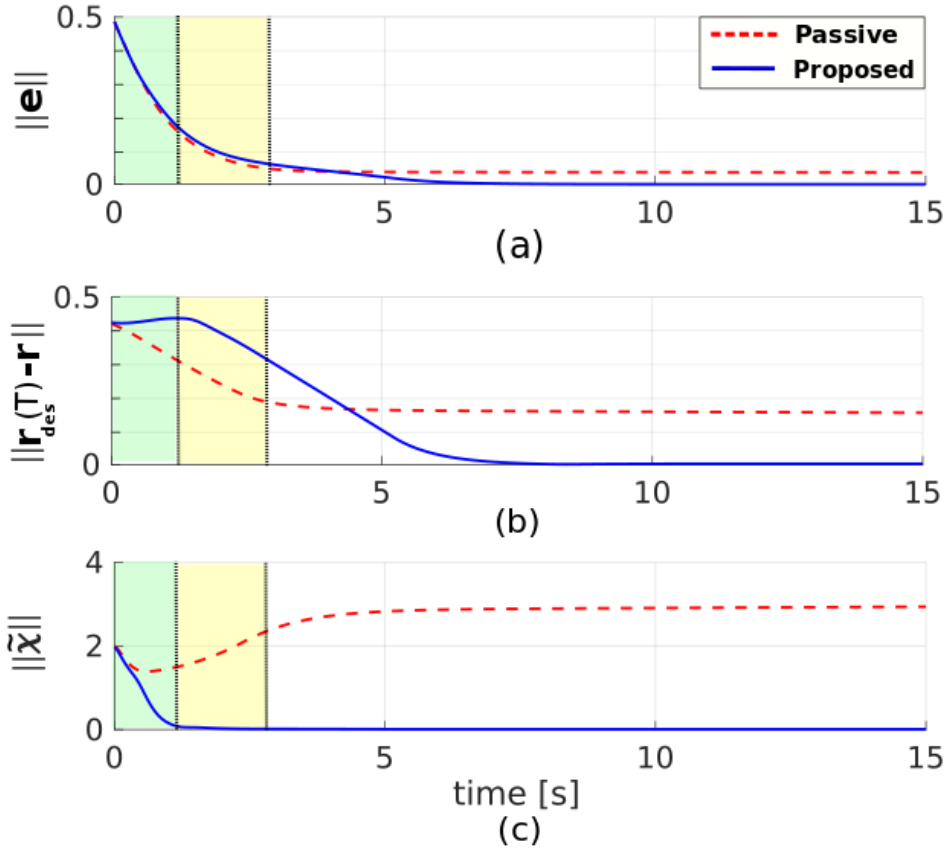


FIGURE 9.8: A 4-point scenario for the multiple depth estimation and visual servoing problem.(a) shows the visual servoing error, (b) the translation error with respect the desired final position of the camera as required by the visual servoing task, and (c) the depth estimation error. In the green region the constraints of Theorem 8.2 are active, in the yellow are the constraints are partially relaxed, and in the white region the constraints are totally relaxed.

constraints imposed by the proposed active method may lead to a slower performance in some scenarios. Notice, however, that the median time of the proposed method was less than 1 s longer than the ideal case (versus 0.2 s for the passive strategy). Moreover, irrespective of the number of points, the worst case performance of the passive method is almost two times longer than the active method.

A scenario with $n = 4$ is explored in more details in Fig. 9.8 and Fig. 9.9. The color code is similar as before: results for the active method are shown in a blue continuous line and for the passive in a dashed red line. For the proposed strategy, consider as a relaxation metric the value of c in (9.44). Fig. 9.8 contains three colored regions that describes the different relaxation steps. In the green area the constraints stated in Theorem 8.2 area active ($c = 0$), in the yellow region the constraints are partially relaxed ($0 < c < v_{\max}$), and in the white area the constraints are totally relaxed ($c = v_{\max}$). As shown in Fig. 9.8a,

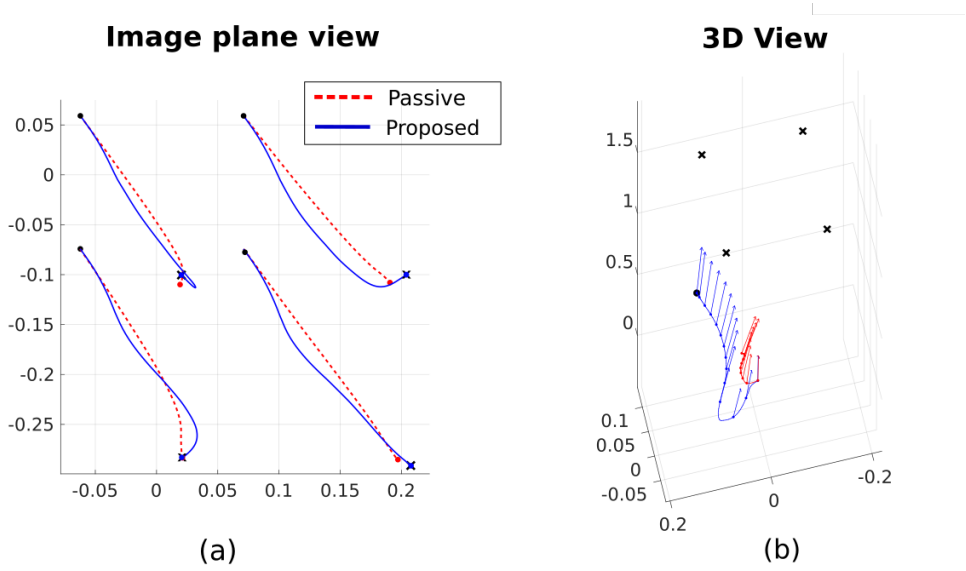


FIGURE 9.9: Mutiple points depth estimation and visual servoing: For the same 4-point setup presented in Fig. 9.8 (a) shows the trajectories of the points in the image frame. The initial features coordinates are painted in black circles and the desired final features coordinates are illustrated by the symbol $\times$. (b) shows the 3D view of the trajectory of the camera for the passive (red) and active strategy (blue). The desired final position is represeted by a black circle at the world coordinates $[0.1, 0.1, 0.4]^T$.

the visual servoing error decreases almost as fast in both cases. However, for the passive method, about $t = 4$ s, the error stops decreasing in a non-zero value and there it remains until the end of the simulation ($\|\mathbf{e}(T)\| \approx 0.03$ m). The method fell into a local minima. Meanwhile, the proposed method is able to continuously decrease the visual servoing error, such that $\|\mathbf{e}(T)\| \approx 10^{-5}$ m. Although it leads to a relatively small visual servoing error, the final pose of the camera using the passive method is not close to the desired value (see Fig. 9.8b). Once again, the proposed active method presents better results, leading the camera to the desired final pose. The reason the passive method does not drive the camera to the desired pose and, consequently, presents a non-zero visual servoing error is because the depth estimation does not converge - nor does the interaction matrix $\hat{L}$. As shown in Fig. 9.8c, the final value of the inverse depth estimation error is almost 3 m^{-1} for the passive method, while for the active it converges to zero.

For the same 4-point setup, the trajectory of the points in the image frame throughout the simulation is illustrated in Fig. 9.9a. The initial and the desired coordinates of the features are represented by black dots and black symbols $\times$, respectively. Colored circles indicate the final position of the features using the passive (red) and the proposed (blue) strategies. Considering the bottom left feature, it seems that passive method is able to drive this particular feature to its desired image coordinates. While for the remaining

features there is a reasonable mismatch. In contrast, the active strategy drives all the features to the desired position in the image plane. The difference in the final visual servoing error and in the trajectories of the features in the image plane suggest that the trajectory of the camera itself is not the same for both methods. The 3D vizualization of the trajectory of the camera and the 3D points is illustrated in Fig. 9.8b. An arrow indicates the z-axis of the camera frame. For both strategies, the camera starts at the origin of the world. While the camera converges to the desired pose (goal position is represented by a black circle) using the active method, for the passive strategy it does not.

These simulation results show that the active strategy increases the success rate of the simultaneous multiple point depth estimation and visual servoing problem when compared to a passive method. Although the proposed method may slightly increase the task time in some configurations, in the worst case scenario it performs significantly faster than the passive strategy.

Chapter 10

Conclusions and Future Work

This chapter presents final remarks and future work that have the potential to improve the methods proposed in this thesis. Sec. 10.1 covers B-spline SLAM, while Sec. 10.2 concludes the 3D depth estimation track.

10.1 Range-based SLAM

The first research problem investigated in this thesis concerns two issues that arise in state-of-the-art range-based Simultaneous Localization and Mapping: memory consumption and map degradation due to discrete resolution. This thesis proposes two techniques that tackle each of these problems individually.

B-spline curve SLAM, presented in Chapter 6, addresses the memory consumption issue by representing the world using B-spline curves. Results show that the proposed framework may be one order of magnitude more efficient than discrete occupancy-grid based strategies. When contrasted to other geometric SLAM techniques, e.g. line and circle based, B-spline curve SLAM is more versatile for describing complex geometric shapes. Localization and mapping is formulated as a single optimization problem that finds the best place (in parametric variable space) for merging two B-spline curves that refer to the same structure in the environment, as well as the pose of the robot that best explains the current measurements based on the current map. The curves are updated with a short processing time, by taking advantage of the fact that B-spline curves that have been previously aligned can be merged by a weighted sum of their control points. The proposed strategy can be specially valuable as a state estimation module in long-term autonomous operations, for which map scalability is a major concern. Furthermore,

since a B-spline curve lies in the convex hull of its control points, the map itself is suitable for path and motion planning algorithms that require representing the obstacles with a convex shape (e.g. OMG-Tools[93]).

Each B-spline curve in the map represents a continuous artifact in the environment, for instance such as walls and boxes. In future work, the map can be further explored for high level task planning and semantic navigation, by enhancing the current implementation to deal with queries such as: *Where is the next obstacle? What is the length of the obstacle with a certain ID?*. Moreover, re-detection of B-spline curves can be improved by integrating data structures that are specialised in fast spatial search, such as R-Tree.

B-spline surface SLAM, proposed in Chapter 7, tackles the map degradation problem due to discretization. The corner stone of the method is the B-spline surface map, a 2.5D continuous map based on B-spline surfaces. This approach allows for more accurate representations of the world, combining floating point resolution of continuous metric maps with the fast update, access and merging of discrete metric maps. In terms of representation error, it outperformed a vanilla state-of-the-art implementation using occupancy grid maps. As for speed, the proposed technique achieved shorter processing times in comparison with Gaussian process maps, a state-of-the-art approach that also generates continuous maps. This work shows that discretizing the map results in quality degradation, as measurements of the world are not registered at their observed locations, but might be slightly displaced to fit in cells. As a consequence localization also degrades, as it queries the map for estimating the pose of the robot that best explains the current sensor measurements. Results reveal that B-spline surfaces handle the discretization problem more effectively. In fact, even though loop closure is not explicitly performed, results using public data sets show that the proposed B-spline surface SLAM framework can build accurate maps at sensor rate speed. Quantitative results also show that the proposed online B-spline SLAM outperforms multi-hypothesis grid-based SLAM (GMapping), and competes with state-of-the-art solutions that perform offline optimisation (Google Cartographer and TSDF-SLAM). Thus, B-spline surface SLAM presents a favourable trade-off between the accuracy of geometric maps and the speed of discrete grid maps.

In future work, the accuracy of B-spline surface SLAM can be further improved by adding loop closure detection and offline optimisation. As for computational performance, implementing the code in a lower level language such as C or C++ will improve

the speed of the algorithm.

10.2 3D Depth Estimation

The second part of this thesis focuses on active incremental depth estimation for image systems. In particular, we investigate the problem of actively controlling the camera such that depth estimation converges with stability guarantees. Unlike most state-of-the-art approaches, the proposed method does not require moving the observed features to the origin of the image plane.

The active depth estimation framework presented in Chapter 8 builds on top of the incremental depth estimator addressed in [24], where some stability guarantees are presented, as well as a way to maximize its convergence speed in an active fashion. However, asymptotic stability only holds for a single point for which the unknown depth must be kept constant throughout the estimation process. This leads to the issues raised in [26], where the previous framework is extended to couple depth estimation and visual servo control, in a scenario with multiple points. The strategy strives to increase the convergence speed, but stability guarantees are not met, as at most one feature can be at the image origin at a given time.

In Chapter 8 we take a step back to first analyze the camera actuation policies that provide global stability guarantees to the depth estimation of a single feature. More importantly, it is shown that, for a set of actuation inputs, stability holds regardless of the position of the feature in the image plane. Afterwards, in Chapter 9 these theoretical results are applied in the concurrent depth estimation and visual servoing problems, using both single and multiple points. The proposed *single point* framework is asymptotically stable. In contrast to previous works with similar stability properties, the tracked feature does not have to lie in (or move to) the origin of the image frame. Moreover, the unknown depth does not need to be kept constant throughout the estimation process. For *multiple points*, the proposed strategy imposes constraints on the control input of the camera, ensuring that asymptotic stability holds when the associated uncertainty is high. As uncertainty decreases, these constraints are relaxed, allowing the camera to achieve the visual servoing goal. Generally, this requires an accurate depth estimation, and thus we can conclude that our method increases the chance of successfully solving the coupled depth estimation and visual servoing problem. This is supported through several comparative numerical simulations.

In future work, the active depth estimation framework may be put into practice in 6 degree-of-freedom robotic platforms, such as airborne systems, underwater vehicles or manipulators. Meanwhile, the theoretical stability of the filter when the camera's velocity is subjected to noise is still an open problem.

Appendix A

Optimal solution for Problem 9.1

This appendix provides a a proof for the following Theorem:

Theorem A.1. *Consider the non-convex problem*

$$\begin{aligned} \underset{\lambda_1,\lambda_2,\mathbf{v}_r}{\textit{maximize}} \quad & \lambda_1 \\ \textit{subject to} \quad & \lambda_1\mathbf{v}_1 + \lambda_2\mathbf{v}_2 = r\mathbf{v}_r \\ & \|\mathbf{v}_r\| = 1 \\ & 0 \leq \lambda_1 \leq 1 \\ & -b \leq \lambda_2 \leq b \end{aligned} \quad , \tag{A.1}$$

where $\lambda_1, \lambda_2 \in \mathbb{R}^+, \mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^n$, $\|\mathbf{v}_1\| > 0$, *and* $\|\mathbf{v}_2\| = 1$. *The problem is always feasible if* $r \leq b$.

Proof. The problem in (A.1) can be seen as computing the constrained weighted sum of two vectors $(\mathbf{v}_1, \mathbf{v}_2)$ that lies in a circle of radius r, while maximizing the weight λ_1 associated with $\mathbf{v}_1$. Figure 1 supports the geometric proof described here. Consider the equality described by the first constraint in (A.1):

$$\lambda_1\mathbf{v}_1 + \lambda_2\mathbf{v}_2 = r\mathbf{v}_r. \tag{A.2}$$

Applying the norm function and squaring both sides yields

$$\|\mathbf{v}_1\|^2\lambda_1^2 + 2\mathbf{v}_1^T\mathbf{v}_2\lambda_1\lambda_2 + \|\mathbf{v}_2\|^2\lambda_2^2 = r^2\|\mathbf{v}_r\|^2, \tag{A.3}$$

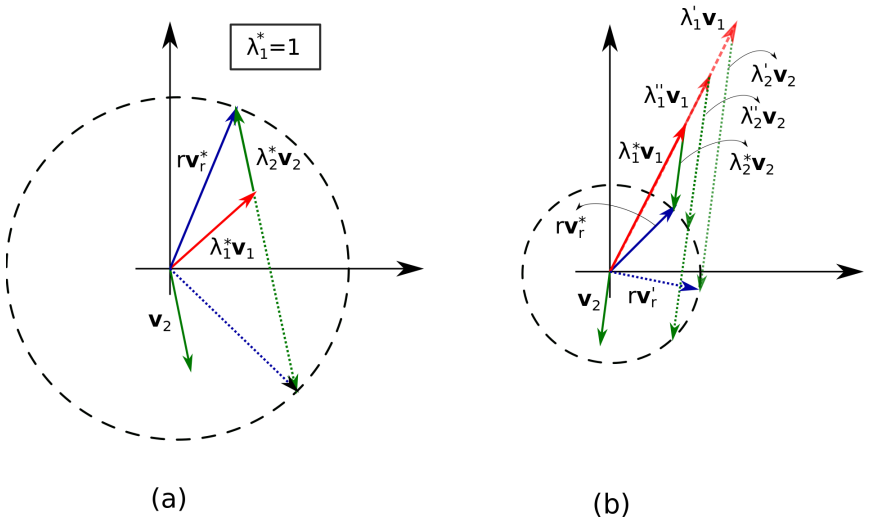


FIGURE A.1: Geometric visualization of the problem in (A.1) in $\mathbb{R}^2$. The problem consists in finding a constrained weighted sum of $\lambda_1\mathbf{v}_1 + \lambda_2\mathbf{v}_2$ that lies in a circle of radius r, while maximizing the weight λ_1. A lagging asterisk indicates an optimal variable. Intermediate steps and alternative solutions are depicted in dashed lines.

where by definition $\|\mathbf{v}_2\| = \|\mathbf{v}_r\| = 1$. Now, let us introduce the quadratic function $f(\lambda_1, \lambda_2)$ defined as

$$f(\lambda_1, \lambda_2) = \|\mathbf{v}_1\|^2\lambda_1^2 + 2\mathbf{v}_1^T\mathbf{v}_2\lambda_1\lambda_2 + \lambda_2^2 - r^2. \tag{A.4}$$

Given admissible $\mathbf{v}_1, \mathbf{v}_2$, and r, the roots of $f(\cdot)$ are candidate solutions for the problem in (A.1). Feasibility requires checking the last two constraints, which define lower and upper bounds for the pair (λ_1, λ_2). The optimal solution $(\lambda_1^*, \lambda_2^*, \mathbf{v}_r^*)$ contains the maximum achievable λ_1. For computing it, we address the problem according to $\|\mathbf{v}_1\|$:

- $\|\mathbf{v}_1\| \leq r$: This case is illustrated in Fig. A.1(a). From (31), $\lambda_1^* = 1$ is always feasible. The optimal weight λ_2^* is obtained by evaluating the roots of (33) using λ_1^*. As $\|\mathbf{v}_1\| \to 0$, $\lambda_2^* \to \pm r$ and the sufficient condition must hold : $r = \|\lambda_2\| \leq b \implies r \leq b$.

- $\|\mathbf{v}_1\| > r$: This more challenging case is shown in Fig. A.1(b). First, we compute a candidate solution for λ_1' that respects (A.2), but not necessarily the bound constraints. If $\mathbf{v}_1 \parallel \mathbf{v}_2$, then $\mathbf{v}_r$ must be parallel to both vectors and we can take $\lambda_1' = 1$ as an initial candidate solution. If $\mathbf{v}_1$ and $\mathbf{v}_2$ are not parallel, define the vector $\mathbf{v}_{2,\perp} = \mathbf{v}_1 - (\mathbf{v}_1^T\mathbf{v}_2)\mathbf{v}_2$, which is perpendicular to the unit vector $\mathbf{v}_2$. A candidate solution that respects (A.2) is given by:

$$\begin{cases} \lambda_1' = r\frac{\|\mathbf{v}_{2,\perp}\|}{\mathbf{v}_1^T\mathbf{v}_{2,\perp}} \\ \lambda_2' = -r\|\mathbf{v}_{2,\perp}\|\frac{\mathbf{v}_1^T\mathbf{v}_2}{\mathbf{v}_1^T\mathbf{v}_{2,\perp}} \\ \mathbf{v}_r' = \frac{\mathbf{v}_{2,\perp}}{\|\mathbf{v}_{2,\perp}\|} \end{cases} \tag{A.5}$$

For the proof, apply the candidate into the left hand side of (A.2):

$$r\frac{\|\mathbf{v}_{2,\perp}\|}{\mathbf{v}_1^T\mathbf{v}_{2,\perp}}\mathbf{v}_1 - r\|\mathbf{v}_{2,\perp}\|\frac{\mathbf{v}_1^T\mathbf{v}_2}{\mathbf{v}_1^T\mathbf{v}_{2,\perp}}\mathbf{v}_2 = \tag{A.6}$$

$$= r\frac{\|\mathbf{v}_{2,\perp}\|}{(\mathbf{v}_{2,\perp} - (\mathbf{v}_1^T\mathbf{v}_2)\mathbf{v}_2)^T\mathbf{v}_{2,\perp}}(\mathbf{v}_1 - (\mathbf{v}_1^T\mathbf{v}_2)\mathbf{v}_2) \tag{A.7}$$

$$= r\frac{\|\mathbf{v}_{2,\perp}\|}{\|\mathbf{v}_{2,\perp}\|^2 - (\mathbf{v}_1^T\mathbf{v}_2)\mathbf{v}_2^T\mathbf{v}_{2,\perp}}\mathbf{v}_{2,\perp} \tag{A.8}$$

$$= r\frac{\mathbf{v}_{2,\perp}}{\|\mathbf{v}_{2,\perp}\|} = r\mathbf{v}_r'. \tag{A.9}$$

Geometrically, the projection $\lambda_1'\mathbf{v}_1$ onto $\mathbf{v}_r'$ and $\mathbf{v}_2$ have length r and $|\lambda_2|$, respectively. Also, the pair (λ_1', λ_2') have the maximum absolute value that satisfies (A.2). Now, we have to ensure that the bound constraints hold. The next candidates are $\lambda_1'' = \min(1, \lambda_1')$ and λ_2'' is a root of $f(\lambda_1'', \lambda_2)$. Choose λ_2'' to be the smallest root in absolute value. Finally, checking the bounds for λ_2 leads to the optimal solution $\lambda_2^* = \text{sign}(\lambda_2'')\min(b, |\lambda_2''|)$ and λ_1^* is the largest positive root of $f(\lambda_1, \lambda_2^*)$.

Algorithm 9: Optimal solution for problem (A.1).

Input: $\mathbf{v}_1, \mathbf{v}_2, r, b$;
Output: $\lambda_1^*, \lambda_2^*, \mathbf{v}_r^*$;
if $\|\mathbf{v}_1\| \leq r$ **then**
  $\lambda_1^* \leftarrow 1$;
  $\lambda_2^* \leftarrow$ roots of $f(\lambda_1^*, \lambda_2) \in [-b, b]$;
else
  $\mathbf{v}_{2,\perp} \leftarrow \mathbf{v}_1 - (\mathbf{v}_1^T\mathbf{v}_2)\mathbf{v}_2$;
  if $\mathbf{v}_1^T\mathbf{v}_{2,\perp} == 0$ **then**
    $\lambda_1' \leftarrow 1$
  else
    $\lambda_1' \leftarrow r\frac{\|\mathbf{v}_{2,\perp}\|}{\mathbf{v}_1^T\mathbf{v}_{2,\perp}}$
  end
  $\lambda_1'' \leftarrow \min(1, \lambda_1')$;
  $\lambda_2'' \leftarrow$ root of $f(\lambda_1'', \lambda_2)$ with the smallest absolute value;
  $\lambda_2^* = \text{sign}(\lambda_2'')\min(b, |\lambda_2''|)$;
  $\lambda_1^* \leftarrow$ most positive root of $f(\lambda_1, \lambda_2^*)$;
end
$\mathbf{v}_r^* \leftarrow (1/r)(\lambda_1^*\mathbf{v}_1 + \lambda_2^*\mathbf{v}_2)$

After computing the optimal values $(\lambda_1^*, \lambda_2^*)$, evaluate (A.2) to obtain $\mathbf{v}_r^*$. □

A closed-form strategy for computing the optimal solution of Problem (A.5) is summarized in Algorithm 9.

Bibliography

[1] "Abi: Robotics market to shift from fixed automation to mobile systems," Dec 2019. [Online]. Available: https://industryeurope.com/abi-robotics-market-to-shift-from-fixed-automation-to-mobile-systems/ [Cited on page 1.]

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots, Second Edition*, 2nd ed. The MIT Press, 2011. [Cited on page 2.]

[3] B. Bennett, "Why your robot vacuum's random patterns make total sense," Jun 2019. [Online]. Available: https://www.cnet.com/news/how-to-choose-the-best-robot-vacuum-for-your-home-roomba-neato-ecovacs-2019/ [Cited on page 3.]

[4] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, 2007. [Cited on pages 4, 5, 38, 63, 71, and 114.]

[5] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, 2011. [Cited on pages 4, 5, 7, 38, 63, 71, 77, 79, 93, 109, 111, 115, and 116.]

[6] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, 2018. [Cited on page 4.]

[7] F. Hidalgo and T. Bräunl, "Review of underwater slam techniques," in *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, Feb 2015, pp. 306–311. [Cited on page 4.]

[8] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping:

Toward the robust-perception age," *IEEE Transactions on Robotics*, 2016. [Cited on pages 4, 6, and 65.]

[9] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2010, pp. 22–29. [Cited on page 5.]

[10] "IEEE standard for robot map data representation for navigation," *1873-2015 IEEE Standard for Robot Map Data Representation for Navigation*, pp. 1–54, Oct 2015. [Cited on page 5.]

[11] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, 1989. [Cited on pages 5 and 69.]

[12] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278. [Cited on pages 5, 7, 38, 63, 72, 73, 77, 79, 98, 111, and 117.]

[13] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003. [Cited on pages 8, 9, 125, 127, and 133.]

[14] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, vol. 47, pp. 2280–2292, 2014. [Cited on pages 8 and 124.]

[15] H. C. Longuet-Higgins, *A Computer Algorithm for Reconstructing a Scene from Two Projections*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, p. 61–62. [Cited on page 9.]

[16] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113. [Cited on pages 9, 125, and 127.]

[17] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment — a modern synthesis," in *Vision Algorithms: Theory and Practice*, 2000, pp. 298–372. [Cited on page 9.]

[18] A. Martinelli, "Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination," *IEEE Trans. Robotics (T-RO)*, vol. 28, no. 1, pp. 44–60, 2012. [Cited on page 9.]

[19] R. Benyoucef, L. Nehaoua, H. Hadj-Abdelkader, and H. Arioui, "Depth estimation for a point feature: Structure from motion amp; stability analysis," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 3991–3996. [Cited on page 9.]

[20] O. Tahri, D. Boutat, and Y. Mezouar, "Brunovsky's linear form of incremental structure from motion," *IEEE Trans. Robotics (T-RO)*, vol. 33, no. 6, pp. 1491–1499, 2017. [Cited on page 9.]

[21] E. Bjørne, T. A. Johansen, and E. F. Brekke, "Cascaded bearing only slam with uniform semi-global asymptotic stability," in *2019 22th International Conference on Information Fusion (FUSION)*, 2019, pp. 1–8. [Cited on page 9.]

[22] F. Chaumette, S. Boukir, P. Bouthemy, , and D. Juvin, "Structure from controlled motion," *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 18, no. 5, pp. 492–504, 1996. [Cited on page 10.]

[23] R. Spica, P. Robuffo Giordano, and F. Chaumette, "Active structure from motion: Application to point, sphere, and cylinder," *IEEE Trans. Robotics (T-RO)*, vol. 30, no. 6, pp. 1499–1513, 2014. [Cited on pages 10, 129, 130, 149, 150, 151, and 152.]

[24] ——, "Plane estimation by active vision from point features and image moments," in *IEEE Int'l Conf. Robotics and Automation (ICRA)*, 2015, pp. 6003–6010. [Cited on pages 10, 130, 139, 143, and 161.]

[25] A. Mateus, O. Tahri, A. P. Aguiar, P. U. Lima, and P. Miraldo, "On incremental structure from motion using lines," *IEEE Trans. Robotics (T-RO)*, pp. 1–16, 2021. [Cited on page 10.]

[26] R. Spica, P. R. Giordano, and F. Chaumette, "Coupling active depth estimation and visual servoing via a large projection operator," *The International Journal of Robotics Research*, vol. 36, no. 11, pp. 1177–1194, 2017. [Cited on pages 10, 146, 153, and 161.]

[27] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley - Cambridge Press, 2016. [Cited on page 17.]

[28] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 1st ed. Springer Publishing Company, Incorporated, 2013. [Cited on pages 20 and 33.]

[29] A. Hájek, "Interpretations of Probability," in *The Stanford Encyclopedia of Philosophy*, Fall 2019 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2019. [Cited on page 21.]

[30] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000. [Cited on page 31.]

[31] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965. [Online]. Available: http://www.jstor.org/stable/2333709 [Cited on page 36.]

[32] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009, p. 5. [Cited on page 38.]

[33] C. de Boor, *A Practical Guide to Splines*. New York, NY: Springer-Verlag, 1978. [Cited on pages 41 and 43.]

[34] T. Lyche and K. Morken, *Spline Methods*. Norway: Unpublished, 2018. [Cited on page 43.]

[35] H. Kano, H. Nakata, and C. F. Martin, "Optimal curve fitting and smoothing using normalized uniform b-splines: a tool for studying complex systems," *Applied Mathematics and Computation*, vol. 169, no. 1, pp. 96–128, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300304007556 [Cited on page 48.]

[36] , "IEEE standard for robot map data representation for navigation," *1873-2015 IEEE Standard for Robot Map Data Representation for Navigation*, pp. 1–54, Oct 2015. [Cited on page 61.]

[37] J. O. Wallgrün, *Hierarchical voronoi graphs: Spatial representation and reasoning for mobile robots*, 1st ed. Springer Publishing Company, Incorporated, 2010. [Cited on page 62.]

[38] J. Nieto, J. Guivant, E. Nebot, and S. Thrun, "Real time data association for fastslam," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 1, 2003, pp. 412–418 vol.1. [Cited on page 62.]

[39] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, Mar 1985, pp. 116–121. [Cited on pages 63 and 68.]

[40] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. [Cited on pages 63, 64, 65, and 101.]

[41] J.-D. Fossel, K. Tuyls, and J. Sturm, "2d-sdf-slam: A signed distance function based slam frontend for laser scanners," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 1949–1955. [Cited on pages 63 and 72.]

[42] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 303–312. [Online]. Available: https://doi.org/10.1145/237170.237269 [Cited on page 63.]

[43] K. Daun, S. Kohlbrecher, J. Sturm, and O. von Stryk, "Large scale 2d laser slam using truncated signed distance functions," in *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2019, pp. 222–228. [Cited on pages 63, 73, 79, and 117.]

[44] D. Sack and W. Burgard, "A comparison of methods for line extraction from range data," in *In Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV*, 2004. [Cited on page 64.]

[45] J. L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging," in *Proceedings, 1989 International Conference on Robotics and Automation*, 1989, pp. 674–680 vol.2. [Cited on pages 64 and 73.]

[46] D. Wolter, L. J. Latecki, R. Lakämper, and X. Sun, "Shape-based robot mapping," in *KI 2004: Advances in Artificial Intelligence*, S. Biundo, T. Frühwirth, and G. Palm, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 439–452. [Cited on page 64.]

[47] R. Vázquez-Martín, P. Núñez, A. Bandera, and F. Sandoval, "Curvature-based environment description for robot navigation using laser range sensors," *Sensors*, 2009. [Cited on page 64.]

[48] A. Caccavale and M. Schwager, "Wireframe Mapping for Resource-Constrained Robots," in *2018 IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2019. [Cited on page 64.]

[49] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research*, vol. 31, no. 1, pp. 42–62, 2012. [Cited on page 64.]

[50] M. G. Jadidi, J. V. Miro, and G. Dissanayake, "Gaussian processes autonomous mapping and exploration for range-sensing mobile robots," *Autonomous Robots*, vol. 42, no. 2, pp. 273–290, 2018. [Cited on page 64.]

[51] Y. Yuan, H. Kuang, and S. Schwertfeger, "Fast gaussian process occupancy maps," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2018, pp. 1502–1507. [Cited on page 65.]

[52] R. Smith, M. Self, and P. Cheeseman, *Estimating Uncertain Spatial Relationships in Robotics*. New York, NY: Springer New York, 1990, pp. 167–193. [Online]. Available: https://doi.org/10.1007/978-1-4613-8997-2{_}14 [Cited on page 65.]

[53] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "A computationally efficient solution to the simultaneous localisation and map building (slam) problem," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 1009–1014 vol.2. [Cited on page 67.]

[54] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, 2001. [Cited on page 67.]

[55] K. P. Murphy, "Bayesian map learning in dynamic environments," in *Advances in Neural Information Processing Systems*, 2000. [Cited on page 67.]

[56] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the National Conference on Artificial Intelligence*, 2002. [Cited on page 67.]

[57] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably

converges," in *IJCAI International Joint Conference on Artificial Intelligence*, 2003. [Cited on page 67.]

[58] F. Lu and E. E. Milios, "Robot pose estimation in unknown environments by matching 2d range scans," *Journal of Intelligent and Robotic Systems*, vol. 18, pp. 249–275, 1997. [Cited on page 68.]

[59] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, p. 239–256, Feb. 1992. [Online]. Available: https://doi.org/10.1109/34.121791 [Cited on page 69.]

[60] D. Hahnel, D. Schulz, and W. Burgard, "Map building with mobile robots in populated environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2002, pp. 496–501 vol.1. [Cited on pages 69 and 70.]

[61] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, pp. 321–328 vol.1, 2000. [Cited on page 70.]

[62] D. Hähnel, W. Burgard, D. Fox, and S. Thrun, "An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements," in *IEEE International Conference on Intelligent Robots and Systems*, 2003. [Cited on page 71.]

[63] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2005. [Cited on page 71.]

[64] B. Holý, "Registration of lines in 2d lidar scans via functions of angles," *Eng. Appl. Artif. Intell.*, vol. 67, pp. 436–442, 2018. [Cited on pages 73 and 74.]

[65] L. Pedraza, D. Rodriguez-Losada, F. Matia, G. Dissanayake, and J. V. Miro, "Extending the limits of feature-based slam with b-splines," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 353–366, April 2009. [Cited on pages 74, 78, and 83.]

[66] M. Liu, S. Huang, G. Dissanayake, and S. Kodagoda, "Towards a consistent slam algorithm using b-splines to represent environments," in *2010 IEEE/RSJ International*

Conference on Intelligent Robots and Systems, Oct 2010, pp. 2065–2070. [Cited on page 74.]

[67] J. Zhao, L. Zhao, S. Huang, and Y. Wang, "2d laser slam with general features represented by implicit functions," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4329–4336, 2020. [Cited on page 74.]

[68] B. Li, Y. Wang, Y. Zhang, W. jie Zhao, J. Ruan, and P. Li, "Gp-slam: laser-based slam approach based on regionalized gaussian process map reconstruction," *Auton. Robots*, vol. 44, pp. 947–967, 2020. [Cited on page 75.]

[69] R. Yagfarov, M. Ivanou, and I. M. Afanasyev, "Map comparison of lidar-based 2d slam algorithms using precise ground truth," *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1979–1983, 2018. [Cited on page 76.]

[70] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of slam algorithms," *Autonomous Robots*, vol. 27, no. 4, p. 387, Sep 2009. [Cited on pages 76, 77, 78, 116, 117, 118, and 119.]

[71] A. Howard and N. Roy, "The robotics data set repository (radish)," 2003. [Online]. Available: http://radish.sourceforge.net/ [Cited on pages 77 and 116.]

[72] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, winter 2010. [Cited on pages 77 and 114.]

[73] "Tu-darmstadt ros pkg," 2011. [Online]. Available: https://code.google.com/archive/p/tu-darmstadt-ros-pkg/downloads [Cited on page 78.]

[74] E. B. Olson, "Real-time correlative scan matching," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 4387–4393. [Cited on page 79.]

[75] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007. [Cited on page 93.]

[76] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. USA: Cambridge University Press, 2007. [Cited on page 105.]

[77] J. T. Barron, "A general and adaptive robust loss function," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019. [Cited on page 108.]

[78] Y. Yuan, H. Kuang, and S. Schwertfeger, "Fast gaussian process occupancy maps," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2018, pp. 1502–1507. [Cited on page 114.]

[79] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. [Cited on page 115.]

[80] H. Xiong, Y. Chen, X. Li, and B. Chen, "A two-level optimized graph-based simultaneous localization and mapping algorithm," *Industrial Robot: An International Journal*, 2018. [Cited on page 117.]

[81] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011. [Cited on page 124.]

[82] A. Bartoli and P. Sturm, "Structure-from-motion using lines: Representation, triangulation, and bundle adjustment," *Computer Vision and Image Understanding (CVIU)*, vol. 100, no. 3, pp. 416–441, 2005. [Cited on page 125.]

[83] J. J. Koenderink and A. J. van Doorn, "Affine structure from motion," *J. Opt. Soc. Am. A*, vol. 8, no. 2, pp. 377–385, 1991. [Cited on page 125.]

[84] S. Soatto, R. Frezza, and P. Perona, "Motion estimation via dynamic vision," *IEEE Trans. Automatic Control (T-AC)*, vol. 41, no. 3, pp. 393–413, 1996. [Cited on page 127.]

[85] A. P. Dani, N. R. Fischer, and W. E. Dixon, "Single camera structure and motion," *IEEE Trans. Automatic Control (T-AC)*, vol. 57, no. 1, pp. 238–243, 2012. [Cited on page 127.]

[86] R. Spica and P. Robuffo Giordano, "A framework for active estimation: Application to Structure from Motion," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 7647–7653. [Cited on pages 130, 133, 149, and 152.]

[87] A. Mateus, O. Tahri, and P. Miraldo, "Active estimation of 3d lines in spherical coordinates," in *American Control Conf. (ACC)*, 2019, to appear. [Cited on page 130.]

[88] I. M. Y. Mareels and M. Gevers, "Persistency of excitation criteria for linear, multivariable, time-varying systems," *Mathematics of Control, Signals and Systems*, vol. 1, pp. 203–226, 1988. [Cited on page 131.]

[89] J.-J. E. Slotine and W. Li, *Applied nonlinear control*. Prentice-Hall, 1991. [Cited on page 132.]

[90] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *IEEE Robotics & Automation Magazine*, vol. 13, pp. 82–90, 2006. [Cited on pages 146 and 153.]

[91] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. [Cited on page 149.]

[92] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag Limited, 2008, pp. 95–110. [Cited on page 149.]

[93] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 6, pp. 2182–2189, 2018. [Cited on page 160.]