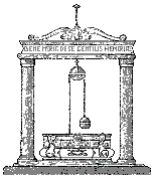
SAPIENZA
UNIVERSITÀ DI ROMA

SAPIENZA UNIVERSITY OF ROME



DOCTORATE OF PHILOSOPHY IN ENGINEERING IN COMPUTER SCIENCE
XXIX CYCLE

Using Extended Measurements and Geometric Features for Robust Long-Term Localization and Mapping

Candidate
Jacopo Serafin
ID Number 1162243

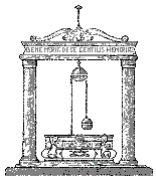
Thesis Advisor
Prof. Giorgio Grisetti

Thesis Co-Advisor
Prof. Marco Schaerf

A.Y. 2016/2017

SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome



Doctorate of Philosophy in Engineering in Computer Science
XXIX Cycle

Using Extended Measurements and Geometric Features for Robust Long-Term Localization and Mapping

Candidate
Jacopo Serafin
ID Number 1162243

Thesis Committee
Prof. Giorgio Grisetti (Thesis Advisor)
Prof. Marco Schaerf (Thesis Co-Advisor)

Reviewers
Martin Magnusson
Rainer Kümmerle

A.Y. 2016/2017

Using Extended Measurements and Geometric Features for Robust Long-Term Localization and Mapping

Ph.D. Thesis - Sapienza University of Rome

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering in Computer Science.
Thesis defended on the 21^{st} of February 2017.

This thesis has been typeset by LaTeX.
Version: February 16, 2017.

AUTHOR'S ADDRESS:
Jacopo Serafin
Department of Computer, Control and Management Engineering "Antonio Ruberti"
Sapienza - University of Rome
Via Ariosto 25, I00185 Rome, Italy
E-MAIL: serafin@dis.uniroma1.it
WEB-SITE: http://www.dis.uniroma1.it/~serafin

"I don't know half of you
half as well as I should like;
and I like less than half of you
half as well as you deserve."
— *J.R.R. Tolkien, The Lord of the Rings, 1954*

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

"Happiness only real when shared."
— Christopher Johnson McCandless "Alexander Supertramp", 1968 - 1992

In these three years, there are many people that have been fundamental for me to complete this Ph.D, and that I should thank for. The first person I would like to thank is Giorgio Grisetti, my advisor but mostly important my friend. This is the characteristic that makes Giorgio a unique person, and I am grateful for having had him as my advisor. It is only thanks to Giorgio's deep knowledge and experience that I could grow so much professionally, and develop such a passion for mobile robots. A Ph.D. with Giorgio is a fun Ph.D., always, under all aspects and regardless of the situation. No matter whether you are facing problems, failures or successful achievements, Giorgio is always present and ready to help, with a big smile in his face, either coding with you (be sure to install Emacs, even better if you use it too!), or by sharing great comments and suggestions. Thank you Giorgio for the passion you put in what you do since this radically changes in a positive direction the life of people around you, including mine.

I want to thank my family which has been the foundation for who I am today. In particular, I thank my love Elena that has always been near me giving support in all kind of situations along all these three years (but they are much more, trust me). Without her help and understanding, even in cases where it was hard just speaking, it would have been much more tough to solve the problems and issues that I have faced during this Ph.D. path. But this is the reason for which I love her. Thanks to my mother Anna and my father Ilario since if I am the person that I am today, it is because they made a great job in growing and educating a crazy head like mine. They have been always available to support me, from bringing me to and from the train station every day, to reducing as much as possible common duties that people must deal with in everyday life. This allowed me to concentrate only on my study, thus significantly contributing in obtaining better research results. I thank my brother Nicola for being my reference point since from when I was a child. He has always represented for me

the limit and example to look at to try to do better. I want to thank my creative sister Giulia that, despite the appearances, I greatly admire and from which I draw inspiration under multiple aspects. If this thesis is much more appealing and attractive, it is only thanks to my sister and her skills with graphic design tools. Indeed, many of the images in this manuscript have been generated by her. Thanks also to all members of Elena's family that, whenever possible, gave unconditioned help and support by treating me as part of the family.

A particular thanks goes to my office mate Roberto for the great atmosphere, good discussions, and more in general for being a good friend. I want also to thank all the guys of the Ro.Co.Co laboratory, including Prof. Nardi and Prof. Iocchi for being always available for fruitful comments and suggestions.

Special thanks go to Prof. Edwin Olson. Thank you Edwin for letting me be part of the April laboratory at University of Michigan, giving me the chance of further improving my skills and the quality of my Ph.D. path. Thank you for all fundamental suggestions and the help you provided, and in particular I am grateful to you for the opportunity you have given to me for joining Toyota Research Institute, a dream come true. Moreover, I would like to thank all the guys in the April laboratory for welcoming me for the months I worked next to them.

Thanks to all other people that collaborated with me during this Ph.D. and that contributed to improve my knowledge and expertise, including Prof. Cyrill Stachniss, Vittorio Ziparo, and more in general all the guys involved in the ROVINA project.

I would like to thank all my friends, in particular the "train guys" that contributed to make lighter the time spent over trains (yes, 2 hours, in the luckiest of the cases!), every day, for three years.

It is impossible to list all people that would actually deserve acknowledgment for being part of my life, and for having been important during this Ph.D., each one contributing in his own way. For these reasons, I say thank you to all of you.

Lastly, I want to thank myself since, if I reached this goal, it is also for the continuous dedication and passion that I put every single day in my work. This means for example, in the case of a crazy nerdy person like me, multiple weekends spent writing code or papers even late in the night, and just for fun. And I have to say, thank you again Elena for putting up with such a nerd like me, and for accepting all connected weirdness. Thank you because you love me, and thank you because you are the only one capable of introducing a pinch of normality and salt in my life.

Jacopo Serafin

This work has partly been supported by the European Commission under FP7-600890-ROVINA, and by the NSF project CCF1442773.

Abstract

In robotics, the term Simultaneous Localization And Mapping (SLAM) refers to the ability of a robot to create a map of the surrounding environment, and to localize itself in that map, by using its on-board sensors. SLAM is fundamental for mobile robots that want to efficiently solve common tasks such as search and rescue, space exploration and self-driving. Indeed, SLAM allows robots to perform more complex actions and reasoning, and for all these reasons it is considered an enabling technology toward the construction of truly autonomous mobile robots.

For more than three decades, the robotics research community focused on the problem of localization and mapping. This led to the development of many different SLAM approaches, and solutions. However, while for 2D indoor applications SLAM reached stable results, when dealing with large-scale environments, or when extending the problem to 3D data, multiple issues arise. Under such assumptions, indeed, the complexity and the memory consumption increase significantly, in particular if the robot has to operate for long periods of time. Despite the recent advancements in the state-of-the-art of SLAM, current 3D localization and mapping methods can robustly handle only scenarios of moderate size (e.g. buildings). Moreover, in order for humans beings to be safe, novel solutions must be capable to detect and recover from failures, as well as be able to deal with noise such as dynamics aspects of the environment and perceptual aliasing.

In the last years the demand for robots capable of helping human beings in their daily environment, including agriculture and autonomous driving, is constantly growing. As a result of this, researchers concentrate on the development of long term SLAM systems that are able to cope with indoor/outdoor challenging environments such as urban scenarios, or underground tunnels. This also means that these systems must be designed to face sensor limitations, such as data sparsity and perceptual aliasing, and common noise factors like seasonal changes and moving objects. In this thesis we present multiple novel solutions that improve the accuracy, efficiency and robustness of the current state-of-the-art in localization and mapping, under such hard conditions.

Modern SLAM systems are built as the aggregation and connection of multiple independent modules like algorithms for feature/key-point detection (data enhancement), measurement registration methods and optimization frameworks. A fundamental block is that one related to loop closures. By detecting areas already visited in the past by the robot, loop closing allows for a drastic reduction of the error affecting the global map estimate. Advancing any of these parts has the effect of improving the entire SLAM system. In this work we focused on developing innovative solutions for some of the most important modules composing modern localization and mapping approaches.

We present a novel point cloud registration method based on an augmented error function associated to the alignment problem. Through an extensive experimental evaluation, our approach demonstrates the ability to outperform other state-of-the-art algorithms. We detail a robust and effective loop closing search heuristic that, based on the trajectory followed by the robot, significantly reduces the overall computation time. Additionally, similarly to visual SLAM approaches, it allows for recovering from inconsistent global map estimates due to miss detected closures. Such an algorithm is able to correctly map hard and harsh environments like catacombs. Also, we introduce a method that exploits 3D sensor data to build highly informative two dimensional measurements. These measurements are then used for achieving robust indoor/outdoor 2D SLAM. Our algorithm also makes efficient use of memory, making the approach particularly suitable for large-scale mapping. Finally, we present a very fast method for computing robust 3D geometric features from sparse sensor data. We demonstrate the effectiveness of that approach by performing 3D feature based SLAM with an autonomous car at the MCity test facility.

Chapter 1
Introduction

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning", the King said, gravely,
"and go on till you come to the end: then stop."
— Lewis Carroll, Alice's Adventures in Wonderland, 1865

An essential skill for a robot that operates in an unknown environment is the capability of constructing an internal model of the surroundings (the map), and be aware of its location in this model. The knowledge of the location, and of the environment, has to be computed by collecting data with the robot's on-board sensors. Noise in the measurements and recurrent patterns in the surroundings makes this estimation problem, known as SLAM [1], a hard one that has attracted the attention of researchers in the last thirty years. The need of map generation with robots is mainly concerned with that of indoor applications. In such cases, indeed, it is not possible to rely on GPS (Global Positioning System) sensors to limit the localization error. Moreover, SLAM represents an automated alternative to maps manually generated by humans. Thus, it also demonstrates that robots can operate regardless of the presence of an ad-hoc localization infrastructure.

The robot's ability of simultaneously building a map of the surrounding environment, and localize itself within that map, enables higher levels of reasoning and allows the execution of more complex actions. More specifically, SLAM is usually at the core of complex robotic systems addressing specific tasks such as vacuum cleaning, autonomous surveying, search and rescue, space exploration and self-driving. A global map limits the drifting error accumulated while incrementally estimating the pose of the robot. Indeed, it is well known that the odometry of a robotic platform drifts over time. However, given a reference map, the robot can re-localize itself by detecting already visited areas (denoted loop closure). This has the effect of drastically reduce the error due to the drift. Additionally, a global map estimate is fundamental for providing

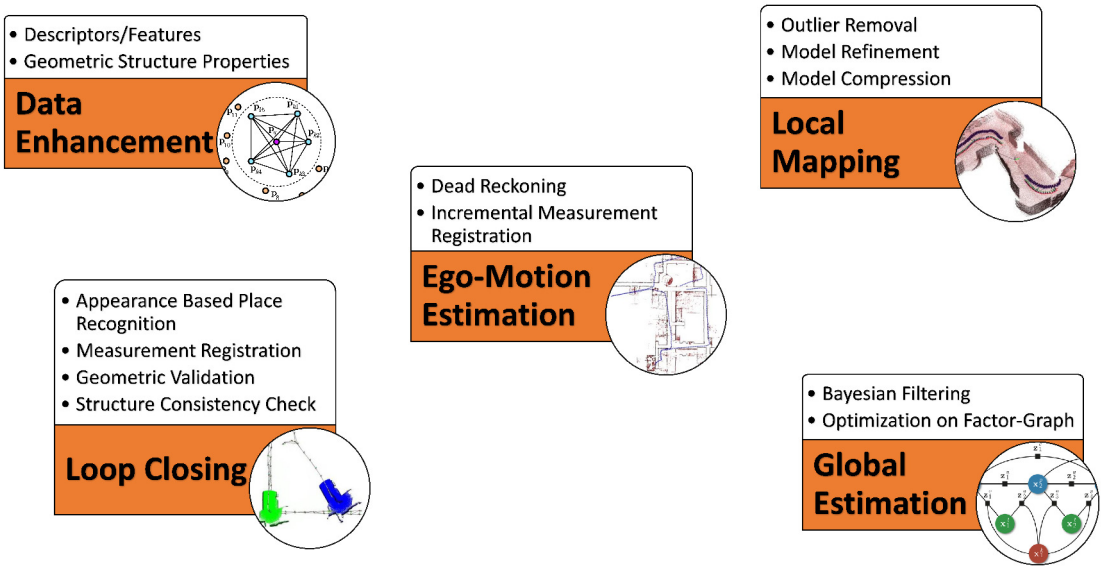


Figure 1.1: Taxonomy of the components that can be part of a SLAM system.

an intuitive visualization of the robot's state to human operators who interact with the mobile platform.

A SLAM system is composed by the aggregation and interconnection of multiple modules, and each module finds its own application even outside of that system (see Fig. 1.1). A first group of components that can be found in SLAM is that of data enhancement. This group has the goal of extracting additional information from raw measurements generated by the sensors, including descriptors [2][3], features [4][5] and geometric structure properties [6] of the surrounding environment. Another fundamental part of a localization and mapping system is the one that computes the ego-motion estimate of the robot. Common techniques used for solving this problem rely on approaches such as dead reckoning [7], and incremental measurement registration [8][9]. Sequences of measurements are eventually aggregated, and merged together, to form local maps. These local maps are refined by removing possible outliers (i.e. dynamic objects), and they can be further optimized [10] and stored in memory efficient data structures [11]. Loop closing, an other fundamental block of a SLAM system, is commonly performed by using algorithms that search, based on appearance similarities, for places already visited in the past. Loop closing relations, connecting far in time local maps, are computed by using methods for registering measurements [12][13]. Since just a single wrong closure can lead the SLAM system to diverge, this group of modules usually also contains approaches for verifying the correctness of the data association. The last set of components is the one related to the global estimation of the map.

This set mainly depends on the type of SLAM approach used in the system: Bayesian filtering [14][15] or optimization [16] on factor graphs [17].

SLAM can be applied in many cases, in particular when a prior reference map is not available and needs to be generated. For some robotics applications, however, the map of the environment in which the robot moves is known a priori. As an example, consider a mobile robot such as an autonomous vehicle, whose given world reference model is an Open Street Map (OSM) from Google. Localization and mapping represents therefore a core enabling technology for solving more advanced tasks, and it is "de facto" of utmost importance toward the construction of truly autonomous robots. For all these reasons, the solution of such a problem is regarded as the holy grail in the mobile robotics research community.

Nowadays there is an always increasing request for mobile robots, such as autonomous cars, to operate and to help human beings in their daily environment. The significantly wider workspace, and human safety concerns, led the research community to focus on the development of long term robust localization and mapping methods that are able to cope with large-scale environments. In this thesis we present multiple novel solutions that improve the accuracy, efficiency and robustness of the current state-of-the-art in localization and mapping, in such scenarios. We focused on advancing some of the most important modules that are part of modern SLAM systems, in particular we present:

- a point cloud registration method based on an augmented error function associated to the alignment problem [18][19];
- a path based loop closing search approach for a 3D SLAM system in hard to access environments such as catacombs [20][21][22];
- an algorithm for space dimension reduction of 3D sparse laser data for reliable indoor/outdoor 2D SLAM [23];
- a fast and robust 3D geometric feature extraction method, from sparse data, for outdoor robot localization and mapping [24].

1.1 Anatomy of a Modern SLAM System

Modern SLAM systems are mainly based on a formulation of the localization and mapping problem known as graph-based SLAM [17]. The problem is modelled using a weighted graph where nodes correspond to robot poses or landmarks positions, and edges represent relation constraints among nodes. The global estimate of the map is computed by finding the configuration of the nodes that better explains the constraints generated through the measurements.

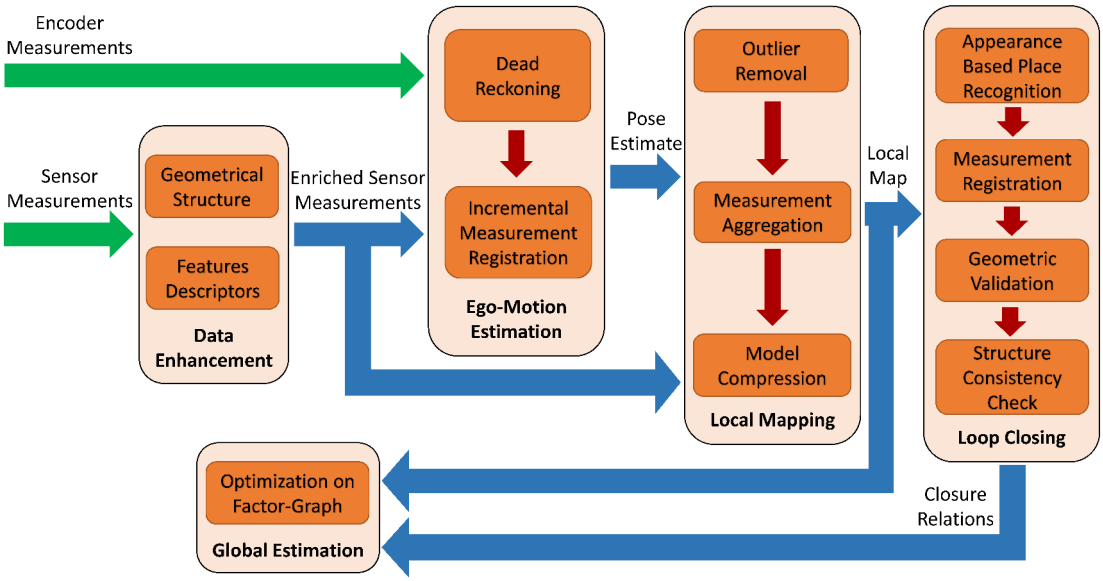


Figure 1.2: Example pipeline of a modern SLAM system. Green, blue and red arrows represent respectively raw input measurements generated by the sensors, input/output data of macro blocks of the system, and results produced by internal modules.

A typical pipeline of a graph based SLAM system, and its modules, is depicted in Fig. 1.2. Raw sensor measurements are first preprocessed in order to extract additional information from the data. This means computing features or descriptors, and in the case of 3D data also geometric properties such as surface normals or tangents. In parallel to this, encoder readings and GPS measurements are integrated to compute an initial estimate of the pose of the robot, denoted dead reckoning. The drift error, intrinsically encoded in the dead reckoning, is then reduced by performing incremental registrations of the augmented sensor measurements. This is done using common techniques such as Structure from Motion (SfM) [25], or variants of the Iterative Closest Point (ICP) [26] algorithm. The pose estimate computed by the dead reckoning is used as initial guess of the alignment, and the output of these modules is referred to as ego-motion pose estimation. The measurements are then aggregate to form a bigger data structure called local map, or key-frame. In this phase, possible outliers such as dynamic objects in scene are removed. Additionally, the local map can be optimized and modelled using more memory efficient representations. The local map, and the transform relation with respect to the previous one, are added to the graph. At this point, the system is capable of generating sequential local maps. However, due to the accumulating drift error, this is not sufficient for obtaining a consistent estimate of the global map. In order to re-localize the robot, and further reduce the drifting,

loop closing is performed. Commonly, based on appearance metrics, an ad-hoc module search for places (local maps) already visited in the past. The candidate closures are registered together and validated through consistency checks before being added to the graph. Optimization frameworks, like the very fast, and open source, g2o library [16], can be used to optimize the graph and compute the global estimate of the map.

1.2 Leveraging Geometric Properties for Improving SLAM

The problems of generating a global map, and the localization of the robot within that map, are strictly correlated. Indeed, an accurate model of the world leads to a better estimate of the robot's state and, similarly, a good estimate of the current robot's state results in the computation of a better global map. Due to this dependency, SLAM is considered a hard problem to solve.

Besides the case of 2D indoor applications, where the research community reached stable and accurate results, SLAM is still an open problem. SLAM is such a broad topic that it can be considered solved only for particular robot and environment combinations. Moreover, the current state-of-the-art in localization and mapping can only cope with scenarios of moderate size and complexity. When the problem is extended to outdoor applications, or to 3D sensor data, multiple issues arise. Also, dealing with large-scale environments, like dynamic cites, poses critical challenges for current solutions. For all these reasons it is always more important to develop methods that can detect and recover from failures. There is the need to find solutions that are able to deal with noise such as perceptual aliasing, and dynamics aspects of the environment (i.e. moving objects, or appearance/geometric changes due to artificial and natural phenomena like seasons). Additionally, from a computation and space complexity point of view, operating with 3D data in large-scale environments produces a huge jump in the dimension of the problem. This calls for algorithms designed to be parallelized, as well as better model representations that store information in more compact forms.

As introduced before, a modern SLAM system is the aggregation of multiple modules such as point cloud registration methods and optimization frameworks. More specifically, point cloud registration is widely used for reducing the robot's odometry drift, and for computing and validating loop closures. Complementing this, optimization procedures are necessary to compute the map and the robot estimate from the constraints arising by processing sensor data. Since all these sub-systems are interconnected together they can not be considered independent of each other. As a consequence of this, improving accuracy, robustness and efficiency of any part of the SLAM system, naturally leads to better and more robust results of the whole system.

The available solutions for localization and mapping, and their sub-systems, do not completely exploit the underlying geometric structure of the environment surrounding the robot. In this thesis, we present a set of novel algorithms that leverage geometric

properties to improve the results, robustness and efficiency of current state-of-the-art methods for localization and mapping.

1.2.1 Minimizing Surface Normal Difference in Point Cloud Registration

Surface normals, or tangents, are important geometric properties that better characterize the local surface described by a patch of points. Current solutions for registering point clouds exploit such characteristics only for computing the data association, and for scaling the error of corresponding points. This has the result of neglecting relevant clues that indeed play a relevant role during the minimization of the error. Extending the minimization function with such descriptive geometric properties makes the whole system more constrained, thus it contributes to enlarge the convergence basin of the whole system.

Through extensive evaluations and simulations, we show that by including geometric features directly in the minimization function, the convergence of the error is faster and more robust to outliers. We present and detail a complete incremental point cloud registration system that additionally exploits the surface normals also in the minimization function associated to the alignment problem. We named the method Normal Iterative Closest Point (NICP). Moreover, we develop a novel approach for merging point clouds, which is also capable to cope with relatively small dynamics in the environment. Our point cloud registration system demonstrates to be able to outperform other current state-of-the-art methods.

1.2.2 Robot Trajectory as Feature for Loop Closing

During ROVINA, the European project that aims at building robots for the exploration, digital preservation and visualization of archeological sites, we have been committed of developing the 3D SLAM system used by the robot. In this specific case, the goal was the exploration of catacombs. Catacombs are networks of narrow tunnels that can extend for hundreds of meters under the ground, and they are considered hard for mapping. Indeed, in these sites there is almost no illumination, and the appearance of the surroundings looks similar nearly everywhere (sensor aliasing). Under such conditions standard SLAM techniques tend to fail.

In environments like catacombs, the robot movements are very constrained. As a consequence of this, the path followed by the robot represents a salient descriptor in and of itself. Thus, it could be used for improving the overall results of the SLAM system. We propose a method that, by exploiting the geometric shape of the robot's trajectory, reduces the search space for finding candidate loop closures. This, significantly decreases the computation time. Moreover, path shape comparisons provide an additional discriminating factor that contributes to lower the perceptual aliasing

typical of environments like catacombs. Finally, similarly to methods using bag of words on RGB features extracted from color images, our system is potentially capable of recovering from inconsistent global map estimates. We demonstrates the validity of our SLAM approach on real data acquired in the catacombs of Priscilla in Rome.

1.2.3 Bridging 3D Sensor Data and 2D SLAM

While 3D SLAM is still an open problem, the two dimensional case has been widely studied and explored. We live in a transition period of time, where the research community focuses on reaching the well established results of current state-of-the-art 2D SLAM approaches, also in the 3D case. While concentrating in this direction, there is a natural need of solutions that take advantage of the higher information encoded by 3D measurements, in current 2D SLAM methods. At the same time, it is also extremely important, in terms of space and time complexity, to find ways of compressing the information encoded in 3D sensor data into a 2D descriptive model.

We developed a 2D indoor/outdoor SLAM pipeline, named FLAT2D, where we exploit the data produced by 3D range sensors (e.g. Velodyne HDL-32E) to generate rich 2D information. Our method collapses the data into 2D by taking in consideration, and analyzing, the vertical structure contained in the measurements. In this way, we construct two dimensional laser scans that contain a higher and more informative quantity of information with respect to those generated by pure 2D lasers. We demonstrated, through experiments in both indoor and outdoor scenarios, how this information can be used with the most recent and effective 2D SLAM methods. Additionally, FLAT2D shows an efficient management of the memory consumption, that is of fundamental importance when dealing with large-scale environments like those encountered in autonomous driving.

1.2.4 Robust 3D Geometric Primitive Extraction from Sparse Data

Enabling localization using sensors that generate 3D sparse data is still an open problem. In such cases, indeed, standard measurement registration techniques fail. The natural consequence of this is to fall back to the usage of features to determine salient properties of the surroundings. For this reason, developing methods capable of computing robust 3D geometric features from sparse data to be used to localize the robot is of fundamental importance. The advantages of such a system are twofold. First, in terms of costs, the sparser is the data the cheaper is the sensor. Second, it would make the localization insensitive to noise like the season changes, that usually affect normal cameras.

Focusing in this direction, and to achieve robust localization and mapping, we developed an algorithm that exploits base geometric features, extracted from the sparse data generated by a single Velodyne HDL-32E. In this specific case, we efficiently

extract clusters from the point clouds that can be well approximated by 3D lines, or 3D planes. The clusters are computed thanks to an efficient pipeline that, despite the heavy non-uniform and sparse data, is able to compute accurate surface normals. We showed the validity of our approach with experiments on real data acquired with an autonomous car in the MCity test facility.

1.3 Thesis Organization

This thesis is organized in four parts: BASICS, POINT CLOUD REGISTRATION, LOCALIZATION AND MAPPING, and CONCLUSIONS. For completeness, and clarity purposes, we included an additional APPENDICES part. Each part and chapter of the thesis is structured and organized as follows:

Part I, BASICS: Related work for localization and mapping, feature extraction and point cloud registration; Main background techniques and theories that are employed in the thesis.	
Chapter 2	Critical analysis and review of the literature related to localization and mapping, feature extraction and point cloud registration; Analysis of relations among the approaches in literature and discussion of their limitations.
Chapter 3	Review of the fundamental concepts behind the topics discussed in this thesis; Introduction of the notations and terminology.

Part II, POINT CLOUD REGISTRATION: Augmented measurements, point cloud merging, scene update and dynamic object handling for point cloud registration.	
Chapter 4	Least-squares error analysis of a point cloud registration algorithm based on augmented measurements.
Chapter 5	Algorithm using extended measurements and scene merging for efficient and robust point cloud registration.

Part III, LOCALIZATION AND MAPPING: Dense and features based localization and mapping methods for ground robots.	
Chapter 6	Simultaneous Localization And Mapping system for catacomb exploration.
Chapter 7	Fast method for indoor/outdoor localization and mapping exploiting 2D measurements built from 3D sparse data.
Chapter 8	3D feature based localization and mapping for an autonomous vehicle equipped with a sparse sensor.
Part IV, CONCLUSIONS: Brief discussion and concluding remarks.	
Chapter 9	Summary, conclusions and final remarks of the thesis.
APPENDICES: Review of mathematical concepts used by the methods presented in this thesis.	
Appendix A	Mathematical and geometric description of the minimal representations for 3D rotations, 3D lines and 3D planes.
Appendix B	Mathematical derivation of the derivative of a 3D rotation matrix in function of the parameters of a unit quaternion.
Appendix C	Detailed mathematical derivation of the Jacobian used by the point cloud registration approaches presented in this thesis.
Appendix D	The concept of integral image in detail.

1.4 Publications

The content of this thesis is the result of papers published in international journals and conferences. In the remainder of this section these publications are listed chronologically:

2016

- J. Serafin and G. Grisetti, **"Using Extended Measurements and Scene Merging for Efficient and Robust Point Cloud Registration"**, *Robotics and Autonomous Systems (RAS)(submitted, in revision)*, 2016

- J. Serafin, E. Olson, and G. Grisetti, **"Fast and Robust 3D Feature Extraction from Sparse Point Clouds"**, *in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, 2016, pp. 4105-4112. DOI: 10.1109/IROS.2016.7759604

- R. Goeddel, C. Kershaw, J. Serafin, and E. Olson, **"FLAT2D: Fast Localization from Approximate Transformation into 2D"**, *in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, 2016, pp. 1932-1939. DOI: 10.1109/IROS.2016.7759305

- J. Serafin, M. Di Cicco, T. M. Bonanni, G. Grisetti, L. Iocchi, D. Nardi, C. Stachniss, and V. A. Ziparo, **"Robots for Exploration, Digital Preservation and Visualization of Archeological Sites"**, *in Artificial Intelligence for Cultural Heritage, L. Bordoni, F. Mele, and A. Sorgente, Eds., Cambridge Scholars Publishing*, 2016, pp. 121-140. ISBN: 9781443890854

2015

- J. Serafin, and G. Grisetti, **"NICP: Dense Normal Based Point Cloud Registration"**, *in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015, pp. 742-749. DOI: 10.1109/IROS.2015.7353455

- R. Capobianco, J. Serafin, J. Dichtl, G. Grisetti, L. Iocchi, and D. Nardi, **"A Proposal for Semantic Map Representation and Evaluation"**, *in Proc. of the European Conference on Mobile Robots (ECMR)*, Lincoln, United Kingdom, 2015, pp. 1-6. DOI: 10.1109/ECMR.2015.7324198

2014

- J. Serafin, and G. Grisetti, **"Using Augmented Measurements to Improve the Convergence of ICP"**, *in Proc. of the Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, Bergamo, Italy, 2014, pp. 566-577. DOI: 10.1007/978-3-319-11900-7_48

- V. A. Ziparo, D. Calisi, G. Grisetti, J. Serafin, M. Prosmans, L. Van Gool, B. Leibe, M. Di Stefano, L. Petti, W. Burgard, F. Nenci, I. Bogoslavskyi, O. Vysotska, M. Bennewitz, and C. Stachniss, **"A User Perspective on the ROVINA Project"**, *in Proc. of the 18th ICOMOS General Assembly and Scientific Symposium "Heritage and Landscape as Human Values"*, Florence, Italy, 2014, pp. 578-582. ISBN: 9788849530575

2013

- V. A. Ziparo, M. Zaratti, G. Grisetti, T. M. Bonanni, J. Serafin, M. Di Cicco, M. Proesmans, L. Van Gool, O. Vysotska, I. Bogoslavskyi, and C. Stachniss, **"Exploration and mapping of catacombs with mobile robots"**, *in IEEE Int. Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Linköping, Sweden, 2013, pp. 1-2. DOI: 10.1109/SSRR.2013.6719380

- I. Bogoslavskyi, O. Vysotska, J. Serafin, G. Grisetti, and C. Stachniss, **"Efficient Traversability Analysis for Mobile Robots using the Kinect Sensor"**, *in Proc. of the European Conference on Mobile Robots (ECMR)*, Barcelona, Spain, 2013, pp. 158-163. DOI: 10.1109/ECMR.2013.6698836

Part I

BASICS

Chapter 2
Related Work

"Professor Johnston often said that
if you didn't know history, you didn't know anything.
You were a leaf that didn't know it was part of a tree."
— Michael Crichton, Timeline, 1999

This Chapter contains an overview of the current state-of-the-art regarding the main subjects addressed in this thesis. More in detail, in Sec. 2.1 we present a survey of the previous literature for robot localization and mapping. In Sec. 2.2 and Sec. 2.3, instead, we explore related work respectively for the problem of feature extraction, and point cloud registration.

2.1 Simultaneous Localization And Mapping

Simultaneous Localization And Mapping (SLAM)

Simultaneous Localization And Mapping is the problem for which a mobile robot can build a map of the surrounding environment, while at the same time localize itself in that map. The localization and mapping processes are not independent, and for this reason SLAM is considered a hard problem to solve. Indeed, it can be seen as the chicken egg problem. In order to build a good map, it is necessary for the robot to be correctly localized. At the same time, a correct localization depends on the availability of a good map.

In SLAM, maps can be classified in sparse and dense maps. The choice of a particular map representation depends on the type of sensors, on the characteristics of the environment, and on the estimation algorithm used. Sparse maps [27][28][29] are usually preferred in environments where locally distinguishable features can be identified, and especially when RGB cameras are employed. On the other hand, dense representations [11][30][31][32] are commonly used in conjunction with range sensors like lasers or depth cameras. Independently of the type of the representation, the map is defined by

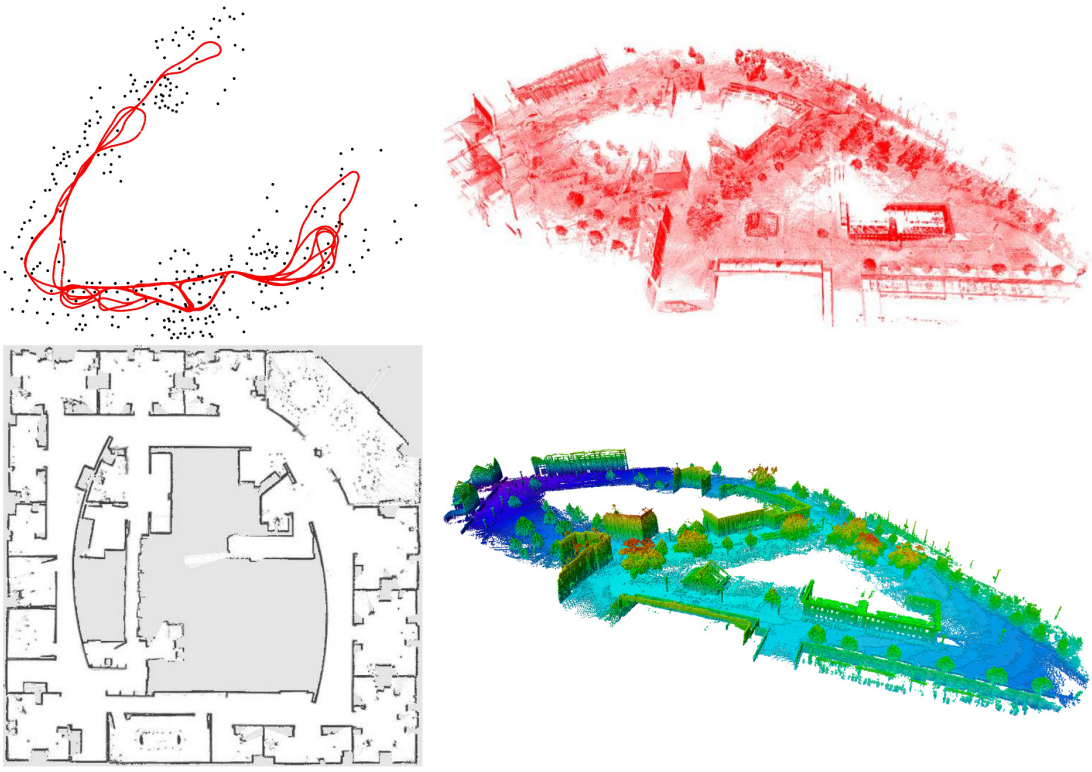


Figure 2.1: Examples of map representations. From left to right and from top to bottom: Victoria Park 2D feature/landmark map (sparse), University of Freiburg campus 3D octree map (dense), Intel Research Lab 2D occupancy grid map (dense) and University of Freiburg campus 3D point cloud map (dense). (Images from [11][17])[35][36].

the measurements and the locations where they have been acquired [33][34]. Fig. 2.1 illustrates typical map representations.

The SLAM problem has been studied for more than 30 years. As suggested in the survey of Cadena *et al.* [1], this time window can be divided in three different main periods:

- classical age (1986-2004)
- algorithmic-analysis age (2004-2015)
- robust-perception age (2015-today)

During the classical age the main probabilistic formulations of SLAM have been introduced. The acronym SLAM and its first definition was introduced by Durrant-Whyte *et al.* in [37], however the first probabilistic SLAM problem born in a series of

papers by Cheeseman and Smith [27][38]. These works demonstrated that landmark estimates are all correlated with each other by the common error of the estimated pose of the vehicle [39].

Filtering

SLAM algorithms can be classified accordingly to the estimation technique used: filtering or smoothing. Filtering approaches rely on recursive Bayesian estimation, also known as Bayes filtering. A Bayes filter is a general probabilistic algorithm for estimating the current belief of a robot from input measurements and control data. In SLAM, the term belief indicates the robot's internal knowledge about the state which includes all relevant aspects of the robot, and of the environment, that can impact the future: for example robot poses and landmark positions. The estimate is augmented and refined by incorporating new measurements as soon as they become available. To highlight their incremental nature, filtering approaches are usually referred to as on-line SLAM methods. Dynamic Bayesian Networks (DBN) are a natural representation to describe filtering processes that can be used to tackle the SLAM problem. Indeed, a DBN highlights the temporal structure of the process described by the network.

The big issue with the general Bayes filter formulation is that in most of the cases it is not computationally tractable. A solution to this problem was given by the introduction of a family of recursive state estimators called Gaussian filters. The first Gaussian filters used for solving SLAM problems are the Kalman (KF) and Extended Kalman Filters (EKF) [27][40]. The difference between KF and EKF is that the former works under the assumption that the system is linear, the latter instead can be used also on non-linear applications. However, both filters require the transition and the sensor models to be Gaussian distributed. The strength of such methods can be found in the fact that they compute a fully correlated posterior probability over landmark positions and robot poses. At the same time, their major weakness resides in the strong assumptions that have to be made on the robot motion model and sensor noise. Indeed, if even only one of these assumption are violated, these kind of filters easily diverge. Always in this direction, Paskin [41] solved the SLAM problem using thin junction trees, thus decreasing the complexity with respect to EKF approaches.

Kalman Filter (KF)

Information Filter (IF)

Another type of Gaussian filters widely used in SLAM are the so called Information (IF) and Extended Information Filters (EIF). The information filter is a variant of the Kalman filter that uses the canonical parameterization, instead of the moment parameterization, to represent the Gaussians [42]. IF and EIF exhibit properties similar to Kalman filters, but they can better handle cases where variables are affected by infinite noise (zero information), or in other words when there is global uncertainty. Whereas the computational bottleneck in the Kalman filter is the update, in the information filter the expensive step is the prediction. Another issue of the IF is that to retrieve the mean from the information vector an inversion of the information matrix is required. On the other hand, the main advantage is that keeping an extended representation for the states results in a sparse pattern of the information matrix that allows substantial speedups in the computation. Eustice *et al.* [14] and Thrun *et al.* [43] proposed methods

based on Sparse Extended Information Filters (SEIF). The former showed a technique to accurately compute the error-bounds in the SEIF framework, reducing in this way the possibility to become overly confident. The latter, instead, used the inverse of the covariance matrix to reduce the error on the robot poses.

Particle Filter

A third family of Bayes filters is that of Particle filters. Particle filters work by representing the posterior belief with a set of random state samples drawn from the posterior distribution itself. Such samples are called particles. This kind of filters perform three main steps. Firstly, they compute a prediction of the next state for each particle and, successively, they calculate an importance factor for each of them. Importance factors are used to incorporate the last measurement into the particle set, and the weighted particle set represents an approximation of the Bayes filter posterior belief. Finally, they perform an importance resampling. The algorithm draws and replace all particles from the current set of particles. The probability of drawing each particle is given by the importance weight computed at previous step. The resampling process transforms a particle set into another set of the same size. By incorporating the importance weights into the resampling process the distribution of the particles change, and after the resampling they are distributed approximately according to the posterior belief. Murphy in [44] used Rao-Blackwellized particle filters to solve the SLAM problem, in this kind of approach each particle represents a possible trajectory of the robot. This implementation has been extended by Montemerlo *et al.* [28][29] to handle SLAM with landmarks. A combination of the work of Hähnel *et al.* [31] and Montemerlo was proposed by Grisetti *et al.* [15][32] where they apply fastSLAM-2 [29] to the case of grid maps.

Rao-Blackwellized Particle Filter

Despite all the improvements obtained with several years of research, filtering SLAM methods suffer of multiple issues. In particular, due to the way they model the localization and mapping problem, the existing families of Bayes filters poorly scale with the dimension of the state, and of the map. Moreover, they do not smoothly handle non-linearities. It has also been demonstrated by Strasdat *at al.* in [45] that, in the case of visual SLAM, key-frame bundle adjustment [46][47][48] outperforms filtering.

Smoothing

Conversely from filtering methods, smoothing approaches estimate the full trajectory of the robot from the full set of measurements [49][50][51]. These approaches address the so called full SLAM problem, and they typically rely on least-squares error minimization techniques. These use an alternative SLAM representation to the DBN known as graph-based or network-based formulation. Such a formulation highlights the underlying spatial structure of the problem. More in detail, in graph-based SLAM landmarks and robot poses are represented by nodes, or vertices. Instead, spatial constraints among nodes, that result from robot observations or from odometry measurements, are encoded as edges, or factors. Once the graph is constructed the goal is to find the configuration of the nodes that best satisfies the constraints given by the edges. An important and challenging aspect of this representation is the computation of the data association, known also as loop closure detection. The loop closure problem

Graph-Based SLAM

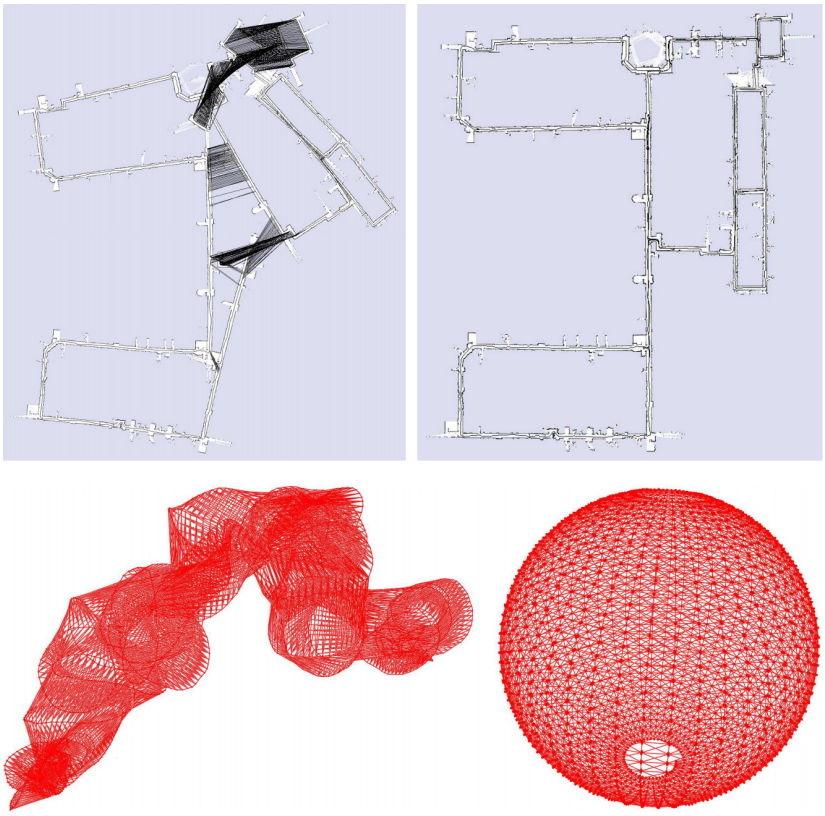


Figure 2.2: Examples of factor graphs before (left) and after (right) the optimization. Top: 2D pose-graph corresponding to the MIT Killian Court dataset. Bottom: 3D pose-graph generated by simulating a robot moving on the surface of a sphere. (Images from [17]).

aims at finding past robot poses (far in time) that are near to its current position (close in space). Identifying when a robot has returned back to a previous location allows to reduce drastically the error on the map, and it increases the robustness of the entire algorithm. The main drawback of the data association is that even only one incorrect correspondence can lead the algorithm to terrible mapping results. During a loop closure two kinds of errors could occur: false positive or miss detected matchings. The former happens when the robot closes a loop with a wrong match. The latter, instead, is the result of a missed closure due to a failure of the measurement registration.

The graph-based SLAM problem is usually decoupled in two different steps: the generation of the graph from the raw measurements, and the computation of the most likely configuration of the nodes given the edges (graph optimization). These two steps are called respectively front-end and back-end. The former is heavily sensor dependent, while the latter relies on an abstract representation of the data which is sensor agnostic.

Front-End/Back-End

Smoothing methods represent the current state-of-the-art for solving SLAM problems. Fig. 2.2 shows examples of factor graphs before, and after the optimization.

During the past years many approaches that rely on network-based SLAM have been proposed. The concept of graph SLAM born in the seminal paper of Lu and Milios [49]. Gutmann and Konolige in [52] developed an efficient method for constructing the graph and for detecting loop closures while running an incremental estimation algorithm. The ATLAS framework [53] creates a two-level hierarchy of graphs and employs a KF to build the bottom level. A global optimization approach is then used to align the local maps at the second level. Like ATLAS, Estrada *et al.* proposed Hierarchical SLAM [54] as a way for using independent local maps. Olson [55] presented a front-end with outlier rejection based on spectral clustering. To perform data association, statistical approaches such as the joint compatibility [56], or the χ^2 error tests are often applied. Nüchter *et al.* in [57] built an integrated SLAM system for 3D mapping. The main goal was to improve the construction of the network, while using for the optimization a 3D variant of the approach of Lu and Milios [49]. Dellaert *et al.* [51] and Ranganathan *et al.* [58] developed an approach known as square root Smoothing And Mapping (SAM). With respect to EKF methods, it handles non-linearities better and it is faster to compute. Moreover, SAM gives an exactly sparse factorization of the information matrix and it can be used to incrementally acquire 2D and 3D maps. Successively, Kaess *et al.* [59] proposed iSAM, an on-line mapping approach. More recently other full graph-based SLAM systems have been proposed like Kintinuous [60], RTAB-SLAM [61] and ORB-SLAM [62]. Whelan *et al.* [63] presented ElasticFusion, where they use surfels, instead of poses, to populate the graph.

ATLAS Framework

Smoothing And Mapping (SAM)

Incremental SAM

Kintinuous, ORB-SLAM, ElasticFusion

In the so called algorithmic-analysis age, researchers focused on the fundamental properties of the SLAM problem. These include observability, convergence, and consistency. In this period, the key role of sparsity as a requirement for efficient SLAM solvers was discovered, and the principal open-source SLAM libraries were developed. Current SLAM libraries like g2o [16], GTSAM [64], Ceres [65], SLAM++ [66] and iSAM [59] can solve problems with tens of thousands of variables in few seconds. In coincidence with these libraries, many approaches for minimizing the error in the constraint network have been proposed. Howard *et al.* [67] used relaxation to localize the robot and build a map. Frese *et al.* in [68] developed a variant of Gauss-Seidel relaxation known as Multi-Level Relaxation (MLR). The idea behind this approach it to apply relaxation at multiple resolutions. Dellaert and Kaess [51] were the first to exploit sparse matrix factorization to solve the linearized problem in off-line SLAM. Kaess *et al.* [59], in iSAM, use partial reorderings of the variables to compute the sparse factorization. Recently, Konolige *et al.* in [69] proposed an open-source implementation of a pose-graph method that constructs the linearized system in an efficient way. Olson *et al.* [50] developed an optimization approach based on the stochastic gradient descent that is able to efficiently correct even graphs of large size. Grisetti et al. [70] extended Olson's approach to use a tree parameterization of the nodes in both 2D and 3D cases. This

g2o, GTSAM, Ceres, SLAM++

Multi-Level Relaxation (MLR)

has the effect of increasing the convergence speed. GraphSLAM [71] applies variable elimination techniques to reduce the dimension of the optimization problem. *GraphSLAM*

2.1.1 LIDAR-Based Localization

In the literature, the idea of using 2D matching methods for solving 3D tasks is not new. Thrun *et al.* [72] developed a 3D map-building system based on 2D scan-matching. They construct a full 3D polygon map upon the data generated by an additional, upwards facing sensor. In order to keep a 3D sparse representation, they eliminate overly similar polygons. However, this system only uses a planar LIDAR for scan matching purposes, thus missing potentially useful off-plane features. Nobili *et al.* [73] directly compared the performance of a 2D SLAM system based on both a planar and a 3D LIDAR. The conclusion of this analysis is that, while the planar configuration resulted in a slightly more accurate localization, the maps built with a VLP-16 were more robust against environmental changes.

Accurate registration of three dimensional data is a significant challenge for 3D methods. In 2D, Olson, Diosi, and Kleeman [74][75] developed approaches for fast and accurate alignment of scans. Despite this, in 3D spaces, most applications still employ some variation of the ICP [26] algorithm. ICP can work well with dense data, but it performs poorly when dealing with matching between scans consisting of non-uniform rings of data, as in the case of Velodyne sensors. Extended ICP versions [12] have been proposed to improve the overall convergence properties of the algorithm. However, they are still designed to work with dense data in mind, such as that from RGB-D sensors or depth cameras. Nonetheless, some 3D SLAM variants based on 3D LIDAR exist.

Moosmann and Stiller [76] introduced a SLAM method specifically targeting the Velodyne HDL-64E sensor. Relative transformations between scans are recovered using ICP. The authors modify the nearest-neighbor data association search to also take into account the surface normals at each point, making their search more robust to errors and outliers. Wolcott and Eustice [77], instead, demonstrated a method for localization based on a prior 3D ground map. Such an approach exploits the intensity information to match this prior model against data from a inexpensive monocular camera. By focusing on ground-only matching, they partially eliminate the need of storing a 3D dense representation of the prior. This algorithm performs comparably to purely LIDAR-based methods. However, it does not handle well occlusions due to dynamic obstacles or weather conditions (i.e. snow or puddles).

To address these issues, Wolcott and Eustice [78] developed a modified version of the approach proposed in [77]. Dense 3D data is compressed into a 2D grid, where each cell contains a Gaussian mixture map summarizing the vertical structure. This greatly reduces the storage needed for the prior map. They extended Olson's multi-resolution scan matcher [75] to handle mixture map representations, obtaining scan matches for maps with a resolution of 20cm, and working frequency of 3-4Hz.

2.1.2 Camera-Based Localization

FAB-MAP

Another class of 3D localization methods focuses on extracting high level features rather than aligning the full scans. For example, Cummins and Newman [79] presented FAB-MAP, an appearance-based SLAM system. FAB-MAP recognizes previously visited places based on a visual vocabulary, even in the presence of varying visual conditions. However, the system is better suited for identifying large-scale loop closures, rather than precisely localizing the vehicle within the environment.

Sattler *et al.* [80] developed a method for registering images to a database of 3D point models. The authors find correspondences between an image, and a model, by comparing Scale Invariant Feature Transform (SIFT) [81] descriptors extracted from each. RANdom SAmple Consensus (RANSAC) [82] can be then used to estimate the camera pose. The method, however, is only sufficient for registering pose within several meters, and still heavily depends on dense 3D priors.

2.1.3 Extracting 2D Map Information From 3D Data

Strip Histogram Grid (SHG)

Korah *et al.* [83] introduced the Strip Histogram Grid (SHG), a variation of voxel grid methods, based on polar coordinates, created to improve recognition of structure in the environment. SHG is able to encode information over larger areas than traditional voxel-based representations, thus allowing the identification of vertical structure over larger regions. The authors validate its application to 3D scene segmentation, for dense models in urban like environments.

Height-Length-Density (HLD)

Morton and Olson [84] presented the Height-Length-Density (HLD) classifier for identifying terrain based on 3D LIDAR data. This classifier handles both positive and negative obstacles by distinguishing cells based on height, slope, and data density. Additionally, they propagate this information through the map to fill in unobserved regions. This method is tailored for obstacle detection needed by navigation systems, but it can be adapted for localization purposes. However, the message-passing part of the algorithm is expensive, thus posing heavy limitations in high-speed environments.

Stanford's DARPA Grand Challenge Team [85] extracted obstacle information from Velodyne data by comparing the distance between the concentric rings generated by each laser during a complete sweep of the sensor. As the terrain changes, the rings compress. Measuring the distance between beams, and comparing it with expected distance values, very small obstacles (on the scale of a curb) can be reliably detected.

2.2 Feature Extraction

Feature extraction is the problem of computing salient descriptors, or key-points, from a set of input measurements. Such descriptors have the goal of capturing unique and discriminant local properties of the input data, while at the same time being invariant

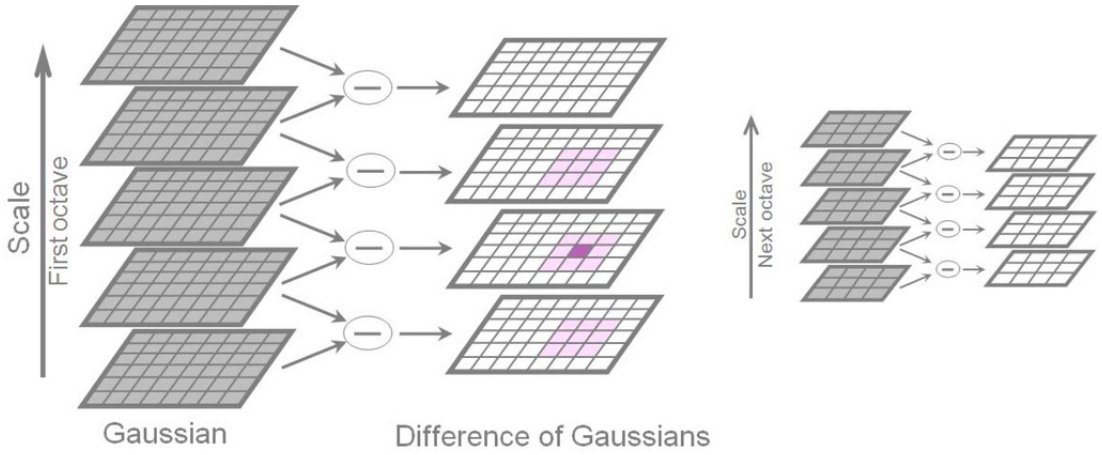


Figure 2.3: Diagram showing the construction of the pyramid of Difference of Gaussians, and the scale-space extrema detection process of SIFT. To build the first octave, the input image is convolved with Gaussians generated with different standard deviation values of σ. Contiguous Gaussian images are then subtracted to produce Difference of Gaussians. Once the octave is computed, the source image is scaled by a factor of 2 and the process repeated. Light pink squares highlight the neighborhood used for searching local extrema on a candidate pixel (dark pink) over scale and space. (Image from [86]).

to the point of view from which they are observed. Moreover, by defining and using distance metrics between descriptors, features can be compared between each other. For more than two decades the research community proposed possible solutions to this problem, leading to the birth of many different algorithms. These methods can be divided in classes based on the input data type, namely full point clouds and RGB, range or RGB-D images.

Scale Invariant Feature Transform (SIFT)

Among the approaches relying on RGB images, the Scale Invariant Feature Transform algorithm by Lowe [81] is considered one of the major breakthrough for the computation of feature descriptors. SIFT extracts key-points by performing two main steps designed to make the features invariant to scale and orientation. First of all a scale filtering is applied to the input image. For computation time reasons, this is done by using Difference of Gaussians (DoG) [87], an approximation of the Laplacian of Gaussian (LoG) filter [88]. A DoG is obtained through the difference of the Gaussian blurring of an image computed with two different standard deviations σ_1 and σ_2. Using Gaussian kernels with high σ values lead to the detection of small corners and vice versa. In this case σ acts as a scaling factor. In order to find blobs belonging to corners of arbitrary size (scale invariance), for a fixed dimension of the input image SIFT computes DoGs for different values of σ. This set of Difference of Gaussians is called octave. To further enhance scale invariance, SIFT performs this process following a pyramidal approach by computing DoG octaves for different sizes of the input image. At this point local

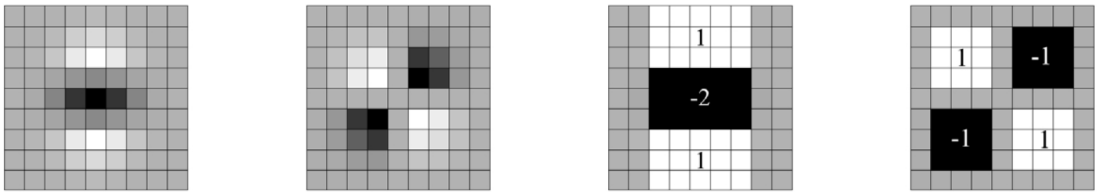


Figure 2.4: Example of box filters used in SURF [90]. From left to right: discretized and cropped Gaussian second order partial derivatives along the y and xy directions, and the associated box filter approximations. The gray regions are equal to zero. (Image from [90]).

extrema, or key-points candidates, are computed analyzing neighborhoods of pixels in the DoGs over scale and space. As an example, one pixel in an image is compared with its 8 neighbors points and the 9 pixels in the next and previous scales. Fig. 2.3 contains a diagram showing graphically the DoGs and the scale-space extrema detection process. As a refining step, in order to have more accurate locations of local maxima, SIFT uses the Taylor expansion of the scale-space function. Additionally, since DoGs are sensitive to edges, they look at the principal curvature (or the eigenvalues ratio) of the 2×2 Hessian matrix associated to the key-point candidates. To generate the final descriptor, one first extracts the "principal orientation" of the key-point by means of a voting scheme applied to the gradient directions. This is done taking in consideration a neighborhood of the key-point. These gradient directions, normalized with the extracted principal orientation, are then exploited to generate a 128 bins histogram that reports the distributions of such directions in a number of cells around the key-point. The highest peaks in the histogram are taken, and any peak above 80% of it is also considered to calculate the orientation. Note that this can lead to the generation of key-points with same location and scale, but different directions. This has been shown to contribute to the stability of the matching phase. SIFT descriptors are reported to be both robust and accurate, albeit computationally expensive. This makes the approach hard to be used in real-time applications. Successively Ke and Sukthankar in [89] extended SIFT to use Principal Component Analysis (PCA). PCA based local descriptors are more distinctive, more robust to image deformations, and more compact than the standard SIFT representation.

Speeded Up Robust Features (SURF)

At the same time, Bay *et al.* [90] published a new algorithm called SURF (Speeded Up Robust Features), an extension of SIFT that relies on integral images to perform image convolution. They further simplify the Difference of Gaussians used in SIFT with approximations represented by box filters (see Fig. 2.4 for an example). Note that box filters can be computed efficiently by using integral images, and they can be evaluated in parallel for different scales. The orientations, instead, are computed by using wavelet responses along the horizontal and vertical directions of a neighborhood of the key-points. SURF estimates the principal orientation by taking the sum of the

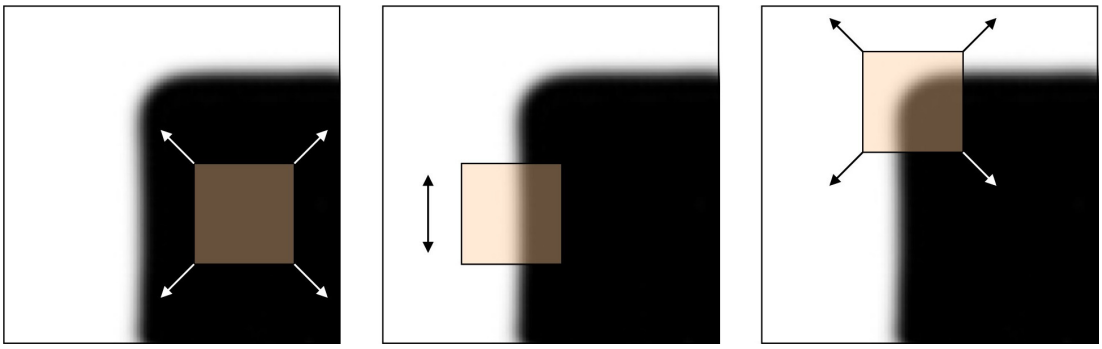


Figure 2.5: Schema highlighting the idea behind the Harris corner detector [91]. Left: no change in all directions (flat region). Middle: no change along the edge direction (edge). Right: significant change in all directions (corner). (Image from [92]).

responses within a sliding orientation window. Like for box filters, also in this case integral images are used to enhance the computation time of the whole algorithm. SURF descriptors need less memory than SIFT and it has been demonstrated they are 3 times faster while having comparable results. Also, SURF can reject matching features based on the sign of Laplacian at no additional costs. However, this approach is not well suited to handle viewpoint and illumination changes.

Features from Accelerated Segment Test (FAST)

Rosten and Drummond in [2] developed a corner detector algorithm called FAST (Features from Accelerated Segment Test), an enhanced version of the original approach proposed by Harris and Stephens in [91]. Fig. 2.5 shows a graphical example of the idea behind the Harris corner detector. For each pixel, instead of doing a derivative of the image, FAST checks the circle of 16 pixels surrounding the query picture element. If there exists a minimum set of n contiguous pixels, with higher or lower intensity, then the pixel is added as key-point candidate. Usually n is selected to be 12 so that the high speed test on the 16 neighboring pixels can be performed. Fig. 2.6 shows the 12 point segment test corner detection used in FAST. However, this kind of features present some limitations:

- the high-speed test cannot be generalized well for $n < 12$;
- the efficiency of the detector depends on the choice and ordering of the selected surrounding pixels, and it is unlikely that the chosen pixels are optimal;
- multiple features are detected adjacent to one another.

To solve these issues Rosten *et al.* [93][94] developed FAST-ER. In this improved version of the algorithm, the first 2 problems are addressed by applying a machine learning approach for determining the best ordering for the chosen neighboring pixels, thus boosting the computation time. Additionally, to overcome the third point listed above,

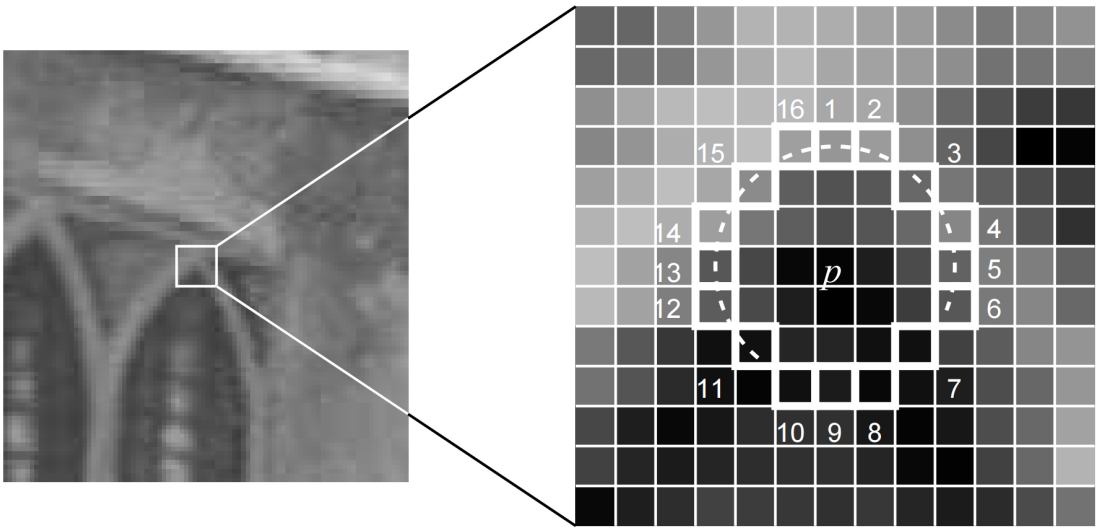


Figure 2.6: The 12 point segment test corner detection used in FAST [2]. The image on the right is a magnification of a part of the picture on the left. The highlighted squares are the 16 circle pixels used in the corner detection. The pixel at p is the center of a candidate corner. The arc indicated by the dashed line passes through 12 contiguous pixels which are brighter than p of more than a certain threshold. (Image from [2]).

FAST-ER uses a non-maximum suppression method for rejecting adjacent key-points. Differently from SIFT and the Harris corner detector, FAST and FAST-ER have shown to be able to run in real-time at the cost of being more sensible to high levels of noise, and different parameter values.

Oriented FAST and Rotated BRIEF (ORB)

More recently, Rublee *et al.* [4] developed ORB (Oriented FAST and Rotated BRIEF [3]) features, a faster and more efficient alternative to SIFT and SURF. ORB is a fusion between FAST and BRIEF (Binary Robust Independent Elementary Features) descriptors with many modifications for increasing the performance. This method firstly computes key-points using FAST, and it applies a corner measure to find the best n point features among them. This is done following a pyramidal approach to obtain scale invariance. One of the problems with FAST features is that they do not have orientations. ORB solves this issue by computing the centroid of the patch centered at the located corner. The direction of the vector from this corner point to the centroid represents the orientation of the key-point. To increase rotation invariance, moments are computed on a circular region centered in the key-point, of radius equal to the size of the patch. The features are represented using binary descriptors based on BRIEF. The main advantage of these kind of descriptors is that they have high variance. High variance makes a feature more discriminative since it responds differentially to inputs. On the other hand, BRIEF performs poorly with rotations. For this reason, to enhance

the descriptors with respect to the orientation, the feature vectors are rotated according to the orientation of the key-points. Unfortunately, once the rotation is applied, the BRIEF descriptor properties introduced before do not hold anymore. To overcome this problem, ORB runs a greedy search among all possible binary tests to find those having both high variance and low correlation. The result is called rBRIEF. ORB features are considerably faster than SURF and SIFT, and they show better results with respect to SURF. ORB is particularly well suited also for low-power systems.

Point Feature Histograms (PFH)

Another class of feature detectors are those that work directly on point clouds. In this direction, a major contribution was given by Rusu *et al.* in [95] who developed the Point Feature Histograms (PFH), an extension of the work of Wahl *et al.* [96]. PFH formulation tries to encode the geometric properties of the neighborhood of a point by generalizing the mean curvature around it. This is done by using a multi-dimensional histogram of values. Such a histogram represents an informative signature for the feature, and it is invariant with respect to the 6D pose of the underlying surface. Moreover, it is resilient versus different sampling densities or noise levels of the neighborhood. The PFH descriptor tries to capture the surface variations looking at all the interactions between the directions of the estimated normals. All pairs of points inside a 3D sphere centered on a query point $\mathbf{p}_q$ are taken in consideration. For each pair of points, a fixed Darboux frame coordinate system is defined on one of the two. This frame is then used to express the difference between the surface normals. Such a difference is represented through the pan, tilt and yaw angles that, together with the Euclidean distance between the points, form a quadruplet. This quadruplet represents the relation between a pair of points. To create the final PFH descriptor, the set of all quadruplets is binned into a histogram. The left picture of Fig. 2.7 shows the influence region diagram for the computation of PFH for a query point $\mathbf{p}_q$. While PFH has shown to be able to produce distinctive descriptors, it has his main drawback in the computational complexity. Indeed, since a PFH feature is computed as a histogram of relationships between all pairs of points of a neighborhood, it has a computational complexity of $\mathcal{O}(k^2)$ with k being the number of points in the neighborhood of $\mathbf{p}_q$.

Fast Point Feature Histograms (FPFH)

To overcome this issue, Rusu *et al.* in [97] developed a variant called Fast Point Feature Histograms (FPFH). FPFH reduces the computational complexity of PFH from quadratic to linear in the number of neighbor points k. This is done by following two main steps. First FPFH computes the angle relationships, used in the PFH case, between the query point $\mathbf{p}_q$ and all its neighbors. These are called Simplified Point Feature Histogram (SPFH). Then, for each point, the k neighbors are re-determined and the neighboring SPFH values are used to weight the final histogram of $\mathbf{p}_q$. The weighting factor is represented by the Euclidean distance between the query point $\mathbf{p}_q$ and its neighbors. The right picture of Fig. 2.7 shows the influence region diagram for the computation of FPFH of a query point $\mathbf{p}_q$. FPFH descriptors have shown to be much faster than PFH features while maintaining a similar level of descriptive power.

Successively Rusu *et al.* in [98] further extended the FPFH descriptors by devel-

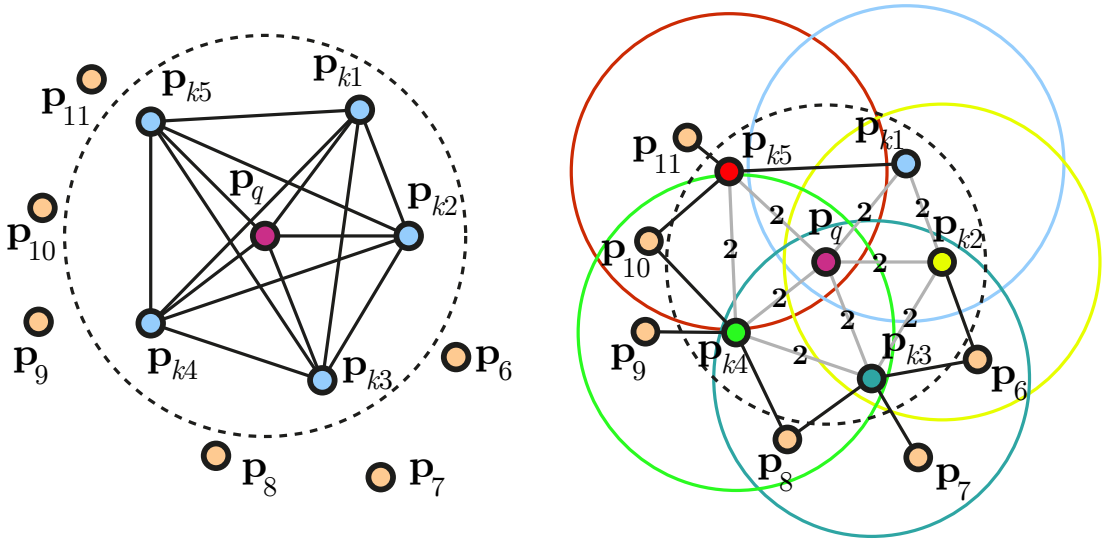


Figure 2.7: Schema showing the diagram of influence of a k-neighborhood set of points centered at $\mathbf{p}_q$ for the computation of PFH (left) and FPFH (right) descriptors. (Images from [97]).

Viewpoint Feature Histogram (VFH)

oping the Viewpoint Feature Histogram (VFH). The VFH combines two factors: a viewpoint and an extended FPFH component. The viewpoint is mapped by iterating over all points, and calculating the angle between the surface normals and the direction between the viewpoint and the centroid of the point cloud. For the extended component, instead, VFH just computes the FPFH at the centroid considering all the points of the cloud as neighbors. At this point the two histograms are added together. The main issue with VFH features is that, if fundamental points belonging to the object are missing due to occlusion or sensor limitations, the computed descriptors can be very different. To overcome this issue, Aldoma *et al.* [99] extended VFH by publishing an enhanced version called Clustered Viewpoint Feature Histogram (CVFH). The idea behind CVFH is to determine stable regions of the point cloud called clusters. From the point cloud a new cluster is created from a random point that has not yet been assigned to any cluster. A point is added to that cluster if there exists a point already in the cluster whose normal is similar and it is in a direct neighborhood. Clusters having too few points are simply rejected. More recently, Wohlkinger and Vincze [100] published the Ensemble of Shape Functions (ESF) features, their idea is to integrate information from multiple points of views. All these methods, however, show their limitations when they have to deal with sparse point clouds.

Ensemble of Shape Functions (ESF)

Li *et al.* [101] adapted the Kanade-Tomasi corner detector [102] to work with 2D and 3D LIDAR data. One of the main problems with this kind of sensors is related to the very low density of measurements in the point clouds. Li *et al.* [103] also published

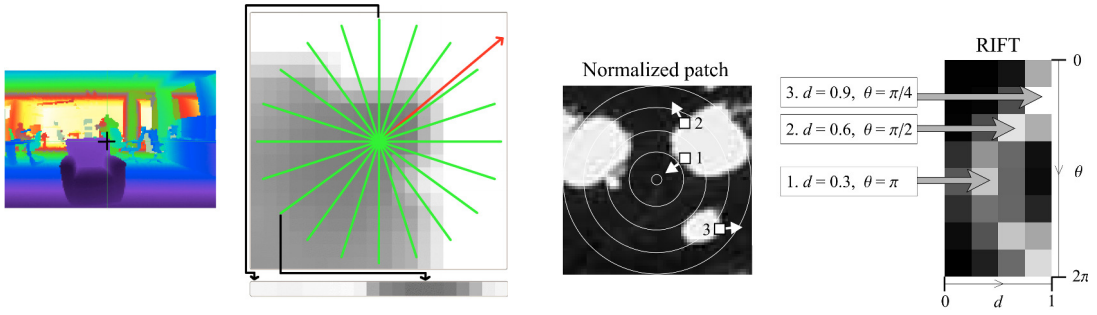


Figure 2.8: Left: a NARF descriptor is visualized on the bottom of the magnification of the key-point highlighted on the left image. Each of the 20 cells of the descriptor corresponds to one of the beams (green) visualized in the patch, with two of the correspondences marked with black arrows. The red arrow shows the dominant orientation. Right: construction of RIFT descriptors. Three sample points, in the normalized patch, map to three different locations in the descriptor. (Images from [5][104]).

a general purpose feature detector using structure tensors of surface normals.

Normal Aligned Radial Feature (NARF)

Steder *et al.* [5] developed the Normal Aligned Radial Feature (NARF). This descriptor is computed directly on a range image that, in case of need, it can be generated from an input raw point cloud. The idea is to look for regions whose surface is stable, but that also exhibits substantial change in the neighborhood. For each point NARF creates a small image patch by looking at it along its normal. The normal is considered as the z-axis of the image patch's local coordinate system where the point is at $(0,\ 0)$. The y-axis is the same as the one of the world coordinate system, while the x-axis aligns accordingly. All neighbors within a certain radius around the point are transformed with respect to this local coordinate system. At this point, a star pattern with n beams having score in the interval $[-0.5,\ 0.5]$ is projected on the image patch. Beams will have a high score if there are many changes of intensity in the cells lying under the beam. This computation is done by comparing each cell with the adjacent one. Moreover, pixels closer to the center contribute to the score with a higher weight. As last step, NARF computes the dominant orientation of the patch to achieve invariance against rotations. The left picture of Fig. 2.8 shows an example of NARF descriptor.

Rotation-Invariant Feature Transform (RIFT)

Lazebnik *et al.* [104] developed the Rotation-Invariant Feature Transform (RIFT) feature. The particularity of this approach is that it fuses ideas from both the RGB and the Euclidean space domains. Given a query point, a circular normalized patch centered in that point is selected. This patch is then divided into concentric rings of equal width, and a gradient orientation histogram is computed within each ring. RIFT assigns all neighbors of the query point to a histogram bin according to their distance and gradient angle position. This angle is the angle between the gradient direction, and the vector starting from the center and pointing outwards the circle. The right picture of Fig. 2.8 shows an example of RIFT descriptor.

2.3 Point Cloud Registration

Point cloud registration is the problem of finding the rigid transformation between two input clouds that maximizes the overlap between them. This problem has been tackled for more than 20 years, and the available methods can be categorized in two main groups: sparse approaches (global) that rely on few meaningful points in the scene, and dense methods (local) that directly operate on the entire set of points. Sparse approaches compute data association based on the local appearance of points (features). It is possible to use these methods also when no prior information about the relative pose between the clouds is available, at the cost of a more complex system. On the other side, dense approaches align two clouds by considering every point, and using simple heuristics to perform the data association. Dense methods are usually faster and easier to implement than sparse approaches, and therefore they are well suited for many robotics applications, like tracking, that require high frame rate. As a drawback however, they are more sensitive to wrong initial guesses. Since for the problems addressed in this work dense (or local) approaches are of utmost importance, only a brief overview of global methods is given.

When no initial guess is available, the registration aims at solving the full global problem of finding the best alignment over the 6 Degrees of Freedom (DoF) space of all possible rigid transforms (rotation and translation). Since rigid transforms are uniquely determined by 3 non-degenerate pairs of corresponding points, a popular approach is to use RANSAC [82] to find the registering triplet of point pairs [82][105][106]. Such methods, however, show their main limitations in the computational complexity that, in the worst case, is cubic in the number of points. Many variants have been proposed to overcome this problem: non-linear optimization to reduce distance between scan pairs [107][108], hierarchical representation in the normal space [109], branch-and-bound using pairwise distance invariants [110], stochastic super-symmetric tensors to represent the constraints between the tuples [111] or, as an alternative to RANSAC, evolutionary game theoretic matching [112][113]. In this category the reference method has been the 4PCS algorithm by Aiger *et al.* [114] which proposed to use special four points instead of triplets as basis in RANSAC. 4PCS brought the computational complexity from cubic to quadratic. Successively, Mellado *et al.* in [115] developed Super 4PCS an enhanced version of 4PCS.

Iterative Closest Point (ICP)

When point clouds are acquired at nearby locations, or a initial guess is given, local registration algorithms are usually used to refine their alignment. The Iterative Closest Point algorithm [26] is one of the earliest and most used dense techniques for registering point clouds. It is an iterative algorithm that refines an initial estimate of the relative transformation. At each step ICP tries to match pairs of points between the two clouds starting from the current transform estimate. Once these correspondences are determined, an improved transformation is usually computed through the Horn [116] formula that minimizes the Euclidean distance between corresponding points. How-

ever, multiple optimization strategies have been presented and a good overview is given by Rusinkiewicz and Levoy in [117]. For simple point distance error metrics, additional techniques for finding a solution to the alignment problem are the Singular Value Decomposition (SVD) [118], quaternions [119] and dual quaternions [120]. Other objective functions for point cloud alignment rely on histogram correlation [121], tensor voting [122], or the Hough transform [81][123]. After its introduction ICP has been subject of several improvements, and many variants have been proposed. It is possible to classify these variants depending on modifications to one of the following stages of the original formulation of ICP:

- input data filters (data enhancement);
- data association (or correspondence search);
- error metric minimization (or optimization).

The input point clouds are usually preprocessed by applying some types of data filters. These filters try to increase the distinctiveness of the measurements. When using range images this can mean decimating the points by computing features or descriptors. As an example, a depth image has a uniform (a grid) distribution of measurements in the image plane and a one dimensional descriptor (the range) associated to each pixel. After applying data filters, only few points in the image will be kept as features and the descriptors are augmented with information from neighboring pixels. In the case of pure point clouds, a common feature enhancement is the computation of geometric properties like the surface normal vectors, or feature reduction techniques can be applied for example to make uniform the density of points in the cloud.

Point clouds can be sparse and not uniformly distributed, moreover the sensor can produce a huge amount of measurements in a short time window that is impossible to process in real-time. In the past, several techniques have been proposed for reducing the number of points and make them uniformly distributed: uniform grids [124][125], random sampling [126][127], octree representations [128][129], bounding boxes [130][131] and grid projection [127].

Data enhancement is really common among variants of ICP. Range sensors generate an approximation of the surrounding environment by discretizing it in a set of points. It is possible to extract multiple differential geometry primitives from smooth regions of such a representation of the world. Among these primitives we find points (no derivative), lines and planes (first derivative), curves and quadrics (second derivative). The first derivative group uses the tangent vector t (i.e. parallel to a line or plane) and the normal vector n (i.e. orthogonal to a line or plane) to approximate the underlying surface. Normal and tangent vectors can be seen as a dual representation and the choice of using one or the other depends by the minimum information required to express the primitive. For example, in the case of a 3D line, using the tangent vector is more

Differential geometry primitives

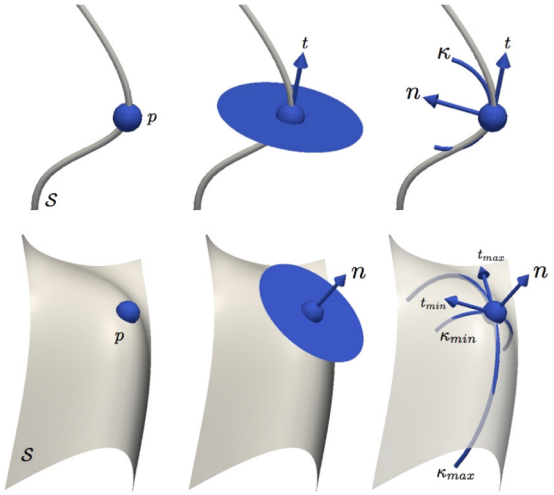


Figure 2.9: Complex shapes (gray) approximations as a 1D (top) and 2D (bottom) manifolds. From left to right and from top to bottom: no derivative (point), first derivative (line), second derivative (curve), no derivative (point), first derivative (plane), second derivative (quadric). (Images from [133]).

convenient since the normal requires to define a plane perpendicular to that line. For a surface, instead, it is true the opposite. Another difference between normal and tangent vector is that the former has a direction while the latter not. The viewpoint is usually used to compute the direction of the normal vectors. For the second derivative group, a curve can be parameterized by a scalar curvature k, while in the case of a quadric it is parameterized by the principal direction vectors t_{min}, t_{max} and the principal curvature scalars k_{min}, k_{max}. In theory, higher derivatives could be used for better approximations of the underlying shape at one point. However, higher derivatives are more sensitive to noise when applied on real cases. Indeed, as shown by Pomerleau *et al.* in [132], even the first derivative requires a large supporting surface to overcome the noise affecting common range sensors. Fig. 2.9 contains a set of approximations for complex shapes using 1D and 2D manifolds.

The heuristics proposed in the past for performing data association between two input clouds are many. The easier and most common correspondence criteria is based on the Euclidean distance between two points in the clouds [12][127][134][135][136][137]. Other heuristics are those between points and a planes [138] and between quadrics [139]. Distances based on point positions and normal angles is also used by Armesto *et al.* in [140]. Also, examples of mixed associations through the application of conversion functions between geometric and appearance features have been proposed in the past using surface orientations [141][125], surface orientations and color [142], color [143] and laser intensities [144][145]. Usually, to find the neighborhood of a point, two different strategies can be used: KD-tree or image projection. The former obtains more precise

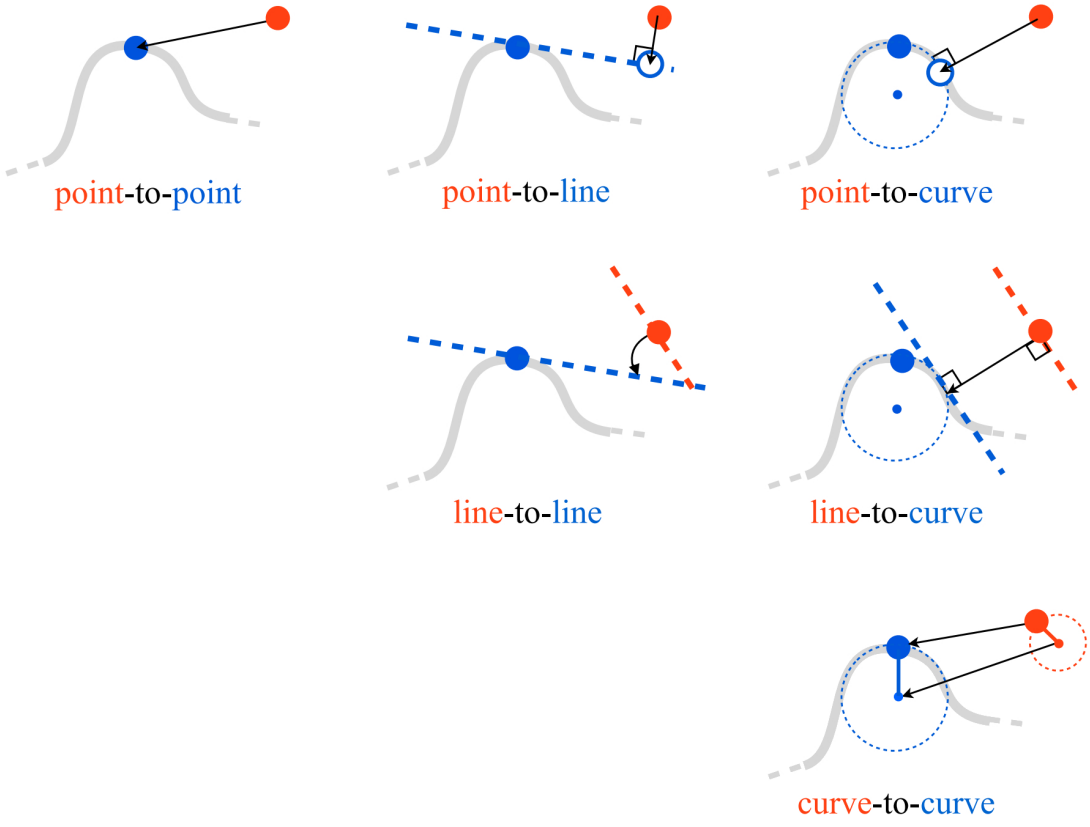


Figure 2.10: Set of 2D error metrics used in ICP variants. (Image from [133]).

results at the cost of a greater computation time. The latter is based on projecting the measurements in the image space. The ordered nature of images allow for a less accurate but faster neighborhood selection.

Error metrics

Most of the ICP variants use a global-rigid error metric. This error metric is parameterized by translation and rotation parameters leading to 6 DoFs in case of 3D point clouds, or 3 DoFs in 2 dimensions. The most simple metric is the point-to-point error that uses the base primitive introduced before. This metric is used in the original ICP formulation of Besl and McKay [26], and it has been successively exploited in [127][134][135][137][144]. Using this metric it is possible that during the registration geometric primitives of different types are aligned. To overcome this issue, many error metrics that deviate from the point-to-point schema have been proposed. Fig. 2.10 shows a set of possible error combinations for the 2D space case.

One of the most effective methods for point cloud registration is the one proposed by Chen *et al.* [146]. The classical Euclidean distance error metric (point-to-point) used

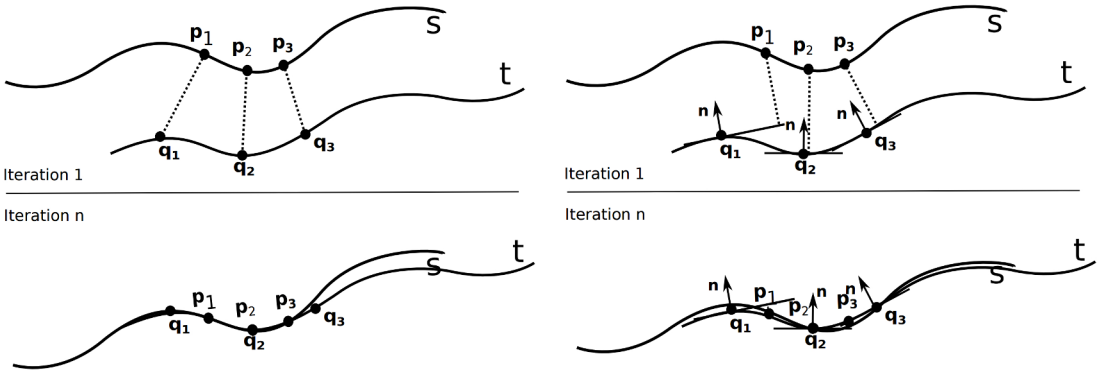


Figure 2.11: Left: point-to-point based ICP registration. Right: point-to-plane based ICP registration. (Images from [150]).

Point-to-plane error metric

by ICP is replaced in favor of a more robust point-to-plane criterion. This captures the idea that the points measured by the sensors are sampled from a locally continuous and smooth surfaces. Fig. 2.11 shows the different behavior between the point-to-point and the point-to-plane based alignment. This error metric has been then reused

Point-to-line error metric

in [138][147][148][149]. A closed form solution of its 2D version (point-to-line) has been presented by Censi in [136], and it is commonly used in robotics [125].

Generalized ICP (GICP)

Plane-to-plane error metric

Segal *et al.* [12] developed a probabilistic version of ICP called Generalized ICP (GICP). GICP models the sensor noise and utilizes the local continuity of the surface sampled through the cloud. This algorithm uses higher complexity 3D primitives, and it is a plane-to-plane variant of ICP that exploits the surface normals when assigning the weight to each matching correspondence in the objective function. Correspondences scaled with such weights allow the points to move along the tangent direction at the surface while they heavy penalize any motion along the normal direction. Fig. 2.12 illustrates the effects of the weights used by GICP. The work developed by Feldmar and Ayache in [151], is an example of variant of ICP employing a quadric-to-quadric error metric.

Quadric-to-quadric error metric

Dense Visual Odometry (DVO)

Steinbrücker *et al.* [13] proposed Dense Visual Odometry (DVO) that takes advantage of the additional light intensity channel available on RGB-D sensors. The main idea behind this approach is to compute an image containing the neighborhood of the edges extracted from the RGB image. Thanks to the depth channel these edges correspond to 3D points, and thus they can be straightforwardly reprojected in the image plane. The transformation is found by minimizing the reprojection distance of the edges on that plane. The objective function thus minimizes a set of 2D distance and this further reduces the observability of the transform resulting in a narrower convergence basin. On the other end, thanks to the reduced amount of data considered by DVO, it can run at high frame rates. This makes the assumption of a good initial

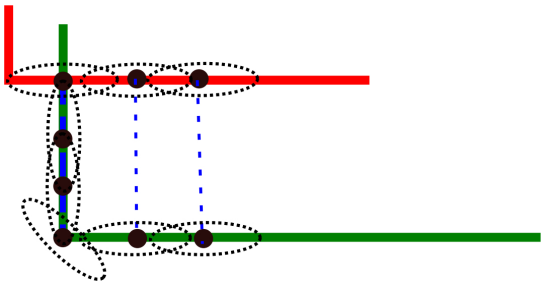


Figure 2.12: Example showing the effects of the weights assigned by GICP to the correspondences. (Image from [12]).

guess almost always verified. Unfortunately, DVO suffers from the noise and the blur affecting moving RGB cameras, and it is sensitive to illumination conditions. In fact, this is more a limitation of the sensor than of the algorithm, however these issues make the approach inadequate to operate in scenarios characterized by poor illumination and robots moving fast. The main advantage of using the RGB channel is that in case of poor structure in the 3D data (e.g. when looking at a flat wall), the algorithm is still able to track the camera pose without getting lost if some texture is present.

Kinect Fusion (KinFu)

Newcomb *et al.* [8] proposed Kinect Fusion (KinFu), which represents a major breakthrough in dense depth mapping. It is a complete system that combines components of mapping and point cloud registration. The implementation takes advantage of the brute force of the GPU to effectively update the environment representation. The camera tracking is a point-to-plane version of ICP that uses image projection to determine the correspondences. This tracker, however, can easily fail if the sensor is moved too fast, resulting in ICP to get lost.

Normal Distribution Transforms (NDT)

Magnusson *et al.* [152] proposed to represent the points with 3D Normal Distribution Transforms (NDT). The idea behind this method is to model the scene, similarly to [12], with a set of small Gaussian distributions computed from the neighborhood of each point. Using this representation it is possible, like in the other algorithms, to apply standard numerical optimization methods to compute the transformation between the two point clouds. NDT is reported to be faster and more robust with respect to other ICP based algorithms for 3D registration. This is possible thanks to the fact that NDT does not execute data association, thus it does not perform expensive nearest-neighbor searches. Since there is not a search for correspondences, NDT can not be considered a variant of ICP. Not less important, by using combined normal distributions to represent the point clouds, NDT is significantly more efficient in terms of memory consumption. This is particularly useful when mapping large-scale areas, and when it is necessary to keep a dynamic map available in memory for long time windows.

Chapter 3
Fundamentals

"If I were again beginning my studies,
I would follow the advice of Plato
and start with mathematics."
— Galileo Galilei, 1564 - 1642

In this Chapter we describe in detail some fundamental concepts that are at the base of the main subjects addressed in this thesis. More in particular, in Sec. 3.1 we introduce the pinhole camera model; in Sec. 3.2 we describe in detail the least-squares problem and its mathematical derivation; in Sec. 3.3 we illustrate the well known ICP algorithm for point cloud registration and finally, in Sec. 3.4, we explore the concept of graph-based SLAM.

3.1 Pinhole Camera Model

To understand how vision can be modeled and replicated on a computer, it is necessary to understand the image acquisition process. The role of the camera in computer vision is similar to that of a biological eye. A camera represents a mapping between the 3D world (object space) and a 2D image. This mapping is commonly known as *perspective projection*, and perspective geometry is fundamental for any understanding of image analysis.

Perspective projection

The *pinhole camera model*, as widely explained by R. Hartley *et al.* in [25], is one of the most simple and used abstraction of a camera. With this term, we denote a simple camera without lens, and with an infinitesimally small single aperture. It can be seen as a light-proof box with a small hole on one side. Light coming from a scene passes through the hole, and projects as a flipped image on the opposite side of the box. This abstraction can be seen in Fig. 3.1, where the candle in the scene is projected reverted on the back side of the box used to model the camera.

Pinhole camera model

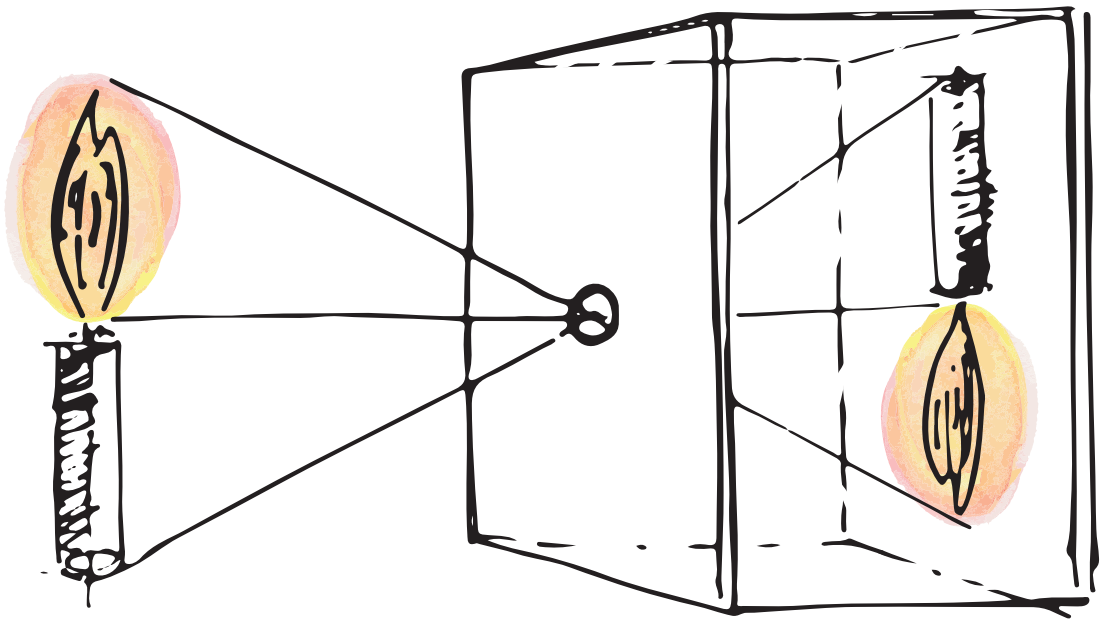


Figure 3.1: Pinhole camera model abstraction.

Camera aperture

Image plane

Focal length

The geometry behind the pinhole camera model is depicted in the left image of Fig. 3.2. Let the center of projection, or *camera aperture*, be the origin $\mathbf{O}$ of an Euclidean coordinate system, and consider the plane $X3 = -f$ commonly known as *image plane* or focal plane. The distance $f > 0$ between the center of projection and the image plane is called *focal length*. The point in the object space (3D) with coordinates $\mathbf{P} = (x_1,\ x_2,\ x_3)^T$ is mapped to the 2D point $\mathbf{Q} = (y_1,\ y_2)^T$ in the image space as the intersection of the image plane with the line passing through $\mathbf{P}$ and the camera aperture $\mathbf{O}$. As shown on the right picture of Fig. 3.2, from the top view of the geometric model two similar triangles can be identified. Both triangles have parts of the projection line (green) as their hypotenuses. From the properties relating similar triangles, assuming $x_3 > 0$ it follows that

$$-\frac{y_1}{f} = \frac{x_1}{x_3} \quad \Rightarrow \quad y_1 = -\frac{fx_1}{x_3} \tag{3.1}$$

$$-\frac{y_2}{f} = \frac{x_2}{x_3} \quad \Rightarrow \quad y_2 = -\frac{fx_2}{x_3} \tag{3.2}$$

which can be summarized in the more compact form

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = -\frac{f}{x_3}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{3.3}$$

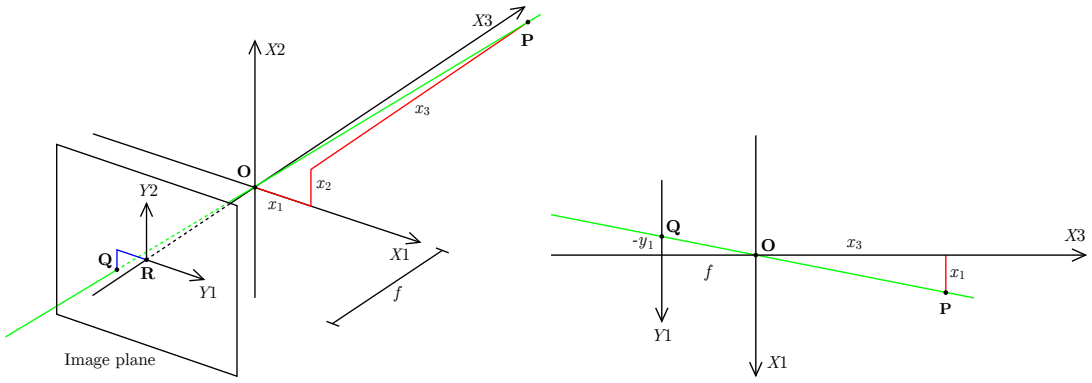


Figure 3.2: Left: geometric model of a pinhole camera. Right: top view of the geometric model of a pinhole camera.

This relation describes the mapping of a 3D point to its image space coordinates (2D).

To simplify the model, it is common to assume the image plane placed between the focal point of the camera and the scene, thus resulting on an image that is not flipped. In this case the image plane is called *virtual image plane*, and it provides a pinhole camera model simpler to analyze with respect to the previous one. Note that the model with the virtual image plane cannot be implemented in practice. With this modification in mind, the relation mapping 3D points to the 2D image plane changes as follows

Virtual image plane

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{3.4}$$

Eq. 3.4 is a non-linear relation, however by using *homogeneous coordinates* it is possible to rewrite it as a linear mapping via matrix multiplication

$$\begin{aligned} \begin{pmatrix} y'_1 \\ y'_2 \\ y'_3 \end{pmatrix} &= \begin{pmatrix} fx_1 \\ fx_2 \\ x_3 \end{pmatrix} \\ &= \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \end{aligned} \tag{3.5}$$

The matrix in the previous equation is known as *camera projection matrix*, or more simply camera matrix. Note that, since the resulting vector $(fx_1,\ fx_2,\ x_3)^T$ represents the image point in homogeneous coordinates, in the case that $x_3 \neq 1$ the entire vector

Camera projection matrix

has to be homogenized. This is done by dividing all the elements of the vector by x_3

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} fx_1/x_3 \\ fx_2/x_3 \\ 1 \end{pmatrix} \tag{3.6}$$

Without loss of generality, by neglecting the last element of the previous expression we obtain the same result of Eq. 3.4.

Principal point *Optical axis*

In Eq 3.5 there is the assumption that the origin of the coordinate system of the image plane coincides with the *principal point* $\mathbf{R}$ (see Fig. 3.2). The principal point is defined as the intersection of the image plane with the *optical axis* $X3$. Usually, this is not the case and the mapping relation assumes the more general form

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{3.7}$$

where $(p_x,\ p_y)^T$ are the coordinates of the principal point $\mathbf{R}$. As before, this equation can be rewritten as a linear mapping using homogeneous coordinates

$$\begin{aligned} \begin{pmatrix} y'_1 \\ y'_2 \\ y'_3 \end{pmatrix} &= \begin{pmatrix} fx_1 + x_3p_x \\ fx_2 + x_3p_y \\ x_3 \end{pmatrix} \\ &= \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \end{aligned} \tag{3.8}$$

Once more, the result must be homogenized to get rid of the homogeneous coordinates

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} fx_1/x_3 + p_x \\ fx_2/x_3 + p_y \\ 1 \end{pmatrix} \tag{3.9}$$

Camera calibration matrix

The 3×3 left block of the matrix in Eq. 3.8 is called *camera calibration matrix* $\mathbf{K}$

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

This matrix represents the most general mapping between points in the 3D world and the 2D image space with a pinhole camera model. It is widely used to manipulate data acquired with sensors like the Microsoft Kinect or the Asus Xtion.

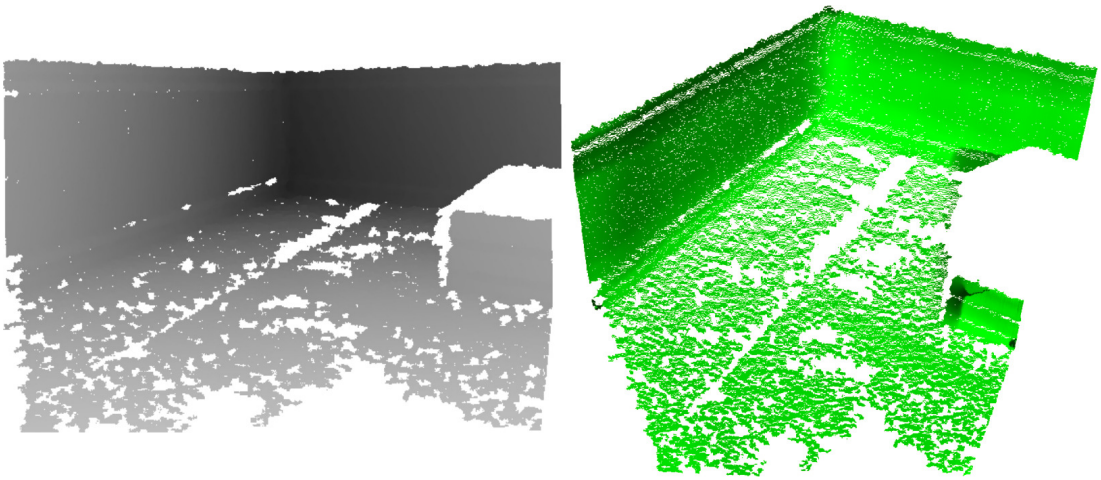


Figure 3.3: Example of the effect of applying the function π^{-1} (right), described by Eq. 3.11, to the input raw measurement (left).

3.1.1 Range Image to 3D Point Cloud and Vice Versa

Typically, 3D sensors provide an indirect measure of a point cloud. As an example, depth cameras generate images where the value of the pixel $(u,\ v)$ contains the depth d of the object closest to the observer, and lying on a beam passing through that pixel. These images are normally called depth or range images. To extract a 3D point cloud, it is necessary to apply a function π^{-1} that depends on the intrinsic camera parameters. In the case of a depth camera, this function can be defined as

Point unprojection

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \pi^{-1}(u,\ v,\ d) = \mathbf{K}^{-1} d \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} (u - p_x)d/f \\ (v - p_y)d/f \\ d \end{pmatrix} \tag{3.11}$$

where

$$\mathbf{K}^{-1} = \frac{1}{f^2} \begin{bmatrix} f & 0 & -fp_x \\ 0 & f & -fp_y \\ 0 & 0 & f^2 \end{bmatrix} \tag{3.12}$$

is the inverse of the camera calibration matrix defined by Eq. 3.10. This function is applied to all pixels contained in the depth image, except for those whose value is zero. In such a case, indeed, the pixels would collapse to the origin, thus to degenerated points.

Similarly, a 3D laser provides for each point azimuth θ, elevation ϕ and the range d measured at that elevation. Normally, both the azimuth and the elevation angles are subject to discretization, and for this reason it is possible to see a 3D scan as a range image. In this case $(\theta,\ \phi)$ are the coordinates of a pixel on a spherical surface,

and the value d of the pixel is its depth. Therefore, without loss of generality, we can define a projection function $\mathbf{s} = \pi(\mathbf{p})$ that maps a point $\mathbf{p}$ from the Cartesian to the measurement space. The point in the measurement space $\mathbf{s}$ can be either a $(u,\ v,\ d)^T$ or $(\theta,\ \phi,\ d)^T$ triplet depending on the type of sensor used. Let also $\pi^{-1}(\mathbf{s})$ be the inverse of the projection function that maps a raw sensor measurement to a point in the Cartesian space. Fig. 3.3 shows an example result of the application of the function π^{-1} (right), described by Eq. 3.11, to an input range image (left) generated by a depth camera.

Projection function

Unprojection function

3.2 Least-Squares Estimation

Least-squares problem

The least-squares method is a common technique used to compute approximated solutions of over determined systems (i.e. systems where there are more equations than unknowns variables). In particular, the term "least-squares" means that the final solution minimizes the sum of the squares of each error term. With this in mind, we can define the error $\mathbf{e}_i$ as the difference between the current measurement generated from the sensor, and a predicted one

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i = \widehat{\mathbf{z}}_i - \mathbf{z}_i \tag{3.13}$$

with $\mathbf{h}_i(\mathbf{x})$ being the function that maps the state vector $\mathbf{x}$ to the predicted measurement $\widehat{\mathbf{z}}_i$. In general $\mathbf{h}_i(\mathbf{x})$ is a non-linear function of the state, however it is possible to approximate it in the neighborhood of a linearization point $\breve{\mathbf{x}}$ using its first order Taylor expansion

$$\mathbf{h}_i(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) \simeq \mathbf{h}_i(\breve{\mathbf{x}}) + \underbrace{\left.\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{J}_i} \cdot \boldsymbol{\Delta}\mathbf{x} \tag{3.14}$$

Jacobian Matrix

where $\mathbf{J}_i$ is called *Jacobian matrix*. Now, assuming the error to have zero mean, and to be normally distributed with *information matrix* $\boldsymbol{\Omega}_i$, the squared error of a measurement depends only on the state, and it is a scalar value that can be computed using the following expression

$$e_i(\mathbf{x}) = \mathbf{e}_i(\mathbf{x})^T \boldsymbol{\Omega}_i \mathbf{e}_i(\mathbf{x}) \tag{3.15}$$

Given a set of N measurements, the goal is to find the state $\mathbf{x}^*$ that minimizes the error of all measurements

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \{ \overbrace{\sum_i^N \underbrace{\mathbf{e}_i(\mathbf{x})^T \boldsymbol{\Omega}_i \mathbf{e}_i(\mathbf{x})}_{e_i(\mathbf{x})}}^{F(\mathbf{x})} \} \tag{3.16}$$

Assuming to have a good initial guess $\breve{\mathbf{x}}$ of the optimum solution, we can rewrite the summands in the previous equation by using the Taylor approximation of Eq. 3.14

$$\begin{aligned} e_i(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}) &= (\mathbf{h}_i(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x})-\mathbf{z}_i)^T\boldsymbol{\Omega}_i(\mathbf{h}_i(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x})-\mathbf{z}_i) \\ &\simeq (\mathbf{J}_i\boldsymbol{\Delta}\mathbf{x}+\mathbf{h}_i(\breve{\mathbf{x}})-\mathbf{z}_i)^T\boldsymbol{\Omega}_i(\mathbf{J}_i\boldsymbol{\Delta}\mathbf{x}+\mathbf{h}_i(\breve{\mathbf{x}})-\mathbf{z}_i) \\ &= (\mathbf{J}_i\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_i)^T\boldsymbol{\Omega}_i(\mathbf{J}_i\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_i) \\ &= \boldsymbol{\Delta}\mathbf{x}^T\mathbf{J}_i^T\boldsymbol{\Omega}_i\mathbf{J}_i\boldsymbol{\Delta}\mathbf{x}+2\mathbf{e}_i\boldsymbol{\Omega}_i\mathbf{J}_i\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_i^T\boldsymbol{\Omega}_i\mathbf{e}_i \\ &= \boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}_i\boldsymbol{\Delta}\mathbf{x}+2\mathbf{b}_i\boldsymbol{\Delta}\mathbf{x}+c_i \end{aligned} \tag{3.17}$$

where the matrix $\mathbf{H}_i = \mathbf{J}_i^T\boldsymbol{\Omega}_i\mathbf{J}_i$ and the vector $\mathbf{b}_i = \mathbf{e}_i^T\boldsymbol{\Omega}_i\mathbf{J}_i$ are commonly known respectively as *Hessian matrix* and *Coefficient vector*, and $\mathbf{e}_i \overset{\text{def.}}{=} \mathbf{e}_i(\breve{\mathbf{x}})$. Substituting Eq. 3.17 in Eq. 3.16 we obtain

Hessian matrix and Coefficient vector

$$\begin{aligned} F(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}) &\simeq \sum_{i=1}^{N}\boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}_i\boldsymbol{\Delta}\mathbf{x}+2\mathbf{b}_i\boldsymbol{\Delta}\mathbf{x}+c_i \\ &= \boldsymbol{\Delta}\mathbf{x}^T\left[\sum_{i=1}^{N}\mathbf{H}_i\right]\boldsymbol{\Delta}\mathbf{x}+2\left[\sum_{i=1}^{N}\mathbf{b}_i\right]\boldsymbol{\Delta}\mathbf{x}+\left[\sum_{i=1}^{N}c_i\right] \\ &= \boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}\boldsymbol{\Delta}\mathbf{x}+2\mathbf{b}\boldsymbol{\Delta}\mathbf{x}+c \end{aligned} \tag{3.18}$$

The last equation is the expression of the objective function $F(\mathbf{x})$ under a linear approximation of $\mathbf{h}_i(\mathbf{x})$, in the neighborhood of the initial guess $\breve{\mathbf{x}}$. In other words, if the initial estimate is fixed, the value of the function can be approximated by a quadratic form with respect to the increments $\boldsymbol{\Delta}\mathbf{x}$. It is possible therefore to minimize this quadratic equation, and compute the optimal increment $\boldsymbol{\Delta}\mathbf{x}^*$ that applied to the current estimate leads to the better solution

Optimal increment

$$\mathbf{x}^* = \breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}^* \tag{3.19}$$

The optimal increment $\boldsymbol{\Delta}\mathbf{x}^*$ is calculated by setting the derivative of the function $F(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x})$ (Eq. 3.18) to be equal to zero, and then solving the corresponding equation

$$\frac{\partial(\boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}\boldsymbol{\Delta}\mathbf{x}+2\mathbf{b}\boldsymbol{\Delta}\mathbf{x}+c)}{\partial\boldsymbol{\Delta}\mathbf{x}} = 2\mathbf{H}\boldsymbol{\Delta}\mathbf{x}+2\mathbf{b} = 0 \tag{3.20}$$

This results in finding the solution of the following linear system

$$\mathbf{H}\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{b} \tag{3.21}$$

If the model function $\mathbf{h}_i(\mathbf{x})$ is linear, the solution can be found in just one step. Since as said before this is not the general case, it is necessary to iterate the linearization procedure until an acceptable solution is found. Several methods that solve this problem

Gauss-Newton and Levenberg-Marquardt

Input: state initial guess $\breve{\mathbf{x}}$ and measurements $\mathcal{C} = \{\langle \mathbf{z}_i,\ \mathbf{\Omega}_i \rangle\}$

Output: new solution $\mathbf{x}^*$

$F_{new} \leftarrow F(\breve{\mathbf{x}})$
repeat
  $\breve{F} \leftarrow F_{new}$
  $\mathbf{b} \leftarrow \mathbf{0}, \quad \mathbf{H} \leftarrow \mathbf{0}$
  forall $i = 1 \dots N$ **do**
    $\widehat{\mathbf{z}}_i \leftarrow \mathbf{h}_i(\breve{\mathbf{x}})$
    $\mathbf{e}_i \leftarrow \widehat{\mathbf{z}}_i - \mathbf{z}_i$
    $\mathbf{J}_i \leftarrow \left.\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$
    $\mathbf{H}_i \leftarrow \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{J}_i$
    $\mathbf{b}_i \leftarrow \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{e}_i$
    $\mathbf{H} \leftarrow \mathbf{H} + \mathbf{H}_i$
    $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{b}_i$
  end
  $\mathbf{\Delta x} \leftarrow solve(\mathbf{H}\mathbf{\Delta x} = -\mathbf{b})$
  $\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} + \mathbf{\Delta x}$
  $F_{new} \leftarrow F(\breve{\mathbf{x}})$
until $\breve{F} - F_{new} > \epsilon$
$\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$

Algorithm 1: Gauss-Newton minimization algorithm.

exist, among these we recall the Gauss-Newton (GN) and the Levenberg-Marquardt (LM) minimization algorithms respectively described by Algorithm 1 and Algorithm 2.

3.2.1 Smooth Manifolds

All the mathematical derivations of the previous section rely on the strong assumptions that the measurement functions are smooth, and that the state spans over an Euclidean space. When we compute the optimal increment we sum it to the current solution. This is supposed to work correctly in every situation as long as that sum is a legal operation in the domain of the state variables considered. Unfortunately, in robotics, this is not true in general since angles and rotations are often involved. Summing a triplet of Euler angles is not properly elegant in the sense that it might lead to singular configurations. In fact, although they admit local Euclidean mappings, rotations are not Euclidean but they are smooth manifolds [153].

Let us consider the space of 3D rotations. A 3D rotation can be parameterized with a rotation matrix $\mathbf{R}$. However, it requires 9 parameters to represent only 3 angles.

Input: state initial guess $\breve{\mathbf{x}}$ and measurements $\mathcal{C} = \{\langle \mathbf{z}_i,\ \mathbf{\Omega}_i \rangle\}$

Output: new solution $\mathbf{x}^*$

$F_{new} \leftarrow F(\breve{\mathbf{x}})$
$\mathbf{x}_{backup} \leftarrow \breve{\mathbf{x}}$
$\lambda \leftarrow computeInitialLambda(\mathcal{C},\ \breve{\mathbf{x}})$
repeat
  $\breve{F} \leftarrow F_{new}$
  $\mathbf{b} \leftarrow \mathbf{0}, \quad \mathbf{H} \leftarrow \mathbf{0}$
  forall $i = 1 \dots N$ **do**
    $\widehat{\mathbf{z}}_i \leftarrow \mathbf{h}_i(\breve{\mathbf{x}})$
    $\mathbf{e}_i \leftarrow \widehat{\mathbf{z}}_i - \mathbf{z}_i$
    $\mathbf{J}_i \leftarrow \left.\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$
    $\mathbf{H}_i \leftarrow \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{J}_i$
    $\mathbf{b}_i \leftarrow \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{e}_i$
    $\mathbf{H} \leftarrow \mathbf{H} + \mathbf{H}_i$
    $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{b}_i$
  end
  $t \leftarrow 0$
  repeat
    $\mathbf{\Delta x} \leftarrow solve(\mathbf{H\Delta x} = -\mathbf{b})$
    $\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} + \mathbf{\Delta x}$
    $F_{new} \leftarrow F(\breve{\mathbf{x}})$
    if $F_{new} < \breve{F}$ **then**
      $\lambda \leftarrow \lambda / 2$
      $\mathbf{x}_{backup} \leftarrow \breve{\mathbf{x}}$
      $t \leftarrow -1$
    end
    else
      $\lambda \leftarrow \lambda * 4$
      $\breve{\mathbf{x}} \leftarrow \mathbf{x}_{backup}$
      $t \leftarrow t + 1$
    end
  until $t < t_{max} \wedge t > 0$
until $\breve{F} - F_{new} > \epsilon$
$\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$

Algorithm 2: Levenberg-Marquardt minimization algorithm.

Using a rotation matrix as a parameter in a least-squares problem would result in the computation of non-valid solutions, since the orthogonality constraint is not enforced. The alternative is to use minimal representations (see Appendix A) like Euler angles or unit quaternions. As a drawback, however, they suffer of singularities. Luckily, rotations are manifolds, and this means that they allow for a local parameterization which is homeomorphic to the vector space $\mathbb{R}^n$. For example, consider an arbitrary rotation $\mathbf{R}(\varphi,\ \theta,\ \psi)$, we can than define a mapping between the space of rotation matrices and Euler angles, and vice versa

Minimal representations

$$\mathbf{v} = toVector(\mathbf{R}) \tag{3.22}$$

$$\mathbf{R} = fromVector(\mathbf{v}) \tag{3.23}$$

where $\mathbf{v} = (\varphi,\ \theta,\ \psi)^T$ is the vector containing the 3 Euler angles.

If we are around the origin of $\mathbf{v}$, the Euler angles are far from possible singularities, and the Euclidean distance between two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ around the origin are closely related to the true distance of their orientations. When we are near a singularity, however, this situation changes and a small difference in the orientation might lead to an abrupt difference in the local parameters $\mathbf{v}$. This can represent an important issue on a least-squares based optimization that strongly relies on the smoothness of the problem. A possible solution to this problem is to compute a local mapping to Euler angles that is away from singularities. In the space of rotations this can be done by considering the reference frame so that $toVector(\mathbf{R}_0) = \mathbf{0}$ is the origin

$$\mathbf{v} = \mathbf{R} \boxminus \mathbf{R}_0 = toVector(\mathbf{R}_0^{-1} \cdot \mathbf{R}) \tag{3.24}$$

$$\mathbf{R} = \mathbf{R}_0 \boxplus \mathbf{v} = \mathbf{R}_0 \cdot fromVector(\mathbf{v}) \tag{3.25}$$

Operators $\boxminus$ and $\boxplus$

The $\boxminus$ and $\boxplus$ are powerful operators that can be used to convert a global difference in the manifold space in a local perturbation, and vice versa. Eq. 3.24 computes the Euler angles of the rotation that moves $\mathbf{R}_0$ to $\mathbf{R}$, in the reference frame of $\mathbf{R}_0$. If the two rotations have a small difference, also $\mathbf{v}$ will be small independently if $\mathbf{R}_0$ is close to a singularity or not. Similarly, Eq. 3.25 computes a new rotation by applying a local perturbation $\mathbf{v}$ to $\mathbf{R}_0$.

Let suppose that the state space of our least-squares problem is a smooth manifold, and let the values in this space to be parameterized redundantly by $\mathbf{x}$, and minimally by the vector $\tilde{\mathbf{x}}$. The prediction function has both the forms $\mathbf{h}_i(\mathbf{x})$ or $\mathbf{h}_i(\tilde{\mathbf{x}})$. Given the current estimate $\breve{\mathbf{x}}$, our goal is to compute the perturbation of the predicted measurements, under a small variation of the state variables: $\mathbf{h}_i(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})$. Note that $\breve{\mathbf{x}}$ is constant, however we can explore the state space by varying $\boldsymbol{\Delta}\tilde{\mathbf{x}}$. We can therefore

Input: state initial guess $\breve{\mathbf{x}}$ and measurements $\mathcal{C} = \{\langle \mathbf{z}_i,\ \mathbf{\Omega}_i \rangle\}$

Output: new solution $\mathbf{x}^*$

$F_{new} \leftarrow F(\breve{\mathbf{x}})$
repeat
  $\breve{F} \leftarrow F_{new}$
  $\tilde{\mathbf{b}} \leftarrow \mathbf{0}, \quad \tilde{\mathbf{H}} \leftarrow \mathbf{0}$
  forall $i = 1 \dots N$ **do**
    $\widehat{\mathbf{z}}_i \leftarrow \mathbf{h}_i(\breve{\mathbf{x}})$
    $\tilde{\mathbf{e}}_i \leftarrow \widehat{\mathbf{z}}_i \boxminus \mathbf{z}_i$
    $\tilde{\mathbf{J}}_i \leftarrow \left.\frac{\partial \mathbf{h}_i(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}$
    $\mathbf{J}_{\mathbf{z}_i} \leftarrow \left.\frac{\partial((\widehat{\mathbf{z}}_i + \boldsymbol{\Delta}\mathbf{z}_i) \boxminus \mathbf{z}_i)}{\partial \boldsymbol{\Delta}\mathbf{z}_i}\right|_{\boldsymbol{\Delta}\mathbf{z}_i=\mathbf{0}}$
    $\tilde{\mathbf{\Omega}}_i \leftarrow (\mathbf{J}_{\mathbf{z}_i} \mathbf{\Omega}_i^{-1} \mathbf{J}_{\mathbf{z}_i}^T)^{-1}$
    $\tilde{\mathbf{H}}_i \leftarrow \tilde{\mathbf{J}}_i^T \tilde{\mathbf{\Omega}}_i \tilde{\mathbf{J}}_i$
    $\tilde{\mathbf{b}}_i \leftarrow \tilde{\mathbf{J}}_i^T \tilde{\mathbf{\Omega}}_i \tilde{\mathbf{e}}_i$
    $\tilde{\mathbf{H}} \leftarrow \tilde{\mathbf{H}} + \tilde{\mathbf{H}}_i$
    $\tilde{\mathbf{b}} \leftarrow \tilde{\mathbf{b}} + \tilde{\mathbf{b}}_i$
  end
  $\boldsymbol{\Delta}\tilde{\mathbf{x}} \leftarrow solve(\tilde{\mathbf{H}}\boldsymbol{\Delta}\tilde{\mathbf{x}} = -\tilde{\mathbf{b}})$
  $\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}$
  $F_{new} \leftarrow F(\breve{\mathbf{x}})$
until $\breve{F} - F_{new} > \epsilon$
$\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$

Algorithm 3: Gauss-Newton minimization algorithm for manifold measurement and state spaces.

compute the Taylor expansion of $\mathbf{h}_i(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})$, with respect to $\boldsymbol{\Delta}\tilde{\mathbf{x}}$, around $\boldsymbol{\Delta}\tilde{\mathbf{x}} = 0$

$$\mathbf{h}_i(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}) \simeq \mathbf{h}_i(\breve{\mathbf{x}}) + \underbrace{\left.\frac{\partial \mathbf{h}_i(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}}_{\tilde{\mathbf{J}}_i} \cdot \boldsymbol{\Delta}\tilde{\mathbf{x}} \tag{3.26}$$

$$= \mathbf{h}_i(\breve{\mathbf{x}}) + \tilde{\mathbf{J}}_i \boldsymbol{\Delta}\tilde{\mathbf{x}} \tag{3.27}$$

Differently form the Taylor expansion of Eq. 3.14, the current linearization point is $\breve{\mathbf{x}}$, and it is fixed. Moreover, the expansion gives us the change of the parameters as a function of the variation of the increments, that are applied to the current linearization

point through the $\boxplus$ operator. Once a new solution $\boldsymbol{\Delta}\tilde{\mathbf{x}}^*$ for the increments is found, the perturbation must be applied to the current estimate through the $\boxplus$ operator

$$\mathbf{x}^* = \breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}^* \tag{3.28}$$

Dealing with manifold state spaces is fundamental for direct methods like GN or LM. Local parameterizations have the advantage of increasing the convergence basin.

Non-Euclidean measurements

Note that even the measurements can be non-Euclidean. Also in this case we can exploit the smooth manifolds to obtain a smoother error function, that in general leads to faster and more robust convergence. Consider the alternative error function

$$\tilde{\mathbf{e}}_i(\mathbf{x}) = \tilde{\mathbf{e}}_i(\widehat{\mathbf{z}}_i,\ \mathbf{z}_i) = \mathbf{z}_i \boxminus \mathbf{z}_i \tag{3.29}$$

This error function computes the displacement between the prediction and the observation in a minimal space, and it uses as reference frame the measurement $\mathbf{z}_i$. Since the prediction and the measurement are in general close in the manifold space of the observations, the displacement computed through the $\boxminus$ will also be small and regular. As usual, we can compute the Taylor expansion of this new error, under a small perturbation of the prediction

$$\tilde{\mathbf{e}}_i(\widehat{\mathbf{z}}_i + \boldsymbol{\Delta}\mathbf{z}_i,\ \mathbf{z}_i) = (\widehat{\mathbf{z}}_i + \boldsymbol{\Delta}\mathbf{z}_i) \boxminus \mathbf{z}_i \simeq \widehat{\mathbf{z}}_i \boxminus \mathbf{z}_i + \underbrace{\left.\frac{\partial((\widehat{\mathbf{z}}_i + \boldsymbol{\Delta}\mathbf{z}_i) \boxminus \mathbf{z}_i)}{\partial \boldsymbol{\Delta}\mathbf{z}_i}\right|_{\boldsymbol{\Delta}\mathbf{z}_i = \mathbf{0}}}_{\mathbf{J}_{\mathbf{z}_i}} \cdot \boldsymbol{\Delta}\mathbf{z}_i \tag{3.30}$$

While the computation of these derivatives looks complicated, it is possible to simplify the process by exploiting the rule of partial derivatives, and the fact that the evaluation point is in $\boldsymbol{\Delta}\tilde{\mathbf{x}} = \mathbf{0}$

$$\frac{\partial \mathbf{h}_i(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}} = \underbrace{\left.\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{J}_i} \cdot \underbrace{\left.\frac{\partial(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}}_{\mathbf{M}} \tag{3.31}$$

This means that one can derive a Jocobian on a manifold, from the non-manifold one (Eq. 3.14), by multiplying its non-zero blocks by the derivative of the $\boxplus$ operator computed in $\breve{\mathbf{x}}$

$$\tilde{\mathbf{J}}_i = \mathbf{J}_i \mathbf{M} \tag{3.32}$$

We are able now to derive a manifold version of the GN and LM algorithms as shown in Algorithm 3 and Algorithm 4.

Input: state initial guess $\breve{\mathbf{x}}$ and measurements $\mathcal{C} = \{\langle \mathbf{z}_i,\ \mathbf{\Omega}_i \rangle\}$

Output: new solution $\mathbf{x}^*$

$F_{new} \leftarrow F(\breve{\mathbf{x}}), \quad \mathbf{x}_{backup} \leftarrow \breve{\mathbf{x}}$
$\lambda \leftarrow computeInitialLambda(\mathcal{C},\ \breve{\mathbf{x}})$
repeat
  $\breve{F} \leftarrow F_{new}$
  $\tilde{\mathbf{b}} \leftarrow \mathbf{0}, \quad \tilde{\mathbf{H}} \leftarrow \mathbf{0}$
  forall $i = 1 \dots N$ **do**
    $\widehat{\mathbf{z}}_i \leftarrow \mathbf{h}_i(\breve{\mathbf{x}})$
    $\tilde{\mathbf{e}}_i \leftarrow \widehat{\mathbf{z}}_i \boxminus \mathbf{z}_i$
    $\tilde{\mathbf{J}}_i \leftarrow \left.\frac{\partial \mathbf{h}_i(\breve{\mathbf{x}} \boxplus \mathbf{\Delta}\tilde{\mathbf{x}})}{\partial \mathbf{\Delta}\tilde{\mathbf{x}}}\right|_{\mathbf{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}$
    $\mathbf{J}_{\mathbf{z}_i} \leftarrow \left.\frac{\partial((\widehat{\mathbf{z}}_i + \mathbf{\Delta}\mathbf{z}_i) \boxminus \mathbf{z}_i)}{\partial \mathbf{\Delta}\mathbf{z}_i}\right|_{\mathbf{\Delta}\mathbf{z}_i=\mathbf{0}}$
    $\tilde{\mathbf{\Omega}}_i \leftarrow (\mathbf{J}_{\mathbf{z}_i} \mathbf{\Omega}_i^{-1} \mathbf{J}_{\mathbf{z}_i}^T)^{-1}$
    $\tilde{\mathbf{H}}_i \leftarrow \tilde{\mathbf{J}}_i^T \tilde{\mathbf{\Omega}}_i \tilde{\mathbf{J}}_i, \quad \tilde{\mathbf{b}}_i \leftarrow \tilde{\mathbf{J}}_i^T \tilde{\mathbf{\Omega}}_i \tilde{\mathbf{e}}_i$
    $\tilde{\mathbf{H}} \leftarrow \tilde{\mathbf{H}} + \tilde{\mathbf{H}}_i, \quad \tilde{\mathbf{b}} \leftarrow \tilde{\mathbf{b}} + \tilde{\mathbf{b}}_i$
  end
  $t \leftarrow 0$
  repeat
    $\mathbf{\Delta}\tilde{\mathbf{x}} \leftarrow solve(\tilde{\mathbf{H}}\mathbf{\Delta}\tilde{\mathbf{x}} = -\tilde{\mathbf{b}})$
    $\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} \boxplus \mathbf{\Delta}\tilde{\mathbf{x}}$
    $F_{new} \leftarrow F(\breve{\mathbf{x}})$
    if $F_{new} < \breve{F}$ **then**
      $\lambda \leftarrow \lambda / 2$
      $\mathbf{x}_{backup} \leftarrow \breve{\mathbf{x}}$
      $t \leftarrow -1$
    end
    else
      $\lambda \leftarrow \lambda * 4$
      $\breve{\mathbf{x}} \leftarrow \mathbf{x}_{backup}$
      $t \leftarrow t + 1$
    end
  until $t < t_{max} \wedge t > 0$
until $\breve{F} - F_{new} > \epsilon$
$\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$

Algorithm 4: Levenberg-Marquardt minimization algorithm for manifold measurement and state spaces.

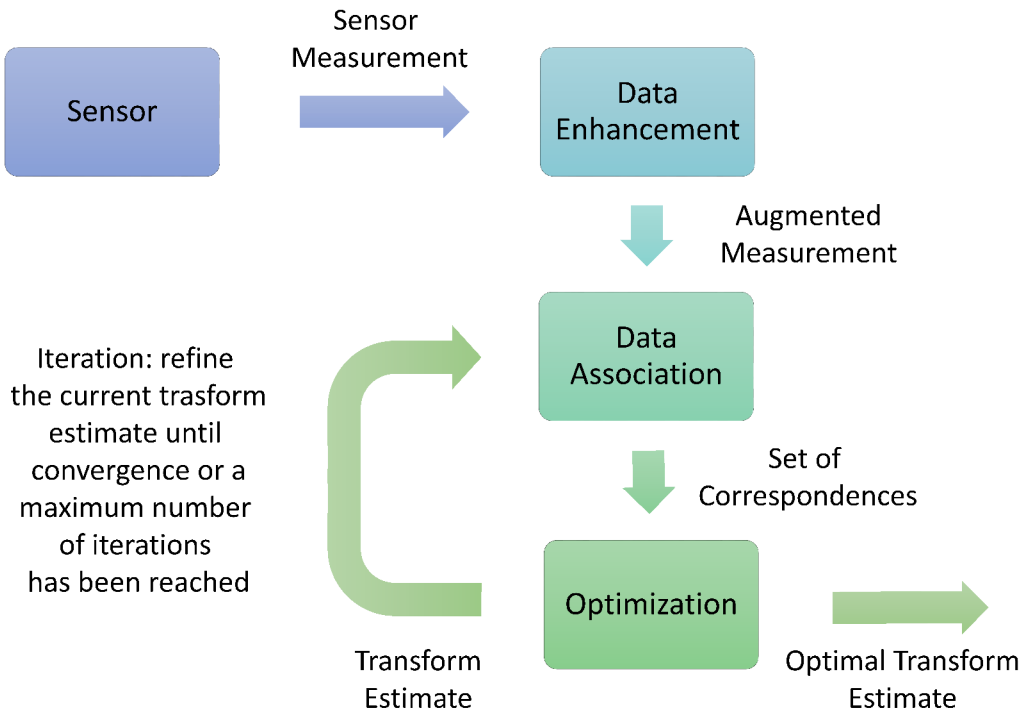


Figure 3.4: Iterative Closest Point algorithm data flow chart.

3.3 Iterative Closest Point

Iterative Closest Point (ICP)

Registering two point clouds consists in finding an isometry transformation that maximizes the overlap between two input clouds. Among all the methods proposed to solve this problem, the Iterative Closest Point algorithm [26] is one of the earliest and most used techniques. ICP is an iterative algorithm that refines an initial estimate of the relative transformation. At each iteration, ICP searches for corresponding pairs of points between the two clouds, starting from the current transform estimate. These correspondences are then used to compute an improved transformation. Fig. 3.4 depicts the data flow chart of ICP.

General ICP minimization function

More formally, let $\mathcal{P}^{\mathrm{r}} = \{\mathbf{p}^{\mathrm{r}}_{1:N^{\mathrm{r}}}\}$ and $\mathcal{P}^{\mathrm{c}} = \{\mathbf{p}^{\mathrm{c}}_{1:N^{\mathrm{c}}}\}$ be the two set of points, we want to find the transformation $\mathbf{T}^*$ that minimizes the distance between corresponding points in the two scenes

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} \overbrace{\underbrace{\left(\mathbf{T} \oplus \mathbf{p}^{\mathrm{r}}_j - \mathbf{p}^{\mathrm{c}}_i\right)^T \mathbf{\Omega}_{ij} \underbrace{\left(\mathbf{T} \oplus \mathbf{p}^{\mathrm{r}}_j - \mathbf{p}^{\mathrm{c}}_i\right)}_{\mathbf{e}_{ij}(\mathbf{T})}}_{\chi^2_{ij}}} \tag{3.33}$$

In Eq. 3.33 the symbols have the following meaning:

- $\mathbf{T}$ is the current estimate of the transformation that maps $\mathcal{P}^{\mathrm{r}}$ in the reference frame of $\mathcal{P}^{\mathrm{c}}$;
- $\mathbf{\Omega}_{ij}$ is an information matrix that takes into account the noise statistics of the sensor and/or the surface;
- $\mathcal{C} = \{\langle i, j \rangle_{1:M}\}$, is a set of correspondences between points in the two clouds. If $\langle i, j \rangle \in \mathcal{C}$, it means that the point $\mathbf{p}_j^{\mathrm{r}}$ in the cloud $\mathcal{P}^{\mathrm{r}}$ corresponds to the point $\mathbf{p}_i^{\mathrm{c}}$ in the cloud $\mathcal{P}^{\mathrm{c}}$;
- $\oplus$ is the standard composition operator (Smith *et al.* [27]) that applies the transformation $\mathbf{T}$ to the point $\mathbf{p}$. If we use the homogeneous notation for transformations and points, $\oplus$ reduces to the matrix-vector product.

Clearly the exact correspondences are not known. However, assuming to have a good initial guess of the relative transform, it is common to compute an approximation of these correspondences through some heuristic (e.g. nearest neighbor). In its most general formulation, ICP iteratively refines the current estimate $\mathbf{T}$ by interleaving the search for correspondences (data association) and the solution of Eq. 3.33 (optimization). In Sec. 3.2 we discussed the mathematical derivation of the least-squares estimation problem used to solve Eq. 3.33. At each new iteration, the data association is recomputed after applying to the cloud $\mathcal{P}^{\mathrm{r}}$ the most recent transformation $\mathbf{T}$. The general ICP formulation is listed in Algorithm 5.

Over time ICP has been modified and extended to a large number of variants of increased robustness and performance. These differ by the choice of the information matrix $\mathbf{\Omega}_{ij}$, or by the heuristic chosen to find the correspondences. Plain ICP uses a diagonal $\mathbf{\Omega}_{ij}$ potentially scaled with a weight capturing the likelihood that a correspondence is correct. In its most general formulation, GICP models $\mathbf{\Omega}_{ij}$ in order to incorporate the surface information from both clouds

Omega scaling

$$\mathbf{\Omega}_{ij} = (\mathbf{\Sigma}_i^s + \mathbf{R}\mathbf{\Sigma}_j^s\mathbf{R}^T)^{-1} \tag{3.34}$$

where

$$\mu_n^s = \frac{1}{|\mathcal{V}_n|}\sum_{\mathbf{p}_k \in \mathcal{V}_n} \mathbf{p}_k \qquad \mathbf{\Sigma}_n^s = \frac{1}{|\mathcal{V}_n|}\sum_{\mathbf{p}_k \in \mathcal{V}_n} (\mathbf{p}_k - \mu_n)^T(\mathbf{p}_k - \mu_n). \tag{3.35}$$

with $\mathcal{V}_n$ being the set of points in the neighborhood of $\mathbf{p}_n$. Such a formulation, commonly known as plane-to-plane, increases the overall symmetry of the model. The covariances $\mathbf{\Sigma}_i^s$ and $\mathbf{\Sigma}_j^s$ are forced to have a disk shape and to lie on the surface from where $\mathbf{p}_i^{\mathrm{c}}$ and $\mathbf{p}_j^{\mathrm{r}}$ were sampled ($\mathbf{\Sigma}_i^s$ and $\mathbf{\Sigma}_j^s$ with a small eigenvalue along the normal direction). When carrying on the minimization in Eq. 3.33, GICP replaces the points

Input: transform initial guess $\mathbf{T}$ and two point clouds $\mathcal{P}^{\mathrm{r}} = \{\mathbf{p}^{\mathrm{r}}_{1:N^{\mathrm{r}}}\}$, $\mathcal{P}^{\mathrm{c}} = \{\mathbf{p}^{\mathrm{c}}_{1:N^{\mathrm{c}}}\}$

Output: the aligning transformation $\mathbf{T}^*$

$cloudPreprocessing(\mathcal{P}^{\mathrm{r}},\ \mathcal{P}^{\mathrm{c}})$
$t \leftarrow 0$
repeat
  $\mathcal{C} \leftarrow findCorrespondences(\mathbf{T} \oplus \mathcal{P}^{\mathrm{r}},\ \mathcal{P}^{\mathrm{c}})$
  $\mathbf{T} \leftarrow \mathrm{argmin}_{\mathbf{T}} \sum_{\mathcal{C}} \left(\mathbf{T} \oplus \mathbf{p}^{\mathrm{r}}_j - \mathbf{p}^{\mathrm{c}}_i\right)^T \mathbf{\Omega}_{ij} \left(\mathbf{T} \oplus \mathbf{p}^{\mathrm{r}}_j - \mathbf{p}^{\mathrm{c}}_i\right)$
  $\chi^2 \leftarrow \sum_{\mathcal{C}} \chi^2_{ij}$
until $\chi^2 > \epsilon \wedge t < t_{max}$
$\mathbf{T}^* \leftarrow \mathbf{T}$

Algorithm 5: Iterative Closest Point algorithm (ICP).

with their means μ^s_i and μ^s_j, and the information matrix $\mathbf{\Omega}_{ij}$ with the result obtained by applying Eq. 3.34.

Note that this formulation of GICP requires at each iteration of the algorithm a double matrix multiplication, and a matrix inversion, for each correspondence. This yield a significant increase in computation time, in particular when the number of correspondences is in the order of tens of thousands, like in the case of dense point clouds. As stated by Segal *et al.* in [12], when the information matrix of the reference surface $\mathbf{\Sigma}^s_j$ is neglected, that means $\mathbf{\Omega}_{ij} = (\mathbf{\Sigma}^s_i)^{-1}$, we obtain a limiting case of GICP known as point-to-plane. Albeit slightly less accurate, this special case of GICP saves computation time since $\mathbf{\Omega}_{ij}$ stays fixed during the entire alignment. This makes the approach particularly suitable when dealing with high frequency dense stereo sensors like the Asus Xtion, or the Microsoft Kinect.

In all cases, the error vector $\mathbf{e}_{ij}(\mathbf{T}) = \mathbf{T} \oplus \mathbf{p}^{\mathrm{r}}_j - \mathbf{p}^{\mathrm{c}}_i$ is computed as the difference between two 3D points, and lives in $\mathbb{R}^3$. Also, consider that two matching points in two different clouds are unlikely to be exactly the same point in space. This introduces an arbitrary error while minimizing the Euclidean distance.

3.3.1 Iterative Closest Point Probabilistic Formulation

Maximum-A-Posteriori (MAP)

We can model the point cloud registration as a Maximum-A-Posteriori (MAP) estimation problem. In particular, the alignment can be seen as the estimation of the pose $\mathbf{x}$ of a robot, given a set of measurements of the surrounding points. In this scenario, one of the cloud is fixed and it represents the global world, while the other it is our current measurement $\mathbf{z}$. More formally, we want to estimate the state $\mathbf{x}^*$ that maximizes the

probability $p(\mathbf{x}|\mathbf{z})$

$$\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\text{argmax}}\{p(\mathbf{x}|\mathbf{z})\} && (3.36)\\
\text{(Bayes Theorem)} &= \underset{\mathbf{x}}{\text{argmax}}\{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})\} && (3.37)\\
\text{(Measurement Independence)} &= \underset{\mathbf{x}}{\text{argmax}}\left\{p(\mathbf{x})\prod_i^N p(\mathbf{z}_i|\mathbf{x})\right\} && (3.38)\\
&= \underset{\mathbf{x}}{\text{argmin}}\left\{-\log\left(p(\mathbf{x})\prod_i^N p(\mathbf{z}_i|\mathbf{x})\right)\right\} && (3.39)
\end{aligned}$$

where the last equality come from the fact that maximizing the posterior is the same as minimizing the negative log-posterior.

Observation model

Let now $\mathbf{h}_i(\cdot)$ be a known function called *observation* or *measurement model* that, given a state $\mathbf{x}$, returns the prediction $\widehat{\mathbf{z}}_i$ of the measurement as if the robot is located in $\mathbf{x}$. Assuming the measurement noise ϵ_i to be zero-mean normally distributed with information matrix $\mathbf{\Omega}_i$, we can rewrite in a more explicit form the likelihood of the measurement

Measurement likelihood

$$p(\mathbf{z}_i|\mathbf{x}) \propto \exp(-\frac{1}{2}(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)^T\mathbf{\Omega}_i(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)) \quad (3.40)$$

and the prior

Prior probability

$$p(\mathbf{x}) \propto \exp(-\frac{1}{2}(\mathbf{h}_0(\mathbf{x}) - \mathbf{z}_0)^T\mathbf{\Omega}_0(\mathbf{h}_0(\mathbf{x}) - \mathbf{z}_0)) \quad (3.41)$$

for some given function $\mathbf{h}_0(\cdot)$, prior mean $\mathbf{z}_0$ and information matrix $\mathbf{\Omega}_0$. In our case $\widehat{\mathbf{z}}_i = \mathbf{h}_i(\mathbf{x}) = \mathbf{T}^{-1}(\mathbf{x}) \oplus \mathbf{z}_i$

At this point, the maximum-a-posteriori probability encoded in Eq. 3.39 can be manipulated as follows

$$\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\text{argmin}}\left\{-\log\left(p(\mathbf{x})\prod_i^N p(\mathbf{z}_i|\mathbf{x})\right)\right\} && (3.42)\\
&= \underset{\mathbf{x}}{\text{argmin}}\left\{\sum_i^N(\widehat{\mathbf{z}}_i - \mathbf{z}_i)^T\mathbf{\Omega}_i(\widehat{\mathbf{z}}_i - \mathbf{z}_i)\right\} && (3.43)\\
&= \underset{\mathbf{x}}{\text{argmin}}\left\{\sum_i^N \mathbf{e}_i(\mathbf{x})^T\mathbf{\Omega}_i\mathbf{e}_i(\mathbf{x})\right\} && (3.44)
\end{aligned}$$

Note that the squared l_2-norm cost function derived in Eq. 3.43 is correct only if the measurement noise is normally distributed. If such assumption changes, and for example the noise follows a Laplace distribution, the cost function will be the l_1-norm. The reader might notice the similarity of the last equation with the general definition of

a least-squares problem described by Eq. 3.16. For a detailed derivation of the solution of a least-squares problem see Sec. 3.2.

3.4 Graph-Based SLAM

Maximum-A-Posteriori (MAP)

The current state-of-the-art methods for Simultaneous Localization And Mapping model SLAM as a Maximum-A-Posteriori estimation problem. More formally, assume that we want to estimate an unknown variable $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$. Usually, in robotics, such variable includes the poses of the robot's trajectory and landmarks positions. Suppose also that a set of measurements $\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ is available so that each measurement can be expressed as a function of $\mathbf{x}$, i.e. $\mathbf{z}_i = \mathbf{h}_i(\mathbf{x}_i) + \epsilon_i$ where $\mathbf{x}_i \subseteq \mathbf{x}$, $\mathbf{h}_i(\cdot)$ is a known function also called *observation* or *measurement model*, and ϵ_i is some sort of measurement noise. In MAP, $\mathbf{x}$ is computed by finding the best configuration $\mathbf{x}^*$ that maximizes the posterior probability, or alternatively, the belief over $\mathbf{x}^*$ given the measurements

Observation model

$$\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmax}}\{p(\mathbf{x}|\mathbf{z})\} && (3.45)\\
\text{(Bayes Theorem)} &= \underset{\mathbf{x}}{\operatorname{argmax}}\{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})\} && (3.46)\\
\text{(Measurement Independence)} &= \underset{\mathbf{x}}{\operatorname{argmax}}\left\{p(\mathbf{x})\prod_i^N p(\mathbf{z}_i|\mathbf{x})\right\} && (3.47)\\
&= \underset{\mathbf{x}}{\operatorname{argmax}}\left\{p(\mathbf{x})\prod_i^N p(\mathbf{z}_i|\mathbf{x}_i)\right\} && (3.48)\\
&= \underset{\mathbf{x}}{\operatorname{argmin}}\left\{-\log\left(p(\mathbf{x})\prod_i^N p(\mathbf{z}_i|\mathbf{x}_i)\right)\right\} && (3.49)
\end{aligned}$$

where the last two equalities come from the fact that $\mathbf{z}_i$ depends only on the subset of variables in $\mathbf{x}_i$, and that maximizing the posterior is the same as minimizing the negative log-posterior.

Factor graph formulation

The problem described by Eq. 3.49 can be represented by using the *factor graph* formulation. More in detail, the variables map to nodes (or vertices) in the graph, while the prior $p(\mathbf{x})$ and measurement likelihood $p(\mathbf{z}_i|\mathbf{x}_i)$ terms correspond to factors (or edges) representing probability constraints among sets of nodes. A first advantage of the graph-based representation is that it gives a smart and easy visualization of the problem, by highlighting the underlying spatial structure of the nodes. Additionally, it allows to perform inference over complex problems with heterogeneous variables and factors, and arbitrary connections. As a drawback, however, the level of connectivity

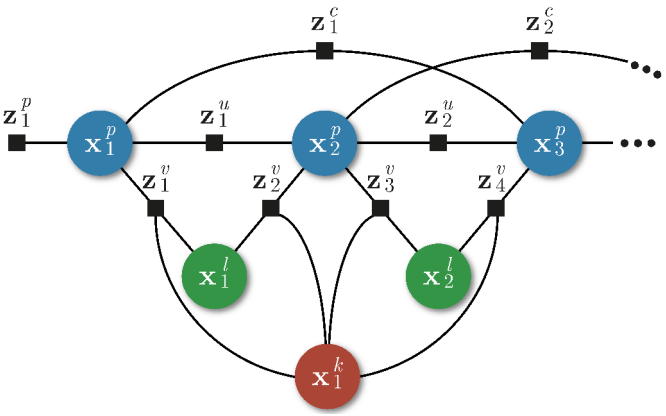


Figure 3.5: Example of factor graph. Blue, green and red circles represent respectively nodes associated to robot poses, landmark positions and the intrinsic camera calibration parameters. Factors are represented by black squares connecting sets of nodes: the "u" label marks odometry constraints, "v" represents camera observations, "c" indicates loop closures, and "p" denotes prior factors. (Image from [1]).

of the graph highly influences the sparsity of the SLAM problem. In fact, the less the graph is connected the more the the problem will be sparse. Sparsity allow for fast computation of the optimization. Fig. 3.5 shows an example of factor graph where blue, green and red circles represent respectively nodes associated to robot poses, landmark positions and the intrinsic camera calibration parameters. Factors are depicted with black squares connecting sets of nodes. In particular "u" labels mark odometry constraints, "v" represents camera observations, "c" indicates loop closures, and "p" denotes prior factors.

Assuming the measurement noise ϵ_i to be zero-mean normally distributed with information matrix $\mathbf{\Omega}_i$, we can rewrite in a more explicit form the likelihood of the measurement

Measurement likelihood

$$p(\mathbf{z}_i|\mathbf{x}_i) \propto \exp(-\frac{1}{2}(\mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i)^T \mathbf{\Omega}_i(\mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i)) \tag{3.50}$$

and the prior

Prior probability

$$p(\mathbf{x}) \propto \exp(-\frac{1}{2}(\mathbf{h}_0(\mathbf{x}) - \mathbf{z}_0)^T \mathbf{\Omega}_0(\mathbf{h}_0(\mathbf{x}) - \mathbf{z}_0)) \tag{3.51}$$

for some given function $\mathbf{h}_0(\cdot)$, prior mean $\mathbf{z}_0$ and information matrix $\mathbf{\Omega}_0$.

At this point, the Maximum-A-Posteriori probability encoded in Eq. 3.49 can be

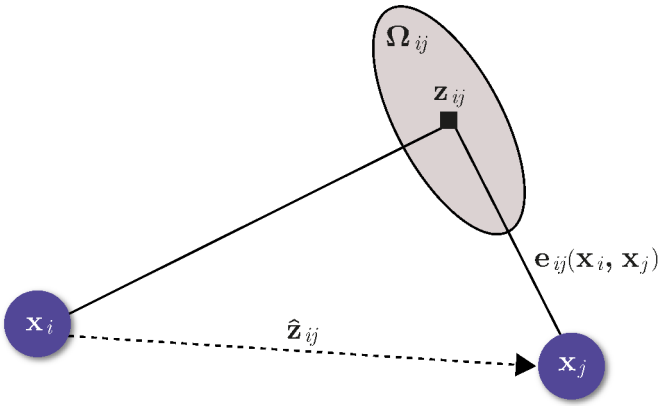


Figure 3.6: Detail of an edge connecting two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. The factor originates from the measurement $\mathbf{z}_{ij}$. From the the two nodes, we can compute the predicted measurement $\widehat{\mathbf{z}}_{ij} = \mathbf{h}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j)$ that represents $\mathbf{x}_j$ seen in the frame of $\mathbf{x}_i$. The error $\mathbf{e}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j)$ is the offset between the expected and the real measurement. A factor is fully described by its error function $\mathbf{e}_{ij}$ and by the information matrix $\mathbf{\Omega}_{ij}$ of the measurement that encodes its uncertainty. (Image from [17]).

manipulated as follows

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ -\log \left(p(\mathbf{x}) \prod_i^N p(\mathbf{z}_i|\mathbf{x}_i) \right) \right\} \tag{3.52}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_i^N (\widehat{\mathbf{z}}_i - \mathbf{z}_i)^T \mathbf{\Omega}_i (\widehat{\mathbf{z}}_i - \mathbf{z}_i) \right\} \tag{3.53}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_i^N \mathbf{e}_i(\mathbf{x}_i)^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x}_i) \right\} \tag{3.54}$$

Note that, like for the ICP probabilistic formulation (see Sec. 3.3.1), the squared l_2-norm cost function derived in Eq. 3.53 is correct only if the measurement noise is normally distributed. If this is not the case, and for example the noise follows a Laplace distribution, the cost function will be the l_1-norm. The reader might notice the similarity of the last equation with the general definition of a least-squares problem described by Eq. 3.16. Indeed, the factor graph formulation of SLAM can be reduced to a non-linear least-squares problem, that can be solved as described in Sec. 3.2.

3.4.1 Least-Squares SLAM

Now, for sake of simplicity of the derivation of the least-squares SLAM solution, let us assume that all factors connect only pairs of nodes. Fig. 3.6 contains a graph showing

the various aspects of an edge that connects two vertices. Under such condition, we can define the error associated to a factor connecting a node i, and a node j, as

$$\mathbf{e}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j) = \mathbf{h}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j) - \mathbf{z}_{ij} = \widehat{\mathbf{z}}_{ij} - \mathbf{z}_{ij} \tag{3.55}$$

With this error definition, Eq. 3.54 can be rewritten as follows

$$\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \overbrace{\sum_{\langle i,j\rangle \in \mathcal{C}} \underbrace{\mathbf{e}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j)^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j)}_{e_{ij}(\mathbf{x}_i,\ \mathbf{x}_j)}}^{F(\mathbf{x})} \right\} && (3.56)\\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_{\langle i,j\rangle \in \mathcal{C}} (\mathbf{h}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j) - \mathbf{z}_{ij})^T \boldsymbol{\Omega}_{ij} (\mathbf{h}_{ij}(\mathbf{x}_i,\ \mathbf{x}_j) - \mathbf{z}_{ij}) \right\} && (3.57)\\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_{\langle i,j\rangle \in \mathcal{C}} (\widehat{\mathbf{z}}_{ij} - \mathbf{z}_{ij})^T \boldsymbol{\Omega}_{ij} (\widehat{\mathbf{z}}_{ij} - \mathbf{z}_{ij}) \right\} && (3.58)
\end{aligned}$$

where $\mathcal{C}$ is the set of pairs of indices for which a factor $\mathbf{z}$ exists between the state variables $\mathbf{x}_i$ and $\mathbf{x}_j$.

As in the case of a standard least-squares problem (see Sec. 3.2), we approximate the error function by using the first order Taylor expansion around a current initial estimate $\breve{\mathbf{x}}$

$$\mathbf{e}_{ij}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) = \mathbf{e}_{ij}(\breve{\mathbf{x}}_i + \boldsymbol{\Delta}\mathbf{x}_i,\ \breve{\mathbf{x}}_j + \boldsymbol{\Delta}\mathbf{x}_j) \simeq \mathbf{e}_{ij} + \mathbf{J}_{ij}\boldsymbol{\Delta}\mathbf{x}_i \tag{3.59}$$

with $\mathbf{J}_{ij}$ being the Jacobian of $\mathbf{e}_{ij}(\mathbf{x})$ evaluated in $\breve{\mathbf{x}}$, and $\mathbf{e}_{ij} \overset{\text{def.}}{=} \mathbf{e}_{ij}(\breve{\mathbf{x}})$.
Recalling the derivation in Eq. 3.17, the local approximation is given by

$$e_{ij}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) = \boldsymbol{\Delta}\mathbf{x}^T \mathbf{H}_{ij} \boldsymbol{\Delta}\mathbf{x} + 2\mathbf{b}_{ij}\boldsymbol{\Delta}\mathbf{x} + c_{ij} \tag{3.60}$$

At this point we can rewrite the sum term in Eq. 3.56 as

$$\begin{aligned}
F(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) &= \sum_{\langle i,j\rangle \in \mathcal{C}} e_{ij}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x})\\
&= \boldsymbol{\Delta}\mathbf{x}^T \left[\sum_{\langle i,j\rangle \in \mathcal{C}} \mathbf{H}_{ij} \right] \boldsymbol{\Delta}\mathbf{x} + 2 \left[\sum_{\langle i,j\rangle \in \mathcal{C}} \mathbf{b}_{ij} \right] \boldsymbol{\Delta}\mathbf{x} + \left[\sum_{\langle i,j\rangle \in \mathcal{C}} c_{ij} \right]\\
&= \boldsymbol{\Delta}\mathbf{x}^T \mathbf{H} \boldsymbol{\Delta}\mathbf{x} + 2\mathbf{b}\boldsymbol{\Delta}\mathbf{x} + c && (3.61)
\end{aligned}$$

As usual, the previous equation is minimized by solving the linear system $\mathbf{H}\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{b}$, and the linearized solution is computed simply by integrating the increments as

$$\mathbf{x}^* = \breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}^* \tag{3.62}$$

Note that the Jacobian $\mathbf{J}_{ij}$ depends only on the unknown increments $\boldsymbol{\Delta}\mathbf{x}_i$ and $\boldsymbol{\Delta}\mathbf{x}_j$. This means that overall matrix will be mainly composed by zero entries except for the parts associated to the nodes that the factors connect. More formally, the Jacobian in this case will have the following form

Jacobian matrix

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0} \cdots \mathbf{0} & \overbrace{\underbrace{\left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{A}_{ij}}}^{\text{node } i} & \mathbf{0} \cdots \mathbf{0} & \overbrace{\underbrace{\left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{B}_{ij}}}^{\text{node } i} & \mathbf{0} \cdots \mathbf{0} \end{pmatrix} \tag{3.63}$$

As a consequence of this sparse structure, the edge will contribute in the construction of the matrix $\mathbf{H}$, and vector $\mathbf{b}$, only in the blocks related to the nodes connected by the factor. This results on the following structure for $\mathbf{H}_{ij}$

$$\mathbf{H}_{ij} = \begin{pmatrix} \ddots & & & \\ & \mathbf{A}_{ij}^T\boldsymbol{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{A}_{ij}^T\boldsymbol{\Omega}_{ij}\mathbf{B}_{ij} \\ & \vdots & \ddots & \vdots \\ & \mathbf{B}_{ij}^T\boldsymbol{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{B}_{ij}^T\boldsymbol{\Omega}_{ij}\mathbf{B}_{ij} \\ & & & \ddots \end{pmatrix} \tag{3.64}$$

and $\mathbf{b}_{ij}$

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{A}_{ij}^T\boldsymbol{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T\boldsymbol{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \end{pmatrix} \tag{3.65}$$

All the other entries will be zero. Since each factor will contribute only to the blocks $\mathbf{H}_{[ii]}$, $\mathbf{H}_{[ij]}$, $\mathbf{H}_{[ji]}$ and $\mathbf{H}_{[jj]}$ of $\mathbf{H}$, and to the blocks $\mathbf{b}_{[i]}$ and $\mathbf{b}_{[j]}$ of $\mathbf{b}$, it is possible to perform a selective and fast update of these structures. Additionally, note that $\mathbf{H}$ is symmetric and therefore we can concentrate only on the upper triangular part of the matrix since $\mathbf{H}_{[ji]} = \mathbf{H}_{[ij]}^T$. Consider also that, the error associated to a factor depends on the relative position between the nodes connecting the factor itself. Since $F(\mathbf{x})$ is invariant with respect to a rigid transformation of all nodes, this makes the linear system under determined. To overcome this problem, it is common practice to constraint one of the increments $\boldsymbol{\Delta}\mathbf{x}_k$ to be zero. This is done by adding an identity matrix, appropriately scaled, to the diagonal block $\mathbf{H}_{[kk]}$ associated to the k^{th} node.

3.4.2 Least-Squares SLAM on Smooth Manifolds

As explained in Sec. 3.2.1, applying the previous solution to SLAM problems that live in non-Euclidean spaces would not work smoothly. However if the space we are working on is a smooth manifold, like in most of the cases in robotics, we can define a $\boxplus$ operator (Eq. 3.25) that applies a local perturbation ($\boldsymbol{\Delta}\tilde{\mathbf{x}}$) in the Euclidean space to a variation in the manifold ($\mathbf{x}$). This operator let us to define the new error function

$$\begin{aligned} \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}) &= \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}}_i \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_i,\ \breve{\mathbf{x}}_j \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_j) \\ &\simeq \tilde{\mathbf{e}}_{ij} + \tilde{\mathbf{J}}_{ij}\boldsymbol{\Delta}\tilde{\mathbf{x}} \end{aligned} \tag{3.66}$$

where $\tilde{\mathbf{e}}_{ij} \overset{\text{def.}}{=} \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}})$ and $\tilde{\mathbf{J}}_{ij}$ is

$$\tilde{\mathbf{J}}_{ij} = \left.\frac{\partial \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}} \tag{3.67}$$

The reader can notice that the previous equation depends only on the increments $\boldsymbol{\Delta}\tilde{\mathbf{x}}_i$ and $\boldsymbol{\Delta}\tilde{\mathbf{x}}_j$, this allows a further expansion that brings us to the following final form of the Jacobian

Jacobian matrix on manifolds

$$\tilde{\mathbf{J}}_{ij} = \begin{pmatrix} \mathbf{0} \cdots \mathbf{0} & \underbrace{\left.\frac{\partial \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}}_{\tilde{\mathbf{A}}_{ij}} & \mathbf{0} \cdots \mathbf{0} & \underbrace{\left.\frac{\partial \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_j}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}}_{\tilde{\mathbf{B}}_{ij}} & \mathbf{0} \cdots \mathbf{0} \end{pmatrix} \tag{3.68}$$

and recalling Eq. 3.32 we get

$$\left.\frac{\partial \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}} = \underbrace{\left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{A}_{ij}} \cdot \underbrace{\left.\frac{\partial(\breve{\mathbf{x}}_i \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_i)}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}}_{\mathbf{M}_i} \tag{3.69}$$

$$\left.\frac{\partial \tilde{\mathbf{e}}_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_j}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}} = \underbrace{\left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{B}_{ij}} \cdot \underbrace{\left.\frac{\partial(\breve{\mathbf{x}}_j \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_j)}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_j}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}}_{\mathbf{M}_j} \tag{3.70}$$

$$\tag{3.71}$$

At this point we solve the linear system $\tilde{\mathbf{H}}\boldsymbol{\Delta}\tilde{\mathbf{x}}^* = -\tilde{\mathbf{b}}$, where

$$\tilde{\mathbf{H}} = \sum_{\langle i,j \rangle \in \mathcal{C}} \tilde{\mathbf{H}}_{ij} = \sum_{\langle i,j \rangle \in \mathcal{C}} \tilde{\mathbf{J}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{J}}_{ij} \tag{3.72}$$

$$\tilde{\mathbf{b}} = \sum_{\langle i,j \rangle \in \mathcal{C}} \tilde{\mathbf{b}}_{ij} = \sum_{\langle i,j \rangle \in \mathcal{C}} \tilde{\mathbf{e}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{J}}_{ij} \tag{3.73}$$

Input: state initial guess $\breve{\mathbf{x}} = \{\breve{\mathbf{x}}_1, \cdots, \breve{\mathbf{x}}_T\}$ and factors $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \boldsymbol{\Omega}_{ij} \rangle\}$

Output: new solution $\mathbf{x}^*$ and new information matrix $\mathbf{H}^*$

repeat

  forall $\breve{\mathbf{x}}_i \in \breve{\mathbf{x}}$ **do**

    $\mathbf{M}_i \leftarrow \frac{\partial (\breve{\mathbf{x}}_i \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_i)}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}\Big|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}$

  end

  $\tilde{\mathbf{H}} \leftarrow \mathbf{0}, \quad \tilde{\mathbf{b}} \leftarrow \mathbf{0}$

  forall $\langle \mathbf{e}_{ij}, \boldsymbol{\Omega}_{ij} \rangle \in \mathcal{C}$ **do**

    $\mathbf{A}_{ij} \leftarrow \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\breve{\mathbf{x}}}, \quad \mathbf{B}_{ij} \leftarrow \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\Big|_{\mathbf{x}=\breve{\mathbf{x}}}$

    $\tilde{\mathbf{A}}_{ij} \leftarrow \mathbf{A}_{ij}\mathbf{M}_i, \quad \tilde{\mathbf{B}}_{ij} \leftarrow \mathbf{B}_{ij}\mathbf{M}_j$

    $\tilde{\mathbf{H}}_{[ii]} \leftarrow \tilde{\mathbf{H}}_{[ii]} + \tilde{\mathbf{A}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{A}}_{ij}, \quad \tilde{\mathbf{H}}_{[ij]} \leftarrow \tilde{\mathbf{H}}_{[ij]} + \tilde{\mathbf{A}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{B}}_{ij}$

    $\tilde{\mathbf{H}}_{[ji]} \leftarrow \tilde{\mathbf{H}}_{[ij]}^T, \quad \tilde{\mathbf{H}}_{[jj]} \leftarrow \tilde{\mathbf{H}}_{[jj]} + \tilde{\mathbf{B}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{B}}_{ij}$

    $\tilde{\mathbf{b}}_{[i]} \leftarrow \tilde{\mathbf{b}}_{[i]} + \tilde{\mathbf{A}}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}, \quad \tilde{\mathbf{b}}_{[j]} \leftarrow \tilde{\mathbf{b}}_{[j]} + \tilde{\mathbf{B}}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}$

  end

  $\mathbf{H}_{[11]} \leftarrow \mathbf{H}_{[11]} + \mathbf{I}$

  $\boldsymbol{\Delta}\tilde{\mathbf{x}} \leftarrow solve(\tilde{\mathbf{H}}\boldsymbol{\Delta}\tilde{\mathbf{x}} = -\tilde{\mathbf{b}})$

  forall $\breve{\mathbf{x}}_i \in \breve{\mathbf{x}}$ **do**

    $\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}$

  end

until $\neg$converged

$\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$

$\mathbf{H}^* \leftarrow \mathbf{0}$

forall $\langle \mathbf{e}_{ij}, \boldsymbol{\Omega}_{ij} \rangle \in \mathcal{C}$ **do**

  $\tilde{\mathbf{H}}^*_{[ii]} \leftarrow \tilde{\mathbf{H}}_{[ii]} + \tilde{\mathbf{A}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{A}}_{ij}, \quad \tilde{\mathbf{H}}^*_{[ij]} \leftarrow \tilde{\mathbf{H}}_{[ij]} + \tilde{\mathbf{A}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{B}}_{ij}$

  $\tilde{\mathbf{H}}^*_{[ji]} \leftarrow \tilde{\mathbf{H}}_{[ij]}^T, \quad \tilde{\mathbf{H}}^*_{[jj]} \leftarrow \tilde{\mathbf{H}}_{[jj]} + \tilde{\mathbf{B}}_{ij}^T \boldsymbol{\Omega}_{ij} \tilde{\mathbf{B}}_{ij}$

end

Algorithm 6: Manifold-based Gauss-Newton SLAM algorithm that computes both the mean, and the information matrix, of the posterior over a set of nodes.

and we apply the increments $\boldsymbol{\Delta}\tilde{\mathbf{x}}^*$ through the $\boxplus$ operator $\mathbf{x}^* = \breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}^*$. Algorithm 6 lists a manifold-based Gauss-Newton algorithm that computes both the mean, and the information matrix, of the posterior over a set of nodes. We omit the non-manifold version since it is just a reduced instance of the manifold-based one.

Part II

POINT CLOUD REGISTRATION

Chapter 4

Augmented Measurements to Improve the Convergence of ICP

"Measure what is measurable,
and make measurable what is not so."
— Galileo Galilei, 1564-1642

REGISTERING two point clouds is a building block of many robot applications such as Simultaneous Localization And Mapping, object recognition and detection, augmented reality and many others. This problem is commonly solved by variants of the Iterative Closest Point algorithm proposed by Besl and McKay [26].

Researchers focused on improving the original formulation of ICP by looking for heuristics that provide "good" correspondences. The original idea of picking up the closest point [26] has been progressively refined to consider features, curvature and other characteristics of the measurements. However, within ICP and its variants, the optimization and the correspondence search are not independent. If the optimization is robust to outliers and exhibits a smooth behavior, the chances that it finds a better solution at the subsequent step increases. Despite the development of approaches like Generalized ICP [12], no existing method improve the objective function of the least-squares problem associated to the alignment.

Since point clouds are the effects of sampling a surface, the local characteristics of such a surface play an important role in the optimization. From this point of view, the objective function has to express some distance between *surface samples*, and the optimization algorithm has to determine the optimal alignment between them. A surface sample, however, is not fully described just by 3D points, but it requires additional cues like the surface normal, the curvature and, potentially, the direction of the edge. GICP minimizes a distance between corresponding points, while it neglects additional cues that can indeed play a role in determining the transformation and in rejecting

outliers.

In this Chapter we present a novel variant of the objective function of ICP that is optimized when computing the transformation. This function depends not only on the relative point distance, but also on the difference between surface normals, or tangents in case the point lies on an edge. We provide an iterative form for the optimization routine and we show, through experiments performed on synthetic data and a standard benchmarking dataset, that our approach outperforms other state-of-the-art techniques, both in terms of convergence speed and robustness.

4.1 Extending the Measurements

Let $\mathbf{n}_i$ be the normal of a point $\mathbf{p}_i$ belonging to a certain surface, and $\boldsymbol{\tau}_i$ its tangent if the point is part of an edge, we can then extend Eq. 3.33 as follows

$$\begin{aligned}\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} & \left(\mathbf{T} \oplus \mathbf{p}_j^{\mathrm{r}} - \mathbf{p}_i^{\mathrm{c}}\right)^T \boldsymbol{\Omega}_{ij}^{\mathrm{p}} \left(\mathbf{T} \oplus \mathbf{p}_j^{\mathrm{r}} - \mathbf{p}_i^{\mathrm{c}}\right) \\ + & \sum_{\mathcal{C}} \left(\mathbf{T} \oplus \mathbf{n}_j^{\mathrm{r}} - \mathbf{n}_i^{\mathrm{c}}\right)^T \boldsymbol{\Omega}_{ij}^{\mathrm{n}} \left(\mathbf{T} \oplus \mathbf{n}_j^{\mathrm{r}} - \mathbf{n}_i^{\mathrm{c}}\right) \\ + & \sum_{\mathcal{C}} \left(\mathbf{T} \oplus \boldsymbol{\tau}_j^{\mathrm{r}} - \boldsymbol{\tau}_i^{\mathrm{c}}\right)^T \boldsymbol{\Omega}_{ij}^{\tau} \left(\mathbf{T} \oplus \boldsymbol{\tau}_j^{\mathrm{r}} - \boldsymbol{\tau}_i^{\mathrm{c}}\right)\end{aligned} \tag{4.1}$$

Here $\mathbf{n}_i^{\mathrm{c}}$, $\mathbf{n}_j^{\mathrm{r}}$ and $\boldsymbol{\Omega}_{ij}^{\mathrm{n}}$ represent respectively the normal of the point $\mathbf{p}_i^{\mathrm{c}}$ and $\mathbf{p}_j^{\mathrm{r}}$, and the information matrix of the correspondence among the two normals. Similarly, $\boldsymbol{\tau}_i^{\mathrm{c}}$, $\boldsymbol{\tau}_j^{\mathrm{r}}$ and $\boldsymbol{\Omega}_{ij}^{\tau}$ are the tangents and the information matrix of the correspondence among the two tangents. We recall that, if $\mathbf{T}$ is a transformation described by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$, the $\oplus$ operator has different definitions depending on its arguments

$$\mathbf{T} \oplus \mathbf{x} = \begin{cases} \mathbf{R} \cdot \mathbf{x} + \mathbf{t} & \text{if } \mathbf{x} \text{ is a point} \\ \mathbf{R} \cdot \mathbf{x} & \text{if } \mathbf{x} \text{ is a tangent or a normal} \end{cases} \tag{4.2}$$

A Mahalanobis distance between two point clouds can be measured by considering also the *distances of corresponding normals* and *corresponding tangents* after applying the transformation $\mathbf{T}$, as shown in Eq. 4.1.

By defining an extended point $\tilde{\mathbf{p}}$ as a vector consisting of a point $\mathbf{p}$, its normal $\mathbf{n}$ and its tangent $\boldsymbol{\tau}$, we have a straightforward modification of the $\oplus$ operator as

$$\tilde{\mathbf{p}} = \left(\mathbf{p}^T,\ \mathbf{n}^T,\ \boldsymbol{\tau}^T\right)^T \qquad \mathbf{T} \oplus \tilde{\mathbf{p}} = \left((\mathbf{R}\mathbf{p} + \mathbf{t})^T,\ (\mathbf{R}\mathbf{n})^T,\ (\mathbf{R}\boldsymbol{\tau})^T\right)^T \tag{4.3}$$

Eq. 4.1 can be, then, compactly rewritten in terms of extended points as

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} \left(\mathbf{T} \oplus \tilde{\mathbf{p}}_j^{\mathrm{r}} - \tilde{\mathbf{p}}_i^{\mathrm{c}}\right)^T \tilde{\boldsymbol{\Omega}}_{ij} \left(\mathbf{T} \oplus \tilde{\mathbf{p}}_j^{\mathrm{r}} - \tilde{\mathbf{p}}_i^{\mathrm{c}}\right) \tag{4.4}$$

where $\tilde{\mathbf{\Omega}}_{ij} = \text{diag}(\mathbf{\Omega}^{\text{p}}_{ij},\ \mathbf{\Omega}^{\text{n}}_{ij},\ \mathbf{\Omega}^{\tau}_{ij})$ summarizes the contribution of $\mathbf{\Omega}^{\text{p}}_{ij}$, $\mathbf{\Omega}^{\text{n}}_{ij}$ and $\mathbf{\Omega}^{\tau}_{ij}$. The function $\text{diag}(\mathbf{a}_1,\ \ldots,\ \mathbf{a}_k)$ stands for a diagonal matrix whose entries, starting from the upper left corner, are $\mathbf{a}_1,\ \ldots, \mathbf{a}_k$. If the point is not sampled from a locally planar surface, nor from an edge, a reasonable distance metric is the Euclidean distance. For these points, we fall back to the ICP case, which is enclosed in Eq. 4.4 by setting the information matrices of the tangent and the normal to the null matrix: $\mathbf{\Omega}^{\text{n}}_{ij} = \mathbf{\Omega}^{\tau}_{ij} = \mathbf{0}$.

When measuring the distance between two planar patches, it is reasonable to neglect displacements along the tangent direction of the plane, while errors along the normal direction should be more severely penalized. Additionally, the normals of the two planes should be as close as possible. However, *this constraint cannot be enforced when using only 3D points*. To obtain this behavior from the error function, we can impose ${\mathbf{\Omega}^{\text{p}}_{ij}}^{-1}$ to be a disc lying on the surface around $\mathbf{p}^{\text{c}}_i$, as done in [12]. Since the tangent is not defined in a planar patch, we set $\mathbf{\Omega}^{\tau}_{ij} = \mathbf{0}$. Additionally, we set the covariance matrix ${\mathbf{\Omega}^{\text{n}}_{ij}}^{-1}$ of the normal to have a shape which is elongated in the normal direction. In this way the error between the normals introduces a strong momentum that "forces" them to have the same direction. Conversely, when measuring the distance between two edges, it is reasonable to slide them onto each other along the tangent direction. This behavior can be obtained by enforcing ${\mathbf{\Omega}^{\text{p}}_{ij}}^{-1}$ to have a prolonged shape, and to lie along the tangent direction. The tangent $\boldsymbol{\tau}$, instead, can be used to penalize two edges not lying on the same direction by setting $\mathbf{\Omega}^{\tau}_{ij}$ to have a shape which is elongated in the direction of $\boldsymbol{\tau}$. Since an edge has no normal, $\mathbf{\Omega}^{\text{n}}_{ij}$ has to be set to $\mathbf{0}$. Table 4.1 shows the different configurations of the matrices $\mathbf{\Omega}^{\text{p}}_{ij}$, $\mathbf{\Omega}^{\text{n}}_{ij}$ and $\mathbf{\Omega}^{\tau}_{ij}$.

The reader might notice that tangents and normals are mutually exclusive. Since the contributions of the tangent and the normal components to the χ^2_{ij} have the same matrix dimensions, we can further simplify the extended point $\tilde{\mathbf{p}}$ by partitioning it into an affine part $\mathbf{p}$, and in a linear part $\mathbf{l}$. The former is subject to translations and rotations, the latter only to the rotation. Without loss of generality, in this way it is possible to reduce the dimension of the error function, and to speed up the calculation.

Table 4.1: This table summarizes the components of the information matrix used in our algorithm, depending on the type of the structure around a point. $\mathbf{R}_{n_i}$ and $\mathbf{R}_{\tau_i}$ are two rotation matrices that bring the y axis respectively along the direction of the normal $\mathbf{n}_i$, or the tangent $\boldsymbol{\tau}_i$. ϵ is a small value (10^{-3} in our experiments).

Case	$\mathbf{\Omega}^{\text{p}}_{ij}$	$\mathbf{\Omega}^{\text{n}}_{ij}$	$\mathbf{\Omega}^{\tau}_{ij}$
planar	$\mathbf{R}_{n_i}\text{diag}(\frac{1}{\epsilon},\ 1,\ 1)\mathbf{R}^T_{n_i}$	$\mathbf{R}_{n_i}\text{diag}(\frac{1}{\epsilon},\ \frac{1}{\epsilon},\ 1)\mathbf{R}^T_{n_i}$	$\mathbf{0}$
edge	$\mathbf{R}_{\tau_i}\text{diag}(\frac{1}{\epsilon},\ \frac{1}{\epsilon},\ 1)\mathbf{R}^T_{\tau_i}$	$\mathbf{0}$	$\mathbf{R}_{\tau_i}\text{diag}(\frac{1}{\epsilon},\ \frac{1}{\epsilon},\ 1)\mathbf{R}^T_{\tau_i}$
none	$\mathbf{I}$	$\mathbf{0}$	$\mathbf{0}$

We therefore define a compact form for an extended point $\tilde{\mathbf{p}}$ as

$$\tilde{\mathbf{p}} = \left(\mathbf{p}^T,\ \mathbf{l}^T\right)^T \qquad \mathbf{T} \oplus \tilde{\mathbf{p}} = \left((\mathbf{R}\mathbf{p} + \mathbf{t})^T,\ (\mathbf{R}\mathbf{l})^T\right)^T \tag{4.5}$$

According to the new formalism, the objective function in Eq. 4.4 becomes

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} \left(\mathbf{T} \oplus \tilde{\mathbf{p}}_j^{\mathrm{r}} - \tilde{\mathbf{p}}_i^{\mathrm{c}}\right)^T \tilde{\mathbf{\Omega}}_{ij} \underbrace{\left(\mathbf{T} \oplus \tilde{\mathbf{p}}_j^{\mathrm{r}} - \tilde{\mathbf{p}}_i^{\mathrm{c}}\right)}_{\tilde{\mathbf{e}}_{ij}(\mathbf{T})} \tag{4.6}$$

where $\tilde{\mathbf{\Omega}}_{ij} = \mathrm{diag}(\mathbf{\Omega}_{ij}^{\mathrm{p}},\ \mathbf{\Omega}_{ij}^{\mathrm{l}})$, and the information matrices must be modified according to Table 4.2.

4.2 Carrying Out the Optimization

In this section we present the procedure for the minimization described in Eq. 4.6 by using a strengthened least-squares procedure. The input of this algorithm are two sets of extended points $\tilde{\mathbf{p}}_{1:n}^{\mathrm{r}}$ and $\tilde{\mathbf{p}}_{1:m}^{\mathrm{c}}$, a (noisy) set of candidate correspondences $\mathcal{C} = \langle i, j \rangle_{1:M}$ and the information matrix $\tilde{\mathbf{\Omega}}_{ij}$, computed according to Table 4.2. The aim of this procedure is to find the transform $\mathbf{T}^*$ that minimizes the following objective or cost function

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} \underbrace{\tilde{\mathbf{e}}_{ij}\left(\mathbf{T}\right)^T \tilde{\mathbf{\Omega}}_{ij} \tilde{\mathbf{e}}_{ij}\left(\mathbf{T}\right)}_{\tilde{\chi}_{ij}^2} \tag{4.7}$$

Each correspondence contributes to the overall cost function by the scalar term $\tilde{\chi}_{ij}^2$.

As previously explained, the minimizing $\mathbf{T}^*$ of Eq. 4.7 can be found by iteratively solving the following linear system

$$\mathbf{H}\mathbf{\Delta}\mathbf{T} = -\mathbf{b} \tag{4.8}$$

where $\mathbf{H} = \sum_{\mathcal{C}} \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{J}_{ij}$ is the Hessian matrix, $\mathbf{b} = \sum_{\mathcal{C}} \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \tilde{\mathbf{e}}_{ij}$ is the coefficient vector and $\mathbf{J}_{ij}$ is the Jacobian of the error function. At each iteration we compute an

Table 4.2: This table summarizes the components of the information matrix for our algorithm when using a reduced representation.

Case	$\mathbf{l}_i$	$\mathbf{\Omega}_{ij}^{\mathrm{p}}$	$\mathbf{\Omega}_{ij}^{\mathrm{l}}$
planar	$\mathbf{n}_i$	$\mathbf{R}_{n_i}\mathrm{diag}(\frac{1}{\epsilon},\ 1,\ 1)\mathbf{R}_{n_i}^T$	$\mathbf{R}_{n_i}\mathrm{diag}(\frac{1}{\epsilon},\ \frac{1}{\epsilon},\ 1)\mathbf{R}_{n_i}^T$
edge	$\boldsymbol{\tau}_i$	$\mathbf{R}_{\tau_i}\mathrm{diag}(\frac{1}{\epsilon},\ \frac{1}{\epsilon},\ 1)\mathbf{R}_{\tau_i}^T$	$\mathbf{R}_{\tau_i}\mathrm{diag}(\frac{1}{\epsilon},\ \frac{1}{\epsilon},\ 1)\mathbf{R}_{\tau_i}^T$
none	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{0}$

improved solution from the previous transform $\mathbf{T}$ using $\mathbf{H}$ and $\mathbf{b}$. By solving the linear system in Eq. 4.8 we determine a perturbation $\mathbf{\Delta T}$ which is applied to the previous transform $\mathbf{T}$ in order to reduce the error. The transform $\mathbf{T}$ of the next iteration is thus computed as

$$\mathbf{T} = \mathbf{\Delta T} \oplus \mathbf{T} \tag{4.9}$$

For readers interested in further details on the derivation of Eq. 4.8 please refer to Sec. 3.2.

In our approach, a perturbation $\mathbf{\Delta T}$ is defined as a vector composed of two parts $(\mathbf{\Delta t}^T,\ \mathbf{\Delta q}^T)^T$ where $\mathbf{\Delta t} = (\Delta t_x,\ \Delta t_y,\ \Delta t_z)^T$ is a translation vector, and $\mathbf{\Delta q} = (\Delta q_x,\ \Delta q_y\ \Delta q_z)^T$ is the imaginary part of the normalized quaternion used to represent an incremental rotation. If $\mathbf{\Delta T} = \mathbf{0}$, the matrix perturbation form is the 4-by-4 identity matrix. Under this parameterization, the Jacobian $\mathbf{J}_{ij}$ with respect to the local perturbation $\mathbf{\Delta T}$ is computed as

$$\mathbf{J}_{ij} = \left.\frac{\partial \mathbf{e}_{ij}\left(\mathbf{\Delta T} \oplus \mathbf{T}\right)}{\partial \mathbf{\Delta T}}\right|_{\mathbf{\Delta T}=\mathbf{0}} = \begin{pmatrix} \mathbf{I} & -2[\mathbf{T} \oplus \mathbf{p}_j^{\mathrm{r}}]_\times \\ \mathbf{0} & -2[\mathbf{T} \oplus \mathbf{l}_j^{\mathrm{r}}]_\times \end{pmatrix} \tag{4.10}$$

where $[\mathbf{v}]_\times$ is the cross product matrix of the vector $\mathbf{v}$ (see Appendix B and Appendix C). In practice, by exploiting the block structure and the sparsity of the Jacobian, it is possible to compute efficiently the linear system in Eq. 4.8.

In order to be robust to the presence of outliers, which usually significantly contribute to the error, the proposed scheme has to be further modified. To reduce the contribution of these wrong terms, we further robustify the error function by clamping the norm of the χ^2_{ij} of each point to a maximum value. This method is known has "winsorization". To this end, we compute a scaling factor γ_{ij} for each information matrix $\tilde{\mathbf{\Omega}}_{ij}$ as

$$\gamma_{ij} = \begin{cases} 1 & \text{if } \chi^2_{ij} < K \\ \frac{K}{\chi^2_{ij}} & \text{otherwise} \end{cases} \tag{4.11}$$

where K is a thresholding value of the χ^2_{ij} error that discriminates when a pair of corresponding points i and j is considered an outlier or not. Even if the correct correspondences are rejected at the beginning of the iterative process, these will be considered again as the system converges towards a better solution, since their χ^2_{ij} will decrease.

Finally, to smooth the convergence we added a damping factor to the linear system in Eq. 4.8. In practice, $\mathbf{\Delta T}$ is found by solving the damped linear system $(\mathbf{H}+\lambda\mathbf{I})\mathbf{\Delta T} = -\mathbf{b}$, since it prevents the solution to take too large steps that might be caused by non-linearities or wrong correspondences.

4.3 Optimization Summary

In this section we wrap-up the ideas discussed above and we provide the sketch of an iterative algorithm for the optimization of Eq. 4.6. At each iteration our algorithm computes an improved transform from the current estimate $\mathbf{T}$ by executing the following steps:

1. compute the information matrices $\tilde{\mathbf{\Omega}}_{ij}$ according to Table 4.2;
2. compute the error vector $\tilde{\mathbf{e}}_{ij}$ as shown in Eq. 4.6;
3. compute the Jacobian $\mathbf{J}_{ij}$ according to Eq. 4.10;
4. compute the $\tilde{\chi}^2_{ij}$ as in Eq. 4.7 and the scaling factor γ_{ij} from Eq. 4.11;
5. compute a scaled version of the Hessian and of the coefficient vector as

$$\mathbf{H} = \sum_{\mathcal{C}} \gamma_{ij} \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{J}_{ij} \qquad \mathbf{b} = \sum_{\mathcal{C}} \gamma_{ij} \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \tilde{\mathbf{e}}_{ij}$$

6. solve the linear system of Eq. 4.8 to find the perturbation $\mathbf{\Delta T}$;
7. compute the improved transformation $\mathbf{T}$ as in Eq. 4.9.

4.4 Experiments

We validated our approach both on real and synthetic data. The real world experiments were conducted on a publicly available benchmarking dataset, and they show the performances of our optimization algorithm when included in a full ICP system. The experiments on synthetic data, instead, allow to characterize the behavior of our approach under different levels of sensor noise and outlier ratios.

4.4.1 Real World Experiments

For the real world experiments we used the benchmarking dataset by Stuerm *et al.* [154]. Each test consists in a sequence of depth and RGB images acquired with a calibrated RGB-D camera in a reference scene. Note that, even if our approach is not restricted to the use of depth images, we decided to use this dataset since it is labeled with the ground-truth values of the poses of the camera. We do not make use of the RGB channels.

In order to provide input data to the algorithm illustrated in the previous section, we processed the point cloud $\mathcal{P}$ generated from each depth image by extracting the local surface characteristics of all measurements. This is done by approximating all points

that lie within a fixed ball centered in $\mathbf{p}_i$ with a Gaussian distribution $\mathcal{N}(\mu_i^s,\ \mathbf{\Sigma}_i^s)$ whose parameters are computed as described in Eq. 3.35.

For determining if a point lies on a corner, an edge or a flat surface, we analyze the eigenvalues of its covariance matrix $\mathbf{\Sigma}_i^s$. If all eigenvalues have more or less the same magnitude, we assume the point is on a corner. If one of the eigenvalues is smaller with respect to the other two, we assume the point lies on an edge. Finally, if one of the eigenvalues is smaller of some order of magnitude with respect to the others then we assume the point is on a planar patch. This discrimination is necessary to compute the correct information matrices, according to Table 4.2.

Given two clouds to be aligned, we search the correspondences using a line of sight criterion over the depth images, we reject the correspondences whose normals are too different, and we execute one iteration of the optimization. Note that ICP and point-to-plane GICP are special cases that can be captured by our algorithm just by modifying the way the information matrices are computed. To focus our analysis on the objective function we left all parts of the system unchanged, including the correspondence selection. This represents an advantage for plain ICP, since normally it does not rely on the surface normals to reject wrong associations.

For each test, we incrementally aligned one frame to the previous one. For each iteration of the alignment, we compared the difference between the current solution and the ground-truth. Each attempted alignment produced a plot which shows the evolution of the rotational and translational error. For compactness, we provide here only the mean error plots obtained by averaging all errors of a run[1]. The reader who is interested in the individual plots of each alignment, can find them at `http://www.dis.uniroma1.it/~serafin/publications/icp-augmented-measurements`.

In order to measure the robustness of the alignments to wrong initial guesses, we performed several runs of the experiments by considering different levels of frame skip rate. If we run an evaluation at a frame skip rate equal to N, this means that we register a point cloud each N, and we discard the others. Table 4.3 shows the average evolution of the rotational and the translational error on three different sequences, and at different frame skip rates.

The plotted results point out that our novel objective function in general performs better than the other approaches, in particular in terms of convergence speed. This is true especially for the rotational part of the error since it is influenced directly by the normals. Also in the case where no frame was skipped (first column of Table 4.3), GICP required twice the number of iterations to converge to the results with respect to our approach. Moreover, ICP and GICP showed much less robustness to frame skipping (second and third column of Table 4.3).

[1] With run we denote all the alignment over a single test with a certain frame skip rate.

Table 4.3: Average evolution of the translational and rotational error for three different sequences of the benchmarking dataset considered, at different frame skip rates. Our approach is labeled "nicp" in the captions of the images.

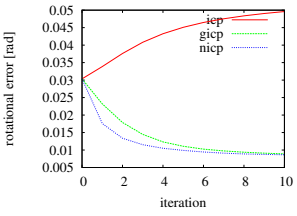
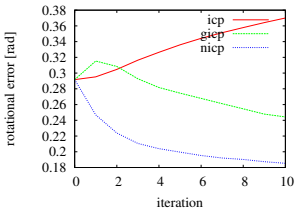
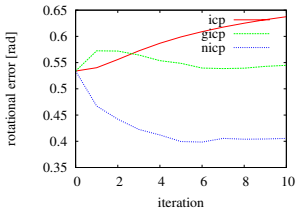
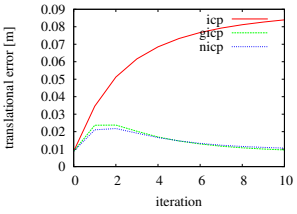
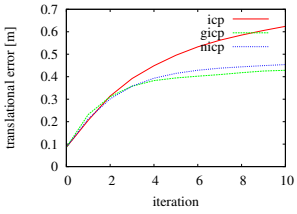
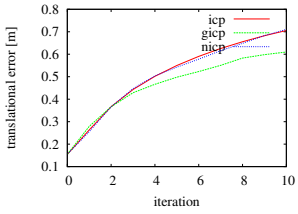
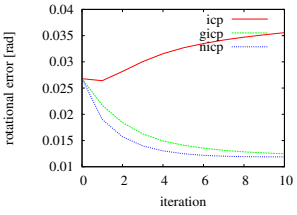
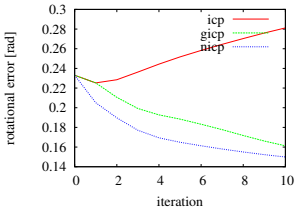
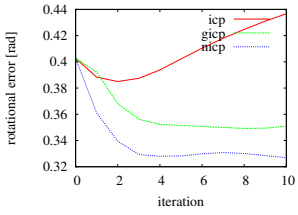
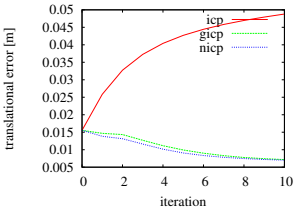
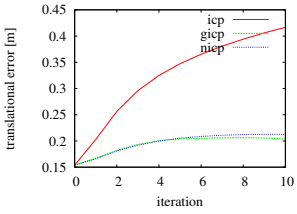
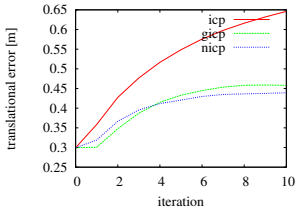
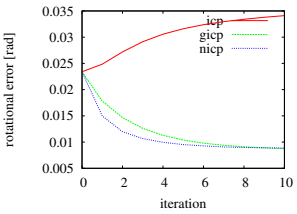
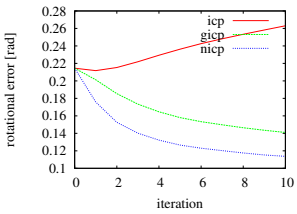
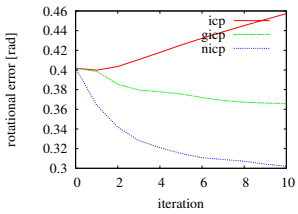
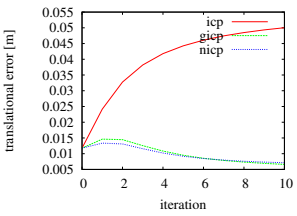
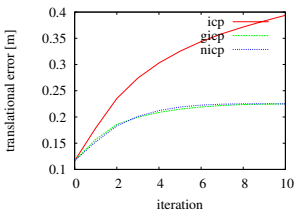
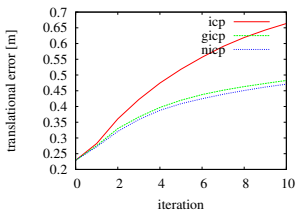
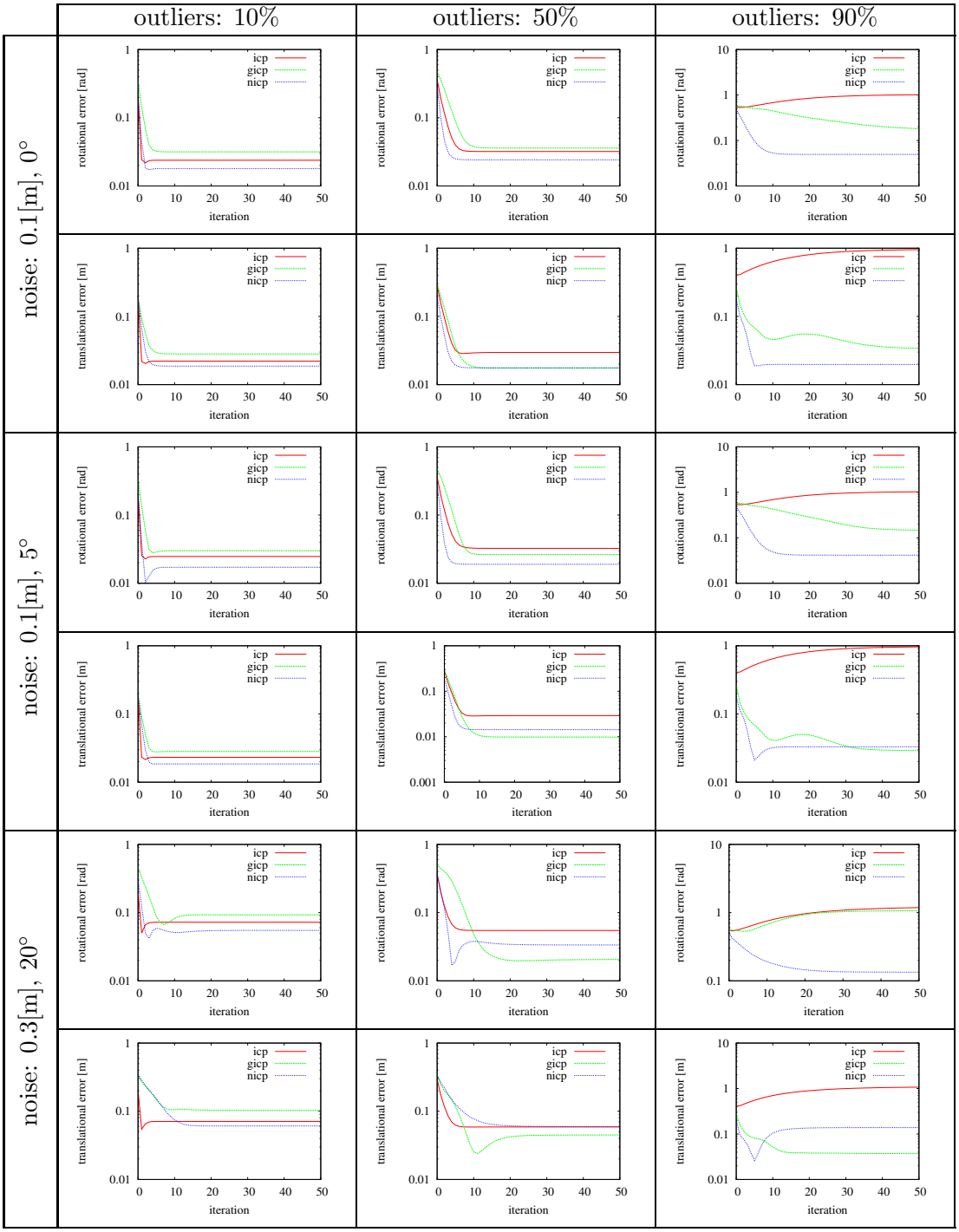
	frame skip rate: 1	frame skip rate: 10	frame skip rate: 20
sequence: fr1/360			
			
sequence: fr1/desk2			
			
sequence: fr1/room			
			

Table 4.4: Average evolution of the translational and rotational error at different outlier ratios and levels of noise affecting the measurements of the points (standard deviation, in meters) and the normals (standard deviation in degrees). Our approach is labeled "nicp" in the captions of the images.



4.4.2 Experiments on Synthetic Data

We conducted experiments on synthetic data in order to assess the effects of the inliers, and of the sensor error, on our optimization function. To this end we generated a scene consisting of about 300k 3D points with normals and tangents arranged on surfaces. Then, we computed the correct correspondences and ideal measurements and we corrupted them. This process has been performed by injecting a variable fraction of random outliers and perturbing the measurements by adding Gaussian noise to the points and normal estimates. For each setting we ran our approach, ICP and GICP, and we plotted the evolution of the translational and rotational error. The results are shown in Table 4.4.

Overall the experiments on synthetic data reflect the behavior of the real world ones. Shortly, using additional information in the error function makes the approach more robust and accelerates the convergence. This is particularly true at high rates of outliers and sensor noise. Not surprisingly, instead, noise in the normals lowers the performances. In the unrealistic scenario in which every normal is affected by a 20° error at 90% of outliers the translational estimate becomes less accurate than GICP, but it still converges to a reasonable solution.

4.5 Conclusions

We presented a novel optimization function to register point clouds using an ICP based algorithm that takes into account an augmented measurement vector. Statistical comparative experiments on real and synthetic data show that our approach outperforms other state-of-the-art methods both in terms of convergence speed and robustness. As expected, the normals and the tangents of the surfaces showed an improvement in particular in the rotational part of the error, while keeping the translational one similar to the other approaches. A further enhancement could be obtained by finding an additional measurement, related to the translation, to be considered in the minimization of the cost function.

Chapter 5

NICP: Dense Normal Based Point Cloud Registration

"The truth is in the details."
— Stephen King, The Duma Key, 2008

Since the introduction of ICP [26], many variants of increased robustness and accuracy have been proposed [8][12][155]. A common intuition behind these state-of-the-art algorithms is the concept of registering surfaces and not points. At the base of their idea there is the observation that point clouds are sampled representation of continuous surfaces, and when seeking for the transformation one has to minimize the distance between corresponding surface elements rather than between corresponding points.

In accordance with this intuition, we present the Normal Iterative Closest Point algorithm, a variant of ICP that can run on-line on a multi-core CPU. The main novelty of NICP is to use a 6D error metric that takes into account the distance between corresponding points *and* corresponding surface normals, and uses the surface curvature as additional cue for data association. NICP leverages our previous analysis on error functions based on surface properties (see Chapter 4). In that work we provided experimental evidence that considering the geometric structure around the points enlarges the basin of attraction. NICP is reported to compare favorably to other state-of-the-art methods on a large publicly available benchmarking dataset.

A common application of point registration methods is the tracking of the position of a moving sensor while constructing local models of the environment. Performing pairwise registrations between consecutive measurements leads to an unavoidable drift in the estimate due to error accumulation. To reduce this effect, it is common to register the current frame against a reference model that is augmented each time a successful registration is performed. Whereas this reduces the drift, a naive implementation would result in a linear growth of the number of points in the model that renders the execution

slower over time. Common strategies to deal with the increasing number of points consist in aggregating the measurements through dense or sparse voxel representations. These approaches, however, do not easily cope with dynamic aspects since removing dynamic parts of the scene requires expensive ray-casting operations.

In this Chapter, we provide a detailed description of NICP. We aim at providing enough information to the interested readers that want to develop their own implementation of the method. Additionally, we discuss a set of algorithmic enhancements to the original system that reduce the computation time by 50% compared to the full implementation. Being more efficient, this system can be used with multiple sensors [156] or run on low-end computers. Moreover, we present a projection-based cloud merging approach that allows to limit the number of points in the scene as new clouds are added, and that naturally deals with moderate dynamics in the scene. Our merging method is straightforward to implement and directly operates on unorganized point clouds. We validated our approaches on a large standard benchmarking dataset, and on real world scenarios, by performing both quantitative and qualitative experiments. An implementation of our approach is publicly available on the web at `http://www.dis.uniroma1.it/~labrococo/nicp`.

5.1 Normal ICP

Our method is a variant of ICP that deviates from the general scheme presented in Section 3.3. Instead of considering only the Euclidean distance between points, we exploit the local properties of the surface, characterized by the surface normals and the local curvature, both in the search for correspondences, and in the computation of the best alignment. In addition to that, to reduce the drift occurring when recovering the trajectory of the sensors through pairwise alignment of consecutive measurements, we construct a global model of the scene by integrating new point clouds.

Figure 5.1 illustrates the different stages of our system that are resumed below:

- as previously described in Sec. 3.1.1, our algorithm computes a Cartesian representation of the 3D point cloud from the raw input data, shall it be a 3D scan, a depth image or a combination of them;
- subsequently, our method adds to each point of the cloud the properties of the surrounding surface, namely normal and curvature (see Section 5.1.1);
- as in a traditional ICP our approach iterates the following steps to refine an initial transform $\mathbf{T}$:
  - search for correspondences performed using a time efficient line of sight criterion (Section 5.1.2).

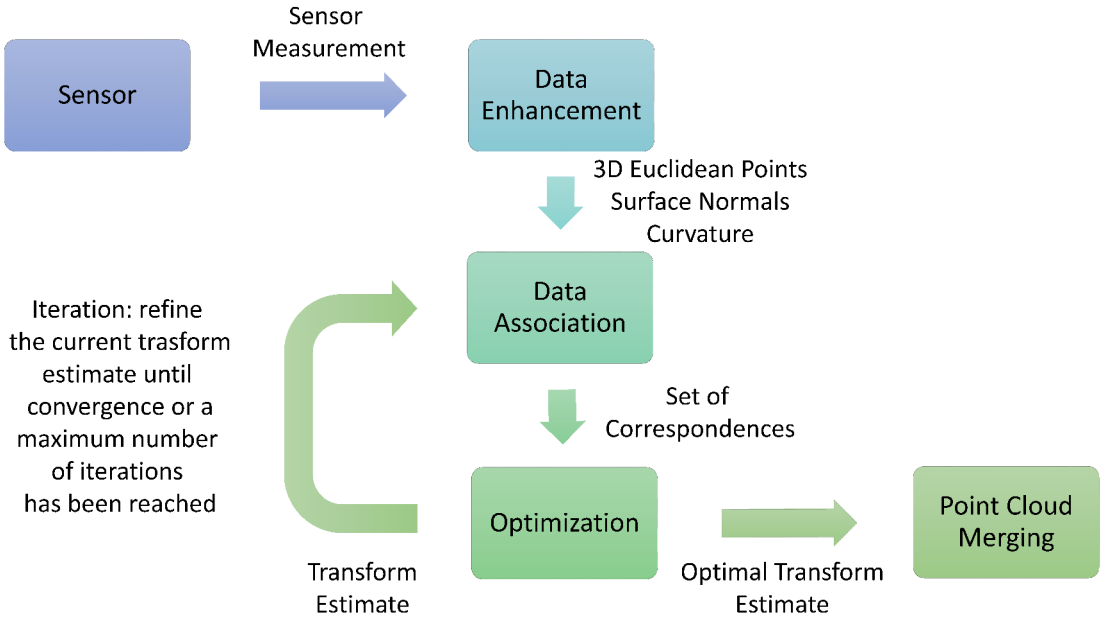


Figure 5.1: Data flow of our approach NICP.

  - computation of the transformation that minimizes the difference between the corresponding points and their normals (Section 5.1.3). This differs from previous variants of ICP that minimize the distance between the euclidean positions of corresponding points. With respect to the common point-to-plane metric, our least-squares formulation allows to solve two additional degrees of freedom in the orientation of the surfaces, thus enlarging the basin of attraction;

- once convergence is reached, or the iterations terminate, our method merges the current cloud into an existing model: the scene (see Section 5.1.4). In this phase, elements belonging to dynamic objects or inconsistencies are eliminated, and nearby points are merged to keep the size of the cloud tractable.

5.1.1 Computing Local Surface Statistics and Normals

The very first step of our method is to apply the unprojection function π^{-1} of Eq. 3.11 to the raw measurements to compute a Cartesian representation of the point cloud. This is done once each time a new measurement becomes available. In case one uses multiple sensors this procedure is performed individually for each of them, producing one Cartesian cloud for each sensor. A point measurement obtained by a range sensor is a sample of a piece-wise continuous surface. This is the core idea of the point-to-plane and plane-to-plane metrics used by Chen *et al.* [146] and Segal *et al.* [12].

We locally characterize the surface around a point $\mathbf{p}_i$ with its surface normal $\mathbf{n}_i$ and curvature σ_i. In order to compute the normal, we extract the covariance matrix of the Gaussian distribution $\mathcal{N}_i^{\mathrm{s}}(\mu_i^{\mathrm{s}},\ \mathbf{\Sigma}_i^{\mathrm{s}})$ of all the points that lie in a sphere of radius R centered in the query point $\mathbf{p}_i$. In our evaluation, we found experimentally that 10 cm is a good value for R in indoor environments. If the surface is well defined, meaning that it is reasonably flat, it can be approximated by a plane, and only one eigenvalue of the covariance will be substantially smaller than the other two. The surface normal is selected as the eigenvector associated to the smallest eigenvalue and, if necessary, reoriented towards the observer point of view.

Similar to GICP, for each point $\mathbf{p}_i$ we compute the mean μ_i^s and the covariance $\mathbf{\Sigma}_i^s$ of the Gaussian distribution by using Eq. 3.35. To determine the set $\mathcal{V}_i$ of points inside the sphere, a standard implementation of the above algorithm would require expensive queries on a KD-tree data structure where the cloud is stored. To speed up the calculation, we adopt an approach based on integral images, described in [6], that allows us to compute Eq. 3.35 in constant time (see Appendix D). Once the parameters of the Gaussian are evaluated, we get its eigenvalue decomposition as follows

$$\mathbf{\Sigma}_i^{\mathrm{s}} = \mathbf{R} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \mathbf{R}^T \tag{5.1}$$

In the previous equation $\lambda_{1:3}$ are the eigenvalues of $\mathbf{\Sigma}_i^{\mathrm{s}}$ in ascending order, and $\mathbf{R}$ is the matrix of eigenvectors that represent the axes of the ellipsoid that approximates the point distribution. We use the curvature $\sigma_i = \lambda_1 / (\lambda_1 + \lambda_2 + \lambda_3) \in [0, 1]$ to discriminate how well the surface is fitted by a plane (see [157] for more details). The smaller it is σ, the flatter is the surface around the point.

In the real case, due to the sensor noise, even surfaces that are perfectly planar will not have an exact 0 curvature (or in other words the smallest eigenvalue null). To reduce the effect of this noise, when needed, we modify the covariance matrix $\mathbf{\Sigma}_i^{\mathrm{s}}$ by forcing a "disc" shape. This can be achieved by modifying the length of the axis of the ellipsoid in the following way

$$\mathbf{\Sigma}_i^{\mathrm{s}} \leftarrow \mathbf{R} \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{R}^T \tag{5.2}$$

where ϵ is a small coefficient, in our experiments ϵ is equal to 0.001. If the surface is not well approximated by a local plane (the curvature is high), we do not modify $\mathbf{\Sigma}_i^{\mathrm{s}}$. This technique has been first introduced by GICP.

After this step, each point $\mathbf{p}_i$ belonging to the cloud is augmented with its own surface characteristics $\langle \mu_i^{\mathrm{s}},\ \mathbf{\Sigma}_i^{\mathrm{s}},\ \sigma_i \rangle$. Figure 5.2 (left image) shows an example of its typical outcome.

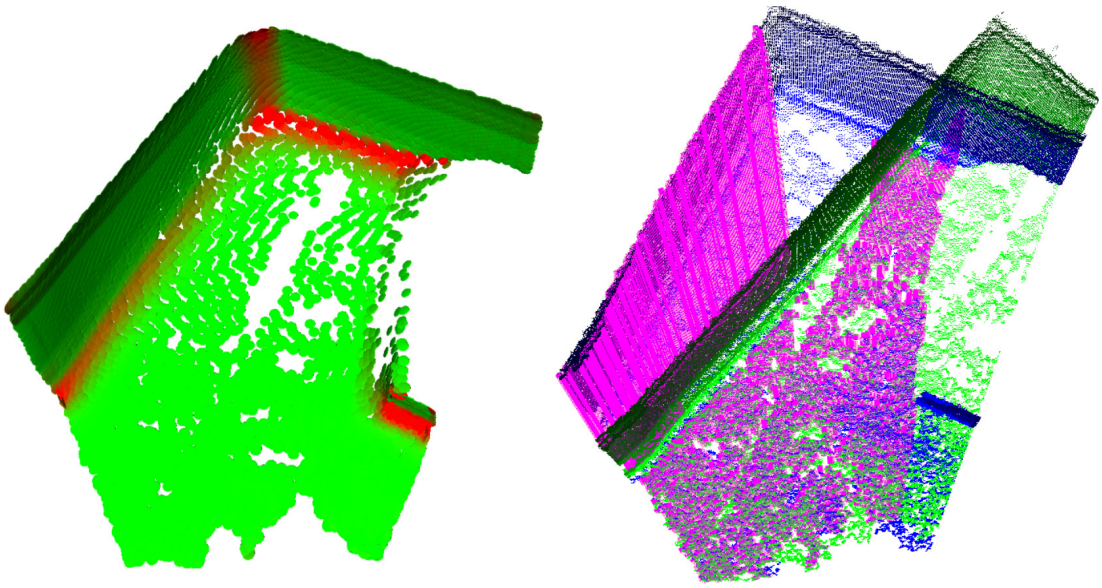


Figure 5.2: Left: example of the effect of extracting the surface statistics. According to their curvature σ, green ellipsoids correspond to points lying on flat regions while red ones to corners. Right: example of the output generated after evaluating the data association. Correspondences are shown as violet lines connecting matching points in the blue and green clouds.

In case one uses multiple sensors, our approach adds the surface normals and curvatures of the points individually for each point cloud. This operation is carried out in the reference frame of the sensor. Let $\mathcal{P}^{\mathrm{c}}_{1:k}$ be these clouds. By knowing the pose of each sensor $\mathbf{K}_{1:k}$ with respect to *the reference frame of the robot* (or more in general a common reference frame), our method computes a global cloud $\mathcal{P}^{\mathrm{c}}$ in that frame as the union of the points in $\mathcal{P}^{\mathrm{c}}_{1:k}$, after applying the corresponding transform

$$\mathcal{P}^{\mathrm{c}} = \bigcup_{\mathbf{K}_i \in \mathcal{S}} \mathbf{K}_i \oplus \mathcal{P}^{\mathrm{c}}_i \tag{5.3}$$

with $\mathcal{S}$ being the set of sensors used.

5.1.2 Line of Sight Correspondence Finding

Similar to [8] and [13] we search for the correspondences by using a line-of-sight criterion. More in detail, we project the reference cloud on a depth image whose viewpoint is the current estimate of the transformation, and points that fall in the same pixel coordinates that have compatible normals and curvature are labeled as a correspondence. Note that this projection operation does not need to be performed for the current cloud, since we have already the corresponding depth image. While the approach by

itself is straightforward, it is necessary to design an efficient implementation due to the potentially large amount of data the algorithm has to manipulate. We describe now a procedure that reduces the memory movements and brings the clouds manipulation to the minimum. The first assumption we make is that the point clouds are stored in arrays not necessarily ordered in any way. To describe our procedure we first introduce the concept of *index image*. Given a projection model $\pi(\mathbb{R}^3) \rightarrow \mathbb{R}^3$ defined as in Sec. 3.1.1, an index image $\mathcal{I}$ is matrix where the element of coordinates $(u,\ v)$ contains the *index* of the point $\mathbf{p}_i$ in the array such that $\pi(\mathbf{p}_i) \rightarrow (u,\ v,\ d)^T$. If more than one point falls in the same pixel, we store the index of the point closest to the observer and having a normal oriented towards the center of projection (the point of view). This operation is simply implemented using a depth buffer.

Now, let $\mathcal{I}(\pi,\ \mathcal{P})$ be an index image computed from the cloud $\mathcal{P}$ using the projection function π. At the beginning of the registration process we compute the current index image by projecting all points of the current cloud $\mathcal{P}^{\mathrm{c}}$

$$\mathcal{I}^{\mathrm{c}} = \mathcal{I}(\pi,\ \mathcal{P}^{\mathrm{c}}) \tag{5.4}$$

Since our optimization procedure keeps fixed the current cloud, $\mathcal{I}^{\mathrm{c}}$ does not move during the entire alignment, thus we do not need to recompute it at every iteration.

Conversely, we need to calculate the index image $\mathcal{I}^{\mathrm{r}}$ of the reference cloud $\mathcal{P}^{\mathrm{r}}$ at each iteration. This is necessary since, at each iteration, we have to re-map the reference cloud in the frame of the current cloud using the most recent transform $\mathbf{T}$

$$\mathcal{I}^{\mathrm{r}} = \mathcal{I}(\pi,\ \mathbf{T} \oplus \mathcal{P}^{\mathrm{r}}) \tag{5.5}$$

Here $\oplus$ applies the transformation $\mathbf{T}$ to the whole cloud $\mathcal{P}^{\mathrm{r}}$.

The reference cloud $\mathcal{P}^{\mathrm{r}}$ can be large, thus this reprojection is an expensive step. An easy optimization consists in clipping $\mathcal{P}^{\mathrm{r}}$ around the current location once before starting the iterations.

Let $\mathcal{I}_{u,v}$ be the value of the pixel of coordinates (u, v) in the index image $\mathcal{I}$. At this point, from $\mathcal{I}^{\mathrm{r}}$ and $\mathcal{I}^{\mathrm{c}}$ we generate a candidate correspondence for each pixel coordinates (u, v) as $\langle i, j \rangle_{u,v} = \left\langle \mathcal{I}^{\mathrm{c}}_{u,v}, \mathcal{I}^{\mathrm{r}}_{u,v} \right\rangle$. A candidate correspondence $\langle i, j \rangle$ between the points $\mathbf{p}^{\mathrm{c}}_i$ and $\mathbf{p}^{\mathrm{r}}_j$ is rejected if one of the following constraints is verified:

- either $\mathbf{p}^{\mathrm{c}}_i$ or $\mathbf{p}^{\mathrm{r}}_j$ do not have a well defined normal;
- the distance between the point in the current cloud and the reprojected point in the reference cloud is greater than a threshold

$$\|\mathbf{T} \oplus \mathbf{p}^{\mathrm{r}}_j - \mathbf{p}^{\mathrm{c}}_i\| > \epsilon_d \tag{5.6}$$

- the magnitude of the log ratio of the curvatures of the points is larger than a

threshold

$$|\log \sigma_i^{\mathrm{c}} - \log \sigma_j^{\mathrm{r}}| > \epsilon_\sigma \tag{5.7}$$

- the angle between the normal of the current point and the reprojected normal of the reference point is greater than a threshold

$$\mathbf{n}_i^{\mathrm{c}} \cdot (\mathbf{T} \oplus \mathbf{n}_j^{\mathrm{r}}) < \epsilon_n \tag{5.8}$$

In our evaluation, we found experimentally that good values for ϵ_d, ϵ_n and ϵ_σ are respectively 0.5 m, 0.95 and 1.3. Figure 5.2 (right image) illustrates an example of the output of the correspondence selection. By using index images, we avoid copying points, normals and covariance matrices in temporary structures, thus increasing the whole speed of the algorithm.

In case of multiple sensors this procedure is repeated independently for each of them, producing two index images for each sensor $\mathcal{I}_k^{\mathrm{c}}$ and $\mathcal{I}_k^{\mathrm{r}}$, by considering the relative pose of the sensor on the mobile base. This leads to the following formulation for equations 5.4 and 5.5

$$\mathcal{I}_k^{\mathrm{c}} = \mathcal{I}(\pi,\ \mathbf{K}_k^{-1} \oplus \mathcal{P}^{\mathrm{c}}) \qquad \mathcal{I}_k^{\mathrm{r}} = \mathcal{I}(\pi,\ \mathbf{T} \oplus \mathbf{K}_k^{-1} \oplus \mathcal{P}^{\mathrm{r}}) \tag{5.9}$$

The remaining operations remain the same.

5.1.3 Computing the Relative Transform

Once we have evaluated a set of correspondence pairs $\mathcal{C} = \langle i, j \rangle_{1:M}$, we compute the relative transformation between the two frames by using an iterative least-squares formulation. Recall that each i^{th} point in a cloud contains the following information: the Cartesian coordinates $\mathbf{p}_i$, the surface curvature σ_i, the surface normal $\mathbf{n}_i$ and the covariance matrix $\mathbf{\Sigma}_i^{\mathrm{s}}$.

Let $\tilde{\mathbf{p}}$ be a point with normal $\tilde{\mathbf{p}} = \begin{pmatrix}\mathbf{p}^T & \mathbf{n}^T\end{pmatrix}^T$ and $\mathbf{T}$ a transformation matrix parameterized by using rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$. The $\oplus$ operator on points with normals can be defined as

$$\tilde{\mathbf{p}}' = \mathbf{T} \oplus \tilde{\mathbf{p}} = \begin{pmatrix}\mathbf{R}\mathbf{p} + \mathbf{t} \\ \mathbf{R}\mathbf{n}\end{pmatrix} \tag{5.10}$$

This means that, given a correspondence pair and the current transform $\mathbf{T}$, the error $\mathbf{e}_{ij}(\mathbf{T})$ is a 6D dimensional vector

$$\mathbf{e}_{ij}\left(\mathbf{T}\right) = \mathbf{T} \oplus \tilde{\mathbf{p}}_j^{\mathrm{r}} - \tilde{\mathbf{p}}_i^{\mathrm{c}} \tag{5.11}$$

Substituting Eq. 5.11 in Eq. 3.33 leads to the following objective function

$$\sum_{\mathcal{C}} \mathbf{e}_{ij}\left(\mathbf{T}\right)^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{e}_{ij}\left(\mathbf{T}\right) \tag{5.12}$$

Here $\tilde{\mathbf{\Omega}}_{ij}$ is a 6×6 information matrix whose goal is to scale the different components of the error. The ideal behavior we want to obtain from that matrix is to rotate corresponding points so that their normals align, while at the same time, the distance along the normal direction is penalized. In addition to this, it must also neglect the distance along the plane tangents. With this in mind, we impose the translational and normal components to be independent between each other, and we select $\tilde{\mathbf{\Omega}}_{ij}$ as follows

$$\tilde{\mathbf{\Omega}}_{ij} = \begin{pmatrix} \mathbf{\Omega}_i^{\mathrm{s}} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega}_i^{\mathrm{n}} \end{pmatrix} \tag{5.13}$$

Here $\mathbf{\Omega}_i^{\mathrm{s}} = \mathbf{\Sigma}_i^{\mathrm{s}-1}$ is the surface information matrix around the current point $\mathbf{p}_i^{\mathrm{c}}$, and $\mathbf{\Omega}_i^{\mathrm{n}}$ is the information matrix of the normal. If the curvature is small enough we force $\mathbf{\Omega}_i^{\mathrm{n}}$ to have a disk shape as follows

$$\mathbf{\Omega}_i^{\mathrm{n}} \leftarrow \mathbf{R} \begin{pmatrix} \frac{1}{\epsilon} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{R}^T \tag{5.14}$$

otherwise we impose $\mathbf{\Omega}_i^{\mathrm{n}}$ to the identity. By generating such information matrices, a correspondence pair is minimized by allowing the points to slide onto each other along the tangential direction of the surface, and rotating them so that their normals align. The reader might notice that setting $\mathbf{\Omega}_i^{\mathrm{n}}$ to zero makes our objective function identical to point-to-plane GICP. To reduce the effect of outliers we further robustify the error function by clamping the norm of the χ_{ij}^2 of each point to a maximum value as described in Eq. 4.11.

Our method minimizes Eq. 5.12 by using a local parameterization of the perturbation in the following form: $\mathbf{\Delta T} = (\Delta t_x,\ \Delta t_y,\ \Delta t_z,\ \Delta q_x,\ \Delta q_y,\ \Delta q_z)^T$. It is composed by the translation vector $\mathbf{\Delta t} = (\Delta t_x,\ \Delta t_y,\ \Delta t_z)^T$ and the imaginary part of the normalized quaternion $\mathbf{\Delta q} = (\Delta q_x,\ \Delta q_y,\ \Delta q_z)^T$. In order to smooth the convergence of the whole system we use a damped Gauss-Newton algorithm. This prevents the solution to take too large steps that might be caused by non-linearities or wrong correspondences. More formally, at each iteration our approach solves the following linear system

$$(\mathbf{H} + \lambda \mathbf{I})\mathbf{\Delta T} = -\mathbf{b} \tag{5.15}$$

Here, $\mathbf{H} = \sum \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{J}_{ij}$ is the approximated Hessian and $\mathbf{b} = \sum \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{e}_{ij}(\mathbf{T})$ is the vector of residuals. Once we computed the perturbation $\mathbf{\Delta T}$ from Eq. 5.15, we refine

the current transformation as

$$\mathbf{T} \leftarrow \boldsymbol{\Delta}\mathbf{T} \oplus \mathbf{T} \tag{5.16}$$

The Jacobian $\mathbf{J}_{ij}$ is calculated by evaluating the derivative of Eq. 5.11 in $\boldsymbol{\Delta}\mathbf{T} = \mathbf{0}$

$$\mathbf{J}_{ij} = \left.\frac{\partial \mathbf{e}_{ij}\left(\boldsymbol{\Delta}\mathbf{T} \oplus \mathbf{T}\right)}{\partial \boldsymbol{\Delta}\mathbf{T}}\right|_{\boldsymbol{\Delta}\mathbf{T}=\mathbf{0}} = \begin{pmatrix} \mathbf{I} & -2[\mathbf{T} \oplus \mathbf{p}_j^{\mathrm{r}}]_\times \\ \mathbf{0} & -2[\mathbf{T} \oplus \mathbf{n}_j^{\mathrm{r}}]_\times \end{pmatrix} \tag{5.17}$$

where $[\mathbf{v}]_\times$ is the cross product matrix of the vector $\mathbf{v}$ (see Appendix B and Appendix C). Note that, it is possible to construct efficiently the linear system in Eq. 5.15 by exploiting the block structure of the Jacobian, and its substantial sparsity.

In case of multiple sensors, we carry out a single optimization step based on all correspondences found in all sensor frames. The aim of this optimization step is to compute the transform of the robot origin that minimizes all correspondences. Since all points in the clouds are expressed in a common frame, we do not need to take care of the sensor transformations $\mathbf{K}_{1:k}$ that have been already embedded in the points.

5.1.4 Point Cloud Merging

To gather a local map by tracking the pose of a robot, it is common to perform several pairwise alignments. Each registration introduces a small error and the map can rapidly become inconsistent. To limit this drift, performing recursive alignments on the same model has been reported to be very effective. After registering each new measurement, a local map is augmented with the new aligned data. This is at the base of KinFu, and many successful 2D mapping approaches. The model generated in this way is typically locally consistent.

A naive implementation of this strategy would result in constant growth of the points in the reference model, and since the performance of the algorithm depends directly on the number of points, we might expect a linear increase in per-frame computation as the time passes. Note that, many of these points are samples of the surface very close in space. To keep the process tractable, and enhance the quality of the model, it is common practice to apply some decimation or aggregation technique. Furthermore, each point is a measurement that comes with its own error that depends by the sensor. In case of depth cameras typically this error grows with the distance measured. Based on the above considerations, while doing this decimation, it is frequent to refine the point statistics.

In our current implementation we keep for each point the coefficients of a 1D normal distribution that describes the isotropic uncertainty of the measurements in the space. In principle, one could represent the full 3D distribution of a point noise, however, our experiments have shown that an isotropic noise provides the best trade off between computation time and accuracy.

The aim of a merging procedure is to fuse the points of two clouds to get a unique

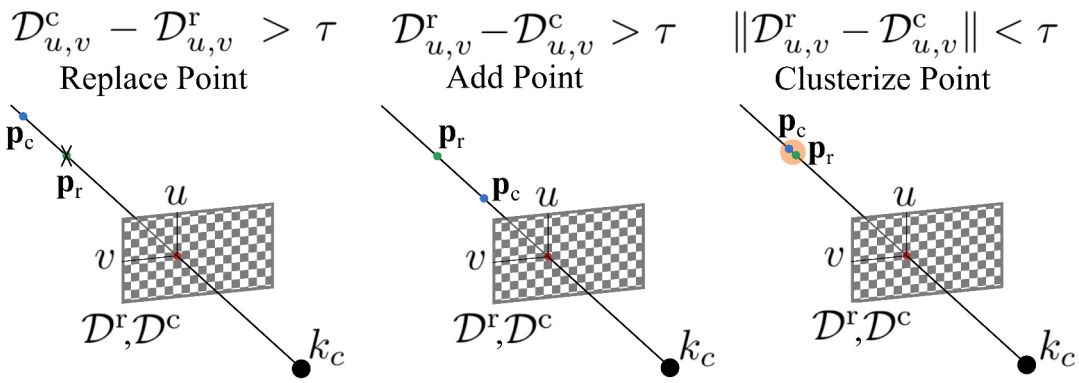


Figure 5.3: Graphical example showing the three possible cases considered by our method during the clustering step of the merging process. In the image, k_c represents the origin of the current point cloud.

consistent model, and at the same time to refine the statistics of the points. The new model should have a regular density of measurements. A merging procedure therefore consists of two components:

- *clustering*, where we partition the input clouds in sets of points that will contribute to a single one in the output cloud;
- *update*, where all statistics of the new point are fused to a new one.

Optionally, if the scene is dynamic we might want also to *remove* points from the model.

In this section we present a projection based approach for clustering that leverages the index-image data structure presented before. This model allows us to efficiently group the points into sets and it handles occlusions. We omit the case of multiple sensor as it is a straightforward extension of the single sensor case.

We assume to have two aligned clouds: $\mathcal{P}^{\mathrm{r}}$ and $\mathcal{P}^{\mathrm{c}}$, and to know the relative transform $\mathbf{T}$ between them. We can then compute a view of the reference as if it was observed from the origin of $\mathcal{P}^{\mathrm{c}}$. This is done already during the computation of the correspondences through Eq. 5.5. In addition to the index image $\mathcal{I}^{\mathrm{r}}$, we also compute the range images $\mathcal{D}^{\mathrm{r}}$ and $\mathcal{D}^{\mathrm{c}}$ for both $\mathcal{P}^{\mathrm{r}}$ and $\mathcal{P}^{\mathrm{c}}$. This operation naturally handles occlusions. Let $\mathcal{D}_{u,v}$ be the value of the pixel of coordinates (u, v) in the depth image $\mathcal{D}$. By selecting a distance threshold parameter τ, we scan the two depth images pixel by pixel and based on the depth comparison we perform the following operations:

- if $\mathcal{D}^{\mathrm{c}}_{u,v} - \mathcal{D}^{\mathrm{r}}_{u,v} > \tau$, the new beam crosses an existing element of the reference surface. In absence of noise, this results in *replacing* the point $\mathcal{D}^{\mathrm{r}}_{u,v}$ with the point $\mathcal{D}^{\mathrm{c}}_{u,v}$. In other words, if we see through a point in the reference cloud we replace it with the corresponding transformed point in the current cloud;

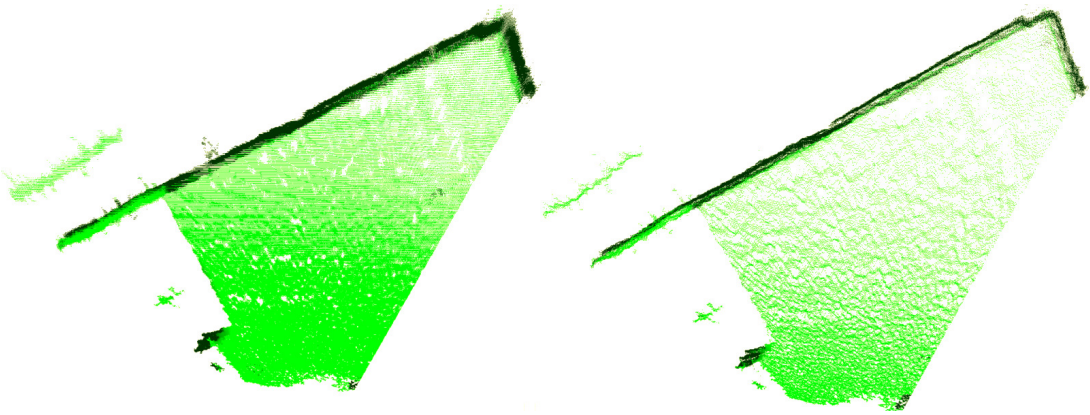


Figure 5.4: Example of merging. Note the difference in the thickness of the walls due to sensor uncertainty before the merging (left), and after (right).

- if $\mathcal{D}^{\mathrm{r}}_{u,v} - \mathcal{D}^{\mathrm{c}}_{u,v} > \tau$, the new beam ends much before a point in the previous cloud. This results in the new point to be added to the reference model as it might be due to a new object entering in the scene;

- if $\|\mathcal{D}^{\mathrm{r}}_{u,v} - \mathcal{D}^{\mathrm{c}}_{u,v}\| < \tau$, the two points are close, so they are likely to belong to the same surface, and they end up in the same cluster.

Figure 5.3 illustrates these three cases.

Please note that, compared to traditional clustering approaches that rely on voxelization, this method makes use of the free space and naturally handles the removal of old points in the scene. In the worst of the cases we have a number of clusters equal to the number of pixels in the image.

Once we are done with the clustering, we update the noise statistics of each point and we adjust its estimate by using an information filter as follows

$$\mathbf{\Omega}_{u,v} = \sum_{i=1}^{N} \mathbf{\Omega}^{\mathrm{m}}_i \qquad \mu_{u,v} = \mathbf{\Omega}^{-1}_{u,v} \sum_{i=1}^{N} \mathbf{\Omega}^{\mathrm{m}}_i \cdot \mathbf{p}_i \tag{5.18}$$

where $\mathbf{\Omega}^{\mathrm{m}}_i$ is the inverse of the covariance matrix of the i-th point $\mathbf{p}_i$ falling in the cluster (u, v). We replace then the estimate of each point in the reference cloud with the mean of the cluster.

After updating the point positions, we need to recompute the normals by applying the procedure described in Section 5.1.1. Note that only the normals within the area of the cloud where points have been inserted or modified are recomputed. Fig. 5.4 highlights how the merging helps to remove artifacts (thick wall) generated by a naive accumulation of the point clouds.

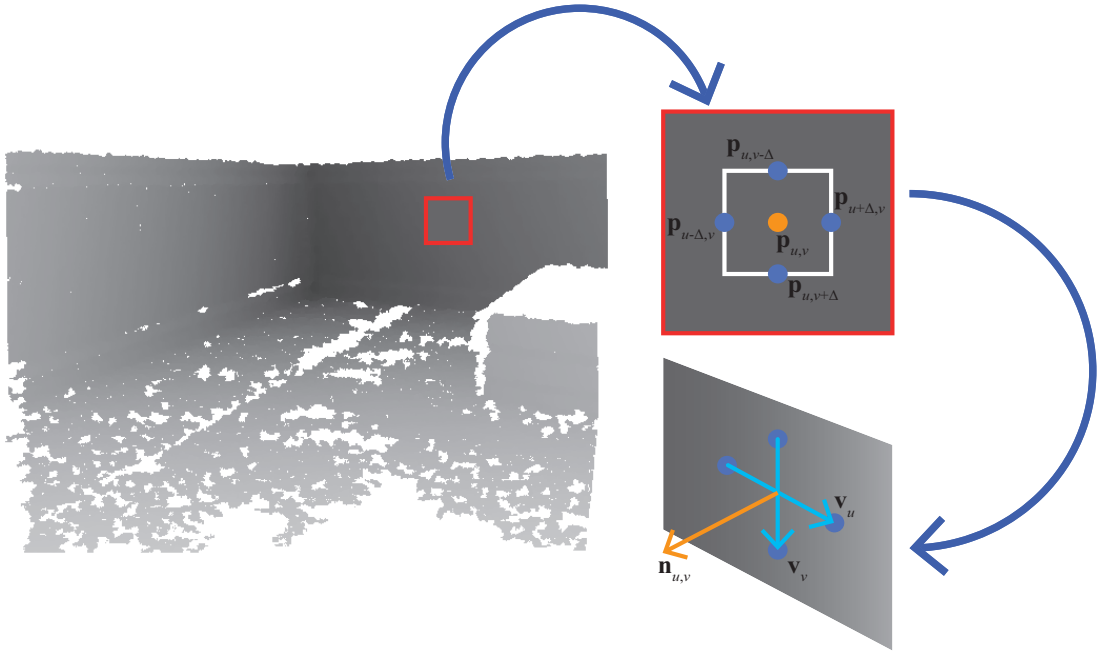


Figure 5.5: Graphical example of cross product surface normal computation. The red square on the top right is a magnification of a part of the range image shown on the left. We select a square region on the depth image, around the query point $\mathbf{p}_{u,v}$, whose neighbor points $\mathbf{p}_{u+\Delta,v}$, $\mathbf{p}_{u-\Delta,v}$, $\mathbf{p}_{u,v+\Delta}$ and $\mathbf{p}_{u,v-\Delta}$ are used to compute the normal through the cross product $\mathbf{v}_u \times \mathbf{v}_v$ (right bottom image).

5.2 Speeded Up Normal ICP

Thanks to its parallel implementation, NICP is able to execute in real-time. However, this can not be ensured when NICP is used in conjunction with low-end computers or with multiple sensors. Albeit a GPU implementation is relatively straightforward, we focused on improving the algorithm and postponed the GPU version to a later stage.

In the remainder of this section we highlight the bottlenecks of the procedure described before, and how we addressed them to further enhance the performance.

5.2.1 Computation of the Normals and the Point Statistics

The computation of the surface statistics (Section 5.1.1), requires the solution of a third degree polynomial for each pixel to calculate the eigenvalue decomposition of the covariance.

A faster but less accurate alternative way, is to compute the normals directly in the depth image, as the normalized cross product of points projecting onto adjacent pixels (see Figure 5.5). Namely, given the point $\mathbf{p}_{u,v}$ that projects in the pixel (u, v),

we compute its normal direction as follows

$$\mathbf{n}'_{u,v} = (\mathbf{p}_{u+\Delta,v} - \mathbf{p}_{u-\Delta,v}) \times (\mathbf{p}_{u,v+\Delta} - \mathbf{p}_{u,v-\Delta}) \qquad \mathbf{n}_{u,v} = \frac{\mathbf{n}'_{u,v}}{||\mathbf{n}'_{u,v}||} \tag{5.19}$$

Here, Δ is an offset on the image of a few pixels. This procedure is substantially faster, however leads to poor quality normals, that improve as Δ increases. Typical values of Δ range from 2 to 5 pixels. To further smooth the normal we apply a block filtering to the normals.

5.2.2 Computing the Relative Transform

During the alignment the reference cloud is "moved" at each iteration. Since the number of points in the reference augments over time, the computation increases accordingly. An alternative is to perform the optimization by computing the transform $\mathbf{T}$ that maps the current cloud onto the reference as follows

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} \left(\mathbf{T} \oplus \mathbf{p}_i^{\mathrm{c}} - \mathbf{p}_j^{\mathrm{r}}\right)^T \mathbf{\Omega}_{ij} \left(\mathbf{T} \oplus \mathbf{p}_i^{\mathrm{c}} - \mathbf{p}_j^{\mathrm{r}}\right) \tag{5.20}$$

This approach, combined with our projection based data association, has the major drawback of "fixing" the reference projection. If the offset is large, the current cloud might be dragged out of the field of view of the reference during the optimization. To cope with this problem, we use a projection function in the correspondence search that has a broader field of view with respect to the original one. In this way, we capture in the reference projection a larger portion of the scene, and during the optimization the current projection is more likely to stay in the frustum used to render the reference. With this slightly different objective function to be minimized, the Jacobian $\mathbf{J}_{ij}$ becomes

$$\mathbf{J}_{ij} = \left.\frac{\partial \mathbf{e}_{ij}\left(\mathbf{\Delta T} \oplus \mathbf{T}\right)}{\partial \mathbf{\Delta T}}\right|_{\mathbf{\Delta T}=\mathbf{0}} = \begin{pmatrix} \mathbf{I} & -2[\mathbf{T} \oplus \mathbf{p}_j^{\mathrm{c}}]_\times \\ \mathbf{0} & -2[\mathbf{T} \oplus \mathbf{n}_j^{\mathrm{c}}]_\times \end{pmatrix} \tag{5.21}$$

Additionally, our time enhanced version performs a set of pyramidal alignments at increasing resolutions rather than operating at a fixed one. This has the benefit of enlarging the convergence of attraction for data association as the pixels become bigger, and to speed up the computation. Accuracy is preserved when performing the alignment at higher resolutions. The reader might notice that this can be done by simply modifying the parameters of the projection function π.

5.2.3 Point Cloud Merging

The update strategy used in the merging procedure requires to keep an information filter per point, and then to recompute the statistics in the regions observed.

To speed up the computation, for each point in a cloud we keep a mono dimensional information $\omega_{u,v}$ representing the uncertainty of both normal and point. The update of the point and its normal is carried out in an information-like form as follows

$$\omega_{u,v} = \sum_{i=1}^{N} \mathbf{\Omega}_i^{\mathrm{m}} \qquad \mu_{u,v}^{\mathbf{p}} = \omega_{u,v}^{-1} \sum_{i=1}^{N} \omega_i \cdot \mathbf{p}_i \qquad \mu_{u,v}^{\mathbf{n}} = \omega_{u,v}^{-1} \sum_{i=1}^{N} \omega_i \cdot \mathbf{n}_i \tag{5.22}$$

The normals are scaled to unit norm after computing the means.

5.2.4 Optimizing Memory Access

From a hardware point of view, the main bottleneck of the procedure lies in the scattered memory access induced by the double indirection through index images. We observed a 20% increase in performance when reordering the points before each alignment. Our point clouds are represented as unordered arrays. During all procedures, the access to the arrays is either linear or follows the order of the pixels in the index image. An alignment operation is likely to access the pixels in these two orders. To guarantee a more "regular" access, before each alignment we sort the points in the reference cloud so that the first block of the array are the points in the index image used for computing the correspondences, in the order set by the index image.

5.3 Experiments

In Chapter 4 we performed a convergence study of the error function of Normal ICP. In this section we aim at comparing both a pairwise and a recursive implementation of NICP that integrates the merging procedure described before. For the remainder of this work, we will refer to the speeded up variant of NICP introduced in Sec. 5.2 with the name of SNICP. We used 3D laser and depth camera based benchmarking datasets. In addition to this, we also show qualitative results of our tracking algorithm working both in a environment with dynamic objects, and with multiple sensors.

We performed an extensive evaluation of NICP and SNICP against the state-of-the-art methods: point-to-plane GICP, DVO, NDT and KinFu. We used the ROS [158] version of DVO and NDT as suggested by the authors and available on the web, while for KinFu we used the implementation provided by the Point Cloud Library (PCL) [159]. Since point-to-plane GICP is a special case of our algorithm, where the error in the normal part of the optimization stage is neglected, and the correspondences are selected based only on point distance, we used our own GICP implementation. Note that our implementation of GICP benefits of all the data structures and the surrounding algorithms that are used in NICP, namely the extraction of the statistics and the calculation of the correspondences. For NDT and KinFu we used the default parameters

found in their implementations. In the case of GICP, SNICP and NICP, instead, we used the values indicated in the previous sections and we forced them to run all the iterations independently by the χ^2 value obtained. In particular, the number of iterations has been set to 10 for both GICP and NICP, while SNICP was configured to perform 3 iterations at three different levels of dimension of the input depth images (1/4, 1/2 and full size). An aggregated list of parameters used in the experiments for NICP can be found on the website linked before.

To measure the performance of an algorithm we used the benchmarking tools of Sturm *et al.* [154], and we computed the Relative Pose Error (RPE). The RPE measures the pairwise alignment error between successive poses. More formally, given the groundtruth transform between two consecutive point clouds $\mathbf{T}_{gt}$, and the transform $\mathbf{T}$ calculated by one of the algorithms considered in the comparison, the RPE computes the translational and rotational difference of the offset transform $\mathbf{\Delta T}_o = \mathbf{T}_{gt}^{-1}\mathbf{T}$. In particular, the translational error t_e is computed as the module of the translation vector $\mathbf{t}_o$ of $\mathbf{\Delta T}_o$. The rotational error R_e, instead, is taken as the rotation angle of the rotation matrix $\mathbf{R}_o$ associated to $\mathbf{\Delta T}_o$. In the ideal case when the transform $\mathbf{T}$ is exactly equal to $\mathbf{T}_{gt}$, the transformation offset $\mathbf{T}_o$ would be the identity matrix, and thus both the translational and rotational errors equal to zero. All datasets have been processed on a i7-3630QM running at 2.4 GHz and with a nVidia GeForce GT 650M graphics card. All the parameters used in the evaluation can be found on the previously linked NICP website.

The depth camera data used in these experiments is the ETH Kinect benchmarking dataset developed by Pomerleau *et al.* in [160]. Each test consists in a sequence of depth and RGB images acquired with a RGB-D camera. For benchmarking purposes, the ground-truth is available. The dataset covers 3 environments of increasing complexity (low, high, medium), with 3 types of motions (rotational, translational, fly) at 3 different speeds (slow, medium, fast). Using tests where the motion of the camera is high allows to evaluate the robustness of the algorithms to poor initial guesses. Indeed a big camera velocity implies an increasing average distance between two processed frames. Note that, since this dataset is recorded with a high frame rate, the RPE is computed on poses with a difference in time of 1 second.

For testing registrations on 3D laser measurements we used the benchmarking dataset provided by Pomerleau *et al.* in [161]. Like for the case of depth camera data, we evaluated the algorithms computing the RPE error. However, instead of con-

Table 5.1: Mean and standard deviation of the pairwise point cloud registration time for each algorithm over all the sequences contained in the ETH Kinect dataset.

	NDT	**GICP**	**DVO**	**NICP**
Mean	105 ms	12 ms	3 ms	12 ms
Std. Dev.	23 ms	1 ms	1 ms	1 ms

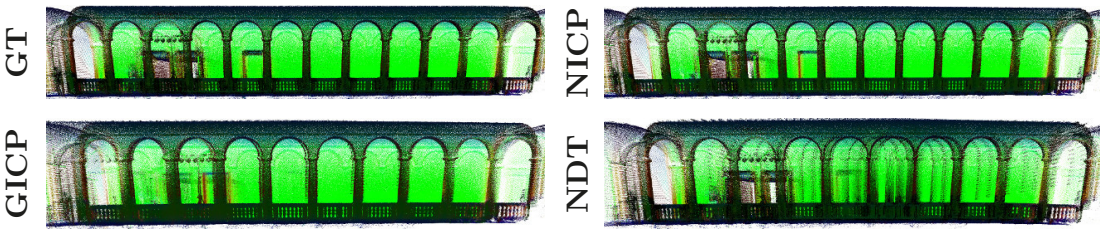


Figure 5.6: Side views of the reconstruction of the ETH Hauptgebaeude dataset. Top-left: ground-truth. Top-right: NICP reconstruction. Bottom-left: GICP reconstruction. Bottom-right: NDT reconstruction. NICP was able to reconstruct consistently the scene while both NDT and GICP generated a shortened corridor (*count the arcs*).

sidering an interval of time of one second between two poses, we computed the error at each frame. This has been done because the frame rate of 3D laser tests are much lower. Also in this case the ground-truth is available.

5.3.1 Pairwise Depth Camera Tracking

In this section we show a comparison of NICP with point-to-plane GICP, DVO and NDT on a depth camera based dataset. Whereas DVO requires the intensity channel, thus it does not belong to the same class of the other algorithms, we included it in the comparison since it is very fast and stable when the lighting conditions are favorable.

As shown in Table 5.1, in terms of processing time, NDT resulted to be the slowest algorithm in computing a single registration. It required in fact about 105 ms. All other approaches, GICP, DVO and NICP, managed to execute in real time computing respectively a registration in about 12 ms, 3 ms and 12 ms. DVO reaches the highest frame rate, since it processes a substantial smaller quantity of data.

Table 5.2 reports the results obtained in processing all the benchmarking sequences of the ETH Kinect dataset with NDT, GICP, DVO and NICP. For each approach and for each test, we processed all the images in sequence and we generated the estimated trajectories. Green cells in the table highlight the best result for each specific test among all the compared algorithms. NICP is almost always more accurate than NDT, GICP and DVO. The effect of using normals in the error function minimization allows to better reduce the rotational error and, as a result, also the translational drift benefits from this. In part of the tests, specifically the fast ones, both the translational and rotational errors are quite large. The cause of these large errors is that the algorithms lost the tracking due to the significant displacements between the clouds.

5.3.2 3D Laser Data Registration

Here we show a comparison between NDT, point-to-plane GICP and NICP on the 3D laser benchmarking dataset provided by Pomerleau *et al.* in [161]. We present the

	Mean / Standard Deviation Translational Error [m]				Mean / Standard Deviation Rational Error [deg°]			
Sequence	**NDT**	**GICP**	**DVO**	**NICP**	**NDT**	**GICP**	**DVO**	**NICP**
high-fast-f	0.452/0.363	**0.284/0.253**	1.150/1.091	0.291/0.259	**17.6/11.0**	20.2/19.2	41.8/34.4	24.1/21.4
high-fast-r	1.142/2.104	0.892/0.864	0.824/0.640	**0.357/0.274**	29.1/42.2	35.4/33.1	24.7/20.0	**14.3/12.6**
high-fast-t	0.132/0.081	0.115/0.070	0.174/0.124	**0.096/0.039**	6.4/2.9	5.9/4.7	5.4/3.1	**5.3/2.6**
high-med-f	0.225/0.178	0.076/0.062	0.249/0.190	**0.069/0.039**	9.8/6.9	**4.8/3.0**	9.4/6.9	5.9/4.1
high-med-r	0.217/0.276	0.123/0.126	0.276/0.234	**0.103/0.108**	10.0/3.4	**8.4/3.3**	8.6/6.0	**8.4/3.3**
high-med-t	0.112/0.081	0.046/0.020	0.147/0.081	**0.043/0.020**	5.6/3.6	**2.5/1.2**	4.6/2.4	**2.5/1.2**
high-slow-f	0.150/0.158	0.055/0.031	0.128/0.061	**0.053/0.029**	4.9/3.1	**3.2/1.8**	4.7/2.1	3.3/1.6
high-slow-r	0.213/0.286	**0.090/0.103**	0.226/0.220	**0.090/0.113**	5.9/3.6	4.0/2.1	6.1/5.4	**3.8/1.9**
high-slow-t	0.064/0.045	**0.038/0.024**	0.150/0.077	0.042/0.033	2.9/1.7	1.7/0.8	4.5/2.4	**1.4/0.7**
low-fast-f	0.704/0.733	**0.282/0.277**	1.204/0.898	0.294/0.265	19.9/37.8	**12.8/12.4**	36.7/30.4	14.6/14.6
low-fast-r	0.614/1.188	0.329/0.522	1.094/0.944	**0.316/0.367**	26.9/41.1	20.8/22.6	29.8/23.0	**15.5/11.7**
low-fast-t	0.175/0.109	0.073/0.031	0.519/0.285	**0.072/0.030**	6.5/3.9	**4.9/2.7**	14.5/8.3	5.0/2.5
low-med-f	0.280/0.257	**0.069/0.041**	0.521/0.371	0.077/0.087	6.9/5.4	**3.8/2.3**	13.9/9.9	4.3/3.8
low-med-r	0.185/0.171	0.075/0.077	0.488/0.285	**0.054/0.037**	8.7/3.7	7.0/2.5	12.7/9.8	**6.7/2.3**
low-med-t	0.094/0.051	**0.063/0.029**	0.346/0.185	**0.063/0.028**	4.9/3.1	3.8/2.4	8.6/5.6	**3.7/2.2**
low-slow-f	0.162/0.146	0.059/0.073	0.407/0.256	**0.052/0.058**	4.0/3.7	2.8/3.4	14.6/9.8	**2.7/2.5**
low-slow-r	0.231/0.303	0.100/0.149	0.466/0.287	**0.089/0.121**	5.9/4.8	4.3/2.8	11.4/8.0	**3.7/2.4**
low-slow-t	0.114/0.113	0.048/0.043	0.418/0.238	**0.039/0.032**	3.0/1.7	2.0/1.1	9.1/5.1	**1.6/0.9**
med-fast-f	0.652/0.459	0.460/0.279	1.282/0.850	**0.343/0.243**	35.0/33.2	33.9/32.0	38.5/28.0	**29.5/27.9**
med-fast-r	0.330/0.512	0.744/0.900	0.614/0.421	**0.285/0.242**	15.6/14.5	27.3/29.9	18.2/12.9	**13.9/8.0**
med-fast-t	0.128/0.060	0.090/0.046	0.217/0.116	**0.086/0.038**	6.3/3.1	5.2/2.6	5.7/2.7	**5.0/2.4**
med-med-f	0.236/0.236	0.070/0.046	0.422/0.222	**0.058/0.039**	11.1/14.1	4.3/2.9	12.7/9.0	**4.1/2.3**
med-med-r	0.160/0.186	0.114/0.110	0.280/0.223	**0.076/0.067**	8.5/3.5	8.6/3.6	**6.2/4.2**	7.7/2.7
med-med-t	0.091/0.053	**0.036/0.016**	0.229/0.147	0.038/0.019	3.9/2.3	**2.0/0.9**	6.0/3.6	**2.0/0.9**
med-slow-f	0.113/0.113	0.050/0.029	0.204/0.135	**0.040/0.023**	3.6/2.6	2.7/1.8	6.1/3.9	**2.6/1.2**
med-slow-r	0.111/0.129	0.060/0.075	0.212/0.174	**0.055/0.064**	4.5/2.4	3.0/1.6	5.2/4.0	**2.4/1.1**
med-slow-t	0.074/0.055	**0.032/0.021**	0.215/0.106	0.034/0.027	3.1/2.1	1.6/0.8	6.2/3.4	**1.4/0.8**

Table 5.2: Pairwise point cloud registration mean and standard deviation of the relative translational and rotational error for all the sequences of the ETH Kinect dataset. Green cells in the table highlight the best mean result for each specific test among all the compared algorithms (NDT, GICP, DVO, and NICP).

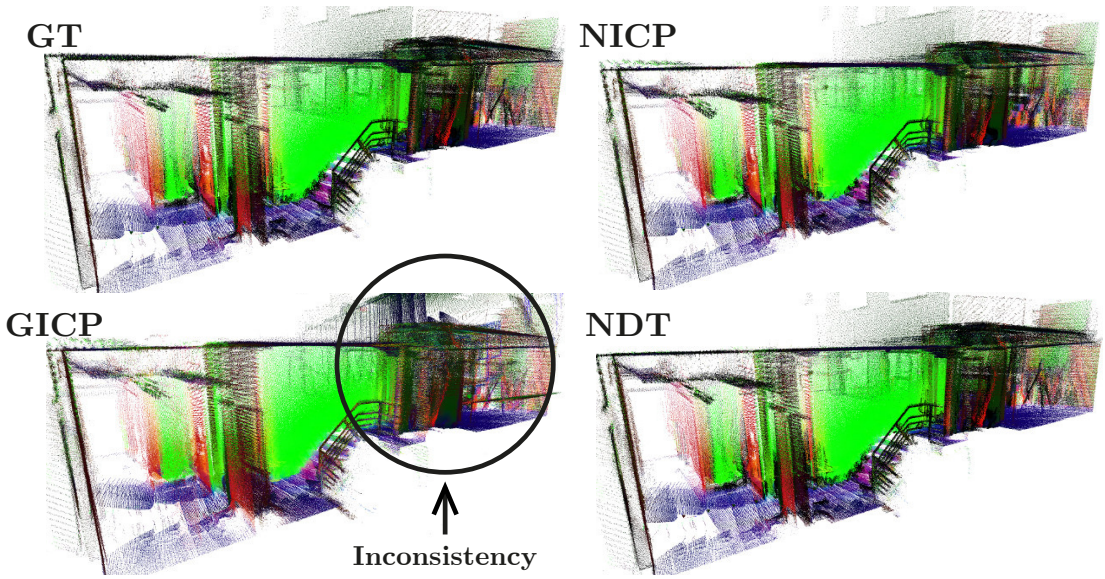


Figure 5.7: Side views of the reconstruction of the ETH Stairs dataset. Top-left: ground-truth. Top-right: NICP reconstruction. Bottom-left: GICP reconstruction. Bottom-right: NDT reconstruction. Both NICP and NDT consistently reconstructed the scene, GICP failed in aligning the part on the back of the stairs.

results on two different sequences:

- **ETH Hauptgebaeude** captures the main building of ETH Zurich and the laser moved basically along a straight direction in a long corridor;
- **ETH Stairs** is a record of the *internal* and *external* part of a building, the laser moved along a corridor and it went outside passing through two doorways. Between the doorways the scanner climbed five steep stairs.

The images in Fig. 5.6 and Fig. 5.7 illustrate the ground-truth, and the reconstructions obtained with NDT, GICP and NICP. While NICP was able to consistently reconstruct both scenes from the datasets considered, GICP and NDT were not. Note that, despite the fact that the ETH Hauptgebaeude dataset is mostly composed of *curved surfaces*, NICP performed very well. NICP is an extended version of point-to-plane GICP that makes use of the normals in the error function. To further stress this aspect, we performed additional experiments on 3D laser data where we gradually transformed GICP in NICP, by increasing the weight of the normal component. To this end we executed NICP with different scaling factors for the information matrices $\mathbf{\Omega}^n_{ij}$ of the normals: 0 (GICP), 0.33, 0.66 and 1 (NICP). Note that when $\mathbf{\Omega}^n_{ij}$ is null, the block of the error vector relative to the normals is always null. This, together with computing the correspondences based only on point distance, results in NICP behaving

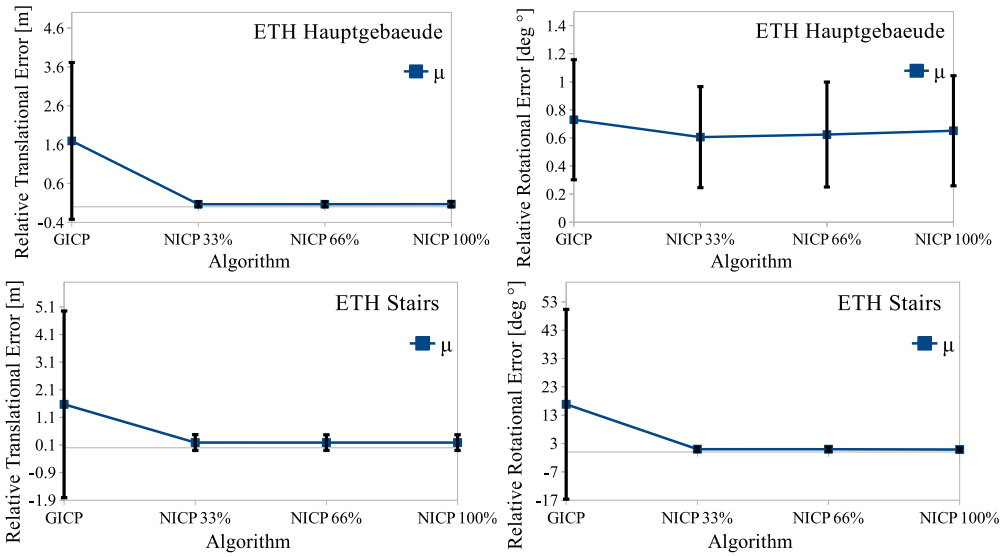


Figure 5.8: Evolution of the mean of the RPE, and its standard deviation, when incrementally increasing the weight of the surface normals in the algorithm. The first and second plots illustrate respectively the translational and the rotational part of the error for the ETH Hauptgebaeude dataset. The third and fourth plots illustrate respectively the translational and the rotational part of the error for the ETH Stairs dataset.

as GICP. The more we increase this scaling factor, the more accurate is the result and the lower the standard deviation (see Figure 5.8). These plots show that the normals provide relevant information that contributes to improve the results and the robustness of NICP.

5.3.3 Incremental Depth Camera Tracking

In this section we present a comparison of the incremental implementations of NICP and SNICP against NDT, point-to-plane GICP and KinFu using the large depth camera standard benchmarking dataset developed by Pomerleau *at al.* in [160] (ETH Kinect). To allow for a fair comparison, we added to GICP and NDT the same components that allows NICP to perform incremental registrations, and we kept the merging parameter

Table 5.3: Mean and standard deviation of the incremental point cloud registration time for each algorithm over all the sequences contained in the ETH Kinect dataset.

	NDT	**GICP**	**KinFu**	**SNICP**	**NICP**
Mean	183 ms	37 ms	52 ms	18 ms	38 ms
Std. Dev.	17 ms	4 ms	4 ms	2 ms	6 ms

fixed to the same value for all of them in all experiments. SNICP and KinFu come with their own merging systems.

As shown in Table 5.3, in terms of processing time, NDT was the slowest algorithm in computing a single registration. This is mainly due to the fact that it does not scale very well when the number of points in the scene increases. KinFu took about 50 ms to process each new point cloud, but this low frame rate is probably caused by the low-end graphics card in our system. GICP and NICP were able to execute close to real-time, while SNICP resulted to be the fastest processing clouds at a rate of almost 60Hz. More precisely, SNICP is more than $\sim$50% faster than NICP.

Table 5.4 reports the results obtained by processing all the sequences in the dataset with NDT, GICP, KinFu, SNICP and NICP algorithms. For each approach and for each test, we processed all the images in sequence and we generated the estimated trajectories. Green cells in the table highlight the best mean result for each specific test among all the compared algorithms. As the reader can see NICP, or its time efficient variant SNICP, are in most of the cases more accurate with respect to the other methods. The surface normals, in conjunction with a new merging method to recursively align point clouds, give a major contribute to increase the overall accuracy and robustness of our approach, thus leading to better camera tracking results. The reader might notice that in some tests, in particular the fast ones, both the translational and rotational errors are quite large. This is due to the fact that, in such experiments, the algorithms lost the tracking because of the large displacements between the clouds.

Table 5.4 also shows that SNICP, despite sacrificing a little of accuracy in favor of computation time efficiency, obtains results comparable with those of NICP. Due to project or environmental constraints, many robotics applications have to rely only on low-end hardware or they need to use multiple sensors simultaneously. In such cases, it is impossible to use methods like KinFu, or even NICP, since they require at least a mid-range laptop to reach near real-time performance. Consider also that in the case of KinFu, a nVidia graphics card is mandatory to run the algorithm, indeed it requires CUDA parallel computing. These results highlight the fact that, by exploiting low computational time methods, SNICP offers a very good alternative in this kind of situations.

5.3.4 Multiple Sensor Tracking

In this experiment we tested our tracking method on a dataset acquired in the Catacombs of Priscilla in Rome during the ROVINA project, and available at the link `http://www.rovina-project.eu/research/datasets`. The robot was equipped with 2 Asus Xtions mounted on the front. More in detail, assume the x axis of the robot reference frame pointing forward, and the z axis going upward. The left and the right Xtions were mounted at a distance of 30 cm between each other, and with a pan angle of respectively $\pi/6$ rad and $-\pi/6$ rad. These kind of catacombs are composed by an

	Mean / Standard Deviation Translational Error [m]					Mean / Standard Deviation Rational Error [deg°]				
Sequence	**NDT**	**GICP**	**KinFu**	**SNICP**	**NICP**	**NDT**	**GICP**	**KinFu**	**SNICP**	**NICP**
high-fast-f	0.906/0.761	**0.232/0.214**	0.902/0.804	0.521/0.271	0.234/0.200	43.1/46.0	**10.3/5.8**	13.4/9.1	22.8/17.4	12.0/7.0
high-fast-r	0.855/1.224	1.066/0.945	1.083/0.700	**0.077/0.034**	0.396/0.307	37.2/44.7	42.0/35.9	17.3/12.5	**10.0/7.3**	14.1/12.3
high-fast-t	0.158/0.296	0.106/0.076	0.198/0.222	0.135/0.092	**0.095/0.039**	7.1/8.7	5.9/5.1	5.2/2.6	**5.0/2.4**	5.2/2.5
high-med-f	0.310/0.371	0.071/0.056	0.090/0.096	0.168/0.129	**0.060/0.032**	13.7/13.3	4.6/2.8	**4.2/2.2**	7.2/4.3	4.9/2.7
high-med-r	0.297/0.694	0.136/0.153	0.515/0.554	**0.040/0.018**	0.107/0.112	18.9/32.4	8.9/3.8	11.2/6.6	8.7/3.4	**8.3/3.2**
high-med-t	0.081/0.073	0.032/0.013	**0.031/0.014**	0.067/0.050	0.032/0.013	3.4/2.8	**2.3/1.1**	**2.3/1.1**	2.5/1.8	**2.3/1.1**
high-slow-f	0.205/0.390	0.024/0.015	0.052/0.102	0.034/0.020	**0.023/0.015**	8.5/21.2	1.7/1.1	**1.3/0.7**	2.4/2.0	1.6/1.0
high-slow-r	0.212/0.382	0.079/0.116	0.196/0.348	**0.026/0.020**	0.079/0.118	15.1/30.0	4.5/3.9	4.0/2.7	**3.2/1.9**	3.5/2.1
high-slow-t	0.043/0.071	0.015/0.006	**0.012/0.005**	0.021/0.009	0.015/0.006	1.6/1.2	1.0/0.4	**0.9/0.4**	1.0/0.4	1.0/0.4
low-fast-f	0.943/0.707	**0.297/0.275**	0.928/0.673	0.445/0.265	0.359/0.390	30.7/42.3	13.2/12.8	16.1/20.3	**13.1/10.8**	13.8/13.6
low-fast-r	0.816/1.171	0.337/0.516	0.724/0.770	**0.087/0.046**	0.325/0.365	35.1/43.9	21.0/23.2	14.1/7.4	**13.5/7.6**	15.5/11.7
low-fast-t	0.392/0.366	**0.064/0.029**	0.201/0.288	0.140/0.087	0.067/0.032	11.5/11.6	**4.9/2.6**	5.4/3.3	5.1/2.6	**4.9/2.5**
low-med-f	0.611/0.707	0.065/0.042	0.301/0.414	0.137/0.120	**0.053/0.035**	13.7/19.3	3.5/2.5	8.8/19.4	4.7/2.8	**3.3/2.0**
low-med-r	0.188/0.248	0.065/0.073	0.423/0.488	**0.035/0.019**	0.061/0.069	8.8/5.3	**6.9/2.3**	7.3/2.4	7.1/2.5	7.1/2.7
low-med-t	0.161/0.232	**0.059/0.027**	0.143/0.172	0.072/0.033	**0.059/0.027**	8.8/17.3	3.8/2.4	**3.7/2.0**	3.8/2.3	3.8/2.3
low-slow-f	0.179/0.211	0.035/0.041	**0.030/0.021**	0.087/0.096	0.033/0.035	5.9/15.7	2.2/1.6	**1.5/0.7**	1.9/1.1	2.1/1.5
low-slow-r	0.154/0.207	0.081/0.149	0.242/0.412	**0.032/0.029**	0.078/0.137	5.8/6.1	3.6/2.3	4.1/4.6	4.1/4.2	**3.3/2.2**
low-slow-t	0.041/0.031	0.026/0.017	0.061/0.137	0.043/0.039	**0.025/0.017**	1.6/1.0	**1.3/0.7**	1.5/0.9	1.4/0.8	**1.3/0.7**
med-fast-f	0.917/0.697	0.522/0.425	1.183/0.852	0.532/0.273	**0.329/0.248**	39.0/31.2	32.1/33.8	21.1/17.2	26.6/19.9	**17.7/10.4**
med-fast-r	0.709/1.234	0.837/1.015	0.715/0.574	**0.072/0.033**	0.299/0.251	24.9/33.2	29.3/32.5	15.9/10.2	**13.4/6.7**	14.3/8.3
med-fast-t	0.173/0.270	0.086/0.045	0.180/0.289	0.138/0.096	**0.082/0.039**	6.2/4.8	5.1/2.5	5.0/2.4	**4.8/2.1**	5.0/2.3
med-med-f	0.542/0.773	0.064/0.049	0.144/0.164	0.128/0.097	**0.050/0.031**	26.0/42.1	4.1/2.6	7.0/9.2	6.0/4.8	**4.0/2.4**
med-med-r	0.119/0.142	0.097/0.102	0.403/0.449	**0.050/0.023**	0.070/0.067	8.8/3.5	8.3/3.2	8.2/3.3	7.8/3.0	**7.4/2.6**
med-med-t	0.059/0.056	**0.029/0.012**	0.033/0.014	0.046/0.025	0.030/0.012	2.4/1.4	2.0/0.9	1.9/0.9	1.9/0.9	**1.9/0.8**
med-slow-f	0.176/0.319	0.030/0.018	0.043/0.073	0.044/0.038	**0.024/0.011**	4.9/10.8	2.0/1.3	**1.9/1.1**	2.5/1.4	**1.9/1.0**
med-slow-r	0.112/0.279	0.035/0.057	0.096/0.233	**0.021/0.018**	0.031/0.052	4.1/5.0	2.3/1.3	2.7/3.9	2.2/1.4	**2.1/1.1**
med-slow-t	0.085/0.226	**0.016/0.007**	0.095/0.244	0.025/0.013	**0.016/0.007**	2.5/4.3	**1.1/0.6**	**1.1/0.6**	1.2/0.6	**1.1/0.5**

Table 5.4: Incremental point cloud registration mean and standard deviation of the relative translational and rotational error for all the sequences of the ETH Kinect dataset. Green cells in the table highlight the best mean result for each specific test among all the compared algorithms (NDT, GICP, KinFu, SNICP and NICP).

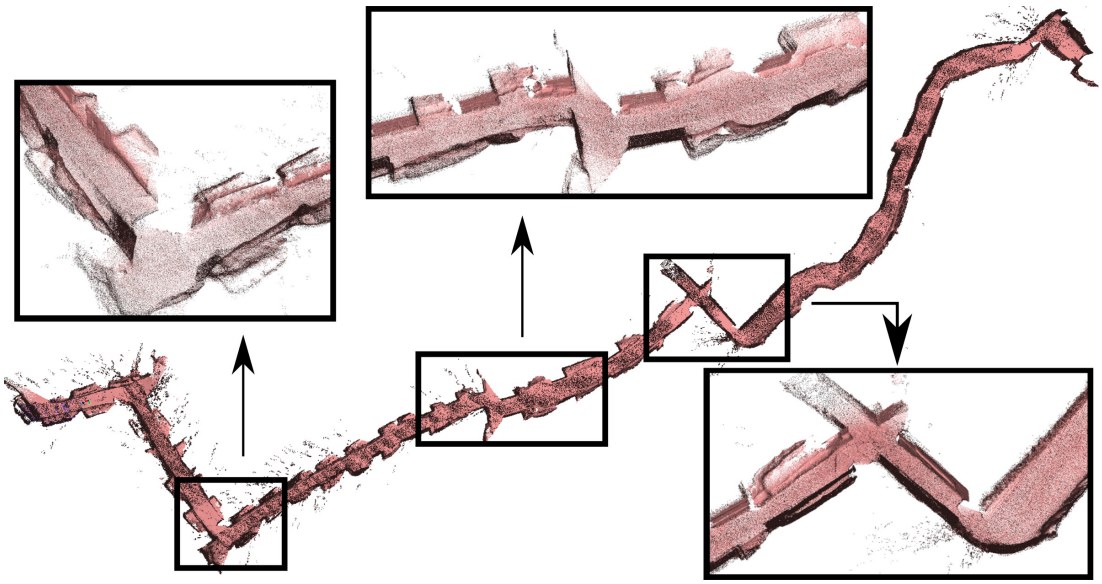


Figure 5.9: Multiple sensor tracking in the catacombs of Priscilla in Rome. In this specific case 2 Asus Xtions have been used.

underground network of very narrow corridors. While exploring the tunnels, multiple sensors are fundamental to help the robot to avoid the possibility of blindness on turns. Fig. 5.9 shows the result of the tracking on a part of the dataset, the magnifications highlight the high quality of the point clouds generated. No surface reconstruction has been applied. To give an idea of the size of the map, consider that the whole dataset has a dimension of $\sim$100$\times$50 meters, and the figure depicts a portion of $\sim$35$\times$10 meters.

5.3.5 Tracking with Dynamic Objects

In this section we show some result of our tracking algorithm working in a environment with dynamic objects. In this test case we run our tracking approach inside an office like environment. While registering, both a chair and a person moved inside the field of view of the camera. Despite these dynamic objects in the scene, our algorithm has been able to track the pose of the sensor, and at the same time remove such elements from the point cloud. Fig. 5.10 illustrates a temporal sequence of snapshot acquired during the tracking. Red points belong to the last depth image registered.

5.3.6 Complexity Analysis

In this section we provide an analysis of the computational complexity of the algorithms treated in this Chapter.

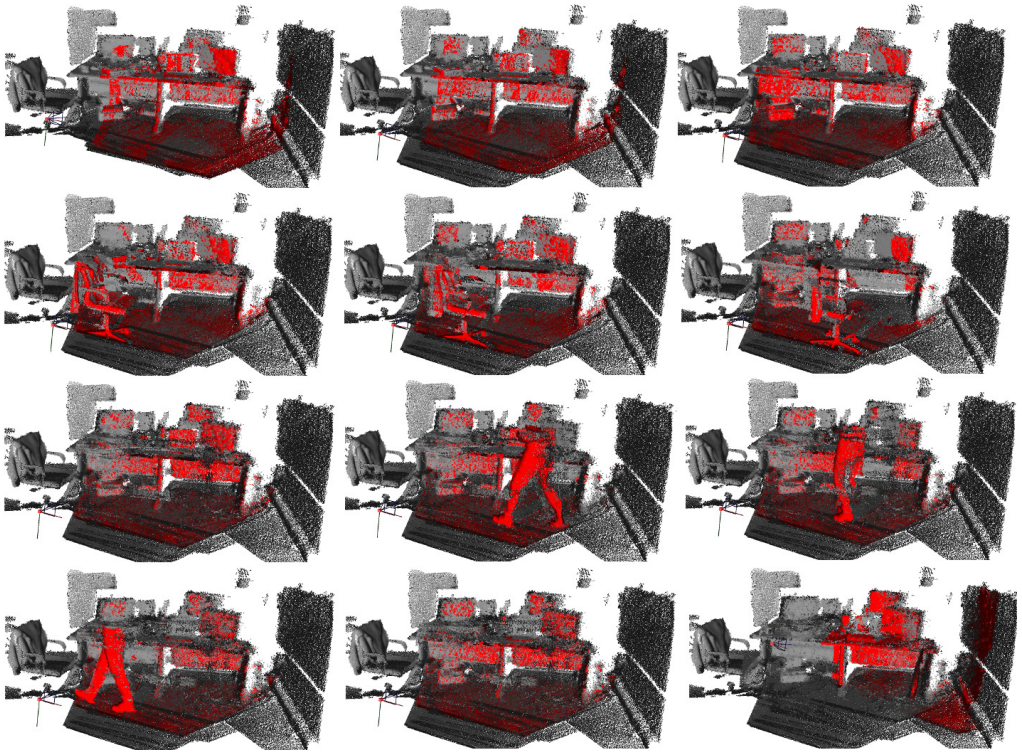


Figure 5.10: Example of camera tracking in an office like environment with dynamic objects. From left to right, and from top to bottom: a temporal sequence of snapshots acquired during the tracking. Red points belong to the last depth image. Despite both a chair and a person moved within the field of view of the camera, our algorithm is able to track the sensor pose and remove the dynamic objects.

The surface normal computation approach plays an important role during an alignment. Indeed, methods using Singular Value Decomposition on the neighborhood of the points have $\mathcal{O}(k \cdot d^3)$ complexity, where k is the number of neighboring points, and d is the dimension of the matrix on which the SVD is computed. In addition to this, it must also be taken into account the complexity associated to the computation of the covariance matrix of each neighborhood. When using KD-trees, this means an additional $\mathcal{O}(p \cdot k \cdot \log(p))$, where p is the number of points in the cloud, plus the KD-tree generation time. Using integral images, instead, adds a $\mathcal{O}(n \cdot m)$ complexity related to the integral image construction, with $(n,\ m)$ being the dimension of the integral image itself. Once generated, the covariance matrix can be computed in constant time $\mathcal{O}(1)$. NICP uses integral image based normal computation through SVD. Using a cross product based method, as done in SNICP, leads to a complexity $\mathcal{O}(k)$, thus to a relevant computation time reduction at the cost of less accurate surface normals.

Another part that must be analyzed is the correspondence search method. By using a line of sight, or projective, criteria the complexity is $\mathcal{O}(n \cdot m)$, where $(n,\ m)$ is

the dimension of the image where the points are projected. Using KD-trees, instead, leads to a complexity of $\mathcal{O}(q \cdot \log(p))$, with q and p being respectively the number of queries in the KD-tree, and the number of points in the cloud. Additionally, as in the case of the surface normal computation, we need to add also the KD-tree construction time. Correspondences estimated using KD-trees are usually slower to compute, but in general more accurate. In our approach we use projective correspondence search.

Finally, the last part to be analyzed is the approach used for computing the relative transformation between the two clouds. The algorithms that solve this problem can be divided in two main classes: direct and iterative approaches. Direct methods like the Horn [116] formula computes the relative translation and rotation between two cloud in one step. Unfortunately, such solutions can be applied only under the assumption that the correspondences are known. Since this is not the general case, we must use iterative methods. All iterative methods have linear complexity in the number of measurements.

5.4 Conclusions

We presented in detail a novel variant of ICP to recursively register point clouds. Our method extends the measurement vector with surface normals information and it uses a line of sight criterion to find correspondences. We discussed the relevant steps needed for the implementation of such a system, and we also provided all the mathematical derivations. Additionally, we introduced a method for point cloud merging that allows to decimate the points on a cloud, while taking into account the sensor intrinsic error of the points. Also, we extended the method to handle dynamic objects and we provided a variant that can process data with a frame rate of ∼60Hz, on a single CPU core.

We showed experiments, on standard benchmarking datasets, comparing our approach to other state-of-the-art methods demonstrating that our algorithm offers better results and higher robustness. We also presented tracking experiments in environments with dynamic objects, or with multiple sensors.

Part III

LOCALIZATION AND MAPPING

Chapter 6

A SLAM System for Mapping Catacombs

"The sign above the door was written in French. It read:
ARRÊTE! C'EST ICI L'EMPIRE DE LA MORT.*"*
"That means," he explained to Gini,
"Stop! It is here the Empire of Death."
— Paul Aertker, Brainwashed, 2014

MAPPING and digitizing relevant archeological sites is a key element to preserve cultural heritage and to make it accessible to the public. Current systems typically build upon static 3D laser scanning technology that has to be moved into archeological sites by humans. While this approach is acceptable in general, it prevents the digitization of cultural heritage that is inaccessible by humans such as large parts of the catacombs of Rome, Naples or other underground sites. Autonomous robots, however, offer a great opportunity to explore and digitize such sites without bringing humans in danger.

In this Chapter, we describe the mapping system that we developed within the European funded project ROVINA, that aims at achieving digital preservation through mobile robots. Our target, and test scenario, is the exploration and digitization of Roman catacombs. Catacombs are ancient Roman cemeteries that were dug several meters below the ground level and form a large network of tunnels. They are commonly found around the main consular Roman roads. Inside the catacombs, it is possible to find relevant archeological material and artifacts such as frescoes, furnishings and ornaments. Besides the archeological interest, exploring the catacombs is of fundamental importance also for safety aspects concerning the construction of new buildings. Unfortunately, catacombs are a dangerous environment due to risks related to structure collapses, and radioactive gas radon.

Mapping catacombs poses substantial challenges to autonomous robot systems, which prevents to use of-the-shelf SLAM solutions. Most of the structures look highly

Figure 6.1: Main components of our robot. On the left, the Element mobile base by Mesa Robotics. On the center, the custom structure built to house computers, sensors, and power supplying of the complete platform. On the right, the assembled robot moving in a catacomb narrow area.

similar with very little texture, and the terrain is mostly uneven and subject to potential collapses. The tunnels are dark, narrow, with many turning points, and they can develop over several kilometers. Moreover, the high humidity challenges the electronic equipment.

Digitizing underground structures, tunnels or mines has been already addressed by other research groups. The system described here mainly differs from previous work because its final goal is to build a tool that can be used by archeologist and people with different background. Thus, robustness and ease of use plays an important role. To monitor the progress of the activity, and to foster autonomous exploration behaviors, our mapping system can operate on-line and is able to produce a full 3D map.

In the following sections we describe the hardware platform used in the project, with particular emphasis on the sensor used by our mapping system. To limit the number of connections between the robot, and the controlling PC, we designed a distributed system that makes use of on-board computers to control the sensors. Each time the robot is dismounted, the sensors used by our system must be calibrated. For this reason, we also describe the procedure we apply for calibrating the robot. Our mapping system can process the output of three depth cameras simultaneously, and it follows the well known graph-based Simultaneous Localization And Mapping formalism [17]. The nodes of the graph are local maps consisting of probabilistic point clouds obtained by registering multiple depth images. This is done through a novel variant of the well known Iterative Closest Point algorithm named SNICP (see Chapter 5), that improves both robustness and accuracy.

We discuss the results of our system on field experiments that demonstrate the effectiveness of our method. The dataset collected in the catacombs are publicly available through the ROVINA project web site `http://www.rovina-project.eu`.

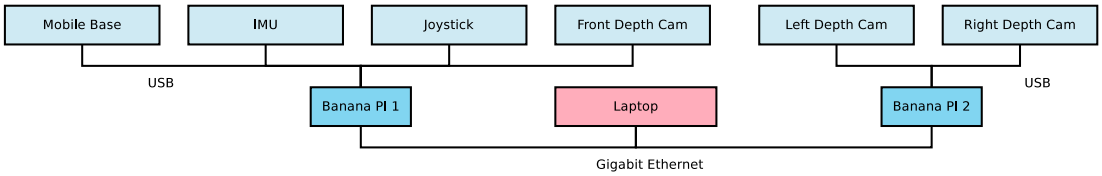


Figure 6.2: Overview of the hardware configuration of the robot.

6.1 Robot Platform Setup

Our robot consists of an Element base by Mesa Robotics equipped with a custom-built housing installed on top of the payload bay. The basic components and the assembled platform are displayed in Fig. 6.1.

The robot is equipped with three RGB-D cameras, whose arrangement guarantees a minimal overlap between the observations, needed for platform calibration, while offering a wide perception volume necessary to simultaneously capture different views of the environment. Fig. 6.2 shows an overview of the hardware configuration.

6.2 Sensor Calibration

Each time the platform is mounted, the sensors need to be calibrated. To this end, we designed quick calibration procedures that allow to set up the robot in a few minutes. Our mapping system heavily relies on RGB-D cameras (Microsoft Kinect or Asus Xtion). The depth measurements of these devices are affected by substantial systematic distortions, and noise, that need to be compensated to maximize the performance of the whole system. In the reminder of this section we first discuss how to perform the calibration of the intrinsic distortion parameters. This has to be done only once. Subsequently, we describe our procedure to determine the relative position of the sensors with respect to a common reference frame on the robot.

6.2.1 Depth Camera Intrinsic

Common RGB-D cameras are affected by a substantial distortion in the depth channel. Ignoring this distortion leads to systematic drifts in the estimate of the robot pose while mapping. The two left most images in Fig. 6.3 illustrates the top view of a reconstruction made with a calibrated, and uncalibrated depth camera. The right pair of images in Fig. 6.3, instead, show a top view of the depth measurements when the camera is facing a planar wall at different distances. As the reader can see, in the case of uncalibrated measurements, planar structures like walls that should be straight result to be curved.

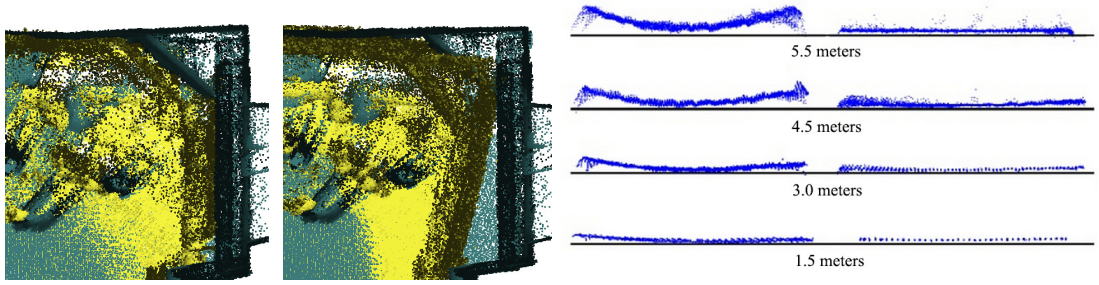


Figure 6.3: The left pair of images show the top view of a 3D reconstruction (yellow), made with a calibrated and an uncalibrated 3D camera, overlaid with the ground-truth (blue). Similarly, the right pair of images image show the top view of a wall acquired at different distances with a calibrated and an uncalibrated 3D camera.

To perform the calibration of the camera parameters we adopt the approach developed by Di Cicco *et al.* in [162]. This method estimates a non-parametric form of the undistortion function, that maps each depth value $d_{u,v}$ of a pixel of coordinates u, v to a calibrated value $d'_{u,v}$. Even if irregular, the distortion of a depth sensor is rather continuous allowing to represent the undistortion function through a coarse grid of multipliers $m_{u,v}$. This grid contains for each depth measurement, and for each pixel, the ratio between the calibrated and the uncalibrated values. Applying the undistortion parameters on-line is straightforward since it only requires a lookup, and a multiplication per depth measurement.

6.2.2 Relative Position of Sensors

To determine the relative offset between the cameras, we construct a set of independent point clouds, one for each camera, while the robot is moving. To generate these point clouds we use the registration system described in the following section. Assuming the robot is standing at the beginning of this operation, each sensor will produce a cloud starting from its own reference frame. At this point we move the robot so that, at different points in time, the field of view of the cameras fall on the same region of the space. At the end of this step, we obtain a small reconstruction of the surrounding environment for each camera. We arbitrary select one of these point clouds as our reference scene, and we compute the relative transformation between it and all the other reconstructions. This allows us to determine the relative translation and rotation between the origins of the clouds, and thus the origin of all cameras with respect to a reference one.

Finally, to determine the relative position of the chosen reference camera with respect to the mobile base, we steer the robot and we estimate the motion of the camera system in a small region based uniquely on the camera measurements. Let $\mathbf{C}_{1:N}$ be the transforms of the camera system along the path of the robot, and $\mathbf{T}_{1:N}$

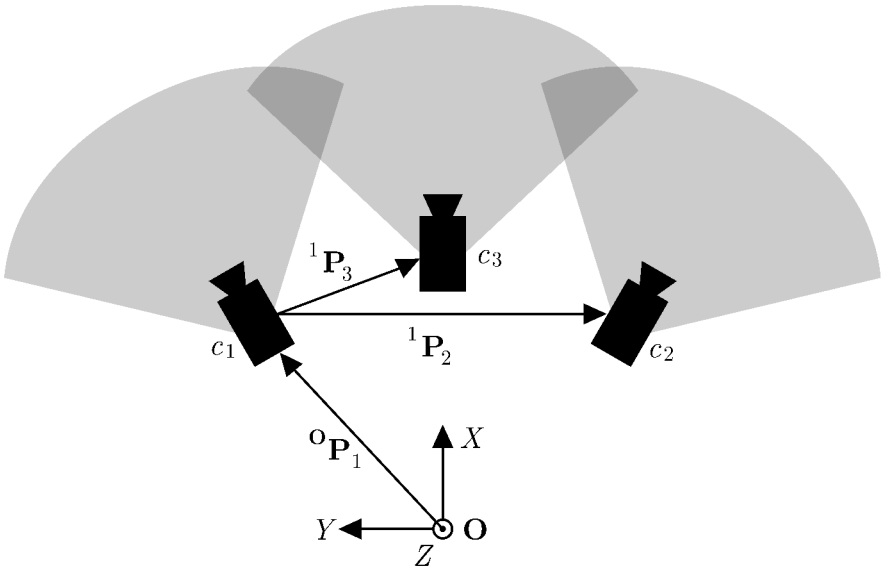


Figure 6.4: Diagram showing the transform tree computed through our calibration procedures, and containing the relative poses of the camera sensors. In the figure, $\mathbf{O}$ is the reference frame of the robot, c_i labels the camera i, and ${}^i\mathbf{P}_j$ represent the pose of the camera i with respect to j. In this specific case, ${}^1\mathbf{P}_3$ and ${}^1\mathbf{P}_2$ are computed through pure point cloud registrations, while ${}^{\mathbf{O}}\mathbf{P}_1$ is estimated by solving Eq. 6.3.

be the path estimated by the odometry. Assuming that the position of the reference camera system $\mathbf{P}$ is known, and also the system to be free from noise, we can write the following relation

$$\hat{\mathbf{C}}_i = \mathbf{T}_i \, \mathbf{P} \tag{6.1}$$

To lessen the effect of accumulating drift, instead of using global offsets as done in the previous equation, we can use incremental displacements between subsequent camera motions as

$$\hat{\mathbf{C}}_{i,i+1}(\mathbf{P}) = (\mathbf{T}_i \, \mathbf{P})^{-1} (\mathbf{T}_{i+1} \, \mathbf{P}) \tag{6.2}$$

We can then estimate the position of the reference camera, with respect to the mobile base, by minimizing the following metric

$$\mathbf{P}^* = \underset{\mathbf{P}}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| \mathbf{C}_{i,i+1}^{-1} \hat{\mathbf{C}}_{i,i+1}(\mathbf{P}) \right\| \tag{6.3}$$

Here $\mathbf{C}_{i,i+1} = \mathbf{C}_i^{-1} \mathbf{C}_{i+1}$ is the relative displacement between two subsequent camera poses. Fig. 6.4 shows a diagram depicting the transform tree containing the relative poses of the camera sensors computed with our calibration procedures.

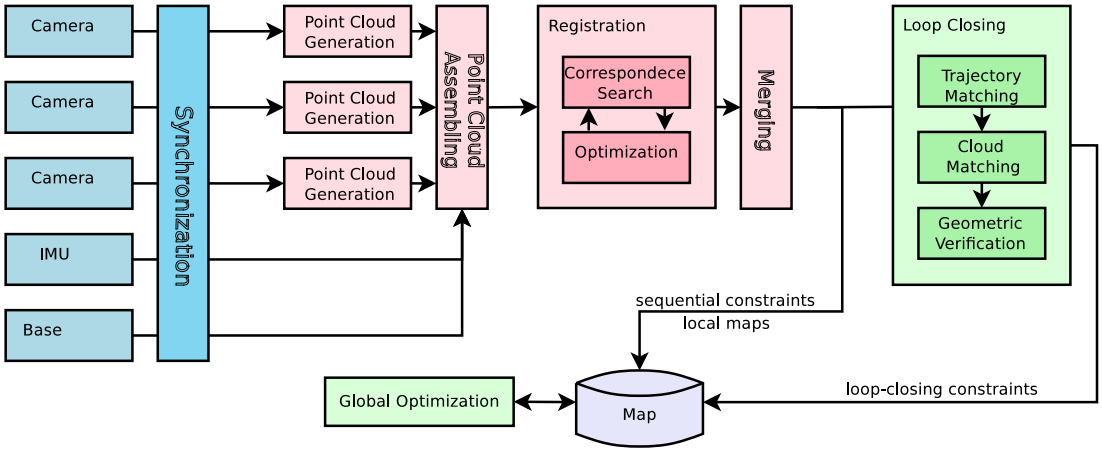


Figure 6.5: Architecture of our mapping system

6.3 Mapping System

To build a consistent map model, and to localize in it, we need to solve the online Simultaneous Localization And Mapping problem. Our approach builds upon the graph-based formulation of SLAM [17], and the overall architecture of our mapping system is illustrated in Fig. 6.5.

To solve the underlying optimization problem, we rely on the well-known g2o library [16] for robust error minimization. In order to use g2o, or any related graph-based SLAM framework, we need to generate the nodes and edges of the pose graph that represents our minimization problem. In the graph-based SLAM formulation the edges encode spatial constraints arising by pairwise registrations of the measurements of the connected nodes. In this section we describe how we generate the pose graph out of the sensor data, in other words the SLAM front-end. Our system builds a pose graph whose nodes are local maps. Each local map is a point cloud constructed by integrating a sequence of sensor data, while the robot moves in the environment. We start a new local map whenever the robot has traveled for a given distance (approximately 5 meters), or if consecutive observations cannot be matched for some reasons.

To create the whole pose graph, we need to solve two problems. First, we have to be able to *build consistent local* maps (Sec. 6.3.1). Second, we need to find loop closures and thus determine if the robot has revisited an already mapped area. Additionally, we must be able to *register* pairs of local maps in order to validate candidate loop closures (Sec. 6.3.2). To enable the generation of large-scale maps, the local maps are always moved from the main memory to the disc. In this way only the underlying graph structure, but not the local point clouds, needs to be maintained in the robot's main memory.

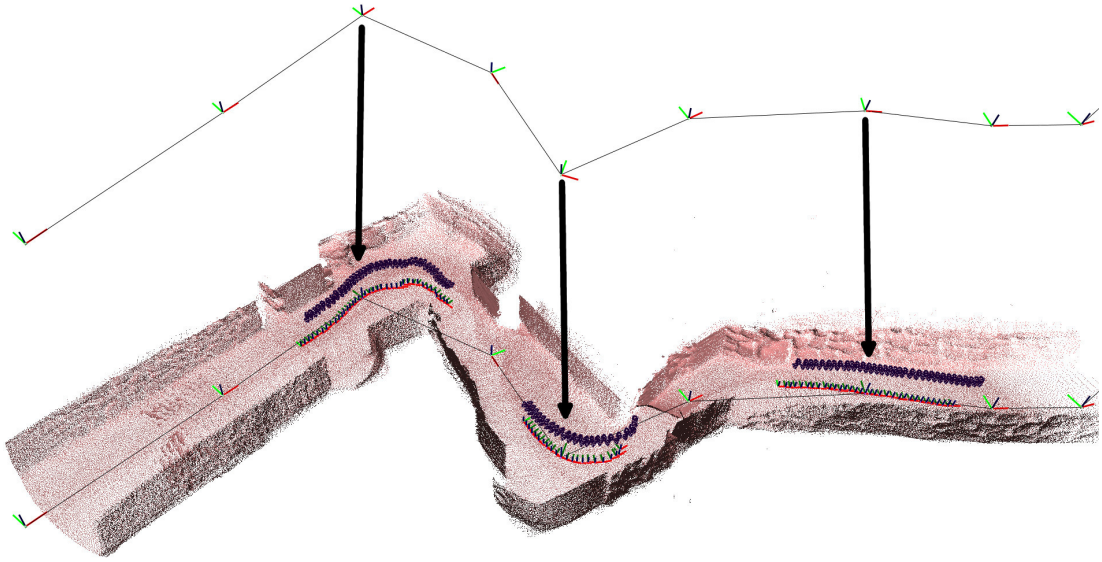


Figure 6.6: Example of pose graph whose nodes are local maps. The top image shows a piece of the graph generated by the local mapper. The bottom picture depicts the same graph with three nodes highlighted (the internal trajectory $\mathcal{T}$ of the three local maps is visible).

6.3.1 Local Map Generation

In our system, a local map consists of a set $\mathcal{P}$ of 3D rich points, and the path $\mathcal{T}$ of the robot from which the points have been recorded. For each reach point we compute the 3D coordinates $\mathbf{p}_i$, the surface normal $\mathbf{n}_i$ and a weight w_i proportional to the information of the point. The path $\mathcal{T}$ is a list of poses $\mathbf{x}_i \in \mathcal{SE}(3)$. Fig. 6.6 shows an example of pose graph whose nodes are local maps generated by our system.

The generation of a local map is performed in an incremental fashion, and in the remainder of this section we describe in detail the central steps of this process. We use each triplet of depth images to extend the current local map. By using the relative transformation between the depth cameras, computed during the calibration step, we generate a point cloud from the triplet of depth images. Then, we register this cloud with the previous one by taking into account also the most recent odometry and IMU measurements. Subsequently we augment the trajectory with the new robot position. As a last step, we merge the most recent sensor measurements with the local map built so far. This is done by performing a per-point filtering operation.

Generation of a Rich Point Cloud from Depth Images

Each time a set of depth images $\{\mathcal{D}_k^{\mathrm{c}}\}$ (one for each camera k) is available, we compute the 3D points, the surface normals and the weights for each image and pixel in the set. Then, based on the relative positions of the cameras, we combine these rich points in a single cloud centered in the reference frame of the robot.

More formally, for each depth measurement $d_{k_{u,v}}$, belonging to the k-th depth image $\mathcal{D}_k^{\mathrm{c}}$, we compute the 3D point coordinates $\mathbf{p}_{k_{u,v}}$ using the relation described in Eq. 3.11 as follows

$$\mathbf{p}_{k_{u,v}} = \mathbf{K}_k^{-1} d_{k_{u,v}} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{6.4}$$

where $\mathbf{K}_k$ is the calibration matrix, associated to the k-th depth camera, containing the intrinsic parameters of the pinhole model. The weight $w_{k_{u,v}} = 1/d_{k_{u,v}}$ of each point is the inverse of its depth. In this way far points, that are affected by larger noise, receive a smaller weight.

Subsequently, we compute the surface normal $\mathbf{n}_{k_{u,v}}$ of each point in the depth image as the cross product of the neighborhood as described in Eq. 5.19. Moreover, to further reduce the effect of the noise, we average the normal estimates of neighboring points based on the weights as

$$\mathbf{n}_{k_{u,v}} = \sum_{i=u-\Delta}^{i=u+\Delta} \sum_{j=v-\Delta}^{j=v+\Delta} w_{k_{ij}} \mathbf{n}_{k_{ij}} \tag{6.5}$$

where Δ is a constant value defining the size of the window on which the average is computed. Finally, we scale all normals to unit length.

The set $\mathcal{P}$ of measurements composing the final local map is obtained by the union of all rich points computed from the depth images generated by each camera

$$\mathcal{P} = \bigcup_{k \in \mathcal{S}} \mathcal{P}_k \tag{6.6}$$

with $\mathcal{S}$ being the set of sensors used, and $\mathcal{P}_k$ the set of rich points $\langle \mathbf{p}_{k_{u,v}},\ \mathbf{n}_{k_{u,v}},\ w_{k_{u,v}} \rangle$ computed from the k-th depth image.

Registration of the Current Cloud in the Local Map

The next step of our algorithm consists in finding a transformation $\mathbf{T}^*$ that maximizes the overlap between the current cloud $\mathcal{P}^{\mathrm{c}} = \{\mathbf{p}^{\mathrm{c}}_{1:N^{\mathrm{c}}}\}$, and the local map built so far. In the reminder of this section we will refer to the cloud of the local map $\mathcal{P}^{\mathrm{r}} = \{\mathbf{p}^{\mathrm{r}}_{1:N^{\mathrm{r}}}\}$ as reference cloud.

Our approach uses a variant of the cost function described by Eq. 4.1, where we simultaneously minimize a weighted distance of corresponding points *and* corresponding normals as follows

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} w_i w_j \left(\left\| \mathbf{T} \oplus \mathbf{p}_i^{\mathrm{c}} - \mathbf{p}_j^{\mathrm{r}} \right\|^2_{\mathbf{\Omega}^{\mathrm{p}}_{ij}} + \left\| \mathbf{T} \oplus \mathbf{n}_i^{\mathrm{c}} - \mathbf{n}_j^{\mathrm{r}} \right\|^2_{\mathbf{\Omega}^{\mathrm{n}}_{ij}} \right) \tag{6.7}$$

where $\|\mathbf{e}\|^2_{\mathbf{\Omega}} = \mathbf{e}^T \mathbf{\Omega}\, \mathbf{e}$, $\mathcal{C}$ is the set of corresponding point pairs $\langle i,\ j \rangle$, while $\mathbf{\Omega}^{\mathrm{p}}_{ij}$ and $\mathbf{\Omega}^{\mathrm{n}}_{ij}$ are respectively the information matrices of the points, and of the normals. The $\oplus$ operator is defined as in Eq. 4.2. The contribution of each correspondence is scaled by the factor $w_i w_j$ that accounts for the noise in the points. Considering also the normals has positive effects on the optimization since it allows for one single correspondence point to resolve also two degrees of freedom in the orientation. These statements have been already discussed in Chapter 4, where we conducted a deep analysis of the objective function in Eq. 6.7.

Correspondence Search

To find the correspondences our system can use two different strategies: a fast approach that relies on image projection, and a slower but more accurate alternative based on KD-trees. During incremental tracking we use the first strategy, since temporally subsequent perceptions are likely to be close. While for loop closures we use the KD-tree based approach. Note that both strategies require an initial guess of $\mathbf{T}$.

Image Projection The basic idea of this strategy is to obtain from the local map a set of depth images, one for each camera sensor, as if the robot is located in $\mathbf{T}$. Let $\mathbf{P}_k$ be the transformation of the k-th depth camera with respect to the origin of the robot, and let $\mathbf{p}_i$ be a point in a cloud, we can determine the $(u_i,\ v_i)$ coordinates of a pixel in the image of the camera by extending Eq. 3.8 as

$$\begin{pmatrix} u'_i \\ v'_i \\ w'_i \end{pmatrix} = \mathbf{K}_k \mathbf{P}_k^{-1} \mathbf{T}^{-1} \mathbf{p}_i \tag{6.8}$$

and then normalizing the resulting coordinates by the factor w'_i

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \frac{1}{w'_i} \begin{pmatrix} u'_i \\ v'_i \end{pmatrix} \tag{6.9}$$

If $\mathbf{T}$ is known, the points in the reference and the current cloud that project on the same pixel and in the same depth image would correspond. In practice this is never the case. However, if $\mathbf{T}$ is small enough, correspondences found using this strategy

and used in the cost function of Eq. 6.7 have shown a good and robust behavior. Our system rejects the correspondences when points are too far in distance, or the angle between their normals is greater than a certain threshold. To further increase the basin of attraction, we use this approach at three different scales: 1/8, 1/4 and 1/2 of the original image size. The correspondences are computed for all cameras, and passed simultaneously to the optimization.

KD-tree Finding correspondences through image projection has the main shortcoming of being heavily dependent on the particular viewpoint, and in the need of a good initial guess. Even if slower with respect to image projection, KD-trees are a more attractive alternative when the guess of the initial transform is poor. In our local maps, however, in addition to the euclidean coordinates of the point, we also have the surface normal components. It is therefore natural to embed this additional information in the KD-tree structure. To this end, we store in the tree 6D vectors $\tilde{\mathbf{p}}_i$, consisting of a point and its scaled normal

$$\tilde{\mathbf{p}}_i = (\mathbf{p}_i^T \ \alpha \mathbf{n}_i^T)^T \tag{6.10}$$

The coefficient α allows to scale the contribution of a normal, when an Euclidean metric for the distance is used. In our default implementation we set α to be equal to 1/3 because we want a difference in the normals of 1 radiant to be punished approximately as a difference in Euclidean distance of 1 meter.

Merging

Iterating the three steps discussed before, leads to an estimate of the transformation $\mathbf{T}^*$ that best aligns the current cloud onto the reference one. To incorporate the new information we need to augment the local map with all the new points, transformed according to $\mathbf{T}^*$, and to append to the trajectory $\mathcal{T}$ the updated robot pose. A naive approach would result in a linear growth of the points in the local map, thus increasing both computation time, and memory consumption.

Our merging procedure aims at updating the points in the reference point cloud $\mathcal{P}^{\mathrm{r}}$, with the new points in current point cloud $\mathcal{P}^{\mathrm{c}}$, under the knowledge of the transform $\mathbf{T}^*$ that maps current onto reference. Note that, depending on the specific case, this update operation might lead also to the removal of points in $\mathcal{P}^{\mathrm{r}}$. Our merging approach works by identifying a set of correspondences between the points in the two clouds. Each corresponding pair is fused using a Kalman filter like update, while points that do not correspond are left untouched.

To find the correspondences, we use the image projection approach described in Section 6.3.1. Again, we render a set of depth images $\{\mathcal{D}_k^{\mathrm{r}}\}$ from the local map $\mathcal{P}^{\mathrm{r}}$, as if the robot is located in $\mathbf{T}^*$, according to the layout of the cameras. Now, let $d_{k_{u,v}}^{\mathrm{r}}$ be a pixel in the k-th reference depth image $\mathcal{D}_k^{\mathrm{r}}$ obtained by projecting the point $\mathbf{p}_{u,v}^{\mathrm{r}}$,

and let $d^{\mathrm{c}}_{k_{u,v}}$ be a pixel in the k-th current depth image $\mathcal{D}^{\mathrm{c}}_k$ obtained by projecting the point $\mathbf{p}^{\mathrm{c}}_{u,v}$. By selecting a threshold τ, for each pixel coordinate we distinguish the three cases discussed in Sec. 5.1.4 that we report here for clarity purposes:

- $d^{\mathrm{c}}_{k_{u,v}} - d^{\mathrm{r}}_{k_{u,v}} > \tau$, in this case the reference point is closer to the observer than the current point of at least τ. In other words the part of the current view sees through the old point cloud. This might be due to a dynamic aspect. To accommodate for that we remove from the reference cloud $\mathbf{p}^{\mathrm{r}}_{u,v}$ and we replace it with $\mathbf{p}^{\mathrm{c}}_{u,v}$;
- $d^{\mathrm{r}}_{k_{u,v}} - d^{\mathrm{c}}_{k_{u,v}} > \tau$, in this case the current point is closer to the observer than the reference point of at least τ. This might be due to an occlusion or to a newly perceived portion of the environment, and we add $\mathbf{p}^{\mathrm{c}}_{u,v}$ to the reference cloud;
- $\|d^{\mathrm{r}}_{k_{u,v}} - d^{\mathrm{c}}_{k_{u,v}}\| < \tau$, the two depth values are closer than τ. In this case we assume they originate from the same portion of the environment, and we merge the two based on their weights as follows

$$\begin{pmatrix} \mathbf{p}^{\mathrm{r}}_{u,v} \\ \mathbf{n}^{\mathrm{r}}_{u,v} \end{pmatrix} \leftarrow \frac{1}{w^{\mathrm{r}}_{u,v} + w^{\mathrm{c}}_{u,v}} \left[w^{\mathrm{r}}_{u,v} \begin{pmatrix} \mathbf{p}^{\mathrm{r}}_{u,v} \\ \mathbf{n}^{\mathrm{r}}_{u,v} \end{pmatrix} + w^{\mathrm{c}}_{u,v} \begin{pmatrix} \mathbf{p}^{\mathrm{c}}_{u,v} \\ \mathbf{n}^{\mathrm{c}}_{u,v} \end{pmatrix} \right] \tag{6.11}$$

$$w^{\mathrm{r}}_{u,v} \leftarrow w^{\mathrm{r}}_{u,v} + w^{\mathrm{c}}_{u,v} \tag{6.12}$$

6.3.2 Loop Closing

The procedure described so far generates a sequential list of local maps, and the relative transformations between them. These estimates, however, are affected by an unavoidable drift that can be mitigated by re-localizing the robot in places already visited in the past. Whenever this operation succeeds, we need to accommodate the spatial location layout of all previously computed local maps based on the new information. The re-localization requires finding a local map generated in the past that can be successfully overlapped with the most recent one. This problem is known as loop closing, and it is fundamental to guarantee the global consistency of the reconstructed map.

Instead of performing a brute-force exhaustive search among all the possible closures, an eventually working yet slow heuristic, our mapping system exploits two different strategies to find candidate loop closures. The first one searches for closures looking at local maps that are spatially near to the current one. More formally, the loop closer adds a local map in the set of candidate closures if the distance between the center of the current local map, and the candidate one, is lower that a certain threshold.

The second strategy exploits the similarities in the robot trajectories. The core idea is that in narrow environments where the robot has limited range of motion, like in the case of catacombs, the *path itself* is a salient feature of the region of the environment the robot is traversing. In other words, if the robot passes through the same tunnel

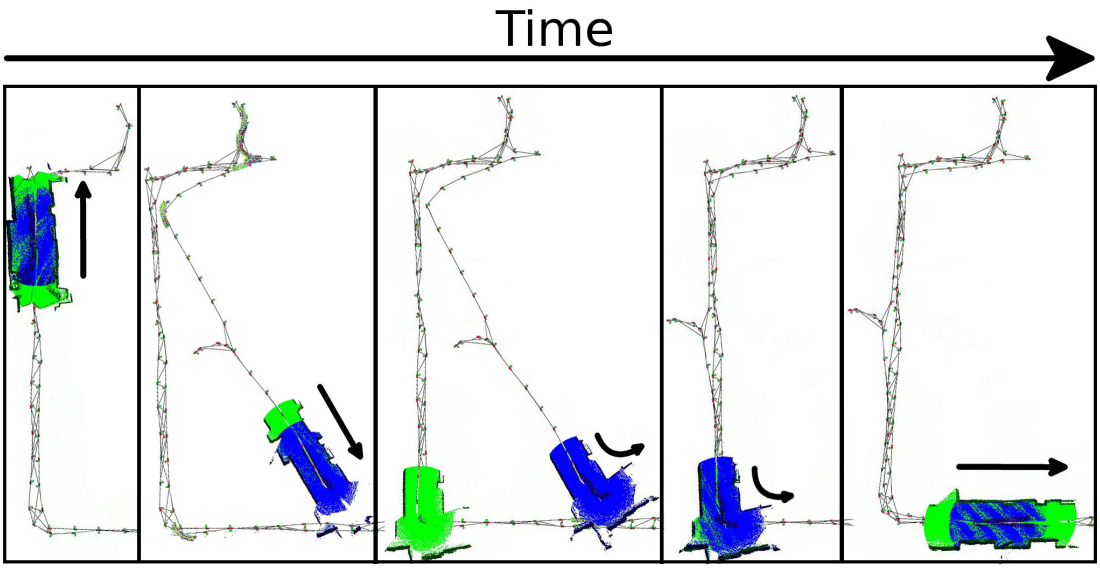


Figure 6.7: Example of loop closures found by our system. Blue point clouds highlight the current position of the robot, while green ones represents accepted loop closures. The small black arrows show the moving direction of the robot. Note how the system is able to recover from a broken situation (2^{nd} picture from left) thanks to the trajectory based loop closing search (3^{rd} and 4^{th} pictures from left).

twice, it is likely it will be constrained to follow very similar trajectories. We can therefore obtain a good initial guess on the similarity between two local maps just by registering their internal path $\mathcal{T}$. More formally, let $\mathcal{P}^{\mathrm{c}}$ be the most recent local map having trajectory $\mathcal{T}^{\mathrm{c}} = \mathbf{T}^{\mathrm{c}}_{1:N}$ consisting of transforms $\mathbf{T}^{\mathrm{c}}_n$, and let $\mathcal{P}^{\mathrm{r}}$ be a local map in the past having trajectory $\mathcal{T}^{\mathrm{r}} = \mathbf{T}^{\mathrm{r}}_{1:M}$ consisting of transforms $\mathbf{T}^{\mathrm{r}}_m$. We want to find a candidate transformation that minimizes the following cost function

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathcal{C}} \left\| toVector\left({\mathbf{T}^{\mathrm{r}}_m}^{-1} \mathbf{T} \mathbf{T}^{\mathrm{c}}_n \right) \right\|_{\mathbf{\Omega_I}} \tag{6.13}$$

where $\mathcal{C}$ is a set of correspondences between the poses of the two trajectories, $\mathbf{\Omega_I}$ is the 6×6 identity matrix, and $toVector(\mathbf{T})$ is a function that maps a transformation matrix in a 6D vector $(\mathbf{t}^T,\ \mathbf{q}^T)^T$ consisting of the translation vector $\mathbf{t} = (t_x,\ t_y,\ t_z)^T$, and the imaginary part $\mathbf{q} = (q_x,\ q_y,\ q_z)^T$ of the unit quaternion associated to the rotation. To generate the initial set of correspondences, we exploit the sequentiality of the trajectory, starting from one single random association.

We repeat this procedure multiple times using different initial guesses, or past local maps, and for each one we solve the problem in Eq. 6.13. In order to reduce the number of initial guesses, we do not consider past local maps that have an internal trajectory which is poorly informative, like in the case of a straight path. A local

map is labeled as informative if the eigenvalue decomposition of the covariance matrix associated to its internal trajectory has at least 2 eigenvalues significantly greater than zero. we proceed then with a geometric validation which consists in registering the point clouds of the two local maps. To this end we use the procedure described in Sec. 6.3.1, and we employ a KD-tree based correspondence search. As initial guess we use the candidate transformation $\mathbf{T}^*$ found while aligning the internal trajectories. If the matching succeeds, we add a constraint between the two local maps to the pose graph and we carry out an optimization step. Fig. 6.7 shows an example of our loop closure system in action. The reader can see how, by exploiting the trajectory followed by the robot, our method also allows to recover from broken situations such as a globally inconsistent map.

6.3.3 Map Optimization

The approach described above does not address the on-line aspects of the problem. To compute the local maps all data should be available at the beginning of the computation. However, a local map is only affected by the measurements in its neighborhoods. Two local maps that are far away will share no information, thus they can be computed independently. In particular, the local maps can be constructed incrementally while the robot moves, and a sparser graph containing only the local maps can be optimized on the fly. This graph is usually one or two orders of magnitude smaller than the input problem, and thus it can be optimized on-line. In our context, since we are optimizing primarily factor graphs consisting of robot poses, we never need to remove a vertex from the graph. This substantially simplifies the bookkeeping of the hierarchical structure, with no major modification of the system. At any point in time, based on the current position of the robot, our system provides a fully optimized estimate of the robot neighborhood, which is done by executing a few rounds of the graph. The system has been implemented within the g2o [16] framework, and it is compatible with its API.

6.3.4 Additional Tools

The inspection and analysis of the state of the SLAM system is more complex, and it requires the analysis of the module on an entire log rather than on the current input. To this end we developed an advanced standalone graphical refiner tool that allows for inspecting the pose graph by interacting with the cloud registration and the loop closure module. In particular it is possible to edit an existing map by adding or removing loop closure constraints between local maps. The tool is typically used at the end of a mission to validate, and eventually correct, the 3D map that the robot has produced online during the exploration. Fig. 6.8 shows two different screenshots of the refining tool before and after a manual addition of a loop closure constraint.

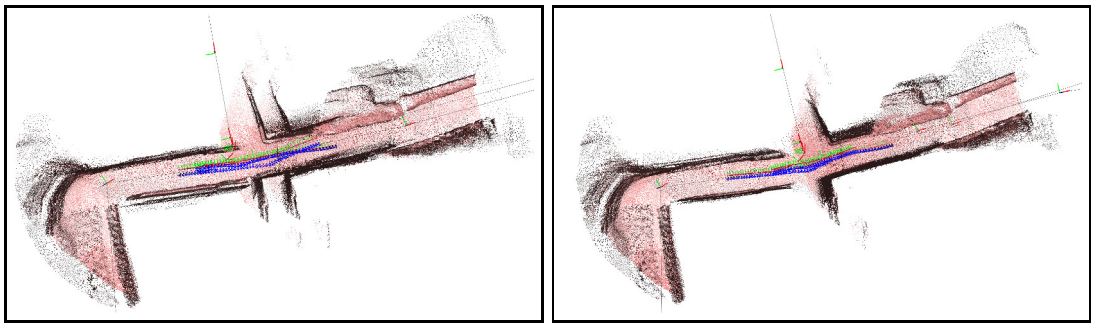


Figure 6.8: The SLAM loop-closure refiner tool developed with our SLAM system. From top to bottom: the map before the refinement, and the map after the manual addition of the loop closure constraints and the optimization.

6.4 Experiments

In this section we provide some quantitative and qualitative results of the performance obtained by our SLAM system in the catacombs of Priscilla in Rome. To the extent of our knowledge, it does not exist any dataset with ground-truth acquired in a catacombs like environment. For this reason, in order to give some quantitative results of our mapping system, a ground-truth dataset has been generated for the catacombs of Priscilla. In particular, in August 2015 an accurate topographic survey has been carried out with a Pentax total station (right picture of Fig. 6.9). During the operative phase we cooperated with archeology experts. Instead of traditional topographic "targets", specific signals with special tags with information recognizable by the robot during its route have been used. These special tags are part of the well known AprilTags developed by Olson in [163]. The AprilTags are a visual fiducial system that can be used to compute accurate 3D position, orientation, and identity of the tags with respect to a camera. The tags are based on a near-optimal lexicographic coding system, and they are similar to the well known QR-codes in the sense that they are 2D bar codes. However, since they are designed to encode small data payloads, they can be detected more robustly and from longer distances. The left picture of Fig. 6.9 shows some examples of tags from the AprilTags system, while the image in the middle illustrates a snapshot of the robot acquiring the dataset in Priscilla using AprilTags.

Based on this topographic data, we computed different measures to quantify the global accuracy of the map generated by our system. More in detail, we used the landmark perceptions gathered through the depth sensors, to register the model constructed by the RGB-D sensors and the one acquired with the total station. This is showed in the left picture of Fig. 6.10. By measuring the distance between the detected landmarks, and the measured ones, we can quantify the absolute error of the mapping process, as illustrated in the right image of Fig. 6.10. On a map of size of more than

Figure 6.9: From left to right: examples of tags from the AprilTags fiducial system, dataset acquisition in Priscilla using the AprilTags fiducial system, and the Pentax total station.

50 meters, the overall mean error distance of a tag computed with our system is ∼1.8 meters, with a standard deviation of ∼0.9 meters. Note that this measure does not fully characterize the system. Indeed a small drift in the estimated local maps might result in globally misaligned map and, as pointed out in [164], this would potentially penalize consistent portions of the map. Additionally, the absence of loops in the Priscilla environment results in an estimate where the positioning drift accumulates. Consider also that the detection of the landmarks in the images is subject to depth errors in the range of 10 centimeters.

To characterize the local accuracy of the map, we used the ground-truth of the landmark positions, and we run an optimization process that "deforms" the robot trajectory to register the ground-truth landmarks. If no deformation occurs, the relative position of adjacent trajectory points would be unchanged after registration. Conversely, a high deformation results in a large "correction" of the relative poses. By measuring the entity of such a deformation between each pair of local maps, we quantify the local accuracy of the map. The outcome of this analysis is reported in Fig. 6.11. The reader should note that these error values are the result of comparisons between local maps of size of approximately 5 meters. The outliers in the results are caused by small local

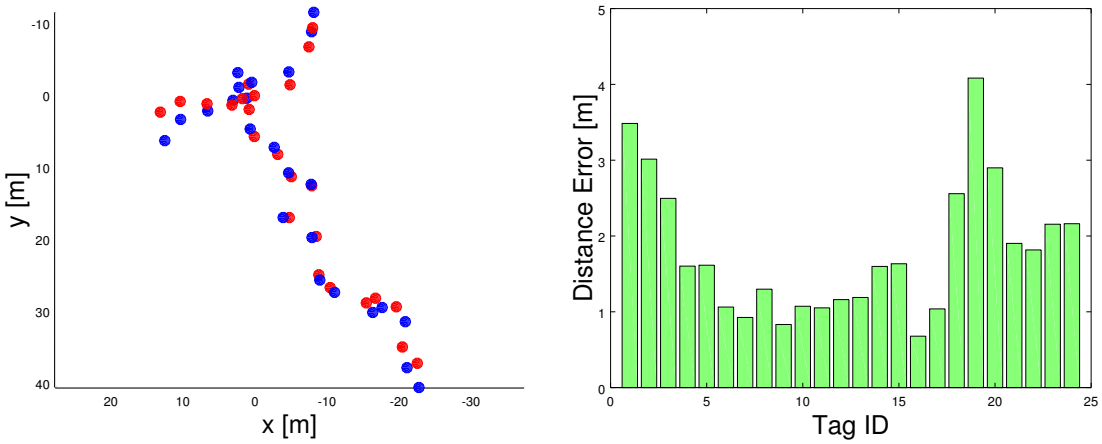


Figure 6.10: Left: top view of the detected landmarks aligned in the same frame. The units are in meters. The red points denote the ground-truth acquired with the total station, while the blue ones indicate the position of landmarks recovered by our on-line mapping system. Right: absolute error of the tags detected by our SLAM system in the overall map.

maps due to sudden broken registrations. The relatively small dimensions of such local maps makes them not enough informative, thus they lead to wrong measurement alignments. More in detail, our system generates an overall mean pose error of 0.44 meters in the translation, and ∼0.071 radians in rotation, between the center point of a local map and its ground-truth value. The standard deviations associated to that means are respectively ∼0.42 meters, and ∼0.22 radians.

Based on these results, experts of archeology stated that the local accuracy of the map is adequate for presentation purposes such as virtual tourism. The topological consistency, that is also ensured by the possibility of manually correcting the final maps through the use of a map refining tool, allows to exploit all data recorded even under sub-optimal conditions. The limited global accuracy is mostly due to sensor noise and the topology of the environment that, being free from loops, does not allow a global redistribution of the error.

To further support the validity of our mapping system we present two additional qualitative results. The first one highlights the robustness of the underlying registration process, that represents one of the fundamental blocks of the entire SLAM approach. More specifically, we compared the incremental alignment of consecutive depth measurements in a corridor of the catacombs of Priscilla in Rome. Fig. 6.12 contains the results of this experiment. While KinFu [8] (right picture) is not able to correctly recover the path followed by the robot when turning, our registration sub-system (left image) succeeds. The second qualitative result can be seen in Fig. 6.13, where the top view of a full map estimate of the catacombs of Priscilla computed by our mapping system is shown.

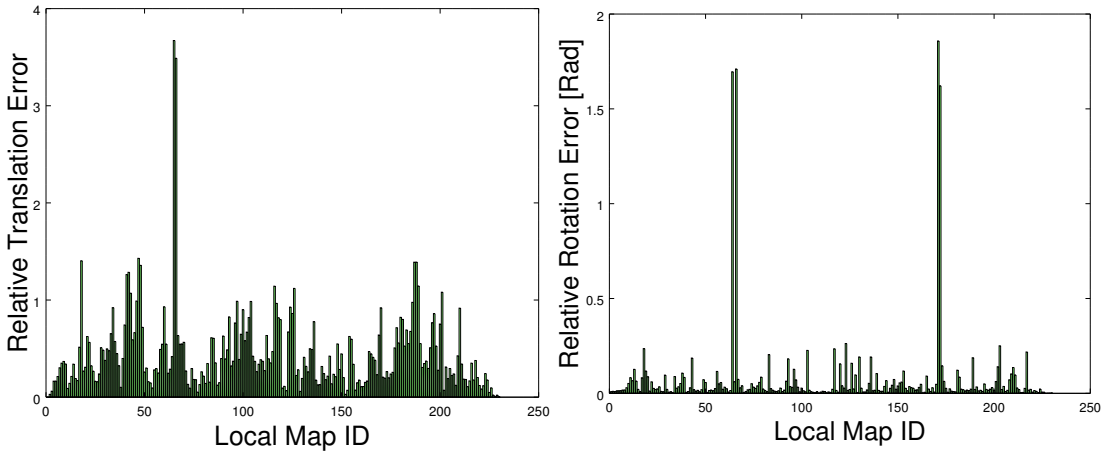


Figure 6.11: Relative translational (left) and rotational (right) error of the trajectory with respect to the ground-truth. The x axis denotes the ID number of a local map, while the y axis denotes the relative error between two temporally subsequent local maps. Units are in meters for the translational error, and in radians for the rotational error.

6.5 Conclusions

In this Chapter we presented in detail a full SLAM system that can be used for mapping hard environments like catacombs. We developed a novel algorithm that exploits the trajectory followed by the robot in the search for loop closures. The results showed by our approach demonstrate to be good, and they are promising for further developments. Our mapping system performed successfully in the context of the European project ROVINA, and it played a fundamental role in obtaining an excellent evaluation from the reviewers.

A possible extension to this framework it to use Voronoi diagrams [165], instead of the robot trajectory. Since Voronoi diagrams are independent by the commands given to the robot, we expect better results even in environments where the robot movements are not constrained.

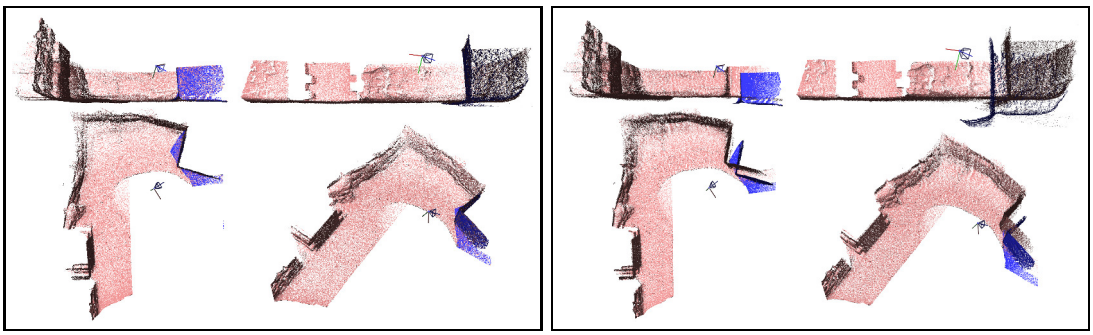


Figure 6.12: This figure shows different views of the reconstruction of a piece of corridor in the catacombs of Priscilla in Rome using our algorithm (left) and KinFu [8] (right). More specifically, for each quadruple of images, from left to right and from top to bottom: frontal view, side view, top view and isometric view.

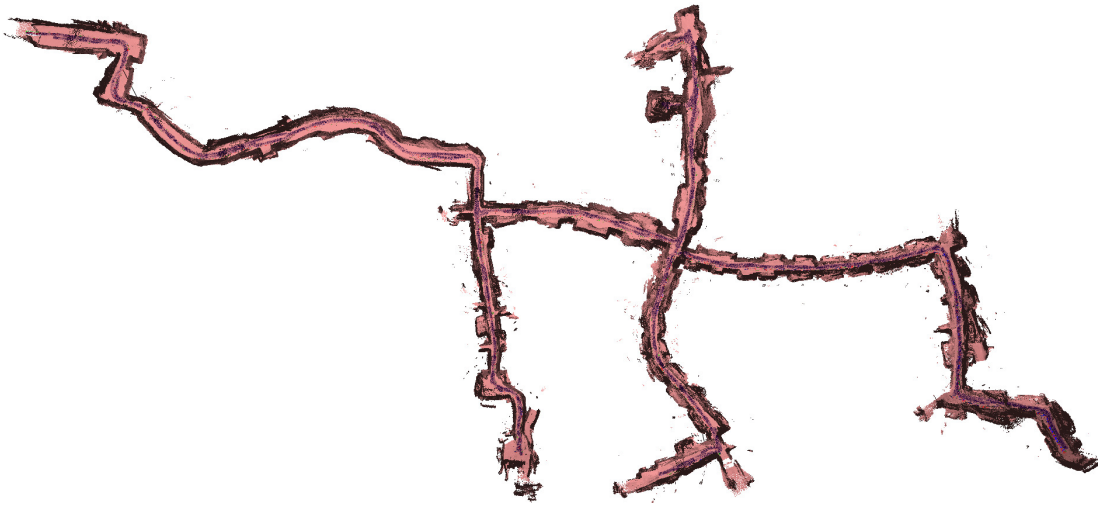


Figure 6.13: Top view of a map generated by our mapping system during a mission in the catacombs of Priscilla.

Chapter 7

FLAT2D: 2D Fast Localization from Approximate Transformation

"His pattern indicates two-dimensional thinking."
— Leonard Nimoy, Star Trek II: The Wrath of Khan, 1982.

Many autonomous vehicles require fast and accurate localization estimates to safely plan and navigate through the environment. Even a 0.5 m lateral position error can be serious for a lane keeping application. Consumer grade GPS is insufficiently accurate to support these needs, which has resulted in the development of algorithms based on data from LIDAR and cameras.

3D LIDAR has proven particularly popular on state-of-the-art autonomous vehicles due to its accuracy and ability to produce fast, 360° Field Of View (FOV) information about the surrounding environment. Additionally, 3D LIDAR sensors are more robust to occlusions than their 2D counterparts. However, the data from these sensors can be challenging to use in localization applications. The dense 3D prior maps necessary to support 3D matching operations take up large amounts of storage space, making it impractical to locally store full 3D maps of large geographic regions. Indeed, it can take hundreds of MBs of storage for even a small urban area. On the other hand, systems cannot tolerate the risks incurred by fetching data on-demand from a remote source.

Assuming the storage challenges have been overcome, alignment and localization in 6 DoFs still present challenges. Traditional 3D alignment algorithms like ICP have poor convergence properties and require good initial registration to produce high quality alignments. Non-uniformity in 3D LIDAR data can exacerbate convergence issues.

2D LIDAR sensors are often used in indoor robotic applications, resulting in prior map representations that are compact and efficient to match against. However, the

planar nature of these sensors makes them susceptible to errors due to occlusion by transient obstacles, or complete failure to observe off-plane hazards. Thus, it is desirable to find a way to use rich, robust 3D LIDAR information as efficiently as we use the existing 2D data.

Unlike a 6 DoFs ICP approach, the system we present in this Chapter determines only three degrees of freedom of the robot: x, y, and θ. However, roll and pitch can be easily extracted from a low-cost accelerometer, and reasonably accurate z estimates (if needed by an application at all) can be encoded in the prior map, since we can safely assume the car stays on the ground.

Our approach is based on 2D scan matching against vertical structure extracted from a 3D point cloud. The verticality constraint ensures crisp, easily matchable scans that produce precise localization results. The structure of the Velodyne VPL-16 and HDL-32E sensors (namely: spinning, vertical fans of laser diodes) make them particularly well suited for the rapid identification of that structure. Switching to a 2D representation significantly reduces map storage needs.

Particle filters are often used for localization systems using prior maps. Our approach represent an alternative based on the Graph-SLAM formulation, allowing us to leverage recovery methods such as max mixtures [166] or switchable constraints [167]. The advantages include immunity from particle depletion, and produce higher precision solutions. In addition to this, we also benefit from the wealth of state-of-the-art 2D methods that already exist for localization and mapping.

7.1 Extracting Structure from 3D Points

In several previous works, 3D LIDAR data is converted into 2D representations for use in localization or navigation. Most systems have need of both of these capabilities, but the type of structure extracted from the 3D point data differs greatly depending on the task at hand. Navigation must identify all hazardous structure, including plants, curbs, and other vehicles. However, the performance of SLAM and localization systems can be negatively affected by an overabundance of structure to match. If "fuzzy," unreliably observable structures such as long grass, a bush, or even a sloped wall are presented to a scan matcher, the variation between scans can result in bad alignments and poor pose estimates.

This section introduces a pair of methods for extracting high-quality 2D structure maps to be used primarily in localization. The first is an improvement over a baseline strategy based on counting the number of points falling into 2D bins. The second exploits structure in Velodyne sensors, classifying structure based on its estimated verticality.

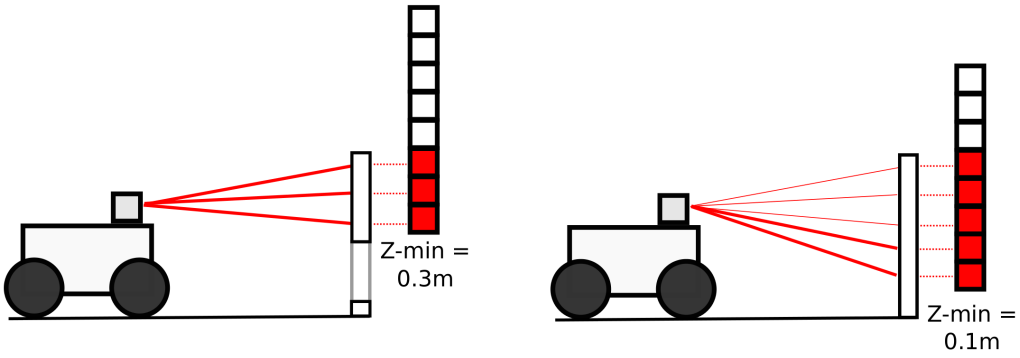


Figure 7.1: An 8-bit example of our compact, bit-based voxel representation. Bits in an integer correspond to bands of z-height. Bits are set to 1 when LIDAR points fall into their band, with the 0^{th} bit corresponding to the band containing the lowest observed height for that XY bin (left picture). If new observations fall below a bin's current z-min, previous measurements are bit-shifted to efficiently make room for the new observations (right picture).

7.1.1 A Baseline Method for 2D Structure Extraction

Many commonly applied 2D structure extraction methods are built around a quantization of the world in XY space [73]. In the simplest case, range returns falling within a specified z-height band relative to the ground are projected down into this grid and marked as obstacles. This works well for hazard estimation, but is very susceptible to noise.

Alternatively, one can partition the world into voxels, only marking structure on a 2D summary map where the number of occupied voxels in a vertical line exceeds the specified threshold. This ensures a certain level of "verticality" (i.e. repeatability) in the detected structure. However, one of the main drawbacks of these methods is their inability to deal with negative obstacles. Negative obstacles may generate no or very distant range returns, causing them to not appear in the map at all. Other issues also arise from quantization errors due to noise affecting the sensor measurements.

7.1.2 Compact Voxel Popcount

We propose a novel voxel representation that is compact, fast, and adaptive to local terrain height. Each XY grid cell stores a floating point minimum z-height and a 64-bit integer. Each bit of this integer corresponds to a fixed-size band of vertical range. Most systems are only concerned with potential obstacles within a relatively limited z-range around the ground; the minimum z-height parameter allows us to focus on a particular region of interest (i.e. 20 cm to 200 cm). When a LIDAR point is received that is lower than the minimum z-height, the XY cell can be updated using a simple bit shift. An 8-bit example can be seen in Fig. 7.1.

In this formulation, the identification of vertical structure can be performed with a popcount operation on the integer: if the number of bits set to 1 exceeds a threshold, the cell passes a verticality test and is marked as structure. Additionally, we may efficiently isolate smaller vertical bands of data with bitwise-AND: for example isolating obstacles the same height as the vehicle.

Our representation may be further improved by working in polar coordinates. Most 3D LIDAR sensors operate by rotating a vertical fan of lasers, resulting in an obvious relationship between the data and a polar coordinate frame. By binning in polar coordinates, we avoid costly projections into Cartesian space. Similarly, quantization error now only occurs in the range dimension, rather than in both X and Y. The Strip Histogram Grid [83] is another instance of a polar binning method.

For sensors without this obvious structure, like nodding planar sensors, this representation has weaknesses. In particular, bins are no longer of a fixed size; instead they grow larger as we move away from the robot. This can further sparsify distance data, eliminating useful features for scan matching.

7.1.3 Slope-based, Multi-Class Structure Detection

We propose a second novel extension to 2D structure representation, namely, to represent multiple classes of structure in one map. In our applications, we find that it is helpful to distinguish between *hazard*, an obstacle to navigation, and *slammable structure*, i.e. vertical structure that we can reliably scan match against. In this way, we may support navigation and localization systems with the same data stream without adversely affecting the performance of either. This extension can be added to the previous popcount methods by setting a separate threshold for each type of structure.

The fan-like structure of Velodyne HDL-32E and VPL-16 sensors lends itself well to the extraction of slope information between points. Due to the shallow angle of observation of the laser diodes in the Velodyne sensors, slope returns on relatively flat ground are not greatly affected by noise in the range measurements. Even small obstacles appear as noticeable jumps in slope, making them easy to detect. In contrast to the previous method, which looks for vertical continuity in a single cell, this method can also detect hazards that span multiple cells, such as steeply sloped ground.

A multi-class representation of structure emerges naturally from this formulation: structure exceeding some minimum slope should be marked as hazardous to navigation, while structure exceeding an additional slope threshold should be upgraded to slammable structure. This is appealing, as hazards may be marked based on the capabilities of the particular system. In our indoor robot system, for example, we find that marking navigation hazards for slopes exceeding 15° and slammable structure at slopes exceeding 80° off the XY plane works well.

We process a single, vertical fan of range returns from the bottom up, conservatively marking obstacles in an occupancy grid at the location of the return nearest to the

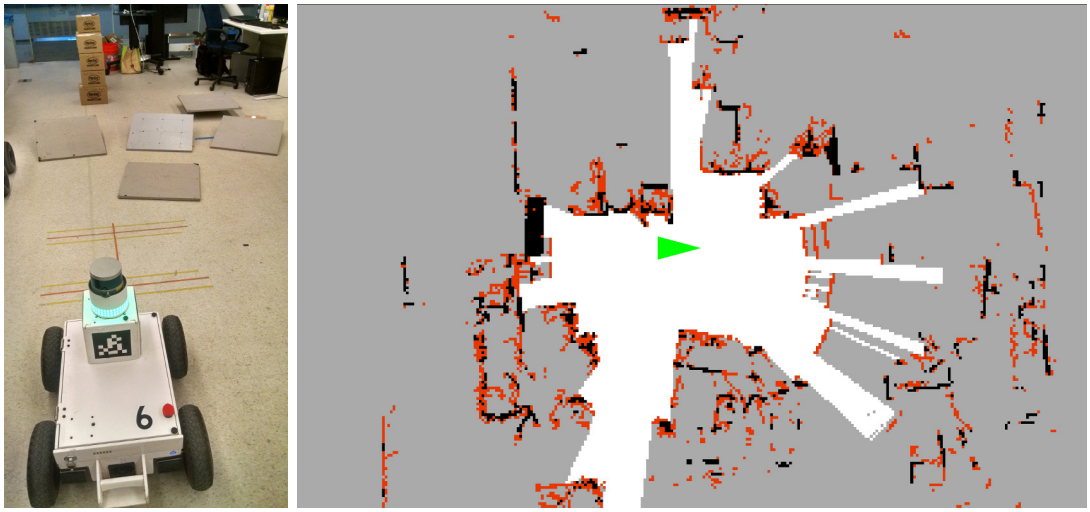


Figure 7.2: An example map (right picture) for a scene (left picture) containing (left to right) a 16°, 14.5°, and 10° ramp. Black denotes scan matching structure, red navigational hazard, light gray unknown, and white observed free space. The slope cutoff for hazards is 15°. Due to noise, the 14.5° ramp is intermittently marked as a hazard.

robot. Additionally, we add to the scan a "dummy return" at $(0,\ 0,\ 0)^T$, the position of the robot, since we assume the robot to be on the ground at all times. Example results of a 2D map constructed from a single rotation of a Velodyne VPL-16 can be seen in Fig. 7.2.

In our system, we find that sufficient performance is achieved by measuring the slope between 2 consecutive vertical measurements. However, particularly in applications where sensor noise is a concern, slopes may be fit across 3 or more points. This can have the effect of smoothing out smaller obstacles, causing them to disappear, but likewise can improve the robustness of "slammable" detection.

Although this method could accumulate structure in a polar coordinate frame, the goal of marking mixed-classes of structure is to support navigation and localization simultaneously based on a unified 2D map. Navigation tasks are most commonly performed in the Cartesian space frames, so we choose to produce our 2D structure maps in Cartesian coordinates, as well. Operating on this unified map saves computation, as it eliminates the need to produce separate representations for each task.

This method still experiences challenges with negative obstacles due to frequent lack of direct observability. In cases where drop offs are observable, the slope-based classification strategy will correctly identify dangerous drop offs, but otherwise will fail to mark a hazard.

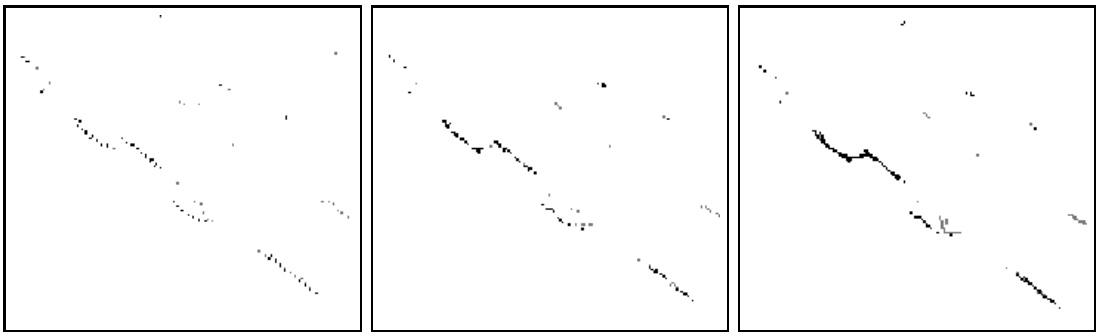


Figure 7.3: From left to right: no fill-in, 25 cm fill-in and 50 cm fill-in. By applying a hole fill-in algorithm during 3D data processing, we densify the resulting 2D map data, facilitating later scan matching. Larger amounts of fill-in greatly increase effective data density, but can negatively affect the system's ability to represent certain fine-detail structures (e.g. a picket fence).

7.1.4 Hole Fill-In

Although Velodyne data is more dense rotationally than vertically (0.38° horizontal vs. 2° vertical spread between observations for a VPL-16 rotating at 20 Hz), at a distance of 15 m, points are already 10 cm apart. These holes in the data present challenges during scan matching, which relies on having sufficient data density to detect overlap in structure. To address this problem, we employ a straightforward technique for filling gaps in the data.

For consecutive vertical slices, returns from each laser diode are considered in turn. If, in both slices, the returns were marked as the same class of structure, we fill-in a line of the same class between the corresponding cells in our 2D map. This allows the scan matcher to make matches at longer ranges. To avoid filling structure where none exists, we only apply fill-in to range returns that are less than a specified distance apart. Example 2D mapping results before and after fill-in can be seen in Fig. 7.3.

7.2 SLAM and Localization

We consider two coordinate systems in which the robot can operate: *local* and *global*. The robot's local coordinates are determined by its open-loop odometry. The robot's global coordinates are the actual position of the robot in the world, e.g. as defined by an existing global map. The goal of localization, then, is to compute a transformation $L2G$ that can convert local into global coordinates.

A robot may have one or more systems capable of generating $L2G$ estimates, including but not limited to GPS and scan matching against a known map. We find it convenient to formulate the challenge of merging these estimates as a equivalent to solving a SLAM problem: each $L2G$ estimator produces noisy observations in which

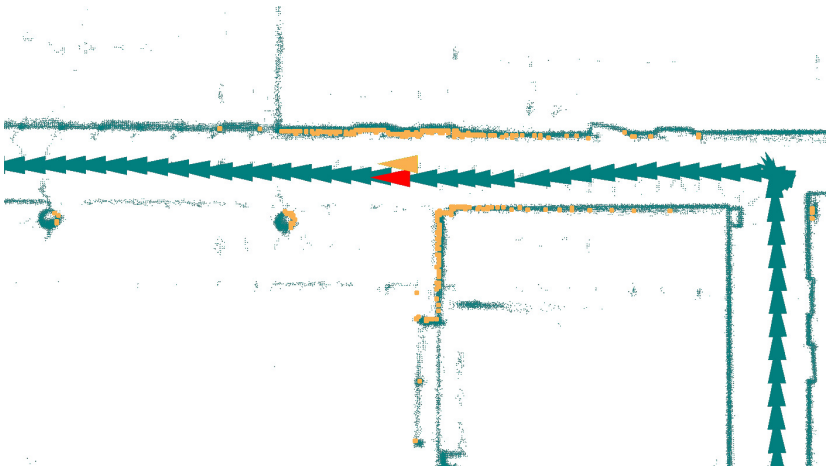


Figure 7.4: A successful scan match in an indoor environment. In this example, we match observations from the current pose (orange) against those from the prior map (teal). The red triangle denotes the historic pose closest to our prior estimate of the robot's current global pose.

we place a certain amount of confidence. Thus, we may find the Maximum Likelihood Estimate (MLE) $L2G$ by minimizing the error among these separate estimates.

7.2.1 SLAM-Graph Formulation

Our SLAM system and our localization formulate Graph-SLAM in distinct ways. The SLAM system creates robot pose nodes at fixed intervals of motion, linking them with relative factors from odometry, camera, and scan matching observations (detailed below). Likewise, global factors provided primary by GPS or human annotation can be added to the graph, though our system only uses these for initial registration of the map to satellite imagery. In conjunction with the previously described 2D structure extraction methods, this allows us to construct globally-consistent maps of the environment.

Our particular system poses localization in terms of factor-graph optimization, as well, where scan match results between the robot and known locations in the map result in factor potentials on the robot's current position. An example instance of such a match can be seen in Fig. 7.4. Additional $L2G$ factors can be added based on GPS observations; these present in our system, but weighted considerably less than scan matching factors. Fig. 7.5 depicts a simple example of a resulting factor graph. This formulation allow the system to initialize the priors based on GPS, switching to more precise scan matching once initialized.

Solving the resulting graph yields the present and historic $L2G$ systems for the robot. One advantage of formulating localization as a graph-solving problem is the ability to incorporate concepts like max mixture edges into solutions. Max mixture

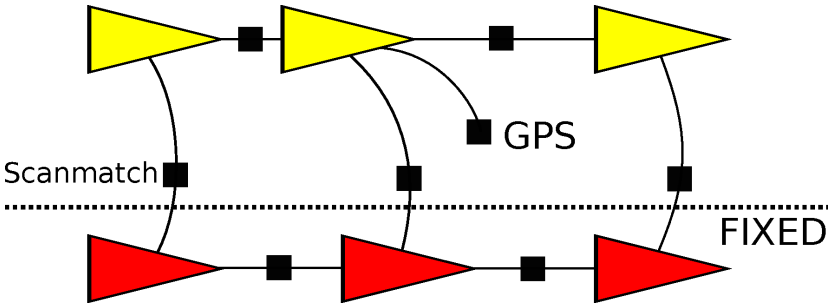


Figure 7.5: A simple example of a factor graph for localization. In addition to standard factors based on GPS or odometry measurements, scan matching factors are added describing transformations back to a known map (in red). During optimization, the portion of the graph corresponding to the known map is held fixed, allowing the localization graph to remain compact.

edges [166] allow the system to reject erroneous observations, for example, due to a bad scan match or bad GPS fix, preventing sudden, catastrophic shifts in the $L2G$ estimate.

Unfortunately, this formulation does not come without a cost. The longer the vehicle is in operation, the more nodes and factors accumulate in the graph, resulting in increasing memory consumption and slower solving times. To prevent excessive accumulation, we fix the maximum number of nodes in the graph. When new observations would result in this threshold being exceeded, the oldest observations are removed, allowing the localization system to operate in a fixed memory footprint and with stable solution times.

7.3 Experiments

We deployed our mapping and localization systems on two platforms: the SmartCarts autonomous golf cart as well as our MAGIC 2.0 mapping robot. Both platforms were equipped with a 2.8 GHz Intel i7-4900MQ and 8 GB of RAM, with code written in C.

7.3.1 Hazard and Structure Detection

We validate our 2D structure classification on two scenes: one to demonstrate that our hazard detection threshold maps well to real-world sloped obstacles, and another to validate that we may classify a wide range of vertical structure, even in the presence of partial occlusions.

To prove that our method is able to correctly distinguish between real-world navigation hazards, we created a scene containing ramps of slope 10, 14.5, and 16°, depicted in the left picture of Fig. 7.2. These angles were chosen to be near the hazard threshold

Figure 7.6: 3D point cloud data allows the robot to observe partially occluded structure (left) and incorporate it into its maps (right). In this scene, the robot observes obstacles of increasing heights, ranging from 10 cm to 1 m. Vertical structure is denoted in black, hazardous terrain in red, unknown in light gray, and observed free space in white.

of 15°. We expect the 10 and 14.5° ramp will be marked as safe, while the 16° ramp will be marked hazardous.

The robot captured the scene from several meters away, viewing the ramps from their lower, ground-level ends. Our terrain classifier successfully marked the 10° ramp as drivable, and the 16° ramp as hazardous, as expected based on the threshold. Due to sensor noise, parts of the 14.5°ramp were marked as hazardous as well. This demonstrates that our hazard detection system is working correctly, but is not entirely robust to noise.

We set up an additional scene to show that our method can preserve information about partially occluded structure, ensuring that useful scan matching features are not removed from the map. The scene consisted of several vertical structures ranging from 0.1 m to 1 m in height, spaced evenly apart in ascending order by height, showed in the left picture of Fig. 7.6. The robot was placed to view this scene such that the taller structures were partially obscured by the shorter structures.

Five bands of structure are clearly visible in front of the robot in the right picture of Fig. 7.6. Only one band of LIDAR strikes fell on the smallest object, a 0.1 m tall box. As a result, its vertical face was not detected and it was only marked as hazardous, non-slammable structure. However, the rest of the structure was correctly identified as sufficiently vertical to be marked as slammable, and accurately placed in the grid map. This shows that our algorithm can correctly identify and denote vertical structure in the environment for use in 2D scan matching.

7.3.2 Map Compactness

One advantage of 2D storage is the compactness of the map representation. Our occupancy grid implementation stores class labels for each cell using a single byte, with each

Figure 7.7: Open-loop odometry during the traversal of the BBB Building at the University of Michigan (left) compared to a trajectory generated using our localization system (right). The robot's open-loop odometry drifts rapidly, but the localization system is able to reliably recover the robot's true pose within the building.

cell corresponding to a 5×5 cm area in the world. A prior map produced by our SLAM system can be seen in Fig. 7.7. The map covers a region of size 74×85 m. For scan matching purposes, we employ a version of the map which preserves information about the structure visible from a series of viewpoints of previous traversal. The resulting scan matching map has a memory footprint of under 4 MB. After applying compression, indeed, this map only consumes 3.6 MB of disk space, compared to the original 830 MB. This is in line with our expectations; maps contain large regions of open space which are easily compressed. Relevant sub-maps encoding particular viewpoints may be decompressed opportunistically when memory usage is a concern.

For comparison, storing the raw 3D range returns corresponding only the selected viewpoints requires 16.3 MB. Converting these returns to 3D points suitable for algorithms such as ICP causes the storage needs to expand to nearly 100 MB.

7.3.3 Indoor Localization

We tested our localization pipeline on our MAGIC 2.0 platform, a system designed for dual indoor/outdoor use. The MAGIC 2.0 robot was equipped with a MEMS grade gyro, wheel encoders, and a Velodyne VPL-16 mounted approximately 0.6 m above

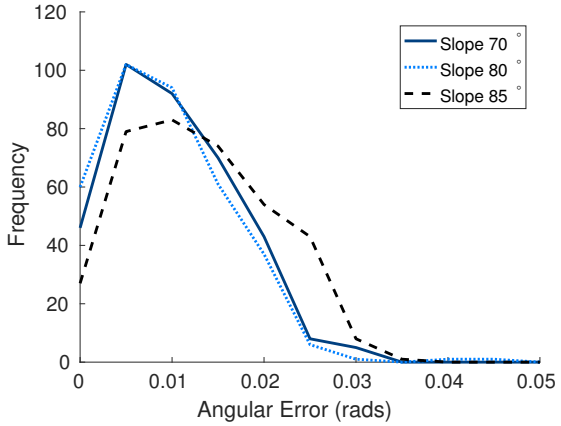


Figure 7.8: Frequency of angular errors in radians for localization based on varying thresholds of vertical structure classification. If set too high, insufficient quantities of structure are extracted to support reliable scan matching. If set too low, too much structure is detected to precisely constrain scan matching results.

the ground. Additionally, we use a Fiber-Optic Gyro (FOG) to acquire ground-truth orientation estimates over the scale of a single pass of the building. We generated a map of the 3^{rd} floor of our building and evaluated the localization system based on a log of a separate, hand-driven traversal.

Results for open loop pose vs. corrected global pose can be seen in Fig. 7.7. Even as the open-loop odometry drifts, the localization system is able to correct for these errors, keeping the robot in the middle of the hallways at all times.

We numerically evaluate the impact of different parts of our system against results from the FOG ground-truth, which measures rotation about θ. By contrasting the FOG θ estimate against the global one produced by localization when employing different occupancy grid generation pipelines, we identify the best combination of features to implement, and the appropriate parameterizations. To quantify the quality of localization, we present distributions of theta error sampled at discrete time steps and binned by steps of 0.005 radians of angular error (roughly equivalent to a 0.25°).

Slope Threshold

First, we examine the impact of various threshold settings for vertical structure detection in the slope based method. We expect overly high and low values to negatively impact scan matching results, resulting in low-quality localization. We tested localization for a traversal of our test environment with the structure threshold set to 70, 80, and 85°. The resulting distributions of observed angular errors can be seen in Fig. 7.8.

We find that the middle threshold of 80° performs best. This is in line with our expectations. Maps constructed based on the 85° threshold mark vertical structure

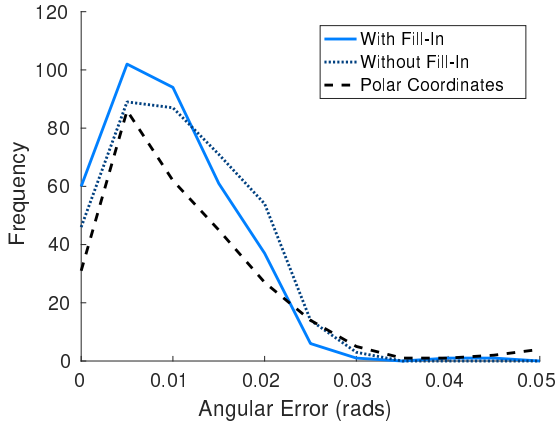


Figure 7.9: Frequency of angular errors in radians for localization based on a polar occupancy map, a Cartesian occupancy grid, and a Cartesian occupancy grid with fill-in applied. The polar map produces more large-scale deviation from truth than it's Cartesian counterpart. However, the results for the Cartesian map are further improved by filling in the regions between vertical slices of observations, aiding the scan matching in achieving good alignments.

less frequently, likely due to sensor noise. The scan matcher, with less or insufficient structure to constrain its matches, produces low quality matches as a result. In contrast, the 70° threshold is too permissive, marking non-vertical structure and sometimes dilating vertical structure. The scan matcher depends on crisp structure to acquire good matches, so this dilation hurts match (and thus localization) quality.

Slope vs. Polar Popcount

Next, we examine the impact of employing a compact, polar-coordinate-based popcount method vs. the slope method with and without fill-in. We expect fill-in to boost performance, as distant observations can be more effectively matched by the scan matcher. It is expected that the polar and Cartesian methods will otherwise perform similarly.

Bin size parameters were tuned to offer the best-case performances for each method. For the slope method, grid resolution was set to 5 cm, and the slammable threshold was set to 80°. Fill-in, when used, was performed for points within 25 cm of each other. The polar popcount method was set to have bins 0.01 rads (0.5°) wide in θ, and 5 cm in range. Structure was marked in bins with at least 2 bits marked. The resulting distribution of observed angular errors can be seen in Fig. 7.9.

Fill-in has the expected impact on the slope-based method, shifting the distribution of error closer to 0. Unexpectedly, the polar popcount performs noticeably worse than both slope methods, exhibiting a large tail of errors in excess of 0.05 rad (2.8°). We hypothesize that this is due to the bin size in the polar frame becoming more spread apart at large distances. The natural spread in bins negatively impacts scan matching,

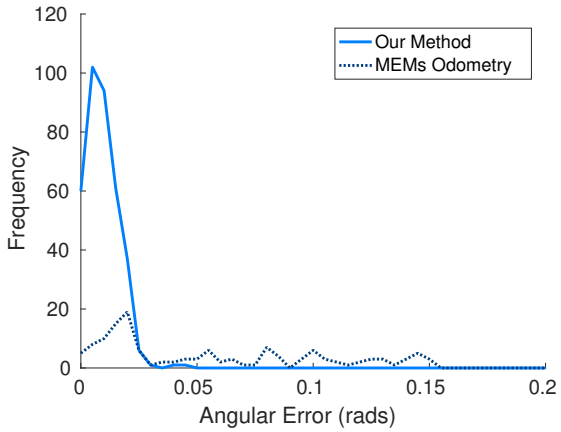


Figure 7.10: Frequency of angular errors in radians for localization for open-loop odometry compared to our final localization system. The orientation estimate provided by a MEMs-grade gyro drifts over time, resulting in divergence from ground-truth of up to 0.155 rads (9°). In contrast, our corrected pose estimates are within 0.02 rads (1.1°) of truth 87% of the time.

as long-range observations are smeared across wider and wider areas. This manifests as the larger angular errors seen in the results.

Slope Method vs. Open-Loop Odometry

Finally, we present a comparison between our best-performing method from the previous analysis, and open-loop MEMs odometry. Odometry-based estimates are initially aligned with the global coordinate frame, but then allowed to drift naturally. The results can be seen in Fig. 7.10.

The θ estimates produced based on the robot's MEMs grade IMU drift significantly over the 10 minutes traversal of the building, accumulating a maximum error of nearly 0.155 rads (9°). Conversely, 2D scan matching based on slope-based structure maps is able to localize the robot within 0.02 rads (1.1°) of ground-truth 87% of the time, 0.025 rads (1.4°) 97% of the time, and in the worst case, deviates from ground truth by just under 0.045 rads (2.5°).

7.3.4 Computational Costs

We present timing data for the indoor dataset to evaluate how frequently our localization system is able to update pose estimates. Our method can be broken up into three distinct stages: map building, scan matching, and graph solving.

To ensure a recent, 360° view of the area surrounding the robot, we produce 2D scan matching maps after every revolution of the Velodyne. At the maximum rotation rate allowed by the hardware, a revolution is completed every 50 ms. We are able to

Figure 7.11: A trajectory based on open-loop odometry on a FOG-equipped golf cart (blue) vs. our localized trajectory (green). Structure from the 2D prior map is shown in yellow. Noisy open-loop odometry causes errors in pose estimation, but 2D scan-matching results are sufficiently accurate for our localization system to correct the errors, keeping the vehicle safely within its lane.

process this data incrementally in real time, resulting in less than 1 ms of delay between the completion of a revolution and publication of a map.

The resulting maps are consumed by a scan matching module, which in turn publishes its best estimate of the robot's current pose in the global map. Scan matching time dominates the cost of our algorithm and varies greatly based on the quality of the initial pose estimate. The median time to acquire the best match is 140 ms, but times as low as 100 ms, and as high as 180 ms, are common. In the rare event that no good match is found, the search can take as long as 400 ms.

Finally, the scan matching results are incorporated into a localization graph which solves for the current and historical robot poses. The graph is mostly sparse, resulting in a linear growth in solve time. We find that solve time increases at a rate of roughly 1 ms per 100 historical poses in the graph. By limiting the size of the graph, we are able to keep solve times under 5 ms. Based on this timing information, our system was able to produce new L2G estimates at a rate of 5 Hz.

7.3.5 Outdoor Localization

We also tested our localization pipeline outdoors on our SmartCarts autonomous golf cart platform. Our vehicle was equipped with a FOG for yaw estimates, an encoder on the left-rear wheel, and a Velodyne HDL-32E for scan matching. We registered hand-annotated lanes against a SLAM map of the environment gathered prior to testing. The robot was then driven to a start location and tasked to drive autonomously between several waypoints in the road network. A comparison of open-loop vs. localization corrected pose can be seen for a portion of the trajectory in Fig. 7.11.

Our open loop odometry slowly accumulates error, even with high-grade sensors like a FOG. Matching scans against a 2D structure map corrects for this error, though, keeping the SmartCart within its lanes throughout the test.

We evaluated the speed with which we are able to process VPL-16 scan data using our slope based approach with 25 cm fill-in. We found that our method can generate 50×50 m grid maps in real time when the Velodyne is rotating at 20 Hz.

For some applications such as autonomous exploration, it can be helpful to explicitly mark free space as explicitly observed out to some range. Using a simple line-of-sight, raycasting approach to mark this space in the grid map out to 5 m adds an additional 6 ms of post-processing per scan. Cropping the maps to remove regions without detail takes another 3 ms.

7.4 Conclusions

Recent advances in localization for autonomous vehicles rely on matching 3D structure in the environment, providing robustness to the effects of weather and road maintenance. However, the resulting maps are computationally costly to match against and have large storage requirements.

In this Chapter, we presented a localization system harnessing the rich structural data of 3D LIDAR sensors, but with the computational and storage efficiency of 2D approaches. We introduced two methods for extracting 2D structural information from 3D LIDAR in support of 2D scan matching, as well as a novel localization method based on graph-SLAM algorithms. We quantified the performance of these methods in a real-world system, evaluating predicted orientation estimates against high-quality estimates produced by a FOG. Our system consistently produces accurate position estimates, even in the presence of partial occlusions and dynamic obstacles.

Chapter 8

3D Lines/Planes Landmark-Based SLAM for Autonomous Vehicles

"Google is working on self-driving cars,
and they seem to work.
People are so bad at driving cars
that computers don't have to be that good
to be much better."
— Marc Andreessen, The New York Times Magazine, 2011

A core challenge in many autonomous vehicle applications is localization: how can a vehicle reliably estimate its position with respect to a prior map, so that the semantic information in that map (e.g. lane information) can be used to support driving? A good localization system should work in a wide range of sensing conditions, produce accurate results, be robust to false positives, and require modest amounts of information to describe landmarks.

GPS, RGB cameras and 2D lasers are often used to solve a wide range of common tasks that a mobile robot needs to face. However, for large-scale outdoor scenarios like in the case of autonomous driving vehicles, these sensors are insufficient. For example, GPS devices have resolutions too coarse to be used for localization. Moreover, they can lose the signal in environments like tunnels or underground parking. 2D lasers can easily go blind due to the very thin region of space they analyze. Indeed, there is high probability for most of the beams to not return back. RGB cameras, instead, suffer of common external noise like changes in the weather conditions. Also, they can be totally useless in absence of sunlight. For all these reasons long range sensors, like 3D LIDAR lasers, are currently widely employed to solve complex tasks as Simultaneous Localization And Mapping [17][168] or obstacle avoidance [84].

Common LIDAR laser sensors generates large-scale point clouds acquiring data on all directions. Unfortunately, such point clouds tend to be very sparse. Standard techniques like point cloud registration [26], that are normally used for localization and mapping, fail. For this reason, to gather a sufficient amount of information, it is often necessary to equip the robot with more than one LIDAR. On the other hand, this kind of sensors are also very expensive. To make mobile robots like autonomous cars more accessible, it is necessary to find ways to work with a minimal set of these sensors. Thus, methods operating on sparse data are critical.

When dealing with sparse data, it is natural to fall back on the usage of features. Instead of relying on the full point cloud, only few meaningful pieces of information extracted from the data are used. A good feature extraction method should be able to find stable and robust features. This means that, given a set of consecutive input point clouds, the same feature must be detected in most of the frames, even in presence of external noise. However, due to the sparsity of the input, this is a challenging task. Common feature extraction methods, that are mainly designed to work with dense point clouds, fail in this setting.

In this Chapter, we present a method for fast and robust extraction of general purpose 3D features, from a single sparse point cloud. In our case the features are 3D lines and 3D planes. Our approach attempts to fit such feature models on significant objects extracted from the input data. An efficient isolation of region of interest in the point cloud, and the adaptive surface normals computation, let our method to compute stable and robust 3D features.

We validate our algorithm by showing the results of a 3D landmark SLAM system whose input are the output features computed by our approach. We also compare such results against the same SLAM system, but fed using the NARF [5] state-of-the-art method for key-point detection. The dataset used in the experiments has been acquired by an autonomous car equipping a single Velodyne HDL-32E LIDAR sensor, in the MCity test facility of the University of Michigan. Additionally, we describe the techniques we used to make the system fast enough for real-time operation.

8.1 Robust 3D Feature Extraction

In this section we describe in detail the complete processing pipeline of our algorithm. After defining the problem of 3D feature extraction in Sec. 8.1.1, we describe the main blocks composing the computation flow of our method (see Fig. 8.1). In particular, in Sec. 8.1.2 we show how to remove non-vertical regions from the input point cloud (i.e. the ground); successively, we explain in Sec. 8.1.3 how we compute the surface normals of the points; lastly, Sec. 8.1.4 describes our point cloud segmentation, and feature extraction procedures.

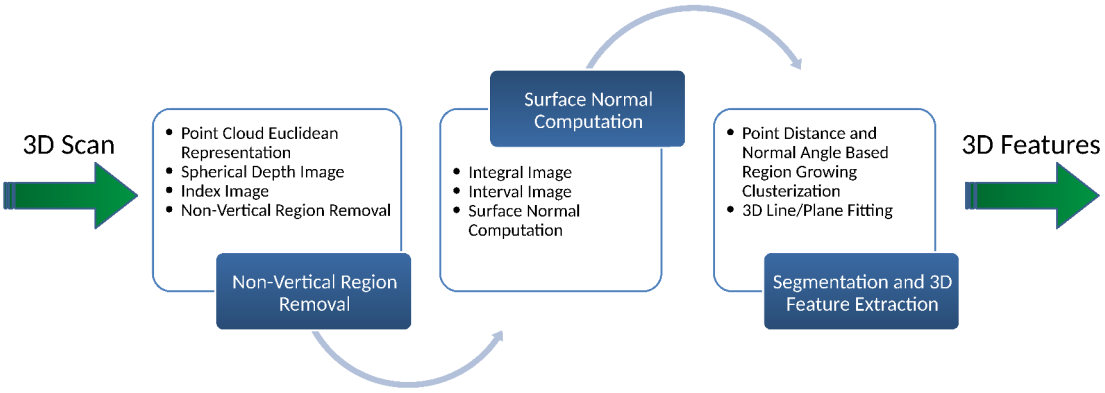


Figure 8.1: The processing pipeline of our 3D feature extraction method.

8.1.1 Problem Formulation

Generally speaking, a good feature should exhibit properties such as:

- sensitivity: there should not be cases where no feature is detected for long periods of time;
- repeatability: the same feature is detected over a wide range of observation conditions;
- robustness: external noise like poor illumination or changes in weather conditions should not compromise the detection of the feature;
- distance measure: it should be possible to define and compute a distance between features. Such a distance must be low or high depending if the two compared features are respectively very similar, or very different.

When dealing with outdoor mobile robots, a significant number of static objects in the surrounding environment can be approximated by geometric entities like 3D lines (lamp posts, trunks, signals) or 3D planes (buildings, walls). The problem we address is the robust detection of 3D line and plane features from sparse point clouds.

Note that, the geometric features selected satisfy all the fundamental properties listed before. The sensitivity and repeatability are assured by the fact that, besides special cases like deserts, most of the environments are always surrounded by objects that can be approximated by lines or planes. Common external noise sources for outdoor mobile robots can be rooted, for example, to changing of illumination or weather conditions. Since the data input of our method is a point cloud, by construction it is mostly insensitive to these kind of noises. Also, being lines and planes well known geometric entities, we can find functions to compute distance values between them.

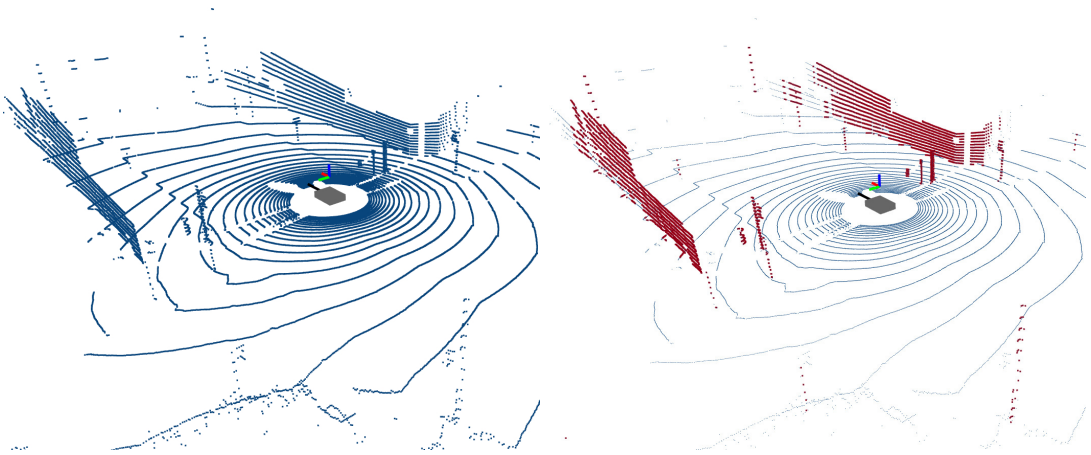


Figure 8.2: Example of our non-vertical region removal approach. The left image shows an input sparse point cloud. The right picture depicts the same point cloud with vertical points highlighted in red. Most of the points belonging to non-vertical regions are successfully detected and removed.

8.1.2 Non-Vertical Region Removal

Many of the objects we want to approximate with lines and planes share a common characteristic: they have a dominant vertical component. Likewise, non-vertical areas (e.g. the ground) tend to not be good features. We exploit this characteristic to make the feature computation more efficient both in terms of CPU time usage, and false positive reduction. The idea is to prune all the points in the input cloud that do not expand along the gravity vector direction. In this way we avoid processing points that would not generate a good feature.

In the case of Velodyne data, it is convenient to represent each point by using the polar coordinates: a triplet composed of azimuth θ, elevation ϕ and the range d. As explained in Sec. 3.1.1, both azimuth and elevation are subject to quantization and we can see a 3D laser scan as a range image. We will refer to this kind of image as *spherical depth image*.

Due to the amount of data our method needs to use, an efficient representation is mandatory. We describe here a simple procedure that minimizes the memory movements without manipulating the cloud. We assume the point clouds are memorized in unordered arrays. To avoid moving large amount of data we use the concept of *index image*. An index image relates the Cartesian points of a cloud with the pixels of its spherical depth image. Given the projection model $\pi(\mathbb{R}^3) \rightarrow \mathbb{R}^3$, defined by the Eq. 3.11, an index image $\mathcal{I}$ is an image where the pixel $\mathcal{I}_{u,v} = i$ contains the index of the point $\mathbf{p}_i$ in the array such that $\pi(\mathbf{p}_i) \rightarrow (u,\ v,\ d)^T$.

Each time the sensor generates a new point cloud, we compute its Cartesian representation $\mathcal{P}$, the associated spherical depth image $\mathcal{D}^{\mathcal{P}}$ by using the projection function

Input: $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$

Output: $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$ pruned from flat regions

foreach *column* u *in* $\mathcal{D}^{\mathcal{P}}$ **do**
  foreach *row* v *in* $\mathcal{D}^{\mathcal{P}}$ **do**
    if $\mathcal{D}^{\mathcal{P}}_{u,v} \neq 0$ && $\mathbf{p}_{u,v}$ *not labeled vertical* **then**
      $n \leftarrow 0$
      $p \leftarrow \mathbf{p}_{u,v}$
      foreach *row* w *in* $\mathcal{D}^{\mathcal{P}}$ *greater than* v **do**
        if $\|\mathbf{p}^{\perp}_{u,v} - \mathbf{p}^{\perp}_{uw}\| < \epsilon_r$ **then**
          $n \leftarrow n + 1$
          $p \leftarrow p \cup \mathbf{p}_{uw}$
        end
      end
      if $n > \epsilon_t$ **then**
        label all points in p as vertical
      else
        delete $\mathbf{p}_{u,v}$ from $\mathcal{P}$
        $\mathcal{D}^{\mathcal{P}}_{u,v} = 0$
        $\mathcal{I}^{\mathcal{P}}_{u,v} = -1$
      end
    end
  end
end

Algorithm 7: Non-vertical region removal

π, and the index image $\mathcal{I}^{\mathcal{P}}$.

We perform the non-vertical region pruning directly in the image plane. Let $\mathbf{p}_{u,v}$ be the 3D point associated to the pixel $\mathcal{D}^{\mathcal{P}}_{u,v}$, and $\mathbf{p}^{\perp}_{u,v}$ be its 2D projection on the ground. Given a column u in $\mathcal{D}^{\mathcal{P}}$, we iterate through all its rows. When we find a valid pixel that is still not labeled as "vertical", we count the number of remaining points $\mathbf{p}_{uw}$ with $w > v$ whose projection $\mathbf{p}^{\perp}_{uw}$ is within a radius ϵ_r from $\mathbf{p}^{\perp}_{u,v}$. If the number of points satisfying this condition is greater than a given threshold ϵ_t, we label all of them as vertical. In the other case, we remove $\mathbf{p}_{u,v}$ from the input point cloud $\mathcal{P}$, and we set $\mathcal{D}^{\mathcal{P}}_{u,v} = 0$ and $\mathcal{I}^{\mathcal{P}}_{u,v} = -1$. We proceed then to analyze the next rows until the last one is reached. The whole procedure is repeated for each column of $\mathcal{D}^{\mathcal{P}}$. At the end of this operation $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$ will contain only valid values for the points labeled as vertical. The idea behind this approach is that, if a group of points creates a vertical structure, then their associated pixels in the spherical depth image $\mathcal{D}^{\mathcal{P}}$ must

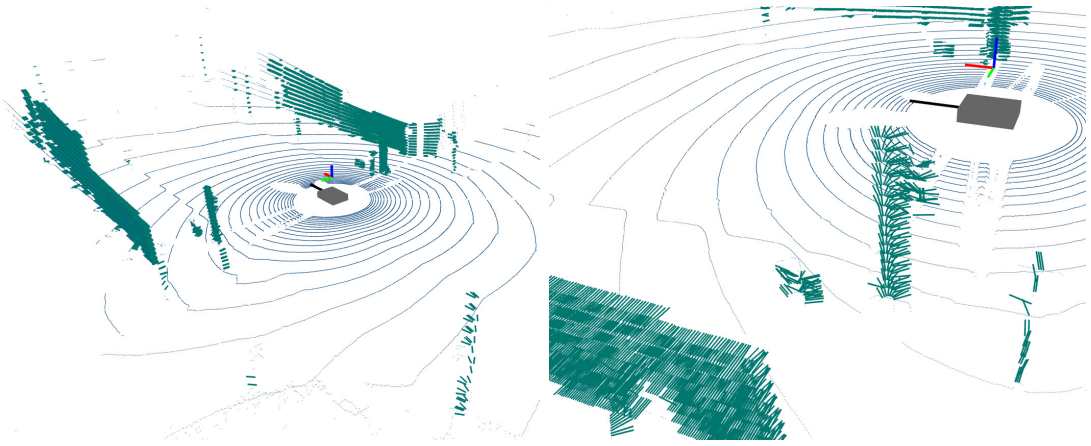


Figure 8.3: Example of our surface normal computation process. The right image is a magnification of the one on the left. The surface normals are drawn as dark green lines. Our adaptive neighborhood selection allows to compute accurate normals also on thin objects like posts.

lie on the same column. This comes from the fact that $\mathcal{D}^{\mathcal{P}}$ is built as a quantization of polar coordinates. Note also that the discretized nature of this method allows to detect structures that are not perfectly vertical. Algorithm 7 describes the non-vertical region removal, while Fig. 8.2 shows respectively its typical input (left) and outcome (right).

8.1.3 Surface Normal Computation

A point measurement gathered by a range sensor is a sample from a piece-wise continuous surface. We extend the spatial information encoded in a point $\mathbf{p}_i$, by locally characterizing the surrounding surface with its normal $\mathbf{n}_i$. The normal $\mathbf{n}_i$ is computed by evaluating the covariance matrix of the Gaussian distribution $\mathcal{N}_i^{\mathrm{s}}(\mu_i^{\mathrm{s}},\ \mathbf{\Sigma}_i^{\mathrm{s}})$ of all the points that lie in a certain neighborhood of the query point $\mathbf{p}_i$. The normal is taken as the eigenvector associated with the smallest eigenvalue and, if needed, flipped towards the observer's point of view. More formally, for each point $\mathbf{p}_i$ we compute the mean μ_i^{s} and the covariance $\mathbf{\Sigma}_i^{\mathrm{s}}$ of the Gaussian distribution as described in Eq. 3.35

A key issue is determining the size of the region over which a normal is computed. A common technique consists in taking all the points lying within a sphere of radius r, centered on the query point $\mathbf{p}_i$. Unfortunately, such heuristic would require CPU expensive queries on a KD-Tree to determine the points inside the sphere.

To speed up the calculation we exploit the matrix structure of the spherical depth image $\mathcal{D}^{\mathcal{P}}$. In particular we use an approach based on integral images described in [6] (see Appendix D). The advantage of this method is that, once the integral image is computed, we can evaluate the quantities listed in Eq. 3.35 in constant time.

We begin by considering all the points within a rectangular region around the pixel

of $\mathcal{D}^{\mathcal{P}}$ associated to $\mathbf{p}_i$. This heuristic can be seen as an approximation of the sphere of radius r centered in $\mathbf{p}_i$. We then adaptively reduce this set to eliminate far-away points, which improves the normal estimates. As an example, suppose we want to compute the normal of a point lying on a lamp post that is located near the wall of a building. Suppose also that we wrongly choose an interval of pixels for the normal computation that includes part of that wall. These outliers, that most likely will be much more than the points on the lamp post, will affect the normal direction generating unwanted artifacts.

We solve this problem by computing adaptive intervals for each pixel in $\mathcal{D}^{\mathcal{P}}$. To this end we construct a new image $\mathcal{R}^{\mathcal{P}}$, of the same size of $\mathcal{D}^{\mathcal{P}}$, where each element contains the exact rectangular boundaries to be used to compute the normal. We will refer to $\mathcal{R}^{\mathcal{P}}$ with the term *interval image*. Given a pixel in $\mathcal{D}^{\mathcal{P}}$, the edges of the interval are calculated by incrementally looking at its neighborhood in all four directions: left, right, up and down. If the depth discontinuity of the pixels on a given direction is smaller than a certain threshold, we expand the boundaries of the region along that line. Otherwise, we analyze the remaining directions until no more expansion is possible.

Once we have the parameters $(\mathbf{\Sigma}_i^{\mathrm{s}},\ \mu_i^{\mathrm{s}})$ of the Gaussian $\mathcal{N}_i^{\mathrm{s}}$, we compute its eigenvalue decomposition as follows

$$\mathbf{\Sigma}_i^{\mathrm{s}} = \mathbf{R} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \mathbf{R}^T \tag{8.1}$$

Here $\lambda_{1:3}^s$ are the eigenvalues of $\mathbf{\Sigma}_i^{\mathrm{s}}$ in ascending order, and $\mathbf{R}$ is the matrix of eigenvectors that represent the axes of the ellipsoid approximating the point distribution.

At the end of this procedure, each point $\mathbf{p}_i$ in $\mathcal{P}$ is labeled with its surface normal $\mathbf{n}_i$. Fig. 8.3 shows an output example of the surface normal computation process.

8.1.4 Segmentation and 3D Feature Extraction

The last block of our processing pipeline has the goal of segmenting the points in the cloud and, subsequently, to extract the 3D features. The general idea is to first compute clusters, or segments, belonging to well defined objects in the scene (i.e. walls, traffic lights, signals, lamp posts). Then, for each cluster, we perform 3D line/plane data fitting. We determine if a cluster is well approximated with a line or a plane feature by computing a measure of the quality of the fitting.

We use a region growing based clusterization algorithm to compute the segmentation. Again, this is performed directly on the image plane. This kind of method incrementally checks the neighboring region of a point seed. Based on some similarity criteria, it determines if the point neighbors should be added to the current cluster or not. When no more points can be added, a new seed and a new cluster are generated. The process is iterated until all the points are segmented. Our criteria to discriminate

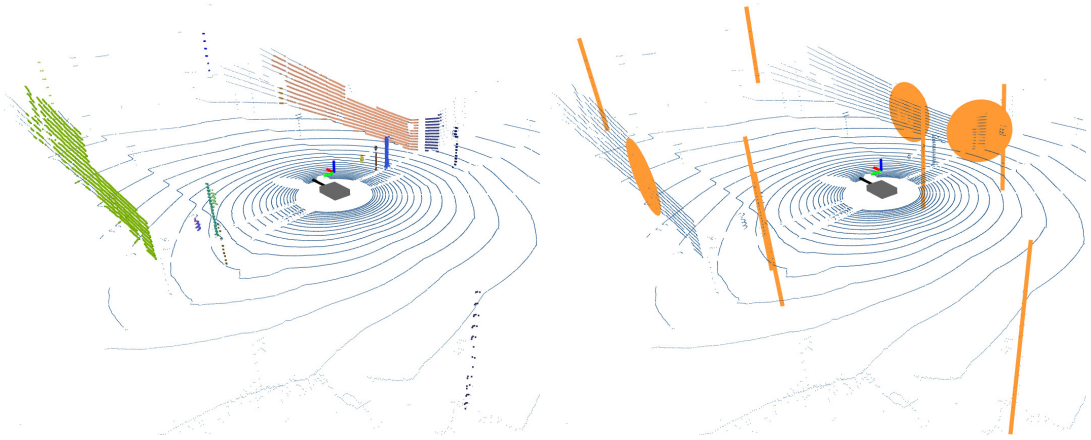


Figure 8.4: Left: example of our segmentation process. Each cluster is drawn with a different color. The ground does not belong to any cluster and is shown only for clarity purposes. Most of the objects of interest for 3D feature extraction are correctly clusterized. Right: example of 3D features extracted by our method from a sparse point cloud. 3D lines and 3D planes (circles) are shown in orange. Most of the walls and posts are correctly detected.

when to add a neighboring point to the current cluster is based both on the point distance, and the angle difference of the normals.

In standard region growing approaches the initial seed is fixed, and all the neighboring points are compared with respect to it. Since our similarity measure is based also on the distance, keeping the seed point fixed could generate multiple clusters actually belonging to the same object. To avoid this, in our algorithm the seed is not fixed. Instead, it changes as we move away from the initial one. In addition to this, we prune all the clusters containing a number of points smaller than a certain threshold ϵ_c. This is necessary to remove all segments too small to be a representation of an object of interest in the surrounding environment.

More formally, a neighboring point $\mathbf{p}_i$ of a point seed $\mathbf{p}_s$ is added to the current cluster if one of the following holds:

- the distance between $\mathbf{p}_i$ and $\mathbf{p}_s$ is lower than a threshold:

$$\|\mathbf{p}_s - \mathbf{p}_i\| < \epsilon_d \tag{8.2}$$

- the angle between the normals $\mathbf{n}_i$ and $\mathbf{n}_s$ is lower than a threshold:

$$\mathbf{n}_s \cdot \mathbf{n}_i > \epsilon_n \tag{8.3}$$

Algorithm 8 describes the pseudo-code that performs the region growing segmentation, while the left picture of Fig. 8.4 shows an output example of our segmentation method.

Data: $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$
Result: a set of clusters $\mathcal{C}$
$\mathcal{C} \leftarrow \emptyset$
for *each row* v *in* $\mathcal{D}^{\mathcal{P}}$ **do**
  for *each column* u *in* $\mathcal{D}^{\mathcal{P}}$ **do**
    if $\mathcal{D}^{\mathcal{P}}(u,\ v) \neq 0$ && $\mathbf{p}^{u,v}$ *not in a cluster* **then**
      $c \leftarrow \emptyset$
      $pool \leftarrow \langle u,\ v,\ \mathbf{p}^{u,v} \rangle$
      while *pool is not empty* **do**
        $\langle y,\ w,\ \mathbf{p}^{yw} \rangle \leftarrow \text{head}(pool)$
        $c \leftarrow c \cup \mathbf{p}^{yw}$
        $pool \leftarrow pool \setminus \langle y,\ w,\ \mathbf{p}^{yw} \rangle$
        for *each* $\mathbf{p}^{ij}$ *neighbors of* $\mathcal{D}^{\mathcal{P}}(y,\ w)$ *that is not in a cluster* **do**
          if $\|\mathbf{p}^{yw} - \mathbf{p}^{ij}\| < \epsilon_d$ && $\mathbf{n}^{yw} \cdot \mathbf{n}^{ij} > \epsilon_n$ **then**
            $pool \leftarrow pool \cup \langle i,\ j,\ \mathbf{p}^{ij} \rangle$
          end
        end
      end
      if $|c| > \epsilon_c$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup c$
      end
    end
  end
end

Algorithm 8: Point distance and normal angle based segmentation.

Once all the clusters are computed, we check which of them can be approximated by a line or a plane. We perform this by fitting the segments with a 3D line or a 3D plane model. Given a cluster c_i, we compute the covariance matrix of the Gaussian distribution $\mathcal{N}_i^{\mathrm{c}}(\mu_i^{\mathrm{c}},\ \mathbf{\Sigma}_i^{\mathrm{c}})$ of all the points in c_i. Let $\lambda_{1:3}^c$ be the eigenvalues of $\mathbf{\Sigma}_i^{\mathrm{c}}$ in ascending order, and let $\mathbf{u}_{1:3}^c$ be the associated eigenvectors. The fitting line $\mathcal{L}_i$, and the fitting plane π_i, are extracted as follows:

Line $\mathcal{L}_i$ is the line with origin μ_i^c, and direction $\mathbf{u}_3^c$;

Plane π_i is the plane with origin μ_i^c, and normal $\mathbf{u}_2^c \times \mathbf{u}_3^c$.

The reader might notice that the eigenvectors $\mathbf{u}_{1:3}^c$ give an indirect measure of the fitting quality. Indeed, in the case of a line where the points change mainly along one direction, we will have two eigenvalues (λ_1^c and λ_2^c) substantially smaller with respect to the third one (λ_3^c). For a plane, instead, only one eigenvalue (λ_1^c) will be significantly

smaller then the others (λ_2^c and λ_3^c). Consider also that, if the feature model $\mathcal{F}$ (in this case a line or a plane) approximates well the cluster, than the residual error e must be small

$$e = \frac{1}{|c_i|} \sum_{\mathbf{p}_j \in c_i} d(\mathcal{F}_i,\ \mathbf{p}_j) \tag{8.4}$$

where $|c_i|$ is the number of points in the cluster c_i, and $d(\mathcal{F}_i, \mathbf{p}_j)$ is a function returning the Euclidean distance between the feature model $\mathcal{F}$ and a point $\mathbf{p}_j$.

Our method checks if a fitting model represents a good approximations for a cluster by looking both at the shape of the eigenvectors $\mathbf{u}_{1:3}^c$, and the value of the residual error e. More formally, we say that a line $\mathcal{L}_i$ represents a valid feature if the following constraints hold

$$\frac{\lambda_1^c + \lambda_2^c}{\lambda_1^c + \lambda_2^c + \lambda_3^c} < \epsilon_l, \quad e = \frac{1}{|c_i|} \sum_{\mathbf{p}_j \in c_i} d(\mathcal{L}_i,\ \mathbf{p}_j) < \epsilon_{dl} \tag{8.5}$$

Similarly, we say that a plane π_i is a valid feature if the following relations are true

$$\frac{\lambda_1^c}{\lambda_1^c + \lambda_2^c + \lambda_3^c} < \epsilon_p, \quad e = \frac{1}{|c_i|} \sum_{\mathbf{p}_j \in c_i} d(\pi_i,\ \mathbf{p}_j) < \epsilon_{dp} \tag{8.6}$$

Note that, during the cluster processing we first try to fit a line and, only in case that fitting fails, we try with a plane. This ordering is fundamental since a plane always represents a good fitting of a line, but not vice versa. The right picture of Fig. 8.4 shows an output example of 3D features computed by our method.

8.2 Experiments

In this experimental evaluation we demonstrate the validity of our feature extraction approach. We show that the stability and the robustness of the computed features can be exploited for solving more complex tasks like localization and mapping, in particular in the context of autonomous driving. We present comparative results of a 3D landmark SLAM system that we feed with the lines and planes computed by our method, and with NARF key-points.

The dataset used in this evaluation has been recorded using an autonomous car mounting a single Velodyne HDL-32E LIDAR sensor. Such a sensor is able to generate point clouds with a horizontal and a vertical field of view of 360 and 40 degrees respectively. The main drawback of this sensor, however, is that it features only 32 laser beams to cover the whole vertical field of view. As a consequence the point clouds are very sparse.

The environment where we acquired the dataset is the MCity test facility of the

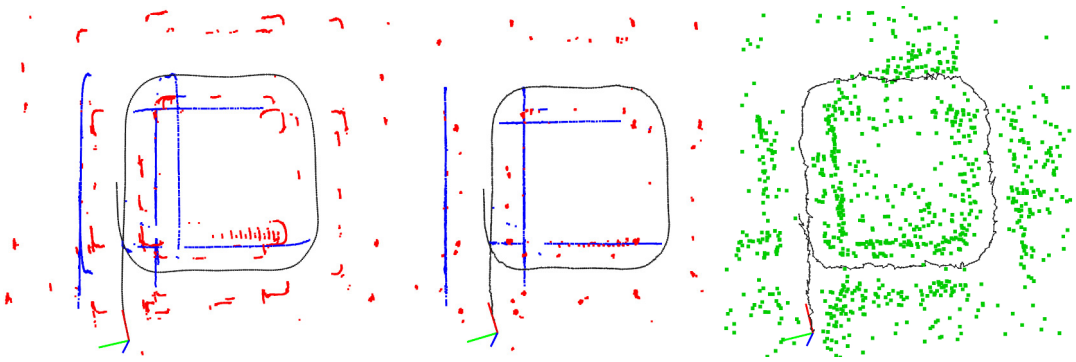


Figure 8.5: Example of 3D landmark based graph SLAM. Left: top view of the graph with our features before the optimization. Center: top view of the graph with our features after the optimization. Right: top view of the graph with NARF key-points after the optimization. Black, blue, red and green points represent in the order odometry, plane, line and NARF key-point measurements. By using our features the trajectory and the measurements are successfully updated to a global consistent state.

University of Michigan. MCity is a realistic off-road environment built for testing the capabilities of autonomous vehicles. The evaluation has been performed using a laptop with a i7-3630QM running at 2.4 GHz.

In the SLAM algorithm we built for these experiments, a landmark can be either a 3D line, a 3D plane or a 3D key-point. As soon as new 3D features are computed, the SLAM system incrementally construct a graph. Each incoming feature can generate either a new factor, or a new node, depending if we can associate it to an existing landmark or not. If a feature is "near" to a landmark L, then we add a factor between the last odometry node and L itself, otherwise a new landmark node is added to the graph.

Fig. 8.5 shows an example of optimization of two graphs constructed by using the output features of our method, and the NARF key-points. In the images the robot trajectory is depicted in black, while measured lines, planes and NARF key-points are colored in red, blue and green respectively. The left image illustrates the graph before any optimization. As the reader can see, the accumulated odometry error results in curved and doubled walls that should be straight. The same problem holds for the lines. However, after performing the graph optimization, the trajectory of the car is corrected, and the measurements are updated to globally consistent positions. This is shown in the central image of Fig. 8.5. The reader could note that, in the optimized graph, the planes can intersect giving the impression of crossing walls. This visual effect is due to the way we represent the measurements (see Appendix A). Within our SLAM algorithm a plane measurement is defined using the coefficients $(a,\ b,\ c,\ d)^T$ of the general equation of a 3D plane. When we draw the graph, each plane measurement is positioned on the point at distance $p = d/\sqrt{a^2 + b^2 + c^2}$ along the normal $(a,\ b,\ c)^T$. This point almost never coincides with the one where the plane was originally sensed.

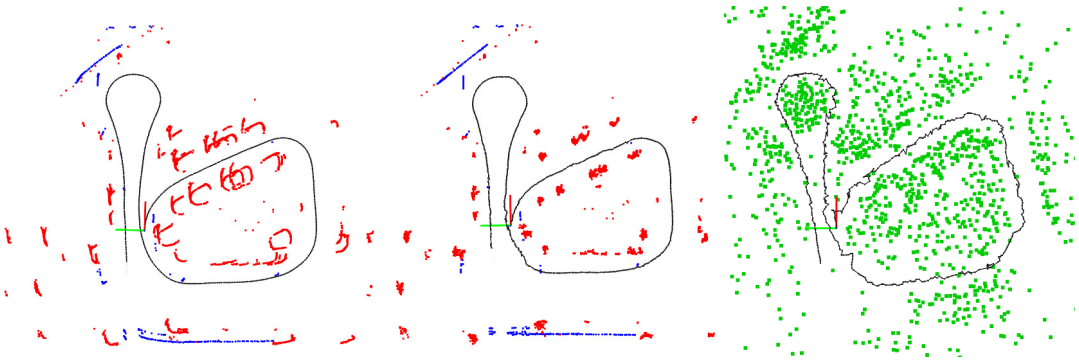


Figure 8.6: Example of 3D landmark based graph SLAM. Left: top view of the graph with our features before the optimization. Center: top view of the graph with our features after the optimization. Right: top view of the graph with NARF key-points after the optimization. Black, blue, red and green points represent in the order odometry, plane, line and NARF key-point measurements. Like in the case of Fig. 8.5, by using our features the robot's trajectory and the other measurements are successfully updated to a global consistent state.

The right image of Fig. 8.5, instead, depicts the optimized graph when using NARF key-points. The robot's trajectory results to be scattered and not globally consistent with the movements that a non-holonomic vehicle like a car can perform. Note also how uniform is the distribution of the NARF key-points. This is a clear evidence that the algorithm is not able to detect unique and repeatable features. As introduced before, this is mainly due to the sparse point clouds generated by the sensor. The robustness and the stability of our features lead to the construction of a well-constrained graph, and to good SLAM results.

Fig. 8.6 shows another example of landmark SLAM in a part of MCity that lacks the presence of buildings. Despite the possibility of relying almost only on lines, the optimization using our features is again good, and the trajectory of the robot is correctly recovered. These results highlights again how our feature extraction algorithm is well-suited to solve tasks like SLAM. Like in the previous case, the car's path obtained by optimizing the graph with NARF key-points results not globally consistent and scattered.

We performed additional experiments to profile our method in terms of time complexity. We tracked the CPU time usage for all three processing blocks depicted in Fig. 8.1. The left plot in Fig. 8.7 contains the mean computation time needed to process all the point clouds of the dataset acquired in MCity. Both results with and without non-vertical region removal are shown. Removing flat regions generates a computation time boost of almost of a factor of 2. An implementation without non-vertical region removal barely runs in real-time (10 Hz), but our full method can process point clouds at more than 20 Hz.

The time complexity of our algorithm is directly dependent on the number of points

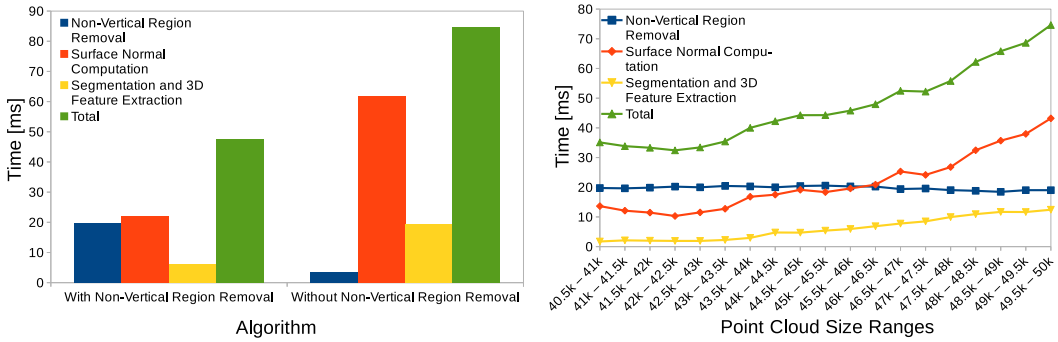


Figure 8.7: Left: mean CPU time usage of our method to extract the 3D features. Removing non-vertical regions lowers the computation time of nearly a factor of 2. Right: computational time needed by our method to extract the 3D features when the number of points in the input cloud increases. The CPU time grows super-linearly with the number of points in the cloud.

in the cloud. To give an idea of the impact that the cloud dimension has on the whole process, we performed another set of experiments. In particular, we profiled the computation time as the number of point in the input cloud increases. The results are shown in the right plot of Fig. 8.7. As expected, the CPU time increases super-linearly with the number of points in the cloud.

8.3 Conclusions

We presented a fast method for extraction of robust 3D features from a sparse point cloud. We also discussed all the relevant steps needed for the implementation of such a system, and we validated our method with an extended experimental evaluation. In addition to this, we presented comparative results against the state-of-the-art NARF key-point detector for 3D point clouds.

We demonstrated that, due to its stability and robustness, our algorithm is suitable to solve more complex tasks like SLAM. All the tests have been performed on a real dataset acquired with an autonomous car at the MCity test facility using a Velodyne HDL-32E. In addition to all of this, our method showed to be fast, enabling real-time use.

Part IV

CONCLUSIONS

Chapter 9
Conclusions and Discussion

"In literature and in life we ultimately pursue,
not conclusions, but beginnings."
— Sam Tanenhaus, Literature Unbound, 1986

Simultaneous localization and mapping is a fundamental building block for any mobile robot. In the recent years, the demand for robots operating in large-scale environments , and for long periods of time, has constantly grown. This is mainly connected to the need of mobile robots working together with people, in their daily environment. As an example it sufficient to think to the rising interest around the business of autonomous cars. This calls for the development of long term SLAM systems capable of coping with large and challenging environments such as entire metropolis. Moreover, considered the nature of the problem, it is necessary to find algorithms that are robust against common noise factors including dynamic aspects and measurement aliasing.

When dealing with 3D large-scale areas, current SLAM solutions are insufficient. Indeed the state-of-the-art for localization and mapping is able to cope only with environments of moderate size (e.g. buildings), and for a limited span of time. This is due to multiple reasons, including the huge jump in the dimension of the problem, and the higher memory consumption. Moreover, application fields such as self-driving and agricultural robots pose significant safety concerns regarding human beings. As a consequence of this, current SLAM solutions must be extended in order to handle noise, and to recover from possible failures.

In this thesis we focused on improving and advancing the current state-of-the-art of specific modules composing common modern SLAM systems. In particular, we introduced a novel point cloud registration method, named Normal ICP, that exploits the surface normal of the points directly in the minimization of the error function. Experimental results showed that NICP is able to outperform other state-of-the-art algorithms. We also developed a reliable and accurate loop closing detection method

that uses as search heuristic the trajectory followed by the robot. Such an approach significantly reduces the overall computation time, and it allows for recovering from inconsistent global map estimates due to miss detected closures. During the evaluation, our approach has been able to correctly map hard and harsh large-scale environments such as the catacombs of Priscilla in Rome. Additionally, we presented a method that exploits 3D sensor data to build dense two dimensional measurements. By relying on recent and well established approaches for 2D localization and mapping, these measurements are then used to perform robust indoor/outdoor 2D SLAM. Moreover, it showed to be memory efficient which makes the approach particularly suitable for large-scale mapping. Finally, we developed a very fast method for computing robust 3D geometric features, in our specific case 3D lines and planes, from sparse sensor data. We evaluated and demonstrated the efficacy of our approach by performing 3D feature based SLAM with an autonomous car in the MCity test facility.

APPENDICES

Appendix A

Minimal Representations

As explained in Sec. 3.2.1, when using over-parameterized representations in conjunction with a least-squares estimation problem, can lead to unstable results. To overcome this issue, it is common to use minimal representations for the state variables involved in the minimization problem. A minimal representation uses the least number of parameters necessary to describe the state space. Normally, it corresponds to the number of degrees of freedom of the least-squares variable involved in the minimization. For example, a planar line has two degrees of freedom. This means that, a minimal representation is given by the Hessian normal form of the equation of a 2D line

$$y \sin\theta + x \cos\theta - p = 0 \tag{A.1}$$

where the parameters p and $\theta = atan2(\sin\theta,\ \cos\theta)$ are respectively the positive length, and the angle of inclination, of the normal segment connecting the origin and the line itself. In the remainder of this Appendix, we will describe the minimal representations employed in the algorithms presented in this thesis. More in particular, we will focus on the the state spaces of three dimensional rotations, planes and lines.

A.1 3D Rotation

A rotation in a three dimensional space has exactly three degrees of freedom. 3D orientations are commonly represented using rotation matrices, the special orthogonal group $\mathcal{SO}(3)$. Having such a formalism nine different parameters, it is an over-parameterized representation. Euler Angles are a simple and intuitive alternative that allow for simple analysis and control of orientations. In particular, they provide a compact way to represent a 3D orientation through the combination of three rotations about different axes.

More formally, consider the basic rotation around one of the axis of a coordinate

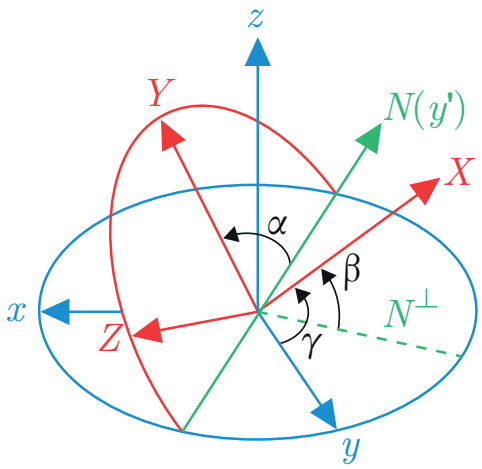


Figure A.1: Tait-Bryan Euler angles convention sequence z-y-x. The original reference system is shown in blue, while the rotated one is depicted in red. N coincides with y.

system. Using the right-hand rule, we can define the following three elemental rotations by an angle θ about the x, y, and z axes

$$\mathbf{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{A.2}$$

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{A.3}$$

$$\mathbf{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.4}$$

A generic rotation matrix can be obtained from these three, simply using matrix multiplications. For example

$$\mathbf{R} = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_x(\gamma) \tag{A.5}$$

it is an intrinsic rotation whose Tait-Bryan angles are α (yaw), β (pitch) and γ (roll) about the z, y and x axes respectively. The Tait-Bryan angles are a convention, widely adopted in robotics, for representing Euler angles. The definitions and notations used for Tait-Bryan angles are similar to those for proper Euler angles, with the difference that the former represents rotations about three distinct axes (x, y and z), while the latter uses the same axis for both the first and third elemental rotations (z, x and z). Fig. A.1 shows a graphical example of the intermediate elemental rotations occurring

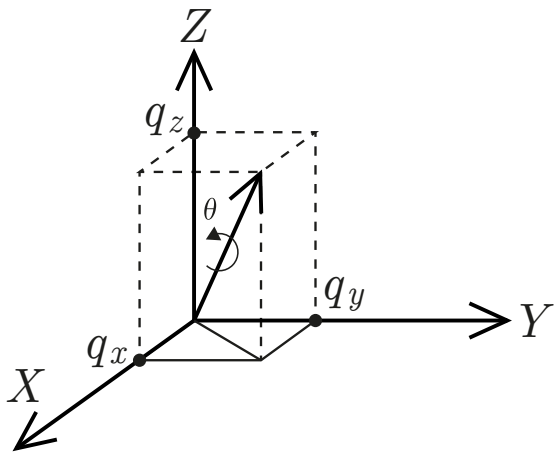


Figure A.2: Geometric abstraction of the concept behind a quaternion. The rotating arrow indicates the positive direction of the rotation.

when using Tait-Bryan angles.

Note that Euler angles are limited by a phenomenon called "gimbal lock". Gimbal lock is a singular configuration for which the rotation can not be uniquely represented using Euler angles. The specific orientation at which a gimbal lock occurs depends on the order of rotations used. In the case of Tait-Bryan angles, this singularity happens when β is equal to $\pi/2$ radians. When the pitch is at that particular configuration, changes to roll and yaw result on the same rotation.

In alternative to Euler angles, it is possible to exploit the quaternions formalism. A quaternion is a four element vector that can be used to encode any rotation in a 3D coordinate system. In particular, it is composed by three complex elements representing the axis about which the rotation happens, and a real element encoding the amount of rotation θ which will occur about that axis. More formally, a quaternion q can be defined as

$$q = q_x i + q_y j + q_z k + q_w \tag{A.6}$$

where q_x, q_y, q_z and q_w are real numbers, while i, j and k are the imaginary components.

The reader might notice that a minimum of four parameters is necessary to represent a quaternion, while a rotation has only three degrees of freedom. We can overcome this issue by using unit quaternions. This particular subset of quaternions satisfy the property for which the norm must be unitary

$$|q| = q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1 \tag{A.7}$$

In this case, the fourth parameter q_w can be computed as a function of the other three

as

$$q_w = \sqrt{1 - q_x^2 - q_y^2 - q_z^2} \tag{A.8}$$

thus reducing the number of independent parameters from four to three, and obtaining in this way a minimal representation. Fig. A.2 contains a graphical example showing the geometric concept behind quaternions. Despite less intuitive, using unit quaternions instead of matrices or Euler angles to represent 3D rotations has many advantages:

- concatenating rotations is computationally more efficient and numerically more stable;
- extracting the angle and axis of the rotation is simpler;
- interpolation is easier;
- quaternions do not suffer from gimbal lock.

A.2 3D Line

A line in the three dimensional space has four degrees of freedom. However, in order to understand how to represent a 3D line with a minimal set of parameters, it is first necessary to give some prior definitions.

Consider a two element vector $\boldsymbol{\sigma} = (\sigma_1,\ \sigma_2)^T \in \mathbb{P}^1$, where $\mathbb{P}^1$ indicates the one dimensional projective space, we can define $\boldsymbol{\sigma}$ by using a 2D rotation matrix $\mathbf{W} \in \mathcal{SO}(2)$ as

$$\mathbf{W} = \frac{1}{||\boldsymbol{\sigma}||} \begin{pmatrix} \sigma_1 & -\sigma_2 \\ \sigma_2 & \sigma_1 \end{pmatrix} \tag{A.9}$$

Note that the first column of the matrix $\mathbf{W}$ is the vector $\boldsymbol{\sigma}$ itself normalized to have unit norm. Let now θ be the update parameter, then a local update can be applied as

$$\mathbf{W} = \mathbf{W}\,\mathbf{R}(\theta) \tag{A.10}$$

with $\mathbf{R}(\theta)$ being a 2D rotation matrix of angle θ. Similarly, a 3D rotation matrix $\mathbf{U} \in \mathcal{SO}(3)$ can be locally updated by using 3 parameters associated to any well-behaved, locally non singular, representation such as Euler angles $\boldsymbol{\theta} = (\theta_1,\ \theta_2,\ \theta_3)^T$. In this case the update can be applied as

$$\mathbf{U} = \mathbf{U}\,\mathbf{R}(\boldsymbol{\theta}) = \mathbf{R}_x(\theta_1)\mathbf{R}_y(\theta_2)\mathbf{R}_z(\theta_3) \tag{A.11}$$

where $\mathbf{R}_x(\theta_1)$, $\mathbf{R}_y(\theta_2)$ and $\mathbf{R}_z(\theta_3)$ are elemental rotations as defined in the previous section.

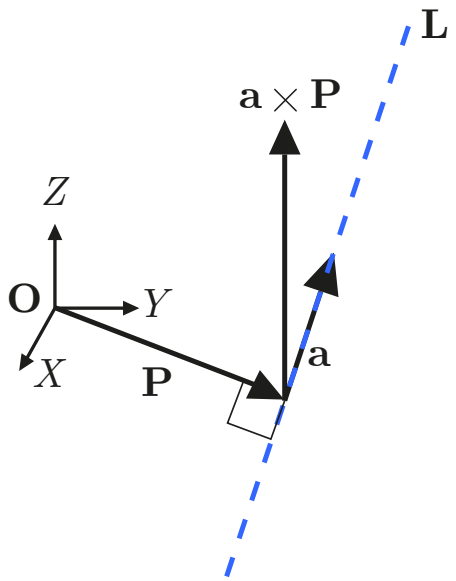


Figure A.3: Geometric visualization of the Plücker representation of a 3D line.

With this mind, any arbitrary projective 3D line $\tilde{\mathbf{L}}$ can be defined by the orthonormal representation [169] $(\mathbf{U},\ \mathbf{W})$

$$(\mathbf{U},\ \mathbf{W}) \in \mathcal{SO}(3) \times \mathcal{SO}(2) \tag{A.12}$$

where $\mathcal{SO}(3)$ and $\mathcal{SO}(2)$ indicate respectively the Lie groups of 3×3 and 2×2 rotation matrices. Note that $(\mathbf{U},\ \mathbf{W})$ is a minimal representation since $\mathcal{SO}(3)$ has three degrees of freedom and $\mathcal{SO}(2)$ exactly one. This means that such a formalism admits a locally non singular parameterization, and thus a 3D line can be locally updated with a minimum set of four independent parameters.

Let be now the vectors $\mathbf{a}$ and $\mathbf{b}$ the Plücker representation (see Fig. A.3) of a 3D line $\mathbf{L}$

$$\mathbf{L} = (\mathbf{a}^T,\ \mathbf{b}^T) = (\mathbf{a}^T,\ (\mathbf{a} \times \mathbf{P})^T)^T \tag{A.13}$$

where $\mathbf{a}$ is the unit vector representing the direction of the line, $\mathbf{b}$ is the moment perpendicular to both the line direction $\mathbf{a}$, and the shortest distance between the origin and the line itself, and $\mathbf{P}$ is an arbitrary point lying on $\mathbf{L}$. We can then define the 3×2 matrix $\tilde{\mathbf{C}}$ whose QR factorization is

$$\tilde{\mathbf{C}} = \underbrace{\left(\frac{\mathbf{a}}{||\mathbf{a}||},\ \frac{\mathbf{b}}{||\mathbf{b}||},\ \frac{\mathbf{a}\times\mathbf{b}}{||\mathbf{a}\times\mathbf{b}||}\right)}_{\mathbf{U}\in\mathcal{SO}(3)} \underbrace{\begin{pmatrix} ||\mathbf{a}|| & 0 \\ 0 & ||\mathbf{b}|| \\ 0 & 0 \end{pmatrix}}_{\mathbf{\Sigma}} \tag{A.14}$$

We recall that the QR factorization of a matrix is the decomposition of an arbitrary matrix $\mathbf{A}$ into the product $\mathbf{A} = \mathbf{Q}\,\mathbf{R}$ of an orthogonal matrix $\mathbf{Q}$, and an upper triangular matrix $\mathbf{R}$. Note that $\mathbf{U} \in \mathcal{SO}(3)$, while the non zero entries of matrix $\mathbf{\Sigma}$ can be represented by a $\mathcal{SO}(2)$ as shown in Eq. A.9, thus yielding to the following matrix $\mathbf{W}$

$$\mathbf{W} = \frac{1}{||\boldsymbol{\sigma}||} \begin{pmatrix} ||\mathbf{a}|| & -||\mathbf{b}|| \\ ||\mathbf{b}|| & ||\mathbf{a}|| \end{pmatrix} \tag{A.15}$$

where $\boldsymbol{\sigma} = (||\mathbf{a}||,\ ||\mathbf{a}||)^T$.

Going back from the orthonormal to the Plücker representation is a trivial task, and it can be easily done as follows

$$\mathbf{L} = (w_{11}\mathbf{u}_1^T,\ w_{21}\mathbf{u}_2^T)^T \tag{A.16}$$

with $\mathbf{u}_i$ being the i-th column of matrix $\mathbf{U}$.

A minimal four parameter update $\mathbf{p} = (\boldsymbol{\theta}^T,\ \theta)^T = (\theta_1,\ \theta_2,\ \theta_3,\ \theta)^T$ of the line $\tilde{\mathbf{L}}$ can therefore be computed as

$$\mathbf{U} = \mathbf{U}\,\mathbf{R}(\boldsymbol{\theta}) \tag{A.17}$$

$$\mathbf{W} = \mathbf{W}\,\mathbf{R}(\theta) \tag{A.18}$$

Under a geometric point of view, matrix $\mathbf{W}$ encodes the ratio $||\mathbf{a}||/||\mathbf{b}||$, hence the distance d from the origin to the line. In other words, the parameter θ acts on d. Matrix $\mathbf{U}$, instead, is related to a 3D coordinate system attached to $\tilde{\mathbf{L}}$. The first parameter θ_1 rotates $\tilde{\mathbf{L}}$ around a circle of radius d, centered in the origin, and lying on the plane defined by the origin and the line. Parameter θ_2 rotates $\tilde{\mathbf{L}}$ around a circle of radius d, centered in the origin, and lying in a plane containing both the origin, and the closest point $\mathbf{Q}$ of $\tilde{\mathbf{L}}$ to the origin, and perpendicular to the line. Finally, the parameter θ_3 rotates $\tilde{\mathbf{L}}$ around the axis defined by the origin and $\mathbf{Q}$.

A.3 3D Plane

A plane in the 3D space has three degrees of freedom. Intuitively what we need to describe an arbitrary 3D plane is an orientation telling the direction the plane is facing, and the distance of the plane from the origin. To this end, consider the general equation of a plane lying at distance p from the origin

$$ax + by + cz + d = 0 \tag{A.19}$$

$$p = \frac{d}{\sqrt{a^2 + b^2 + c^2}} \tag{A.20}$$

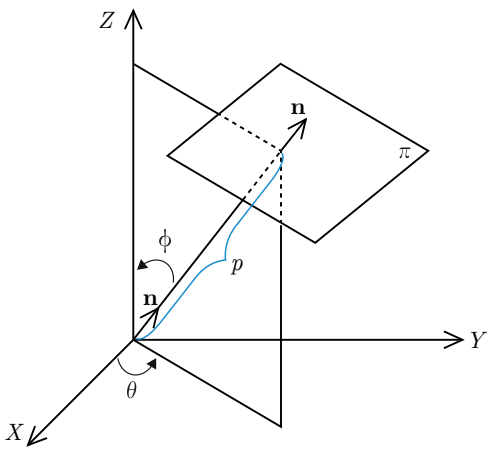


Figure A.4: Geometric idea behind the minimal representation parameters of a 3D plane.

Like in the case of a planar line, we can derive from Eq. A.19 the Hessian normal form of the plane. This can be done by defining the components of the unit normal vector to the plane $\mathbf{n} = (n_x,\ n_y,\ n_z)^T$ as

$$n_x = \frac{a}{\sqrt{a^2+b^2+c^2}} \tag{A.21}$$

$$n_y = \frac{b}{\sqrt{a^2+b^2+c^2}} \tag{A.22}$$

$$n_z = \frac{c}{\sqrt{a^2+b^2+c^2}} \tag{A.23}$$

Then the Hessian normal form of the plane is

$$\mathbf{n} \cdot \mathbf{x} = -p \tag{A.24}$$

Both the general equation, and the Hessian normal form, can describe a 3D plane using a set of four parameters. Obtaining a minimal representation from these two is straightforward. Indeed, the normal direction of the plane can be expressed by using the azimuth θ and elevation ϕ angles with respect to the origin

$$\theta = \tan^{-1}\left(\frac{n_y}{n_x}\right) \tag{A.25}$$

$$\phi = \cos^{-1}\left(\frac{n_z}{r}\right) \tag{A.26}$$

with $r = \sqrt{n_x^2+n_y^2+n_z^2}$. The minimal representation of a plane is thus composed by the triplet $(\theta,\ \phi,\ p)$. Fig. A.4 shows the geometric idea behind such a representation.

Appendix B

Quaternion-Based Rotation Matrix Derivative

Consider a three dimensional rotation defined through the quaternion $\mathbf{q} = (q_x,\ q_y,\ q_z,\ q_w)^T$, then we can also define the associated rotation matrix $\mathbf{R}(\mathbf{q})$ as a function of the quaternion's parameters

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix} \tag{B.1}$$

Now, assuming that $\mathbf{q}$ is *normalized*, which means that $\mathbf{q}$ is a unit quaternion with $|\mathbf{q}| = 1$, we can rewrite the previous matrix as follow

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_z\hat{q}_w & 2q_xq_z + 2q_y\hat{q}_w \\ 2q_xq_y + 2q_z\hat{q}_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_x\hat{q}_w \\ 2q_xq_z - 2q_y\hat{q}_w & 2q_yq_z + 2q_x\hat{q}_w & 1 - 2q_x^2 - 2q_y^2, \end{pmatrix} \tag{B.2}$$

where $\hat{q}_w = \sqrt[2]{1 - q_x^2 - q_y^2 - q_z^2}$.

The partial derivatives with respect to $\mathbf{q}$ of the rotation matrix $\mathbf{R}(\mathbf{q})$ (Eq. B.2),

evaluated in $\mathbf{q} = (0,\ 0,\ 0,\ 1)^T = \mathbf{0}$, have the following form

$$\left.\frac{\partial \mathbf{R}(\mathbf{q})}{\partial q_x}\right|_{\mathbf{q}=\mathbf{0}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & 2 & 0 \end{pmatrix} = \mathbf{S}_x \tag{B.3}$$

$$\left.\frac{\partial \mathbf{R}(\mathbf{q})}{\partial q_y}\right|_{\mathbf{q}=\mathbf{0}} = \begin{pmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ -2 & 0 & 0 \end{pmatrix} = \mathbf{S}_y \tag{B.4}$$

$$\left.\frac{\partial \mathbf{R}(\mathbf{q})}{\partial q_z}\right|_{\mathbf{q}=\mathbf{0}} = \begin{pmatrix} 0 & -2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{S}_z \tag{B.5}$$

From these results, we note also that the derivative of the rotation of a vector $\mathbf{v} = (v_x,\ v_y,\ v_z)^T$, in $\mathbf{q} = (0,\ 0,\ 0,\ 1)^T = \mathbf{0}$, can be expressed as

$$\mathbf{S}(\mathbf{v}) = (\mathbf{S}_x \cdot \mathbf{v} | \mathbf{S}_y \cdot \mathbf{v} | \mathbf{S}_z \cdot \mathbf{v}) = \begin{pmatrix} 0 & 2v_z & -2v_y \\ -2v_z & 0 & 2v_x \\ 2v_y & -2v_x & 0 \end{pmatrix} = -2[\mathbf{v}]_\times \tag{B.6}$$

where $[\mathbf{v}]_\times$ is known as the cross product matrix of the vector $\mathbf{v}$.

Appendix C
Jacobian Derivation

Let us define a generic point with normal as a 6D vector having the form $\mathbf{p} = (\mathbf{c}^T,\ \mathbf{n}^T)^T = (c_x,\ c_y,\ c_z,\ n_x,\ n_y,\ n_z)^T$ composed by its Cartesian coordinates $\mathbf{c}$, and the associated surface normal components $\mathbf{n}$. Now, recalling the definition of the $\oplus$ operator described in Eq. 5.10, the perturbation of the error between two corresponding points $\mathbf{p}$ and $\mathbf{p}'$ is

$$\mathbf{e}(\mathbf{x} \oplus \Delta\mathbf{x}) = \mathbf{e}(\mathbf{t} \oplus \Delta\mathbf{t},\ \mathbf{R} \oplus \mathbf{R}(\Delta\mathbf{q})) = \begin{pmatrix} \mathbf{R}(\Delta\mathbf{q}) \cdot \hat{\mathbf{c}} + \Delta\mathbf{t} \\ \mathbf{R}(\Delta\mathbf{q}) \cdot \hat{\mathbf{n}} \end{pmatrix} - \begin{pmatrix} \mathbf{c}' \\ \mathbf{n}' \end{pmatrix} \tag{C.1}$$

with $\hat{\mathbf{p}} = \mathbf{T}(\mathbf{x}) \oplus \mathbf{p}$.

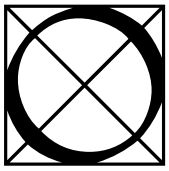
The 6×6 *Jacobian matrix* $\mathbf{J}$ of the error function $\mathbf{e}(\mathbf{x} \oplus \Delta\mathbf{x})$ is computed deriving the previous equation with respect to $\Delta\mathbf{t} = (\Delta t_x,\ \Delta t_y,\ \Delta t_z)^T$ and $\Delta\mathbf{q} = (\Delta q_x,\ \Delta q_y,\ \Delta q_z)^T$, and then evaluating the derivative in $(\Delta\mathbf{t} = 0,\ \Delta\mathbf{q} = 0)$

$$\begin{aligned} \mathbf{J} &= \left(\frac{\partial \mathbf{e}(\mathbf{t} + \Delta\mathbf{t}, \mathbf{R} + \mathbf{R}(\Delta\mathbf{q}))}{\partial \Delta\mathbf{t}} \middle| \frac{\partial \mathbf{e}(\mathbf{t} + \Delta\mathbf{t}, \mathbf{R} + \mathbf{R}(\Delta\mathbf{q}))}{\partial \Delta\mathbf{q}} \right) \Bigg|_{\Delta\mathbf{t}=\mathbf{0},\ \Delta\mathbf{q}=\mathbf{0}} \\ &= \begin{pmatrix} \mathbf{I}_{3\times3} & \frac{\partial \mathbf{R}(\Delta\mathbf{q})}{\partial \Delta q_x} \cdot \hat{\mathbf{c}} & \frac{\partial \mathbf{R}(\Delta\mathbf{q})}{\partial \Delta q_y} \cdot \hat{\mathbf{c}} & \frac{\partial \mathbf{R}(\Delta\mathbf{q})}{\partial \Delta q_z} \cdot \hat{\mathbf{c}} \\ \mathbf{0}_{3\times3} & \frac{\partial \mathbf{R}(\Delta\mathbf{q})}{\partial \Delta q_x} \cdot \hat{\mathbf{n}} & \frac{\partial \mathbf{R}(\Delta\mathbf{q})}{\partial \Delta q_y} \cdot \hat{\mathbf{n}} & \frac{\partial \mathbf{R}(\Delta\mathbf{q})}{\partial \Delta q_z} \cdot \hat{\mathbf{n}} \end{pmatrix} \Bigg|_{\Delta\mathbf{t}=\mathbf{0},\ \Delta\mathbf{q}=\mathbf{0}} \\ &= \begin{pmatrix} \mathbf{I}_{3x3} & -2[\hat{\mathbf{c}}]_\times \\ \mathbf{0}_{3x3} & -2[\hat{\mathbf{n}}]_\times \end{pmatrix} \end{aligned} \tag{C.2}$$

where $[\mathbf{v}]_\times$ represents the cross product matrix as defined in Eq. B.6.

The reader might notice that the left block of the Jacobian matrix has a 3×3 matrix of zeros in the bottom, this is because the normal, unlike the point coordinates, is not translated.

Appendix D
Integral Images



PERATIONS such as the computation of the covariance matrix over the k-nearest neighboring of a point, commonly necessary for the extraction of surface normals, can be computationally expensive. However, if the data follow some sort of organization, like in the case of the matrix structure in depth images, it is possible to speedup this process by using the concept of integral images.

The integral image represents a fast and powerful technique to compute the sum of the values contained in arbitrary rectangular regions of a matrix. The base idea is the construction of a matrix known as *summed area table*. In such a table, the value of a generic element of coordinates $(x,\ y)$ contains the sum of all the elements in the upper left rectangular block of the input matrix, including the element $(x,\ y)$ itself. The top picture of Fig. D.1 shows a graphical example of the region influencing a given element $(x,\ y)$ in the summed area table.

The power of the integral images resides on the fact that, once the summed area table is computed, it is possible to evaluate in constant time ($\mathcal{O}(1)$) the sum of the elements of an arbitrary rectangular region of the original matrix. In addition to this, note that the summed area table can be computed parsing a single time the input matrix. This can be easily done by knowing that an element $(x,\ y)$ of the table is equal to

$$S(x,\ y) = I(x,\ y) + S(x-1,\ y) + S(x,\ y-1) - S(x-1,\ y-1) \qquad \text{(D.1)}$$

where $S(x,\ y)$ indicates the value table element of coordinates $(x,\ y)$, and $I(x,\ y)$ is the value of the element in the original matrix in position $(x,\ y)$. In the case that a coordinate is negative, like when analyzing an edge element, the value of $S(\cdot)$ has to be set to zero. The bottom left and bottom right pictures of Fig. D.1 show respectively a generic input matrix, and the associated integral image.

Once the table is constructed, we can evaluate the sum of all elements in a rectangular block of the original matrix by directly querying the integral image. More specifically, let a rectangular region to be identified by its four vertices A, B, C and D,

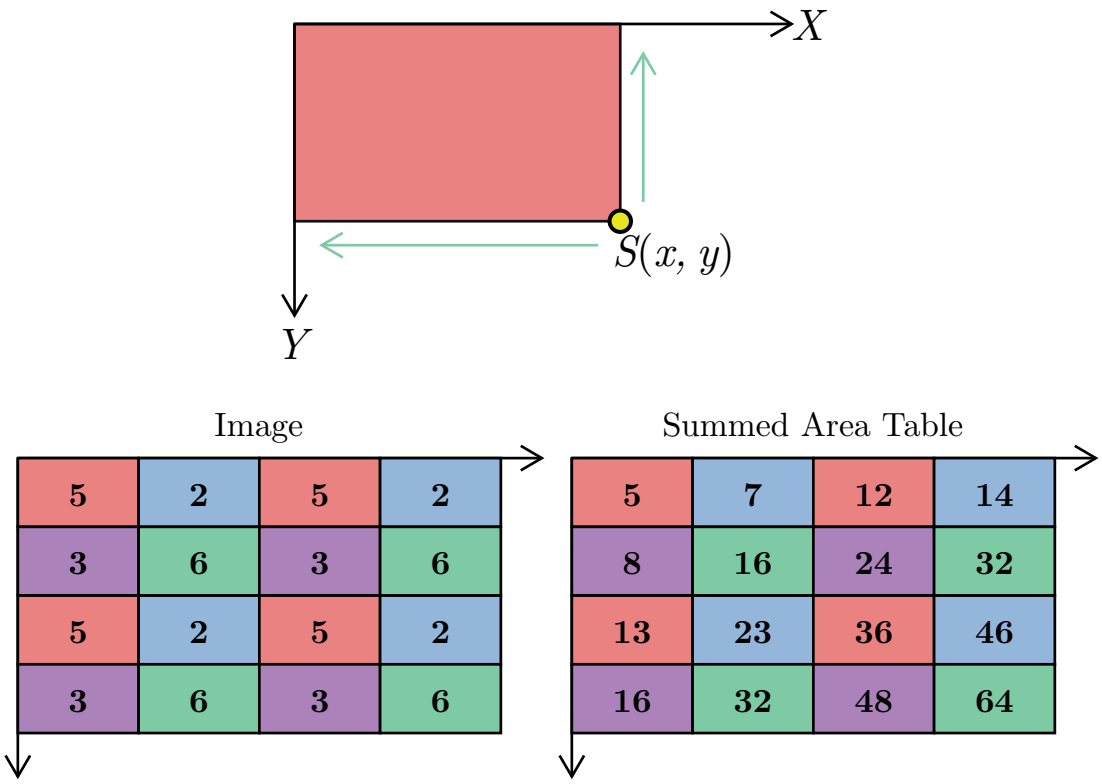


Figure D.1: Top: example showing the rectangular region influencing an arbitrary element $(x,\ y)$ of the summed area table. Bottom: integral image (right) computed from an input matrix (left). (Images from [170]).

the summed area of such a region is computed as

$$S(A,\ B,\ C,\ D) = S(A) + S(D) - S(B) - S(C) \tag{D.2}$$

where $S(A)$ is the value of the integral image at the element identified by the vertex A, and similarly for the other terms. As an example, suppose we want to compute the summed area of the rectangular region highlighted on the top picture of Fig. D.2. Additionally, consider the summed areas defined by the vertices A, B, C and D as illustrated in the rest of images of Fig. D.2. The summed area $S(A,\ B,\ C,\ D)$ is

$$S(A,\ B,\ C,\ D) = S(A) + S(D) - S(B) - S(C) = 16 + 64 - 32 - 32 = 16 \tag{D.3}$$

The reader might be curious of how, for example, these integral images can be used to compute the mean and covariance matrix of Eq. 3.35. Without loss of generality, by

performing some simple manipulations we obtain that

$$
\begin{aligned}
\boldsymbol{\Sigma}_i^s &= \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}(\mathbf{p}_k^c-\mu_i^s)(\mathbf{p}_k^c-\mu_i^s)^T \\
&= \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mathbf{p}_k^{c^T}-\mu_i^s\mathbf{p}_k^{c^T}-\mathbf{p}_k^c\mu_i^{s^T}+\mu_i^s\mu_i^{s^T} \\
&= \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mathbf{p}_k^{c^T}-\frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mu_i^s\mathbf{p}_k^{c^T}-\frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mu_i^{s^T}+\frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mu_i^s\mu_i^{s^T} \\
&= \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mathbf{p}_k^{c^T}-\frac{1}{|\mathcal{V}_i|}\mu_i^s\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^{c^T}-\frac{1}{|\mathcal{V}_i|}\left(\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\right)\mu_i^{s^T}+\frac{1}{|\mathcal{V}_i|}\mu_i^s\mu_i^{s^T}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}1 \\
&= \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mathbf{p}_k^{c^T}-\mu_i^s\mu_i^{s^T}-\mu_i^s\mu_i^{s^T}+\mu_i^s\mu_i^{s^T} \\
&= \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mathbf{p}_k^{c^T}-\mu_i^s\mu_i^{s^T} \qquad \text{(D.4)}
\end{aligned}
$$

Recalling that the mean μ_i^s is equal to

$$
\mu_i^s = \frac{1}{|\mathcal{V}_i|}\sum_{\mathbf{p}_k^c\in\mathcal{V}_i} \qquad \text{(D.5)}
$$

and looking at Eq. D.4, the reader will notice that in order to compute the mean μ_i^s and the covariance matrix $\boldsymbol{\Sigma}_i^s$ it is sufficient to keep track of three quantities regarding the points that are taken into account for the calculation:

- the number of points $|\mathcal{V}_i|$;
- the sum of the points $\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c$;
- the squared sum of the points $\sum_{\mathbf{p}_k^c\in\mathcal{V}_i}\mathbf{p}_k^c\mathbf{p}_k^{c^T}$.

We can therefore define an integral image whose elements are not just real numbers, but a more complex structure that has the function of a point accumulator. This structure has the goal of memorizing the three quantities listed before. By exploiting the organized structure of the image, we can select a rectangular neighborhood around a query point, and compute efficiently the mean and covariance using the rule described in Eq. D.2.

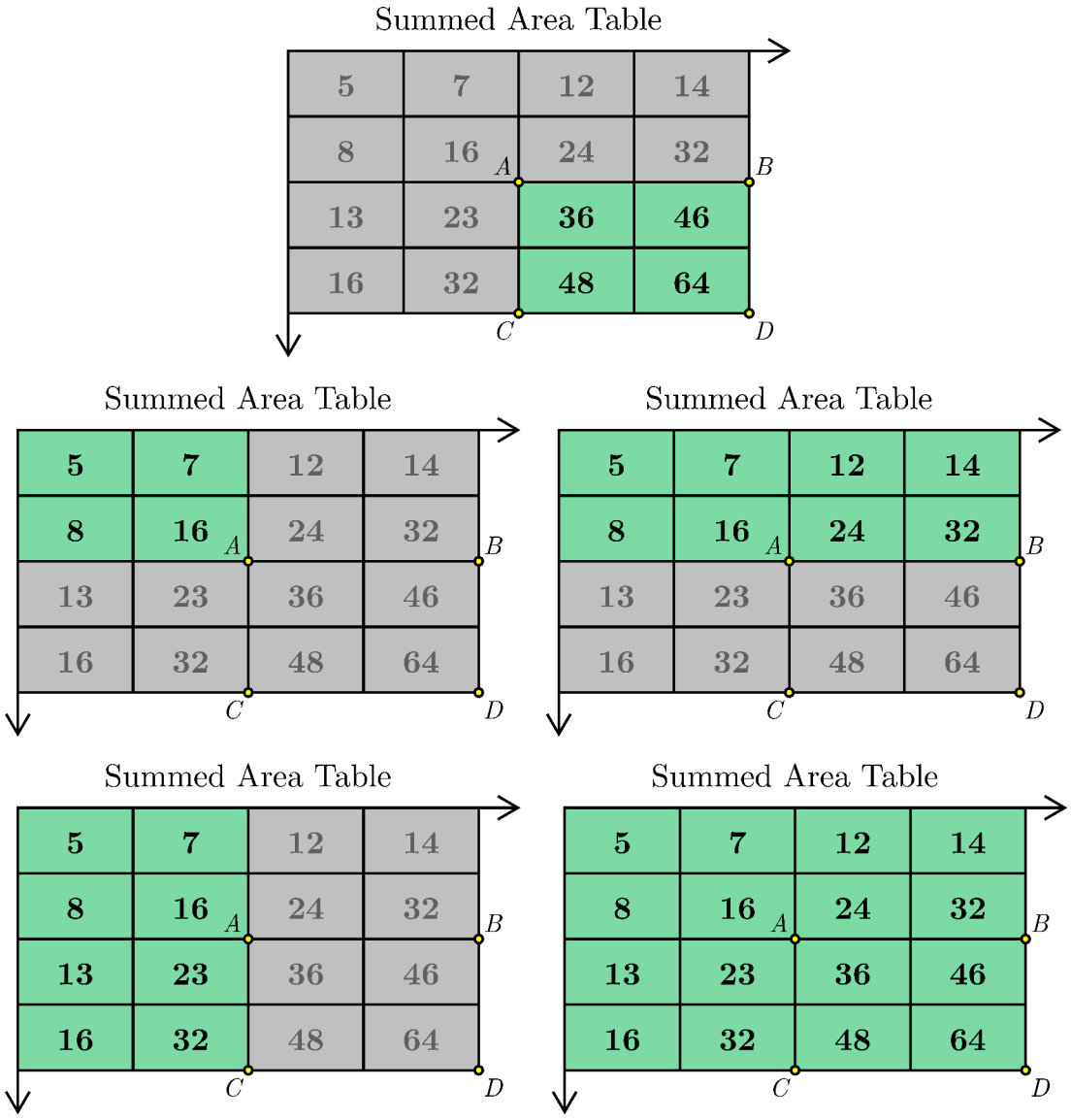


Figure D.2: Summed regions captured by the elements corresponding to the four vertices A (middle left), B (middle right), C (bottom left) and D (bottom right) of the query region highlighted on the picture on the top. (Images from [170]).

Bibliography

[1] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. D., and Leonard, J. J. Simultaneous localization and mapping: Present, future, and the robust-perception age. *arXiv*, (2016). [Cited on pages ix, 3, 16, 55]

[2] Rosten, E. and Drummond, T. Fusing points and lines for high performance tracking. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2, pp. 1508–1515. IEEE (2005). [Cited on pages viii, 4, 25, 26]

[3] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pp. 778–792. Springer (2010). [Cited on pages 4, 26]

[4] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571. IEEE (2011). [Cited on pages 4, 26]

[5] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W. Point feature extraction on 3d range scans taking into account object boundaries. In *Robotics and automation (icra), 2011 ieee international conference on*, pp. 2601–2608. IEEE (2011). [Cited on pages viii, 4, 29, 134]

[6] Holzer, S., Rusu, R. B., Dixon, M., Gedikli, S., and Navab, N. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2684–2689. IEEE (2012). [Cited on pages 4, 76, 138]

[7] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. *Robotics: modelling, planning and control*. Springer Science & Business Media (2010). [Cited on page 4]

[8] A. Newcombe, R., et al. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. of the Int. Symposium on Mixed and Augmented Reality (ISMAR)* (2011). [Cited on pages xi, 4, 35, 73, 77, 114, 115]

[9] Newcombe, R. A., Fox, D., and Seitz, S. M. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 343–352 (2015). [Cited on page 4]

[10] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. Bundle adjustment - a modern synthesis. In *International workshop on vision algorithms*, pp. 298–372. Springer (1999). [Cited on page 4]

[11] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, **34** (2013), 189. [Cited on pages vii, 4, 15, 16]

[12] Segal, A. V., Haehnel, D., and Thrun, S. Generalized-ICP. In *Proc. of Robotics: Science and Systems (RSS)* (2009). [Cited on pages viii, 4, 21, 32, 34, 35, 52, 63, 65, 73, 75]

[13] Steinbrücker, F., Sturm, J., and Cremers, D. Real-time visual odometry from dense rgb-d images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 719–722. IEEE (2011). [Cited on pages 4, 34, 77]

[14] Eustice, R. M., Singh, H., and Leonard, J. J. Exactly sparse delayed-state filters. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on* (2005). [Cited on pages 5, 17]

[15] Grisetti, G., Stachniss, C., and Burgard, W. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, **23** (2007), 34. [Cited on pages 5, 18]

[16] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. g2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3607–3613. IEEE (2011). [Cited on pages 5, 7, 20, 104, 111]

[17] Grisetti, G., Kümmerle, R., Stachniss, C., and Burgard, W. A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, **2** (2010), 31. [Cited on pages vii, ix, 5, 16, 19, 56, 100, 104, 133]

[18] Serafin, J. and Grisetti, G. Using augmented measurements to improve the convergence of ICP. In *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. Springer (2014). [Cited on page 5]

[19] Serafin, J. and Grisetti, G. Nicp: Dense normal based point cloud registration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 742–749. Hamburg, Germany (2015). [Cited on page 5]

[20] Ziparo, V., et al. Exploration and mapping of catacombs with mobile robots. In *IEEE Int. Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–2 (2013). [Cited on page 5]

[21] Ziparo, V. A., et al. A user perspective on the rovina project. In *Proc. of the 18th ICOMOS General Assembly and Scientific Symposium "Heritage and Landscape as Human Values"*, pp. 578–582. Florence, Italy (2014). [Cited on page 5]

[22] Serafin, J., Di Cicco, M., Bonanni, T. M., Grisetti, G., Iocchi, L., Nardi, D., Stachniss, C., and Ziparo, V. A. Robots for exploration, digital preservation and visualization of archeological sites. In *Artificial Intelligence for Cultural Heritage* (edited by L. Bordoni, F. Mele, and A. Sorgente), chap. 5, pp. 121–140. Cambridge Scholars Publishing (2016). [Cited on page 5]

[23] Goeddel, R., Kershaw, C., Serafin, J., and Olson, E. Flat2d: Fast localization from approximate transformation into 2d. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)(to appear)*, pp. 1932–1939. Daejeon, Korea (2016). [Cited on page 5]

[24] Serafin, J., Olson, E., and Grisetti, G. Fast and robust 3d feature extraction from sparse point clouds. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4105–4112. Daejeon, Korea (2016). [Cited on page 5]

[25] Hartley, R. and Zisserman, A. *Multiple view geometry in computer vision*. Cambridge university press (2003). [Cited on pages 6, 37]

[26] Besl, P. J. and McKay, N. D. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1992). [Cited on pages 6, 21, 30, 33, 50, 63, 73, 134]

[27] Smith, R., Self, M., and Cheeseman, P. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehicles*, (1990). [Cited on pages 15, 17, 51]

[28] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence* (2002). [Cited on pages 15, 18]

[29] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *In Proc. of the Int. Conf. on Artificial Intelligence (IJCAI* (2003). [Cited on pages 15, 18]

[30] Triebel, R., Pfaff, P., and Burgard, W. Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pp. 2276–2282. IEEE (2006). [Cited on page 15]

[31] Hähnel, D., Burgard, W., Fox, D., and Thrun, S. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* (2003). [Cited on pages 15, 18]

[32] Grisetti, G., Stachniss, C., and Burgard, W. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on* (2005). [Cited on pages 15, 18]

[33] Konolige, K., Bowman, J., Chen, J., Mihelich, P., Calonder, M., Lepetit, V., and Fua, P. View-based maps. *The International Journal of Robotics Research*, (2010). [Cited on page 16]

[34] Konolige, K. A gradient method for realtime robot control. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 1, pp. 639–646. IEEE (2000). [Cited on page 16]

[35] Stachniss, C., Frese, U., and Grisetti, G. Openslam (2007). Available from: `http://www.openslam.org`. [Cited on pages vii, 16]

[36] Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems* (2007). [Cited on pages vii, 16]

[37] Durrant-Whyte, H., Rye, D., and Nebot, E. Localization of autonomous guided vehicles. In *Robotics Research* (1996). [Cited on page 16]

[38] Smith, R. C. and Cheeseman, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, (1986). [Cited on page 17]

[39] Leonard, J. J. and Durrant-Whyte, H. F. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on* (1991). [Cited on page 17]

[40] Castellanos, J. A., Montiel, J., Neira, J., and Tardós, J. D. The spmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, **15** (1999), 948. [Cited on page 17]

[41] Paskin, M. A. Thin junction tree filters for simultaneous localization and mapping. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (2003). [Cited on page 17]

[42] Schön, T. B. and Lindsten, F. Manipulating the multivariate gaussian density. Tech. rep. (2011). [Cited on page 17]

[43] Thrun, S., Liu, Y., Koller, D., Ng, A. Y., Ghahramani, Z., and Durrant-Whyte, H. Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, (2004). [Cited on page 17]

[44] Murphy, K. P. et al. Bayesian map learning in dynamic environments. In *NIPS* (1999). [Cited on page 18]

[45] Strasdat, H., Montiel, J. M., and Davison, A. J. Visual slam: why filter? *Image and Vision Computing*, **30** (2012), 65. [Cited on page 18]

[46] Klein, G. and Murray, D. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 225–234. IEEE (2007). [Cited on page 18]

[47] Strasdat, H., Montiel, J., and Davison, A. J. Scale drift-aware large scale monocular slam. *Robotics: Science and Systems VI*, (2010). [Cited on page 18]

[48] Lim, J., Frahm, J.-M., and Pollefeys, M. Online environment mapping. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 3489–3496. IEEE (2011). [Cited on page 18]

[49] Lu, F. and Milios, E. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, (1997). [Cited on pages 18, 20]

[50] Olson, E., Leonard, J., and Teller, S. Fast iterative alignment of pose graphs with poor initial estimates. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on* (2006). [Cited on pages 18, 20]

[51] Dellaert, F. and Kaess, M. Square root sam: Simultaneous localization and mapping via square root information smoothing. *Int. J. Rob. Res.*, (2006). [Cited on pages 18, 20]

[52] Gutmann, J.-S. and Konolige, K. Incremental mapping of large cyclic environments. In *Computational Intelligence in Robotics and Automation, 1999. CIRA'99. Proceedings. 1999 IEEE International Symposium on* (1999). [Cited on page 20]

[53] Bosse, M., Newman, P., Leonard, J., Soika, M., Feiten, W., and Teller, S. An atlas framework for scalable mapping. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on* (2003). [Cited on page 20]

[54] Estrada, C., Neira, J., and Tardós, J. D. Hierarchical slam: real-time accurate mapping of large environments. *Robotics, IEEE Transactions on*, (2005). [Cited on page 20]

[55] Olson, E. B. Robust and efficient robotic mapping. (2008). [Cited on page 20]

[56] Neira, J. and Tardós, J. D. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on robotics and automation*, **17** (2001), 890. [Cited on page 20]

[57] Nüchter, A., Lingemann, K., Hertzberg, J., and Surmann, H. 6d slam with approximate data association. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on* (2005). [Cited on page 20]

[58] Ranganathan, A., Kaess, M., and Dellaert, F. Loopy SAM. In *Proc. of the Int. Conf. on Artificial Intellignece (IJCAI)* (2007). [Cited on page 20]

[59] Kaess, M., Ranganathan, A., and Dellaert, F. isam: Fast incremental smoothing and mapping with efficient data association. In *Robotics and Automation, 2007 IEEE International Conference on* (2007). [Cited on page 20]

[60] Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., and McDonald, J. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia (2012). [Cited on page 20]

[61] Labbé, M. and Michaud, F. Online global loop closure detection for large-scale multi-session graph-based slam. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2661–2666. IEEE (2014). [Cited on page 20]

[62] Mur-Artal, R., Montiel, J., and Tardós, J. D. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, **31** (2015), 1147. [Cited on page 20]

[63] Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. Elasticfusion: Dense slam without a pose graph. *Proc. Robotics: Science and Systems, Rome, Italy*, (2015). [Cited on page 20]

[64] Dellaert, F. Factor graphs and gtsam: A hands-on introduction. (2012). [Cited on page 20]

[65] Agarwal, S. and Mierle, K. *Ceres Solver: Tutorial & Reference*. Google Inc. (2016). [Cited on page 20]

[66] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H., and Davison, A. J. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359 (2013). [Cited on page 20]

[67] Howard, A., Mataric, M. J., and Sukhatme, G. Relaxation on a mesh: a formalism for generalized localization. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2, pp. 1055–1060. IEEE (2001). [Cited on page 20]

[68] Frese, U., Larsson, P., and Duckett, T. A multilevel relaxation algorithm for simultaneous localization and mapping. *Robotics, IEEE Transactions on*, (2005). [Cited on page 20]

[69] Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., and Vincent, R. Efficient sparse pose adjustment for 2d mapping. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 22–29. IEEE (2010). [Cited on page 20]

[70] Grisetti, G., Stachniss, C., and Burgard, W. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, **10** (2009), 428. [Cited on page 20]

[71] Thrun, S. and Montemerlo, M. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, **25** (2006), 403. [Cited on page 21]

[72] Thrun, S., Burgard, W., and Fox, D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 321–328. IEEE (2000). [Cited on page 21]

[73] Nobili, S., Dominguez, S., Garcia, G., and Philippe, M. 16 Channels Velodyne Versus Planar LiDARs Based Perception System for Large Scale 2D-SLAM. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 7th Workshop on Planning, Perception and Navigation for Intelligent Vehicles, St Louis, MO, USA 2009.* Hamburg, Germany (2015). [Cited on pages 21, 119]

[74] Diosi, A. and Kleeman, L. Fast laser scan matching using polar coordinates. *The International Journal of Robotics Research*, **26** (2007), 1125. [Cited on page 21]

[75] Olson, E. B. Real-time correlative scan matching. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 4387–4393. IEEE (2009). [Cited on page 21]

[76] Moosmann, F. and Stiller, C. Velodyne slam. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pp. 393–398. IEEE (2011). [Cited on page 21]

[77] Wolcott, R. W. and Eustice, R. M. Visual localization within lidar maps for automated urban driving. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 176–183. IEEE (2014). [Cited on page 21]

[78] Wolcott, R. W. and Eustice, R. M. Fast lidar localization using multiresolution gaussian mixture maps. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2814–2821. IEEE (2015). [Cited on page 21]

[79] Cummins, M. and Newman, P. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, **30** (2011), 1100. [Cited on page 22]

[80] Sattler, T., Leibe, B., and Kobbelt, L. Fast image-based localization using direct 2d-to-3d matching. In *2011 International Conference on Computer Vision*, pp. 667–674. IEEE (2011). [Cited on page 22]

[81] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, **60** (2004), 91. [Cited on pages 22, 23, 31]

[82] Fischler, M. A. and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, **24** (1981), 381. [Cited on pages 22, 30]

[83] Korah, T., Medasani, S., and Owechko, Y. Strip histogram grid for efficient lidar segmentation from urban environments. In *CVPR 2011 WORKSHOPS*, pp. 74–81. IEEE (2011). [Cited on pages 22, 120]

[84] Morton, R. D. and Olson, E. Positive and negative obstacle detection using the hld classifier. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1579–1584. IEEE (2011). [Cited on pages 22, 133]

[85] Montemerlo, M., et al. Junior: The Stanford entry in the Urban Challenge. *J. Field Robot.*, **25** (2008), 569. [Cited on page 22]

[86] Sima, A. A. and Buckley, S. J. Optimizing sift for matching of short wave infrared and visible wavelength images. *Remote Sensing*, **5** (2013), 2037. [Cited on pages vii, 23]

[87] Lowe, D. G. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157. Ieee (1999). [Cited on page 23]

[88] Lindeberg, T. Scale-space theory: A basic tool for analysing structures at different scales. In *Journal of applied statistics*. Citeseer (1994). [Cited on page 23]

[89] Ke, Y. and Sukthankar, R. Pca-sift: A more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, pp. II–506. IEEE (2004). [Cited on page 24]

[90] Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pp. 404–417. Springer (2006). [Cited on pages vii, 24]

[91] Harris, C. and Stephens, M. A combined corner and edge detector. In *Alvey vision conference*, vol. 15, p. 50. Citeseer (1988). [Cited on pages viii, 25]

[92] Lazebnik, S. Corner detection (2016). Available from: `http://slazebni.cs.illinois.edu/spring16/lec08_corner.pdf`. [Cited on pages viii, 25]

[93] Rosten, E. and Drummond, T. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pp. 430–443. Springer (2006). [Cited on page 25]

[94] Rosten, E., Porter, R., and Drummond, T. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, **32** (2010), 105. [Cited on page 25]

[95] Rusu, R. B., Marton, Z. C., Blodow, N., and Beetz, M. Learning informative point classes for the acquisition of object model maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pp. 643–650. IEEE (2008). [Cited on page 27]

[96] Wahl, E., Hillenbrand, U., and Hirzinger, G. Surflet-pair-relation histograms: a statistical 3d-shape representation for rapid classification. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pp. 474–481. IEEE (2003). [Cited on page 27]

[97] Rusu, R. B., Blodow, N., and Beetz, M. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 3212–3217. IEEE (2009). [Cited on pages viii, 27, 28]

[98] Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2155–2162. IEEE (2010). [Cited on page 27]

[99] Aldoma, A., Vincze, M., Blodow, N., Gossow, D., Gedikli, S., Rusu, R. B., and Bradski, G. Cad-model recognition and 6dof pose estimation using 3d cues. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 585–592. IEEE (2011). [Cited on page 28]

[100] Wohlkinger, W. and Vincze, M. Ensemble of shape functions for 3d object classification. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pp. 2987–2992. IEEE (2011). [Cited on page 28]

[101] Li, Y. and Olson, E. A general purpose feature extractor for light detection and ranging data. *Sensors*, **10** (2010), 10356. [Cited on page 28]

[102] Tomasi, C. and Kanade, T. *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh (1991). [Cited on page 28]

[103] Li, Y. and Olson, E. Structure tensors for general purpose lidar feature extraction. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* (2011). [Cited on page 28]

[104] Lazebnik, S., Schmid, C., and Ponce, J. A sparse texture representation using local affine regions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **27** (2005), 1265. [Cited on pages viii, 29]

[105] Irani, S. and Raghavan, P. Combinatorial and experimental results for randomized point matching algorithms. In *Proceedings of the twelfth annual symposium on Computational geometry*, pp. 68–77. ACM (1996). [Cited on page 30]

[106] Chen, C.-S., Hung, Y.-P., and Cheng, J.-B. Ransac-based darces: A new approach to fast automatic registration of partially overlapping range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21** (1999), 1229. [Cited on page 30]

[107] Papazov, C. and Burschka, D. Stochastic optimization for rigid point set registration. In *International Symposium on Visual Computing*, pp. 1043–1054. Springer (2009). [Cited on page 30]

[108] Papazov, C. and Burschka, D. Stochastic global optimization for robust point set registration. *Computer Vision and Image Understanding*, **115** (2011), 1598. [Cited on page 30]

[109] Diez, Y., Martí, J., and Salvi, J. Hierarchical normal space sampling to speed up point cloud coarse matching. *Pattern Recognition Letters*, **33** (2012), 2127. [Cited on page 30]

[110] Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H. Robust global registration. In *Symposium on geometry processing*, vol. 2, p. 5 (2005). [Cited on page 30]

[111] Cheng, Z.-Q., Chen, Y., Martin, R. R., Lai, Y.-K., and Wang, A. Supermatching: Feature matching using supersymmetric geometric constraints. *IEEE Transactions on Visualization and Computer graphics*, **19** (2013), 1885. [Cited on page 30]

[112] Albarelli, A., Rodola, E., and Torsello, A. Loosely distinctive features for robust surface alignment. In *European Conference on Computer Vision*, pp. 519–532. Springer (2010). [Cited on page 30]

[113] Rodolà, E., Albarelli, A., Bergamasco, F., and Torsello, A. A scale independent selection process for 3d object recognition in cluttered scenes. *International journal of computer vision*, **102** (2013), 129. [Cited on page 30]

[114] Aiger, D., Mitra, N. J., and Cohen-Or, D. 4-points congruent sets for robust pairwise surface registration. *ACM Transactions on Graphics (TOG)*, **27** (2008), 85. [Cited on page 30]

[115] Mellado, N., Aiger, D., and Mitra, N. J. Super 4PCS fast global pointcloud registration via smart indexing. In *Computer Graphics Forum*, vol. 33, pp. 205–215. Wiley Online Library (2014). [Cited on page 30]

[116] Horn, B. K., Hilden, H. M., and Negahdaripour, S. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, (1988). [Cited on pages 30, 96]

[117] Rusinkiewicz, S. and Levoy, M. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pp. 145–152. IEEE (2001). [Cited on page 31]

[118] Arun, K. S., Huang, T. S., and Blostein, S. D. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (1987), 698. [Cited on page 31]

[119] Horn, B. K. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, **4** (1987), 629. [Cited on page 31]

[120] Walker, M. W., Shao, L., and Volz, R. A. Estimating 3-d location parameters using dual number quaternions. *CVGIP: image understanding*, **54** (1991), 358. [Cited on page 31]

[121] Bosse, M. and Zlot, R. Map matching and data association for large-scale two-dimensional laser scan-based slam. *The International Journal of Robotics Research*, **27** (2008), 667. [Cited on page 31]

[122] Reyes, L., Medioni, G., and Bayro, E. Registration of 3d points using geometric algebra and tensor voting. *International Journal of Computer Vision*, **75** (2007), 351. [Cited on page 31]

[123] Censi, A. Scan matching in a probabilistic framework. In *ICRA*, pp. 2291–2296. Citeseer (2006). [Cited on page 31]

[124] Magnusson, M., Nuchter, A., Lorken, C., Lilienthal, A. J., and Hertzberg, J. Evaluation of 3d registration reliability and speed-a comparison of icp and ndt. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 3907–3912. IEEE (2009). [Cited on page 31]

[125] Bosse, M. and Zlot, R. Continuous 3d scan-matching with a spinning 2d laser. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 4312–4319. IEEE (2009). [Cited on pages 31, 32, 34]

[126] Jost, T. and Hugli, H. A multi-resolution scheme icp algorithm for fast shape registration. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pp. 540–543. IEEE (2002). [Cited on page 31]

[127] Pan, Y., Dai, B., and Peng, Q. Fast and robust 3d face matching approach. In *2010 International Conference on Image Analysis and Signal Processing*, pp. 195–198. IEEE (2010). [Cited on pages 31, 32, 33]

[128] Fairfield, N. and Wettergreen, D. Evidence grid-based methods for 3d map matching. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 1637–1642. IEEE (2009). [Cited on page 31]

[129] Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2 (2010). [Cited on page 31]

[130] Stewart, C. V., Tsai, C.-L., and Roysam, B. The dual-bootstrap iterative closest point algorithm with application to retinal image registration. *IEEE transactions on medical imaging*, **22** (2003), 1379. [Cited on page 31]

[131] Tsai, C.-L., Li, C.-Y., Yang, G., and Lin, K.-S. The edge-driven dual-bootstrap iterative closest point algorithm for registration of multimodal fluorescein angiogram sequence. *IEEE transactions on medical imaging*, **29** (2010), 636. [Cited on page 31]

[132] Pomerleau, F., Breitenmoser, A., Liu, M., Colas, F., and Siegwart, R. Noise characterization of depth sensors for surface inspections. In *Applied Robotics for the Power Industry (CARPI), 2012 2nd International Conference on*, pp. 16–21. IEEE (2012). [Cited on page 32]

[133] Pomerleau, F., Colas, F., and Siegwart, R. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends in Robotics (FnTROB)*, **4** (2015), 1. [Cited on pages viii, 32, 33]

[134] Pulli, K. Multiview registration for large data sets. In *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on*, pp. 160–168. IEEE (1999). [Cited on pages 32, 33]

[135] Druon, S., Aldon, M.-J., and Crosnier, A. Color constrained icp for registration of large unstructured 3d color data sets. In *2006 IEEE International Conference on Information Acquisition*, pp. 249–255. IEEE (2006). [Cited on pages 32, 33]

[136] Censi, A. An icp variant using a point-to-line metric. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 19–25. IEEE (2008). [Cited on pages 32, 34]

[137] Kim, D. and Kim, D. A fast icp algorithm for 3-d human body motion tracking. *IEEE Signal Processing Letters*, **17** (2010), 402. [Cited on pages 32, 33]

[138] Champleboux, G., Lavallee, S., Szeliski, R., and Brunie, L. From accurate range imaging sensor calibration to accurate model-based 3d object localization. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pp. 83–89. IEEE (1992). [Cited on pages 32, 34]

[139] Feldmar, J. and Ayache, N. Locally affine registration of free-form surfaces. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 496–501. IEEE (1994). [Cited on page 32]

[140] Armesto, L., Minguez, J., and Montesano, L. A generalization of the metric-based iterative closest point technique for 3d scan matching. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 1367–1372. IEEE (2010). [Cited on page 32]

[141] Jost, T. and Hügli, H. Fast icp algorithms for shape registration. In *Joint Pattern Recognition Symposium*, pp. 91–99. Springer (2002). [Cited on page 32]

[142] Schutz, C., Jost, T., and Hugli, H. Multi-feature matching algorithm for free-form 3d surface registration. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, vol. 2, pp. 982–984. IEEE (1998). [Cited on page 32]

[143] Johnson, A. E. and Kang, S. B. Registration and integration of textured 3d data. *Image and vision computing*, **17** (1999), 135. [Cited on page 32]

[144] Godin, G., Rioux, M., and Baribeau, R. Three-dimensional registration using range and intensity information. In *Photonics for Industrial Applications*, pp. 279–290. International Society for Optics and Photonics (1994). [Cited on pages 32, 33]

[145] Yoshitaka, H., Hirohiko, K., Akihisa, O., and Shin'ichi, Y. Mobile robot localization and mapping by scan matching using laser reflection intensity of the sokuiki sensor. In *IECON 2006-32nd Annual Conference on IEEE Industrial Electronics*, pp. 3018–3023. IEEE (2006). [Cited on page 32]

[146] Chen, J. and Medioni, G. Object modeling by registration of multiple range images. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* (1991). [Cited on pages 33, 75]

[147] Gagnon, H., Soucy, M., Bergevin, R., and Laurendeau, D. Registration of multiple range views for automatic 3-d model building. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 581–586. IEEE (1994). [Cited on page 34]

[148] Bergevin, R., Soucy, M., Gagnon, H., and Laurendeau, D. Towards a general multi-view registration technique. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18** (1996), 540. [Cited on page 34]

[149] Gelfand, N., Ikemoto, L., Rusinkiewicz, S., and Levoy, M. Geometrically stable sampling for the icp algorithm. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pp. 260–267. IEEE (2003). [Cited on page 34]

[150] Bellekens, B., Spruyt, V., Berkvens, R., and Weyn, M. A survey of rigid 3d pointcloud registration algorithms. In *Fourth International Conference on Ambient Computing, Applications, Services and Technologies*, pp. 8–13. Citeseer (2014). [Cited on pages viii, 34]

[151] Feldmar, J. and Ayache, N. Rigid, affine and locally affine registration of free-form surfaces. *International journal of computer vision*, **18** (1996), 99. [Cited on page 34]

[152] Magnusson, M., Duckett, T., and Lilienthal, A. J. Scan registration for autonomous mining vehicles using 3D-NDT. *Journal on Field Robotics*, (2007). [Cited on page 35]

[153] Lee, J. M. Smooth manifolds. In *Introduction to Smooth Manifolds*, pp. 1–29. Springer (2003). [Cited on page 44]

[154] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (2012). [Cited on pages 68, 87]

[155] Pomerleau, F., Colas, F., Siegwart, R., and Magnenat, S. Comparing ICP variants on real-world data sets. (2013). [Cited on page 73]

[156] Blais, G. and Levine, M. D. Registering multiview range data to create 3D computer objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **17** (1995), 820. [Cited on page 74]

[157] Pauly, M., Gross, M., and Kobbelt, L. P. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization'02*, pp. 163–170. IEEE Computer Society (2002). [Cited on page 76]

[158] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. Ros: an open-source robot operating system. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA) Workshop on Open Source Software* (2009). [Cited on page 86]

[159] Rusu, R. B. and Cousins, S. 3d is here: Point cloud library (pcl). In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* (2011). [Cited on page 86]

[160] Pomerleau, F., Magnenat, S., Colas, F., Liu, M., and Siegwart, R. Tracking a depth camera: Parameter exploration for fast ICP. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 3824–3829. IEEE (2011). [Cited on pages 87, 91]

[161] Pomerleau, F., Liu, M., Colas, F., and Siegwart, R. Challenging data sets for point cloud registration algorithms. *The International Journal of Robotics Research*, **31** (2012), 1705. [Cited on pages 87, 88]

[162] Di Cicco, M., Iocchi, L., and Grisetti, G. Non-parametric calibration for depth sensors. *Robotics and Autonomous Systems*, **74** (2015), 309. [Cited on page 102]

[163] Olson, E. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3400–3407. IEEE (2011). [Cited on page 112]

[164] Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., and Kleiner, A. On measuring the accuracy of slam algorithms. *Autonomous Robots*, **27** (2009), 387. [Cited on page 113]

[165] Aurenhammer, F. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, **23** (1991), 345. [Cited on page 116]

[166] Olson, E. and Agarwal, P. Inference on networks of mixtures for robust robot mapping. *The International Journal of Robotics Research*, **32** (2013), 826. [Cited on pages 118, 124]

[167] Sünderhauf, N. and Protzel, P. Switchable constraints for robust pose graph slam. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1879–1884. IEEE (2012). [Cited on page 118]

[168] Guivant, J. E., Masson, F. R., and Nebot, E. M. Simultaneous localization and map building using natural features and absolute information. *Robotics and Autonomous Systems*, **40** (2002), 79. [Cited on page 133]

[169] Bartoli, A. and Sturm, P. Structure-from-motion using lines: Representation, triangulation, and bundle adjustment. *Computer vision and image understanding*, **100** (2005), 416. [Cited on page 157]

[170] Kelly, M. Computer Vision - The Integral Image (2010). Available from: `https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/`. [Cited on pages xiv, 166, 168]