Mapping, State Estimation, and Navigation for Quadrotors and Human-Worn Sensor Systems

Slawomir Grzonka

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

September 2011

Dekan:	Prof. Dr. Bernd Becker
Erstgutachter:	Prof. Dr. Wolfram Burgard
Zweitgutachter:	Prof. Dr. Leonhard Reindl
Tag der Disputation:	22.12.2011

Zusammenfassung

Die technische Unterstützung für Ersthelfer, die einen gefährlichen Bereich erkunden müssen um Verletzte zu finden und zu retten, während sie selbst lebensgefährliche Bereiche vermeiden sollen, ist jüngst immer mehr in den Fokus der Forschung gerückt.

Eine Möglichkeit, Ersthelfer bei ihrer Arbeit zu unterstützen, besteht darin, Roboter einzusetzen um für den Menschen lebensgefährliche Bereiche zu entdecken oder Verletzte zu orten. In diesem Zusammenhang werden fliegende Roboter eine Schlüsselrolle spielen, da sie dank ihrer Bewegungsfreiheit über Hindernisse hinwegfliegen können, die radbasierten Robotern den Weg versperren würden. Damit sie innerhalb von Gebäuden eingesetzt werden können, müssen fliegende Roboter in der Lage sein, stationär zu verweilen. All diese Voraussetzungen werden von Quadrotoren erfüllt. Leider ist es sehr zeit- und kostenintensiv, einem Menschen das Fernsteuern solcher Quadrotoren beizubringen. Zudem besteht während des Fernsteuerns immer ein großes Risiko den Roboter zu beschädigen, da innerhalb von Gebäuden meist nur wenig Platz zum Maneuvrieren ist und die Beschaffenheit der Umgebung einen schwerwiegenden Einfluss auf die Funkverbindung haben kann. Daher muss der fliegende Roboter in der Lage sein über einen längeren Zeitraum selbstständig zu operieren. In diesem Fall ist nur eine minimale Interaktion mit einem Menschen notwendig (z.B. die Eingabe des nächsten Wegpunktes den der Roboter selbstständig anfliegen soll).

Eine weitere Möglichkeit, Ersthelfer technisch zu unterstützen, besteht darin, Sensoren in deren Arbeitskleidung zu integrieren. Solche Sensoren können lebenswichtige Informationen über den Bereich liefern, in dem sich der Träger gerade aufhält. Im Zusammenhang mit Ersthelfern die in einem für sie unbekannten Gebäude operieren, können diese Informationen dazu verwendent werden, den Menschen im Notfall zum nächsten Ausgang zu leiten, insbesondere dann, wenn der Mensch beispielsweise durch Rauch oder Feuer orientierungslos ist. Weiterhin können solche Informationen verwendet werden, um die Einsatzteams besser zu koordinieren, indem beispielsweise verhindert wird, dass ein und der selbe Bereich mehrmals abgesucht wird.

Die technischen Systeme die wir bisher beschrieben haben (konkret: Sensoren, die auf einem Quadrotor montiert oder in die Arbeitskleidung integriert sind), müssen ihren aktuellen Zustand, unter anderem ihre Position im Raum, kennen. Um die eigene Position innerhalb eines Gebäudes bestimmen zu können ist im Allgemeinen eine Karte des Gebäudes nötig. Leider ist in den meisten Fällen im Voraus keine Karte des Gebäudes vorhanden und das System muss während des Einsatzes selbstständig eine Karte der Umgebung erstellen. Aufgrund der begrenzten verfügbaren Rechenleistung sind effiziente Kartierungsverfahren notwendig.

In dieser Arbeit entwickeln wir neue Technologien für effizientes Kartieren welche mit einer Reihe von verschiedenen Sensoren verwendet werden können. Weiterhin entwickeln wir ein Navigationssystem, welches einem kleinen fliegenden Roboter (Quadrotor) völlig autonomes

Fliegen ermöglicht. Zuletzt entwickeln wir ein Verfahren, um aus menschlichen Bewegungen Karten von Gebäuden zu erstellen. Hierbei trägt der Mensch einen Datenanzug, welcher aus einer Menge von Inertialsensoren besteht.

Im ersten Teil dieser Arbeit stellen wir neue Ansätze vor um die Trajektorie eines Roboters basierend auf seinen Messungen zu schätzen. Wir werden zeigen, dass unsere entwickelten Verfahren genaue Resultate liefern, jedoch zum Teil um mehrere Größenordnungen schneller arbeiten als alternative zeitgemäße Verfahren. Dies ermöglicht es einem (robotischen) Sensorsystem die eigene Trajektorie auf effiziente Weise zu rekonstruieren, was ihnen wiederum erlaubt, genaue Karten der Umgebung zu erstellen. Die in diesem Teil entwickelten Verfahren werden anschließend im zweiten Teil der Arbeit in den entsprechenden Navigationssystemen eingesetzt.

Im zweiten Teil der Arbeit entwickeln wir zwei Navigationssysteme für verschiedene Sensorkombinationen. Das erste System ermöglicht es einem kleinen Quadrotor, völlig selbstständig in Gebäuden zu fliegen. Zu den hier entwickelten Modulen gehören Positionskontrolle, Lokalisierung, Kartierung, Pfadplanung und Hindernisvermeidung. Außerdem präsentieren wir unser System, das dem Quadrotor erlaubt, Hindernisse unter sich zu kartieren. Das zweite Navigationssystem verwendet Daten von menschlichen Bewegungen, die mit einem Datenanzug, bestehend aus mehreren Inertialsensoren, aufgezeichnet wurden. Wir entwickeln ein Verfahren welches zuverlässig und genau die Trajektorie des Menschen der den Anzug trägt, rekonstruiert und sowohl eine geometrische als auch eine topologische Karte der Umgebung aufbaut. Das Verfahren verwendet hierzu nur menschliche Bewegungen und daraus erkannte Aktivitäten. Der Träger des Anzugs muss daher keine zusätzlichen Sensoren, wie Kameras oder Laser Scanner, tragen. Insbesondere würden letztere in Bereichen mit starker Rauchentwicklung oder mit Feuer, nicht zuverlässig funktionieren.

Obwohl wir unsere Arbeit für den Einsatzbereich von Ersthelfern motiviert haben, können unsere entwickelten Verfahren für eine große Anzahl von verschiedenen Anwendungen verwendet werden und sind nicht auf das oben genannte Einsatzgebiet beschränkt.

Abstract

Technical support for first responders, who have to explore hazardous environments to locate and rescue victims and thereby avoiding dangerous areas, has recently gained a substantial interest in the research community.

One possibility to assist first responders is by using robots to explore the environment and detect dangerous areas or locate victims. Flying robots are envisioned to play a key role in this context as their increased mobility allows them to fly over obstacles where wheeled robots get stuck. In order to be able to operate indoors, the flying robot should be able to keep a stationary pose. All these prerequisites are met by quadrotors. Unfortunately, teaching human personnel to remotely steer such a flying platform is time intensive and costly. Additionally, manual piloting bears the risk of damaging the robot due to the confined space indoors and due to environmental conditions that can have a severe impact on the quality of the radio link. The flying robot therefore needs to be able to operate autonomously over an extended period of time. In this case, only minimal input from a human (e.g., the next location the robot should fly to) is required.

Another possibility to support first responders is by using sensor systems that are integrated into their garment. Such sensors can provide vital information about the current location of the wearer or an approximate map of the environment. In the context of first responders operating in an unknown building, this information could be used to re-route the wearer to the nearest exit in case of emergency, especially if environmental conditions like smoke and fire elicit confusion among first responders. Even more, this kind of information can be employed in a search and rescue mission by improving the delegation of different teams, i.e., by avoiding searching the same area multiple times.

However, systems like the ones described so far (i.e., sensor systems mounted on a quadrotor or integrated into the garment) need to be aware of their current state, including their own location. To estimate the location indoors, a map of the environment is needed in most cases. In general, this map is not known beforehand and the (robotic) system needs to build a map of the environment based on its sensor measurement during the mission. Due to the limited computational power available, efficient mapping techniques are mandatory.

In this thesis we develop novel technologies for efficient mapping that can be used with a variety of sensors. We furthermore develop a navigation system that enables a small-sized flying robot (quadrotor) to fly autonomously indoors. Finally, we develop an approach to map indoor environments based on human motion recorded with a data suit, i.e., an embedded sensor system consisting of several inertial measurement units worn by the human.

In the first part of this thesis, we present an innovative technique for estimating the trajectory of a robot, given its observations. We will demonstrate that compared to other state-of-the-art ap-

proaches our approach is up to several orders of magnitude faster without any loss in accuracy. This allows embedded systems to efficiently and accurately recover their trajectory and thus allows them to build accurate maps of the environment. This general framework is subsequently used in the second part of the thesis in the corresponding embedded sensor systems.

In the second part, we develop two navigation systems for different types of sensor setups. The first navigation system enables a small-sized quadrotor to fly autonomously indoors. This includes pose control, localization, map building, path-planning, and obstacle avoidance. We also present a novel technique to map obstacles underneath the robot. The second system employs information recorded with a data suit consisting of several inertial measurement units worn by a human. We develop a solution to recover the trajectory of the human and to build a geometrical as well as topological map of the environment. In all cases, we solely employ the motions and detected activities of the human. The wearer, therefore, does not need to carry any additional sensors like cameras or laser scanners that would also not work reliably in the case of environmental conditions like smoke and fire.

Above, we motivated our work in particular envisioned for first responders. However, it is important to note that the developed technologies can be applied to a variety of scenarios and are not restricted to the field of search and rescue.

Acknowledgements

It is a pleasure for me to thank all the people who made this thesis possible and I am indebted to many of my colleagues for their support. First of all I would like to thank my adviser Wolfram Burgard. He granted me an exceptional degree of freedom in developing my own ideas and at the same time helped me a lot by having the right balance between high expectations, encouragement, and practical guidance. I really enjoyed from heart being part of the AIS-team. I would also like to thank Leonhard Reindl for reviewing this thesis and acting as an referee.

I owe my deepest gratitude to Giorgio Grisetti for his excellent support and fruitful discussions. It was an honor for me to work with him and it is a matter of fact that this thesis would not be as it is without his advice.

Many thanks to all co-authors and colleagues for the fruitful discussions, their friendship, and the collaborations. In particular, I would like to thank Frederic Dijoux, Andreas Karwath, Rainer Kümmerle, Edwin Olson, Christian Plagemann, Axel Rottmann, Michael Ruhnke, Cyrill Stachniss, Bastian Steder, Gian Diego Tipaldi, and Martin Wehrle. I would also like thank all my other friends and (former) colleagues at the AIS lab for the great atmosphere and the productive collaborations: Kai Arras, Maximilian Beinhofer, Maren Bennewitz, Felix Endres, Barbara Frank, Jürgen Hess, Dominik Joho, Henrik Kretzschmar, Markus Kuderer, Boris Lau, Daniel Meyer-Delius, Jörg Müller, Patrick Pfaff, Christoph Sprunk, Jürgen Sturm, and Kai Wurm.

Furthermore, I would like to thank all the people who made the project muFly possible and for the ongoing collaborations: Roland Siegwart, Christian Bermes, Samir Bouabdallah, Janosh Nikolic, and Dario Schafroth from ETH in Zürich. Robert Hahn, Steffen Krumbholz, and Stefan Wagner from TU-Berlin. Alain Brenzikofer and Pascal Ferrat from CSEM in Zürich as well as Alexandre Pagès from CEDRAT in Cedex. Finally, special thanks to the great people from Xsens, especially Per Slyke, Henk Luinge, Christian Liedtke, and Kiman Velt.

I also would like to thank Susanne Bourjaillat, Kris Haberer, Michael Keser, and Daniela Wack for their administrative and technical support during my time in Freiburg.

Many thanks to all the people who have made their datasets publicly available. In particular, Dirk Hähnel, Rainer Kümmerle, Cyrill Stachniss, and Rudolph Triebel.

My deepest gratitude and thanks go to my family, Beate Garbers, and Julia Sander for their support and love they gave me during my stay in Freiburg.

This work has partly been supported by the European Community under contract number FP6-IST-034120 Micro/Nano based Systems and by the German Research Foundation (DFG) under contract number SFB/TR-8. Their support is gratefully acknowledged.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 3
- 1.2 Contributions to Open-Source Software . . . 4
- 1.3 Publications . . . 4
- 1.4 Collaborations . . . 6
- 1.5 Outline . . . 6

2 Notation **9**
- 2.1 Coordinate System . . . 9
- 2.2 Motion Composition and Spatial Uncertainty . . . 9
- 2.3 Quaternions . . . 11
- 2.4 Symbols and Abbreviations . . . 14

Part I Graph-Based Optimization for Efficient Mapping

3 Basics on Optimization through Error Minimization **17**
- 3.1 The Graph-Based Model Formulation . . . 18
- 3.2 General Least Squares Problem Formulation . . . 22
- 3.3 Preconditioned Gradient Descent-Based Approaches . . . 24
- 3.4 Stochastic Gradient Descent . . . 28
- 3.5 A Variant of Stochasic Gradient Descent for Efficient 2D Optimization . . . 30

4 Tree-Based Graph Optimization **37**
- 4.1 Tree Parametrization and 2D Graph Optimization . . . 38
- 4.2 Analysis of the Algorithm . . . 45
- 4.3 Distributing the Error in 3D . . . 50
- 4.4 Analysis of the Rotational Residual in 3D . . . 54
- 4.5 Node Reduction . . . 57
- 4.6 2D Experiments . . . 58
  - 4.6.1 Simulated Experiments on Large Data Sets . . . 58
  - 4.6.2 Real World Experiments . . . 61
- 4.7 3D Experiments . . . 63
  - 4.7.1 Simulated Experiments on Large Data Sets . . . 63
  - 4.7.2 Real World Experiments . . . 67
- 4.8 Related Work . . . 72
- 4.9 Conclusion . . . 74

Part II State Estimation, Navigation, and Mapping for Embedded Sensor Systems

5 Autonomous Indoor Flying using a Quadrotor Robot 77
5.1 Requirements for Autonomous Indoor Flying . . . 79
5.2 System Architecture . . . 80
5.3 Navigation System . . . 82
5.3.1 Incremental Motion Estimation . . . 82
5.3.2 Localization . . . 87
5.3.3 Simultaneous Localization and Mapping . . . 88
5.3.4 Altitude Estimation . . . 89
5.3.5 High-Level Control for Pose and Altitude . . . 94
5.3.6 Path Planning and Obstacle Avoidance . . . 95
5.4 Experiments . . . 97
5.4.1 Localization . . . 97
5.4.2 Simultaneous Localization and Mapping . . . 99
5.4.3 Multi-Level SLAM and Altitude Estimation . . . 106
5.4.4 Pose Control . . . 106
5.4.5 Path Planning and Obstacle Avoidance . . . 108
5.4.6 On-Board Power Generation using a Fuel-Cell Prototype . . . 109
5.5 Related Work . . . 111
5.6 Conclusion . . . 113

6 Activity-Based Indoor Mapping and Estimation of Human Trajectories 115
6.1 Hardware Architecture . . . 117
6.2 Feature Detection . . . 118
6.2.1 Door Handling Events . . . 118
6.2.2 Stair Detection . . . 121
6.3 Multi Hypothesis Tracking . . . 123
6.4 Simultaneous Localization and Mapping . . . 125
6.5 Room Segmentation and Approximate Mapping . . . 128
6.6 Overall System . . . 130
6.7 Experiments . . . 132
6.7.1 Trajectory Estimation . . . 132
6.7.2 Room Segmentation and Approximate Mapping . . . 144
6.8 Related Work . . . 145
6.9 Conclusion . . . 146

• • • • • •

7 Conclusions and Outlook 149

Chapter 1

Introduction

First responders, (i.e., fire fighters, police men, or human personnel from a technical relief agency) in a search and rescue mission, are envisioned to become one of the key applications of mobile robotics and embedded sensor systems. There exists several possibilities of technical support for first responders, who have to explore a dangerous environment (e.g., a burning house) to locate and rescue victims while in the meantime avoiding hazardous areas.

One possibility to support first responders is by using flying robots that are equipped with several sensors and can operate over an extended period of time. Within the context of search and rescue missions, these robots should be able to operate indoors. Another possibility of support are sensor systems that are embedded into the garment and therefore physically connected to the human. These systems could highly improve the quality of the daily work. Even more, they could protect humans by providing additional information about the environment or proposing the human a path by avoiding potential hazards in unknown environments.

The goal of this thesis is to provide novel approaches that will aid humans, for example first responders, in their daily work. We present two embedded robotic sensor systems. A fully autonomous indoor quadrotor (i.e., a flying robot, see Figure 1.1) and a system to map indoor environments based on human motion obtained from a data suit (see Figure 1.2).

Developing a navigation system that enables a quadrotor to fly autonomously in indoor environments is highly challenging. The reason is its limited payload, the high dynamics of this flying platform and the confined space around the robot. These factors impose special requirements on robustness, computational complexity, and accuracy for the underlying navigation system. Similar requirements are also present in the case of a sensor system physically connected to the human. A system that is envisioned to improve the quality of human personnel or even help saving human lives needs to be highly accurate. Since this system is worn by a human, the limited payload prevents the usage of heavy sensors. Even more, in context of search and rescue, many sensors (e.g., cameras or lasers) will not work (reliably) in the presence of environmental conditions like smoke or fog. We therefore use for this scenario inertial measurement units only.

However, both systems need an accurate estimate of their current state. Indeed, state estimation is said to be one of the most important prerequisites for a truly autonomous system. In the work presented here, the state also contains the current pose of the agent (i.e., robot or human) in the environment. Since the agent has in general no information about the environment beforehand, the system is required to construct a map of the surroundings during the mission. This

Figure 1.1: One goal of this thesis: a navigation system enabling fully autonomous indoor flights using a small-sized quadrotor robot. The image on the left shows a snapshot of the internal state of our navigation system. Here, the quadrotor is flying autonomously in a cluttered office room. The free space around the robot is seriously confined, imposing high demands on pose stability, state estimation, and control. The image in the bottom left shows the office room from a similar view point as the snapshot. Our quadrotor is shown on the right.

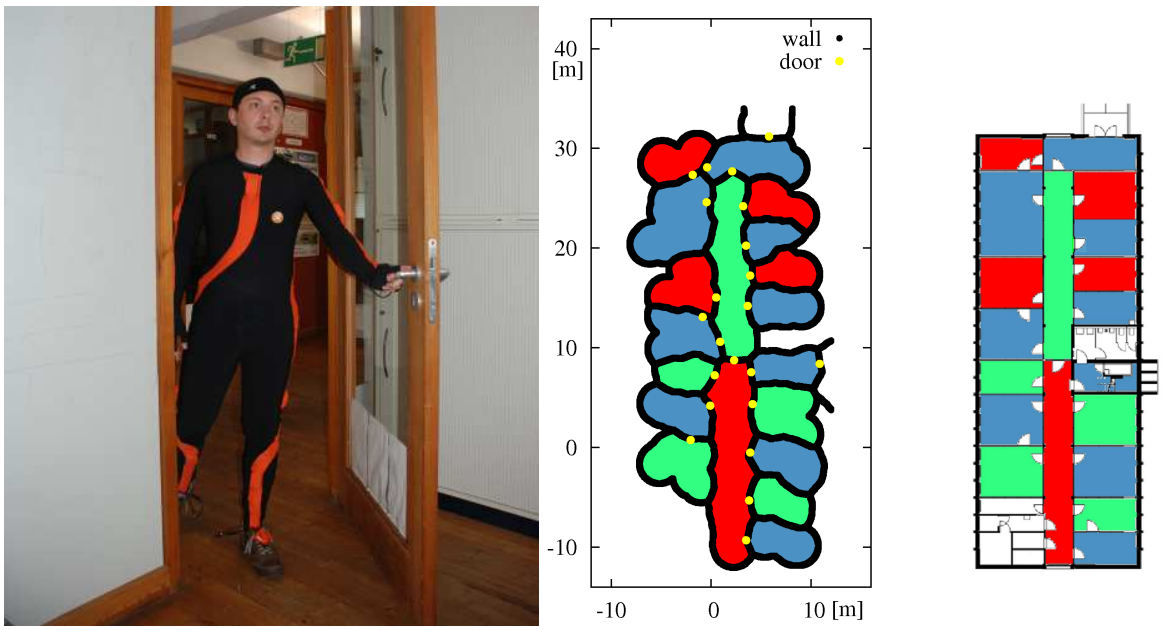

Figure 1.2: One goal of this thesis: a framework for mapping indoor environments based on human motion and activity detected given data from a data suit. The left image shows the user wearing the data suit. The middle image depicts the outcome of our approach. Based on the detection and tracking of doors and stairs we can estimate the most likely trajectory of the subject. Moreover, we can estimate a geometrical and topological map of the environment (middle image). Our estimated floor plan accurately resembles the floor plan of the same building (right image). Note that we used three different colors in total for the topological representation of the environment for better readability.

problem is known as simultaneous localization and mapping (SLAM) and has been in focus of research during the past decades. In our work, we focus on estimating the full trajectory of the agent given all observations, i.e., calculating a solution to the full SLAM problem. In this case, we address the problem by dividing the SLAM approach into two parts, namely the front-end SLAM and the back-end SLAM. The front-end is application dependent and aims to calculate the incremental trajectory of the agent. It is furthermore responsible for recognizing previously encountered parts of the environment (also known as *loop closures*). This type of information can then be used by a back-end SLAM system to estimate the most likely trajectory

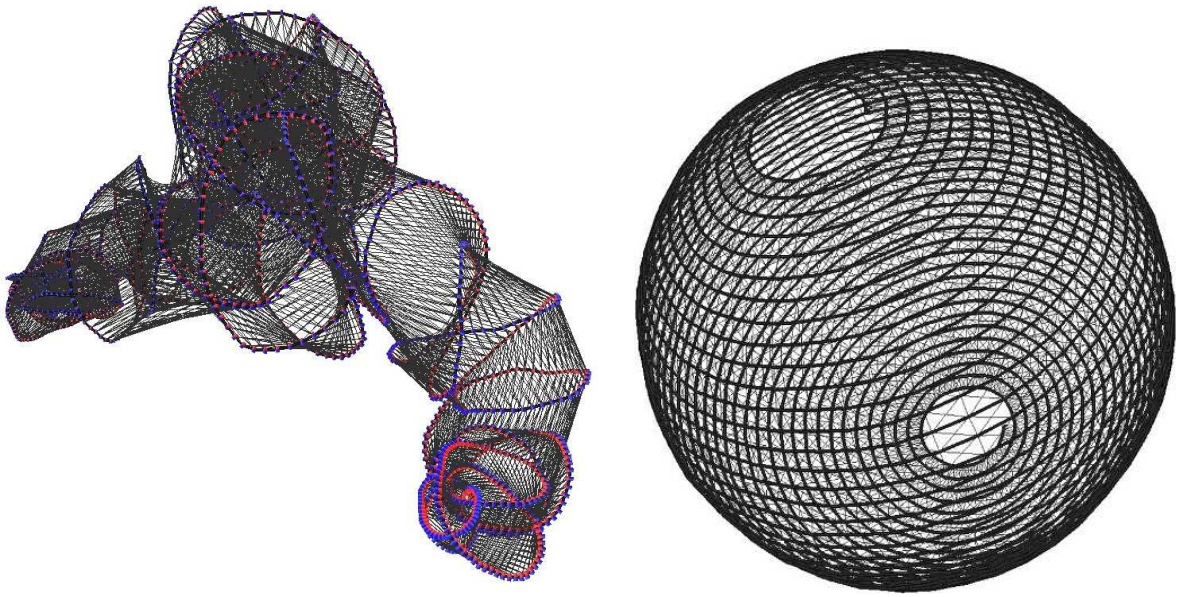

Figure 1.3: One goal of this thesis: a general framework for graph-based optimization. Given a graph consisting of nodes (i.e., robot poses) and edges (i.e., robot observations) the goal is to estimate a configuration of the nodes that minimizes the overall error in the system and accurately recovers the trajectory of the robot. In the example shown above, the robot was moving on a sphere. The left image depicts the trajectory of the robot given it's measured incremental motions only. The right image shows the result using our approach where the estimated trajectory accurately resembles the true trajectory taken by robot.

of the agent (given all sensor measurements). In this thesis, we model the agent's trajectory as a graph consisting of nodes and edges. Nodes in the graph represent agent positions in distinct time steps, whereas edges between two nodes encode an observation made about the corresponding locations. Again, these observations could be incremental motion estimates as well as a detected loop closures. Due to noisy measurements as well as the accumulation of errors over time, there exist in general different sensor readings about the same location that do not match the current configuration. We therefore need a technique able to estimate a configuration of the nodes that maximizes the overall observation likelihood encoded by the edges. This is also known as graph-based optimization. It is important to note that the abstract representation (i.e., nodes and edges) allows us to decouple the back-end system from the overall application.

The back-end system of SLAM, however, is a computationally intensive part. Therefore, a goal of this thesis is to develop a robust and efficient graph optimization technique which can be used for environment modeling (see Figure 1.3). This in turn will allow an agent to estimate it's current state. In this thesis, we will use this developed frame-work as the back-end for both the navigation system of the flying robot and the SLAM system for mapping indoor environments based on human motion.

1.1 Contributions

With this work, we contribute to the field of robotics research in several ways. We develop a fast and accurate graph-optimization framework, that is one of the essential parts of a mapping system. We adapt and extend algorithms developed for wheeled robots to flying ones. These newly developed techniques allow for fully autonomous flights in indoor environments using

a quadrotor. We present a mapping system that is able to accurately and robustly recover the geometric structure of buildings and the subject's trajectory given his movements and detected activities only. In summary, we propose:

- a general framework to graph-based network optimization in 2D and 3D (Chapter 4)
- a navigation system for autonomous indoor flying using a quadrotor robot (Chapter 5)
- an approach to map indoor environments based on human motion and activity (Chapter 6)

1.2 Contributions to Open-Source Software

Parts of our software have been released as open-source. Publishing software as open-source allows other researchers to build upon our results, evaluate our approaches on different data and platforms as well as verify our results. In more detail, we published:

- TORO[1]. This framework implements our tree network optimization algorithm. It has been published under the Creative Commons license (Attribution-NonCommercial-ShareAlike). Furthermore we provided several data sets that have been used for evaluating our approach. This framework was developed in collaboration with Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard.
- The Quadrotor Navigation System[2] provides several software modules for autonomous indoor navigation using a quadrotor flying robot and has been published under the Creative Commons license (Attribution-NonCommercial-ShareAlike). Due to it's modular design, most parts of this software can also be used for different platforms like wheeled robots. This framework was developed in collaboration with Giorgio Grisetti and Wolfram Burgard.

1.3 Publications

This thesis is based on the work published in international journals and conference proceedings. The following list of publications is given in chronological order.

Journal Articles

- S. Grzonka, A. Karwath, F. Dijoux, and W. Burgard. Activity-based Estimation of Human Trajectories. In *IEEE Transactions on Robotics (T-RO)*, 28(1):234–245, 2012.
- S. Grzonka, G. Grisetti, and W. Burgard. A Fully Autonomous Indoor Quadrotor. In *IEEE Transactions on Robotics (T-RO)*, 28(1):90–100, 2012.
- S. Bouabdallah, C. Bermes, S. Grzonka, C. Gimkiewicz, A. Brenzikofer, R. Hahn, D. Schafroth, G. Grisetti, W. Burgard, and R. Siegwart. Towards Palm-Size Autonomous Helicopters. In *Journal of Intelligent & Robotic Systems (IRS)*, 61:1–27, 2011.

[1] http://www.openslam.org/toro
[2] http://www.openquadrotor.org

Conferences and Workshops

- B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place Recognition in 3D Scans Using a Combination of Bag of Words and Point Feature based Relative Pose Estimation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

- S. Grzonka, B. Steder, and W. Burgard. 3D Place Recognition and Object Detection using a Small-sized Quadrotor. In *Workshop on 3D Exploration, Mapping, and Surveillance with Aerial Robots at Robotics: Science and Systems (RSS)*, 2011.

- S. Grzonka, F. Dijoux, A. Karwath, and W. Burgard. Learning Maps of Indoor Environments Based on Human Activity. In *Spring Symposium Series of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2010.

- S. Bouabdallah, C. Bermes, S. Grzonka, C. Gimkiewicz, A. Brenzikofer, R. Hahn, D. Schafroth, G. Grisetti, W. Burgard, and R. Siegwart. Towards Palm-Size Autonomous Helicopters. In *Proc. of the International Conference and Exhibition on Unmanned Aerial Vehicles (UAV)*, 2010. **Best conference paper award**.

- S. Grzonka, F. Dijoux, A. Karwath, and W. Burgard. Mapping Indoor Environments Based on Human Activity. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010. **Finalist best student paper award**. **Finalist best paper award in cognitive robotics**.

- S. Grzonka, G. Grisetti, and W. Burgard. Towards a Navigation System for Autonomous Indoor Flying. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009. **Best conference paper award**.

- S. Grzonka, G. Grisetti, and W. Burgard. Autonomous Indoors Navigation using a Small-Size Quadrotor. In *Workshop Proc. of the International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2008.

- S. Grzonka, S. Bouabdallah, G. Grisetti, W. Burgard, and R. Siegwart. Towards a Fully Autonomous Indoor Helicopter. In *Workshop of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

- B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, and W. Burgard. Estimating Consistent Elevation Maps using Down-Looking Cameras and Inertial Sensors. In *Proc. of the Workshop on Robotic Perception at the International Conference on Computer Vision Theory and Applications*, 2008.

- G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient Estimation of Accurate Maximum Likelihood Maps in 3D. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

- B. Steder, G. Grisetti, C. Stachniss, S. Grzonka, A. Rottmann, and W. Burgard. Learning Maps in 3D using Attitude and Noisy Vision Sensors. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

Outside the scope of this thesis fall the following publications

- S. Grzonka, C. Plagemann, G. Grisetti, and W. Burgard. Look-ahead Proposals for Robust Grid-based SLAM with Rao-Blackwellized Particle Filters. In *International Journal of Robotics Research (IJRR)*, 02:191–200, 2009.
- K. Arras, S. Grzonka, M. Luber, and W. Burgard. Efficient People Tracking in Laser Range Data using a Multi-Hypothesis Leg-Tracker with Adaptive Occlusion Probabilities. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2008.
- S. Grzonka, C. Plagemann, G. Grisetti, and W. Burgard. Look-ahead Proposals for Robust Grid-based SLAM. In *Proc. of the International Conference on Field and Service Robotics (FSR)*, 2007.
- K.O. Arras, B. Lau, S. Grzonka, M. Luber, O. Martinez Mozos, D. Meyer-Delius, and W. Burgard. Range-Based People Detection and Tracking for Socially Aware Service Robots. In *Towards Service Robots for Everyday Environments*. Springer STAR series, 2012.

1.4 Collaborations

Parts of this thesis are the results of collaborations with other people and we would like to thank all the people who put hard work in the joint projects. Especially, our tree network optimization algorithm (Chapter 4) was developed in joint work with Giorgio Grisetti and Cyrill Stachniss. Learning indoor maps based on human activity (Chapter 6) was originally addressed in the co-supervised master thesis of Frederic Dijoux and extended in collaboration with Andreas Karwath.

1.5 Outline

This thesis is structured as follows. We first review some basic mathematical concepts needed for this work in Chapter 2, in particular the notation of motion composition and quaternions. Subsequently, we present our developed techniques in the next chapters.

As already mentioned in the introduction, both the navigation system for the flying robot as well as the SLAM system for mapping indoor environments based on human motion rely on our graph-based optimization. This graph-based optimization is the back-end of the corresponding SLAM system. We therefore first describe our developed tree-based network optimizer in *Part I: Graph-Based Optimization for Efficient Mapping*. We start by discussing the basics of optimization through error minimization in general and present the path-parametrized optimization (PPO) algorithm for two-dimensional robotic mapping (Chapter 3). We then develop a variant of PPO by introducing a novel parametrization of the nodes, extend our approach to three-dimensional error minimization in Chapter 4, and show that our approach yields accurate results up to several orders of magnitude faster than other state-of-the-art approaches.

We furthermore develop two navigation systems for embedded sensor systems in *Part II: State Estimation, Navigation, and Mapping for Embedded Sensor Systems* where our graph-based

optimization algorithm is used as the back-end SLAM system for efficient environment modeling. First, we present our developed techniques enabling fully autonomous indoor flights using a small-sized quadrotor in Chapter 5. This technologies include position control, multilevel SLAM, path-planning, and obstacle avoidance. Subsequently, we present a solution to map indoor environments based on human motion and detected activity in Chapter 6. We demonstrate that our proposed method is able to robustly and accurately recover the trajectory taken by a subject. Furthermore, we demonstrate that we are able to build approximate geometrical as well as topological maps of the environments that accurately resemble the floor plans of the building.

Finally, we recapitulate the contributions and the results of our work in Chapter 7, followed by a discussion of future work.

Chapter 2

Notation

Before describing our developed techniques in the next chapters we will first review some basic mathematical concepts. We start by introducing the coordinate system used in this work in the next section. Subsequently, we briefly describe motion composition and spatial uncertainty in Section 2.2, in particular the operators $\oplus$ and $\ominus$. In Section 2.3 we will review an alternative way to represent rotations, namely quaternions, that allows us to easily compute intermediate rotations by using spherical linear interpolation (slerp). Finally, we describe the symbols and abbreviations used within this work in Section 2.4.

2.1 Coordinate System

Throughout this thesis we use the right-handed coordinate system, i.e., x is pointing forwards, y is pointing to the left, and z is pointing upwards as illustrated in Figure 2.1. Additionally, we describe three dimensional rotations in Euler angles, namely roll (ϕ, rotation along the x-axis), pitch (θ, rotation along the y-axis), and yaw (ψ, rotation along the z-axis) and we assume each rotation to lie within $[-\pi, \pi)$.

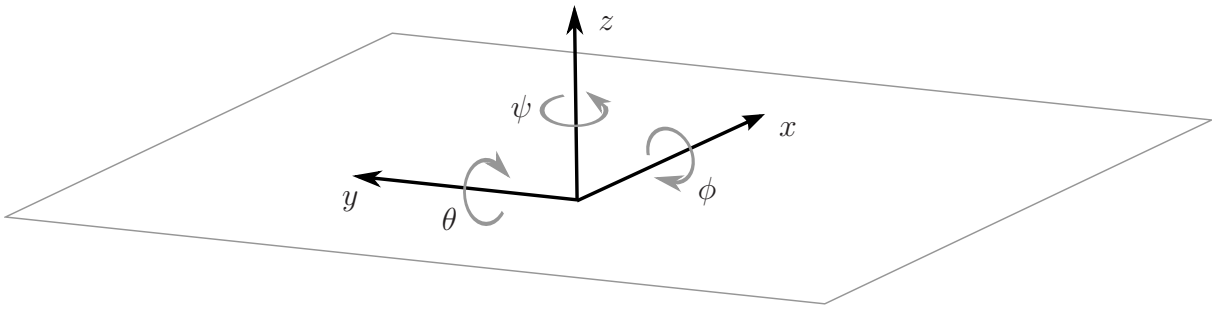

Figure 2.1: We use the right-handed coordinate system within this thesis. Here, x is pointing forwards, y is headed left, and z is pointing upwards. The Euler angles roll (ϕ), pitch (θ), and yaw (ψ) describe a (counter-clockwise) rotation along the x, y, and z axis, respectively.

2.2 Motion Composition and Spatial Uncertainty

We use the notation of motion composition consisting of the operators $\oplus$ ("oplus") and $\ominus$ ("ominus") as proposed by Smith and Cheeseman [138, 137]. These operators generalize the composition operators $+$ and $-$ from vectors to vectors represented in different reference frames,

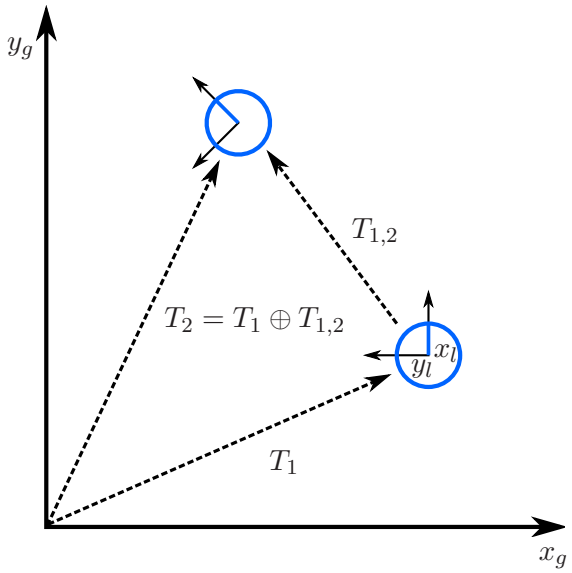

Figure 2.2: A 2D example for motion composition. Let T_1 denote the robots position and orientation with respect to the global reference frame (x_g, y_g). Let furthermore $T_{1,2}$ be the transformation of the robot in the local reference frame of T_1, i.e., given by the axes x_l and y_l. In this case, T_2 denotes the pose of the robot in global reference frame as the head-to-tail concatenation of T_1 and $T_{1,2}$ and is calculated as $T_2 = T_1 \oplus T_{1,2}$.

in particular with respect to a rotation. Figure 2.2 shows an small example. Consider a robot moving in 2D, i.e., a pose of the robot is represented by the coordinates x and y and the rotation ψ (i.e., the yaw). Here, the robot starts at T_1 and moves to some location (which we will call later T_2). However, the robot's sensors are only able to measure a relative transformation $T_{1,2}$, in the robot's local reference frame made of the axes x_l and y_l, whereas the starting position, T_1 is expressed in the global reference frame defined by the axes x_g and y_g. The goal now is to calculate the final location in the global reference frame. In other words, given T_1 and $T_{1,2}$, we want to calculate T_2. This kind of concatenation is also known as *head-to-tail* composition and we calculate T_2 as $T_2 = T_1 \oplus T_{1,2}$.

In general, however, the estimates of a spatial transformations are affected by noise and we assume that this noise is Gaussian. We furthermore assume that the transformations are mutually independent, i.e., the covariance between two different transformations is zero. In the remainder of this section, we will describe the operators $\oplus$ and $\ominus$ in detail for the 2D case (i.e., $\mathbf{x} = (x, y, \psi)$) and refer the reader to [138, 137] for a description of the full 3D case.

Let $\langle T_{ij}, \Sigma_{ij} \rangle$ be an uncertain spatial relationship from node (location) i to node (location) j. An uncertain spatial relationship consists of a transformation T_{ij} (the mean of the relationship) and the corresponding covariance Σ_{ij}. Let furthermore T_{ij} consist of the rotation R_{ij} and the translation t_{ij} with

$$R_{ij} = \begin{pmatrix} \cos\psi_{ij} & -\sin\psi_{ij} \\ \sin\psi_{ij} & \cos\psi_{ij} \end{pmatrix}, \text{ and} \tag{2.1}$$

$$\mathbf{t}_{ij} = (x_{ij}, y_{ij}). \tag{2.2}$$

The motion composition operator, $\oplus$, is then defined as

$$\langle T_{ik}, \Sigma_{ik} \rangle := \langle T_{ij}, \Sigma_{ij} \rangle \oplus \langle T_{jk}, \Sigma_{jk} \rangle, \text{ with} \tag{2.3}$$

$$T_{ik} := T_{ij} \oplus T_{jk}, \text{ consisting of} \tag{2.4}$$

$$R_{ik} = R_{ij} R_{jk}, \text{ and} \tag{2.5}$$

$$\mathbf{t}_{ik} = R_{ij}\mathbf{t}_{jk} + \mathbf{t}_{ij}. \tag{2.6}$$

Similarly, we have

$$\begin{aligned} \Sigma_{ik} &:= \Sigma_{ij} \oplus \Sigma_{jk} && (2.7)\\ &= J_{1\oplus}\Sigma_{ij}J_{1\oplus}^T + J_{2\oplus}\Sigma_{jk}J_{2\oplus}^T, && (2.8) \end{aligned}$$

with $J_{1\oplus}$ and $J_{2\oplus}$ defined as

$$\begin{aligned} \frac{\partial T_{ik}}{\partial T_{ij}} =: J_{1\oplus} &= \begin{pmatrix} 1 & 0 & -(y_{ij}-y_{ik}) \\ 0 & 1 & (-x_{ij}+x_{ik}) \\ 0 & 0 & 1 \end{pmatrix}, \text{ and} && (2.9)\\ \frac{\partial T_{ik}}{\partial T_{jk}} =: J_{2\oplus} &= \begin{pmatrix} \cos\psi_{ij} & -\sin\psi_{ij} & 0 \\ \sin\psi_{ij} & \cos\psi_{ij} & 0 \\ 0 & 0 & 1 \end{pmatrix} && (2.10)\\ &= \begin{pmatrix} R_{ij} & \begin{matrix} 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{pmatrix}. && (2.11) \end{aligned}$$

The inverse operator, $\ominus$ for a spatial transformation, $\langle T_{ij}, \Sigma_{ij}\rangle$ is defined as

$$\begin{aligned} \langle T_{ji}, \Sigma_{ji}\rangle &:= \ominus\langle T_{ij}, \Sigma_{ij}\rangle, \text{ with} && (2.12)\\ T_{ji} &:= \ominus T_{ij}, \text{ consisting of} && (2.13)\\ R_{ji} &= R_{ij}^T, \text{ and} && (2.14)\\ \mathbf{t}_{ji} &= -R_{ij}^T\mathbf{t}_{ij}. && (2.15) \end{aligned}$$

Additionally, we have

$$\begin{aligned} \Sigma_{ji} &:= \ominus\Sigma_{ij} && (2.16)\\ &= J_\ominus\Sigma_{ij}J_\ominus^T, \text{ with} && (2.17)\\ \frac{\partial T_{ji}}{\partial T_{ij}} =: J_\ominus &= \begin{pmatrix} -\cos\psi_{ij} & -\sin\psi_{ij} & y_{ji} \\ \sin\psi_{ij} & -\cos\psi_{ij} & -x_{ji} \\ 0 & 0 & -1 \end{pmatrix} && (2.18) \end{aligned}$$

Given the inverse operator, $\ominus$, for an uncertain transformation $\langle T_{ij}, \Sigma_{ij}\rangle$ we can finally define the inverse of a motion composition, i.e.,

$$\langle T_{jk}, \Sigma_{jk}\rangle = (\ominus\langle T_{ij}, \Sigma_{ij}\rangle) \oplus \langle T_{ik}, \Sigma_{ik}\rangle. \quad (2.19)$$

However, we will use the more intuitive notation of Lu and Milios [100] for the inverse, namely

$$\begin{aligned} \langle T_{jk}, \Sigma_{jk}\rangle = \langle T_{ik}, \Sigma_{ik}\rangle \ominus \langle T_{ij}, \Sigma_{ij}\rangle &:= (\ominus\langle T_{ij}, \Sigma_{ij}\rangle) \oplus \langle T_{ik}, \Sigma_{ik}\rangle, \text{ as well as} && (2.20)\\ T_{jk} = T_{ik} \ominus T_{ij} &:= (\ominus T_{ij}) \oplus T_{ik}. && (2.21) \end{aligned}$$

2.3 Quaternions

There are different ways to represent rotations in a mathematical way. Up to now, we described a rotation by a rotation matrix, R. However, we will also use a different representation, namely quaternions, since they allow us to easily calculate intermediate rotations. The following section gives a brief description about quaternions and we refer to Ken Shoemake's technical report [135] for more details.

A quaternion $\mathbf{q}$ is a vector of size 1×4 and describes a full three-dimensional rotation. More formally, a quaternion $\mathbf{q}$ is defined as (see also [135]):

$$\begin{aligned} \mathbf{q} &:= \langle w, \mathbf{v} \rangle && \text{,with } w \in \mathbb{R}, \mathbf{v} \in \mathbb{R}^3 && (2.22)\\ &= \langle w, \underbrace{(x, y, z)}_{\mathbf{v}} \rangle && \text{,with } w, x, y, z \in \mathbb{R} && (2.23)\\ &= w + \mathrm{i}x + \mathrm{j}y + \mathrm{k}z, && && (2.24) \end{aligned}$$

with w being the real part of the quaternion and $\mathbf{v}$ being the imaginary part respectively. In the equation above, i, j, and k denote the complex dimensions with

$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{i} \cdot \mathrm{j} \cdot \mathrm{k} = -1. \tag{2.25}$$

Note that from the equation above we also obtain

$$\begin{aligned} \mathrm{i} \cdot \mathrm{j} &= \mathrm{k}, \text{ and} && (2.26)\\ \mathrm{i} \cdot \mathrm{j} &= -\mathrm{j} \cdot \mathrm{i}. && (2.27) \end{aligned}$$

The set of quaternions forms a division ring (i.e., a field without commutativity for the multiplication operator) with the addition $+$ and the multiplication $\cdot$ as following. With respect to the addition operator, $+$, the set of quaternions forms a commutative group with

$$\begin{aligned} \mathbf{q}_1 + \mathbf{q}_2 &= \langle w_1, \mathbf{v}_1 \rangle + \langle w_2, \mathbf{v}_2 \rangle && (2.28)\\ &:= \langle w_1 + w_2, \mathbf{v}_1 + \mathbf{v}_2 \rangle && (2.29)\\ &= \mathbf{q}_2 + \mathbf{q}_1. && (2.30)\\ (\mathbf{q}_1 + \mathbf{q}_2) + \mathbf{q}_3 &= \mathbf{q}_1 + (\mathbf{q}_2 + \mathbf{q}_3). && (2.31)\\ \mathbf{q} + \mathbf{0} &= \mathbf{0} + \mathbf{q} && (2.32)\\ &= \mathbf{q}, \text{ with } \mathbf{0} = \langle 0, 0, 0, 0 \rangle, && (2.33)\\ \mathbf{q} + (-\mathbf{q}) &= 0. && (2.34) \end{aligned}$$

In terms of the multiplication operator, $\cdot$, the set of quaternions forms a group without commutativity, i.e., the following holds:

$$\begin{aligned} \mathbf{q}_1 \cdot \mathbf{q}_2 &= \langle w_1, \mathbf{v}_1 \rangle \cdot \langle w_2, \mathbf{v}_2 \rangle && (2.35)\\ &:= \langle w_1 \cdot w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2^T, \mathbf{v}_1 \times \mathbf{v}_2 + w_1 \cdot \mathbf{v}_2 + w_2 \cdot \mathbf{v}_1 \rangle. && (2.36) \end{aligned}$$

Here, $\mathbf{v}_1 \times \mathbf{v}_2$ denotes the cross product between the two vectors. Furthermore, we have

$$\begin{aligned} (\mathbf{q}_1 \cdot \mathbf{q}_2) \cdot \mathbf{q}_3 &= \mathbf{q}_1 \cdot (\mathbf{q}_2 \cdot \mathbf{q}_3), && (2.37)\\ s \cdot \mathbf{q} &= \mathbf{q} \cdot s, \text{ for } s \in \mathbb{R} && (2.38)\\ &= \langle s, (0, 0, 0) \rangle \cdot \langle w, \mathbf{v} \rangle && (2.39)\\ &= \langle s \cdot w, s \cdot \mathbf{v} \rangle. && (2.40)\\ 1 \cdot \mathbf{q} &= \mathbf{q} \cdot 1 && (2.41)\\ &= \mathbf{q}. && (2.42) \end{aligned}$$

Let $\mathbf{q}^*$ denote the conjugate of $\mathbf{q}$ with

$$\begin{aligned} \mathbf{q}^* &= \langle w, \mathbf{v} \rangle^* && (2.43)\\ &:= \langle w, -\mathbf{v} \rangle. && (2.44) \end{aligned}$$

We can now define the inverse quaternion $\mathbf{q}^{-1}$ of $\mathbf{q}$ as

$$\begin{aligned} \mathbf{q}^{-1} &:= \frac{\mathbf{q}^*}{\|\mathbf{q}\|}, \text{ with} && (2.45)\\ \|\mathbf{q}\| &:= \mathbf{q} \cdot \mathbf{q}^* && (2.46)\\ &= \mathbf{q}^* \cdot \mathbf{q} && (2.47)\\ &= w^2 + x^2 + y^2 + z^2. && (2.48)\\ & && (2.49) \end{aligned}$$

We also have

$$\begin{aligned} (s_1 \cdot \mathbf{q}_1 + s_2 \cdot \mathbf{q}_2) \cdot \mathbf{q}_3 &= s_1 \cdot \mathbf{q}_1 \cdot \mathbf{q}_3 + s_2 \cdot \mathbf{q}_2 \cdot \mathbf{q}_3, \text{ and} && (2.50)\\ \mathbf{q}_3 \cdot (s_1 \cdot \mathbf{q}_1 + s_2 \cdot \mathbf{q}_2) &= s_1 \cdot \mathbf{q}_3 \cdot \mathbf{q}_1 + s_2 \cdot \mathbf{q}_3 \cdot \mathbf{q}_2. && (2.51) \end{aligned}$$

For the sake of completeness, we also state the following properties:

$$\begin{aligned} (\mathbf{q}^*)^* &= \mathbf{q}, && (2.52)\\ (\mathbf{q}_1 \cdot \mathbf{q}_2)^* &= \mathbf{q}_2^* \cdot \mathbf{q}_1^* && (2.53)\\ (\mathbf{q}_1 + \mathbf{q}_2)^* &= \mathbf{q}_1^* + \mathbf{q}_2^* && (2.54)\\ \|\mathbf{q}_1 \cdot \mathbf{q}_2\| &= \|\mathbf{q}_1\| \cdot \|\mathbf{q}_2\| && (2.55) \end{aligned}$$

Given a unit quaternion $\mathbf{q}$, i.e., $\|\mathbf{q}\| = 1$, we can reformulate it as

$$\begin{aligned} \mathbf{q} &= \langle w, \mathbf{v} \rangle && (2.56)\\ &= \langle \cos\alpha, \sin\alpha \cdot \hat{\mathbf{v}} \rangle && (2.57) \end{aligned}$$

for a vector $\hat{\mathbf{v}} \in \mathbb{R}^3$ with $\|\hat{\mathbf{v}}\|_2 = 1$. In this case, the quaternion $\mathbf{q}$ describes a rotation of $2 \cdot \alpha$ along the rotational axis $\hat{\mathbf{v}}$.

Spherical Linear Interpolation (SLERP) Using quaternions to represent three-dimensional rotations allows us to calculate intermediate rotations using spherical linear interpolation.
Let $\mathbf{q}_1 \otimes \mathbf{q}_2$ denote the inner product of two unit quaternions, i.e., $\mathbf{q}_1 \cdot \mathbf{q}_2^T$ while treating both quaternions as vectors which results in

$$\begin{aligned} \mathbf{q}_1 \otimes \mathbf{q}_2 &= w_1 \cdot w_2 + x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 && (2.58)\\ &= \cos\beta, \text{ for a } \beta \in [-\pi, \pi). && (2.59) \end{aligned}$$

Given a scalar $u \in [0, 1]$, $\text{slerp}^*(\mathbf{q}_1, \mathbf{q}_2, u)$ calculates an intermediate rotation between the unit quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ as

$$\begin{aligned} \text{slerp}^*(\mathbf{q}_1, \mathbf{q}_2, u) &= \mathbf{q}_1 \cdot \frac{\sin(\beta \cdot (1-u))}{\sin\beta} + \mathbf{q}_2 \frac{\sin(\beta \cdot u)}{\sin\beta} && (2.60)\\ & && (2.61) \end{aligned}$$

However, since all unit quaternion form a sphere, there exits two paths on the sphere from $\mathbf{q}_1$ to $\mathbf{q}_2$. In order to get the "shorter" path, we finally calculate

$$\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, u) = \begin{cases} \text{slerp}^*(\mathbf{q}_1, \mathbf{q}_2, u) & \text{if } \|\mathbf{q}_1 - \mathbf{q}_2\| \leq \|\mathbf{q}_1 + \mathbf{q}_2\| \\ \text{slerp}^*(\mathbf{q}_1, -\mathbf{q}_2, u) & \text{else.} \end{cases} \quad (2.62)$$

In Section 4.3, we will use the notation $\text{slerp}(Q, w)$ for a given a rotation matrix Q. Here, $\text{slerp}(Q, w)$ is short for $\text{slerp}(\langle 1, (0,0,0) \rangle, \mathbf{q}, w)$ with $\mathbf{q}$ being the quaternion expressing the same rotation as the rotation matrix Q. Recall that $\langle 1, (0,0,0) \rangle$ is the neutral element of the multiplication, i.e., expressing a "zero-rotation".

2.4 Symbols and Abbreviations

The following tables summarize the symbols and abbreviations used in our work.

Symbol	Description
x	scalar variable
$\mathbf{x}, (\dots)$	vector
A	matrix
$\{\dots\}$	set
$\langle\dots\rangle$	ordered set (tuple)
$\mathbf{x}_i$	i-th element of vector $\mathbf{x}$
$A_{(ij)}$	element in the i-th row and j-th column of matrix A
$\mathbf{x}^t, A^t, t \in \mathbb{N}$	time indexed vector / matrix
$\mathbf{x}^T, A^T$	transpose of vector / matrix
I	identity matrix
Ω	information matrix
$p(\cdot)$	a probability density function
$p(A \mid B)$	conditional probability of event A, given evidence B
T_{ij}, δ_{ij}	a transformation from reference / node i to reference / node j
$\langle i, j\rangle$	a constraint between node i and node j
$\mathcal{N}(x; \mu, \sigma)$	a Gaussian distribution with mean μ and standard deviation σ.

Abbreviation	Description
DOF	degree of freedom
GPS	global positioning system
IMU	inertial measurement unit
MCL	monte carlo localization
MEMS	microelectromechanical systems
MHT	multi hypothesis tracker
ML	maximum likelihood
MLR	multi-level relaxation
NARF	normal aligned radial features
PDF	probability density function
PID	proportional integral differential
PGD	preconditioned gradient descent
PPO	path-parametrized optimization algorithm
RMSE	root mean square error
SGD	stochastic gradient descent
SURF	speeded up robust features
SLAM	simultaneous localization and mapping
SLERP	spherical linear interpolation
TORO	tree-based network optimization algorithm

Part I

Graph-Based Optimization for Efficient Mapping

Chapter 3

Basics on Optimization through Error Minimization

We review the general problem formulation for error minimization using a graph-based model. We then describe least squares error minimization in general and show that graph-based optimization is an instance of least squares minimization. Subsequently, we focus on preconditioned gradient descent and stochastic gradient descent for error minimization. Finally, we describe path parametrized optimization (PPO), a combination of both approaches entitled for 2D robotic mapping, that forms the basis to our tree-based network optimization (TORO) algorithm presented in the next chapter.

As stated in the introduction the goal of the first part of this thesis is to estimate a configuration of the environment, given the history of observations. A common way of doing this is through a graph-based model consisting of a set of nodes and edges. Although a graph can be used to model arbitrary relationships, our focus is on simultaneous localization and mapping (SLAM). In our case, a node of the graph represents the knowledge of an entity, like a robot pose or a landmark location, whereas an edge between two nodes reflect spatial constraints between those. These spatial constraints arise from observations like odometry, laser scan matching, vision tracking or another sensor, the robot is equipped with. Due to the noisy measurements, all estimates within the problem formulation are affected by noise as well. Therefore, a configuration of the nodes that satisfies a constraint is likely to be a bad configuration for another constraint and vice versa. In this case, we say that these constraints have a contrary effect on each other. In other words, we cannot compute a configuration of the nodes that perfectly satisfies all of the constraints in the general case. Our goal is therefore to calculate the configuration that *minimizes* the overall error introduced by the constraints, thus *maximizing* the probability of a configuration, given the data.

This chapter is structured as follows. We first discuss the graph-based model formulation in Section 3.1 and non-linear least squares in general in Section 3.2. We describe the theory of preconditioned gradient descent (PGD) with a special attention to Gauss Newton in Section 3.3. Subsequently, we describe stochastic gradient descent (SGD) in Section 3.4. Finally, we present PPO, a variant of SGD introduced by Olson *et al.* in Section 3.5. This approach, a combination of SGD and an approximation to Gauss Newton, forms the basis to our tree-based network optimization algorithm (TORO) for robotic mapping presented in the next chapter.

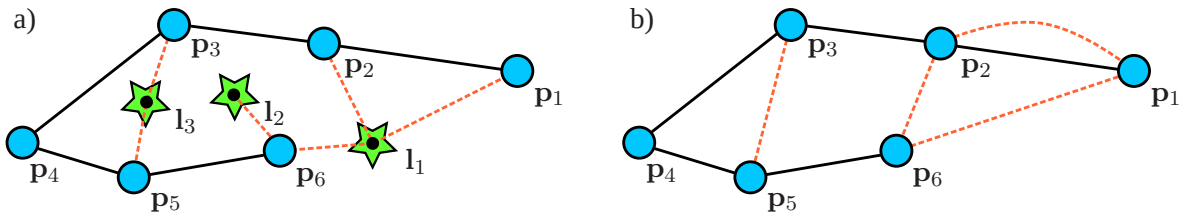

Figure 3.1: A simple graph consisting of robot poses $\mathbf{p}_1, \ldots, \mathbf{p}_6$ (blue circles) and landmarks $\mathbf{l}_1, \ldots, \mathbf{l}_3$ (green stars). Constraints between robot poses are visualized as black lines and constraints between a robot pose and a landmark are displayed as orange dashed lines in (a). The landmarks can be marginalized out of the estimation problem by transforming the edges into observations between robot poses. This is visualized in (b) where constraints between poses and landmarks from (a) are transformed into edges between robot poses, visualized by orange lines in (b).

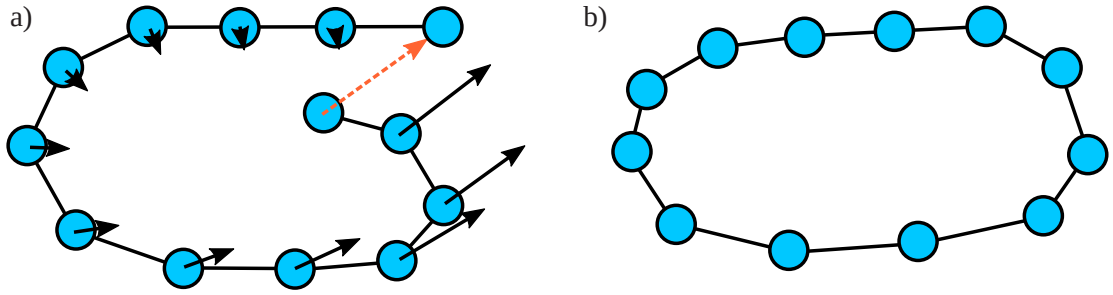

Figure 3.2: Example for optimization: The robot traverses a loop and re-localizes to a previously seen location indicated by the dashed orange arrow in (a). Minimizing the overall error in this network results in moving all nodes as visualized by the black solid arrows. The resulting optimal configuration of the nodes is shown in (b).

3.1 The Graph-Based Model Formulation

A common way of describing a SLAM problem is through its graph-based (also called network-based) model. In general, a Graph $G = (\mathbf{x}, \mathcal{C})$ consists of a set of nodes $\mathbf{x}$ and a set of edges $\mathcal{C}$. In our particular case, the nodes represent either robot poses or landmark locations, whereas edges between two nodes describe the spatial relation between those. Note, that our goal is to find the most likely configuration of the nodes, given the edges. In other words, we want to calculate the most likely configuration of the nodes explained by the data (i.e., satisfying the edges). Thus it is convenient to call the edges also *constraints*. For an edge connecting two robot poses, the spatial constraint can either arise from motion commands or incremental motion estimation. If an edge connects a robot pose and a landmark, the spatial constraint arises from an observation (e.g., visual features). Figure 3.1 (a) shows a small graph consisting of robot poses $\mathbf{p}_1, \ldots, \mathbf{p}_6$ (blue circles) and landmarks $\mathbf{l}_1, \ldots, \mathbf{l}_3$ (green stars). Here, constraints (edges) are visualized through lines for the case of connecting two robot poses (black lines) or indicating an observation of a landmark (dashed orange lines). In general, a constraint does not refer to a perfect observation or measurement but rather represents a distribution. In our case, we assume the error in the measurements to be affected by white noise only, i.e., we assume this distribution to be Gaussian. Thus, each constraint is made of a mean observation and the corresponding uncertainty (which is not visualized in Figure 3.1). As shown by Montemerlo *et al.* [105, 103] we can remove the landmarks from the estimation process by marginalizing them out of the problem formulation. Marginalizing out a landmark, however, results in connecting all nodes

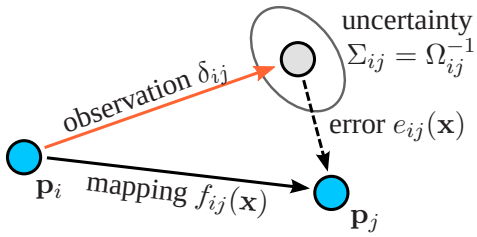

Figure 3.3: Terminology for describing a graph $G = (\mathbf{x}, \mathcal{C})$. The set of parameters $\mathbf{x}$ can be transformed into a set of poses $\mathbf{p}$, with $\mathbf{p}_i, \mathbf{p}_j \in \mathbf{p}$. An observation of node j seen from node i is denoted as δ_{ij} and is associated with an information matrix Ω_{ij} describing the uncertainty of the observation. The function f_{ij} maps the current configuration into a zero noise observation of node j seen from node i, thus computing the expected observation. The difference between the expected and the true observation is the error e_{ij} of the corresponding constraint.

from which this landmark has been observed. By doing so, the information encoded in the observation of the landmark was transformed into an observation made about a robot pose. This is visualized in Figure 3.1 (b). Note that the two constraints between node $\mathbf{p}_1$ and node $\mathbf{p}_2$ can be merged into one but this is omitted here for better visibility. As stated above, by optimizing the graph we seek to find the most likely configuration of the nodes, given the data (constraints). Consider for example the graph shown in Figure 3.2. Here, the robot traverses a cyclic path and re-localizes itself in a previously visited part of the environment. This is known in the literature as *loop closing*. The corresponding constraint is visualized by the dashed orange arrow in (a) where both poses connected through this arrow are the same. Optimizing this graph results in moving all nodes in order to minimize the overall error (indicated by the solid black arrows in (a)) resulting in the configuration shown in Figure 3.2 (b).
Assuming the constraints are affected by white noise we can describe a graph more formally using the following definitions (see also Figure 3.3):

- Let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be a vector of n parameters describing the configuration of the graph. Note that these parameters could for example be absolute poses $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$ or any arbitrary set of variables. In the latter case, we assume there exist a bijective function g mapping these variables to absolute poses (real world coordinates), thus we assume $g(\mathbf{x}) = \mathbf{p}$ and $g^{-1}(\mathbf{p}) = \mathbf{x}$. The parameters are the nodes in the graph structure.

- Let $\delta_{ij} \in \mathbb{R}^{k \times 1}$ be a measurement between node i and node j. It refers to a relative observation about node j seen from node i. Here, k is the dimension of the observation space. It is the mean of the Gaussian observation distribution.

- The uncertainty $\Sigma_{ij} \in \mathbb{R}^{k \times k}$ associated with the observation δ_{ij} is expressed by the information matrix $\Omega_{ij} = \Sigma_{ij}^{-1}$.

- The observation δ_{ij}, together with the information matrix Ω_{ij} form a *constraint*. These constraints are the edges in the graph structure. Note that we will omit to draw the uncertainty ellipsoid in upcoming figures for better readability.

- $\mathcal{C} = (\langle i_1, j_1 \rangle, \ldots, \langle i_m, j_m \rangle)$ is the set of all constraints and the associated information matrices, with $\langle i, j \rangle$ being short for $\langle \delta_{ij}, \Omega_{ij} \rangle$.

- Finally, $f_{ij}(\mathbf{x}) : \mathbb{R}^{\dim(\mathbf{x})} \rightarrow \mathbb{R}^{k \times 1}$ is a function mapping the current configuration to a zero noise observation of node j seen from node i. In other words, $f_{ij}(\mathbf{x})$ computes the *expected* observation given the current configuration of the nodes.

Having this definitions at hand we can formulate the optimization problem as follows. Given the set of poses and constraints encoded by the nodes and edges respectively we want to find the configuration that maximizes the likelihood given the data encoded in the edges. Thus, we want to find $\mathbf{x}^*$ with

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x} \mid \mathcal{C}), \tag{3.1}$$

where $p(\cdot)$ is an appropriate probability density function (pdf). In general it is inconvenient to evaluate $p(\mathbf{x} \mid \mathcal{C})$ since we need to reason about the state given the measurement (i.e., it is a diagnostic system). The idea is to apply Bayes' rule, transform the problem into a causal system and reason about the likelihood of the data, given the current state. Applying Bayes' rule to Eq. (3.1) leads to

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x} \mid \mathcal{C}) \tag{3.2}$$

$$\overset{\text{Bayes' rule}}{=} \underset{\mathbf{x}}{\operatorname{argmax}}\, \frac{p(\mathcal{C} \mid \mathbf{x})\, p(\mathbf{x})}{p(\mathcal{C})}, \tag{3.3}$$

where $p(\mathcal{C})$ is constant, since $\mathbf{x}$ is the random variable of our system. In addition, if no further prior information about the configuration space is available, we can assume $p(\mathbf{x})$ to be uniformly distributed. This allows us to simplify the equation above and formulate the goal to find the configuration $\mathbf{x}^*$, with

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathcal{C} \mid \mathbf{x}). \tag{3.4}$$

However, evaluating this equation is in general still unpractical, if not unfeasible. Therefore it is common within the robotics community to assume independence between individual constraints. We further assume that the constraints, and thus the measurement errors, are affected by white noise only. Although this assumption is violated to some degree in the real world (i.e., a sensor can have a bias in the estimate due to environmental conditions like temperature or humidity) we neglect the error that is introduced by this assumption. This allows us to simplify $p(\mathcal{C} \mid \mathbf{x})$ in Equation (3.4) to

$$p(\mathcal{C} \mid \mathbf{x}) = \prod_{\langle i,j \rangle \in \mathcal{C}} p(\langle i, j \rangle \mid \mathbf{x}). \tag{3.5}$$

Keeping this in mind we need to define the *error* and *residual* of a constraint. Given two nodes i , j and a constraint $\langle i, j \rangle = \langle \delta_{ij}, \Omega_{ij} \rangle$ between these we can define the error e_{ij} and the residual r_{ij} as

$$e_{ij}(\mathbf{x}) = f_{ij}(\mathbf{x}) - \delta_{ij} \tag{3.6}$$

$$r_{ij}(\mathbf{x}) = -e_{ij}(\mathbf{x}). \tag{3.7}$$

The error e_{ij} is also visualized in Figure 3.3. Observe that the residual is just the inverse of the error. In other words, moving node j along the direction of the residual r_{ij} will lower the error e_{ij} until the minimum $e_{ij} = r_{ij} = 0$ is reached. This condition is fulfilled when the observation δ_{ij} perfectly matches the configuration of the nodes i and j and is called the *equilibrium* of that constraint.

Due to the noisy measurements, however, we will in general never be able to reach the equilibrium of all constraints simultaneously. Therefore our goal is to find a configuration that

maximizes the overall likelihood. We will later see, that maximizing the overall likelihood is equal to minimizing the overall error introduced by the constraints. Recall that we assume the observation (Eq. (3.5)) to be Gaussian. Thus, the likelihood $p(\langle i,j\rangle \mid \mathbf{x})$ of a constraint $\langle i,j\rangle$ is

$$p(\langle i,j\rangle \mid \mathbf{x}) = \frac{1}{(2\pi)^{k/2}\,|\Sigma_{ij}|^{1/2}}\,\exp(-\frac{1}{2}(f_{ij}(\mathbf{x})-\delta_{ij})^T\Omega_{ij}(f_{ij}(\mathbf{x})-\delta_{ij})). \tag{3.8}$$

However, maximizing Eq. (3.5) is equal to minimize its negative log likelihood $-\ln(p(\mathcal{C}\mid\mathbf{x}))$. This leads to

$$-\ln(p(\mathcal{C}\mid\mathbf{x})) \overset{\text{Eq. (3.5)}}{=} -\ln\left(\prod_{\langle i,j\rangle\in\mathcal{C}} p(\langle i,j\rangle\mid\mathbf{x})\right) \tag{3.9}$$

$$= \sum_{\langle i,j\rangle\in\mathcal{C}} -\ln\left(p(\langle i,j\rangle\mid\mathbf{x})\right) \tag{3.10}$$

with

$$\begin{aligned}
-\ln(p(\langle i,j\rangle\mid\mathbf{x})) &= -\ln(\frac{1}{(2\pi)^{k/2}\,|\Sigma_{ij}|^{1/2}}\,\exp(-\frac{1}{2}(f_{ij}(\mathbf{x})-\delta_{ij})^T\Omega_{ij}(f_{ij}(\mathbf{x})-\delta_{ij})))\\
&= \ln((2\pi)^{k/2}\,|\Sigma_{ij}|^{1/2}) + \frac{1}{2}(f_{ij}(\mathbf{x})-\delta_{ij})^T\Omega_{ij}(f_{ij}(\mathbf{x})-\delta_{ij}))\\
&\propto (f_{ij}(\mathbf{x})-\delta_{ij})^T\Omega_{ij}(f_{ij}(\mathbf{x})-\delta_{ij}))\\
&= e_{ij}^T(\mathbf{x})\,\Omega_{ij}\,e_{ij}(\mathbf{x}) &(3.11)\\
&= r_{ij}^T(\mathbf{x})\,\Omega_{ij}\,r_{ij}(\mathbf{x}) &(3.12)\\
&=: \chi_{ij}^2(\mathbf{x}). &(3.13)
\end{aligned}$$

Note that $\chi_{ij}(\mathbf{x})$ is the Mahalanobis distance and represents the error between the prediction f_{ij} and the observation δ_{ij} as a multiple of standard deviations. The notation of χ_{ij}^2 has been chosen on purpose because the underlying likelihood is Gaussian and thus the squared Mahalanobis distance follows a χ_p^2 distribution of degree p, equal to the dimension of the error vector $e_{ij}(\mathbf{x})$. This allows us to rewrite Equation (3.9) to

$$-\ln\left(p(\mathcal{C}\mid\mathbf{x})\right) \overset{\text{Eq. (3.5)}}{=} -\ln\left(\prod_{\langle i,j\rangle\in\mathcal{C}} p(\langle i,j\rangle\mid\mathbf{x}\right) \tag{3.14}$$

$$\overset{\text{Eq. (3.10)}}{=} \sum_{\langle i,j\rangle\in\mathcal{C}} -\ln\left(p(\langle i,j\rangle\mid\mathbf{x})\right) \tag{3.15}$$

$$\overset{\text{Eq. (3.13)}}{=} \sum_{\langle i,j\rangle\in\mathcal{C}} \chi_{ij}^2(\mathbf{x}) \tag{3.16}$$

$$=: \chi^2(\mathbf{x}). \tag{3.17}$$

Recalling the original problem formulation, namely finding the configuration $\mathbf{x}^*$ that maximizes the overall observation likelihood, we can finally reformulate this problem into calculating the configuration $\mathbf{x}^*$ with

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\; p(\mathcal{C}\mid\mathbf{x}) \tag{3.18}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}}\; -\ln(p(\mathcal{C}\mid\mathbf{x})) \tag{3.19}$$

$$\overset{\text{Eq. (3.17)}}{=} \underset{\mathbf{x}}{\operatorname{argmin}}\; \chi^2(\mathbf{x}). \tag{3.20}$$

To sum up, maximizing the overall observation likelihood it is equal to find the configuration $\mathbf{x}^*$ that minimizes the observation error encoded in the constraints, namely χ^2. Unfortunately, finding this global minimum in general is very hard and for most minimization problems we can not even guarantee that a solution found by an algorithm is indeed the global minimum. We therefore relax this requirement and focus on calculating a solution that is a local minimum of the error function. In other words, we seek to find an $\mathbf{x}^*$ such that for a small positive number ϵ the following holds:

$$\forall \mathbf{x} : \|\mathbf{x} - \mathbf{x}^*\| < \epsilon \rightarrow \chi^2(\mathbf{x}^*) \leq \chi^2(\mathbf{x}). \tag{3.21}$$

Having this at hand we now need techniques for calculating $\mathbf{x}^*$ (and thus minimize the error). Since the error function in general is non-linear (f.e., due to rotations), we cannot calculate $\mathbf{x}^*$ in one step. Therefore it is common to use iterative approaches, i.e., adjusting the actual configuration $\mathbf{x}^t$ at time t to a new one, $\mathbf{x}^{t+1}$ that lowers the overall error. In the next section we will briefly review least squares in general and show that the graph-based model formulation for robotic mapping is an instance of it. Subsequently, we review preconditioned gradient descend-based and stochastic gradient descent which form the basis of the optimization approach proposed by Olson *et al.*, that in turn is the basis to our tree-based network optimizer (TORO) described in Chapter 4. Note that we will call Olson's approach also PPO throughout this work, that is short for path-parametrized optimization.

3.2 General Least Squares Problem Formulation

The general least squares problem can be formulated as follows (see also Fransen *et al.* [52]). Given m functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ with $m \geq n$ and the *error function* (also called *cost function*) $f = (f_1, \ldots, f_m)^T : \mathbb{R}^n \rightarrow \mathbb{R}^m$, our goal is to find $\mathbf{x}^*$ with

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} (F(\mathbf{x})), \text{ where} \tag{3.22}$$

$$F(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})^2 = f(\mathbf{x})^T f(\mathbf{x}) = \|f(\mathbf{x})\|^2. \tag{3.23}$$

In the special case where in the equation above $f(\mathbf{x})$ is of the form

$$f(\mathbf{x}) = \mathbf{b} - A\mathbf{x}, \tag{3.24}$$

with known vector $\mathbf{b} \in \mathbb{R}^m$ and known matrix $A \in \mathbb{R}^{m \times n}$, we call it a *linear* least squares problem, otherwise we refer to it as a *non-linear* least squares problem.

Note that least squares is a variant of the more general minimization problem, namely finding a minimum $\mathbf{x}^*$ for *some* objective function $\widehat{F}(\mathbf{x})$. In other words, a minimization problem is called least squares, if and only if $\widehat{F}(\mathbf{x}) = F(\mathbf{x})$. However, as already mentioned in the previous section, it is in general hard to find the global minimum of a function $F(\mathbf{x})$, especially on a real valued configuration space $\mathbb{R}^n$. Thus, it is common to solve a "simpler" problem, namely finding a *local* minimum of $F(\mathbf{x})$. In the remainder of this section, we will first show that graph-based optimization as formulated in the last section is indeed a least squares problem and subsequently focus on basic concepts of iterative methods for finding a local minimum for Equation 3.23. Indeed, all general methods for non-linear optimization are iterative.

Observe that our goal in graph optimization is to minimize the overall χ^2 error, that is

$$\chi^2(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} r_{ij}^T(\mathbf{x})\, \Omega_{ij}\, r_{ij}(\mathbf{x}) \tag{3.25}$$

Since the covariance matrix $\Sigma_{ij} = \Omega_{ij}^{-1}$ is positive definite also Ω_{ij} is positive definite [120] and we have:

$$\forall \langle i,j \rangle \in \mathcal{C} : \quad r_{ij}^T(\mathbf{x})\, \Omega_{ij}\, r_{ij}(\mathbf{x}) \geq 0, \text{and therefore} \tag{3.26}$$

$$\forall \langle i,j \rangle \in \mathcal{C} \;\; \exists h_{ij}(\mathbf{x}) : r_{ij}^T(\mathbf{x})\, \Omega_{ij}\, r_{ij}(\mathbf{x}) = h_{ij}(\mathbf{x})^2. \tag{3.27}$$

Here, the equality in Equation (3.26) originates from the fact that $r_{ij}(\mathbf{x})$ can be zero, i.e., when the equilibrium of that constraint is reached. Except of the different indexing between Equation (3.23) and Equation (3.27) we finally obtain

$$F(\mathbf{x}) \overset{\text{Eq. (3.23)}}{=} \sum_{i=1}^{m} f_i(\mathbf{x})^2 \tag{3.28}$$

$$\overset{\text{Eq. (3.27)}}{=} \sum_{\langle i,j \rangle \in \mathcal{C}} h_{ij}(\mathbf{x})^2 \tag{3.29}$$

$$= \chi^2(\mathbf{x}), \tag{3.30}$$

which proofs that graph-based optimization is an instance of least squares minimization.
Looking in more detail on iterative approaches for least squares error minimization, we can summarize them through the following steps:

Algorithm 1 General Error Minimization Technique

1: choose an initial configuration $\mathbf{x}_0$ (also called *initial guess*).
2: $t = 1$ (t is the current *iteration*)
3: **while** convergence criterion is not fulfilled and $t < t_{\text{max}}$ **do**
4: choose direction: $\Delta \mathbf{x}_t$
5: choose preconditioner: K_t (if $K_t = \alpha_t I$, then it is also called *step length*)
6: update actual configuration: $\mathbf{x}_{t+1} = \mathbf{x}_t + K_t \Delta \mathbf{x}_t$
7: $t = t + 1$
8: **end while**

All algorithms for error minimization differ in the calculated direction and the chosen preconditioning matrix (lines 4 + 5 in the algorithm above). However, we will focus on error minimization techniques, following the *descending condition*, i.e,

$$\exists t_0 \in \mathbb{N} : \forall t > t_0 : F(\mathbf{x}_{t+1}) < F(\mathbf{x}_t). \tag{3.31}$$

In other words, we focus on techniques that lower the error in each step $t > t_0$ but relax this condition in the first steps (i.e., $t \leq t_0$) and thus allow the algorithm to escape from a local minimum. There exist different types of convergence criteria but in work we say that an algorithm converged to a solution if for a manually chosen $\epsilon > 0$ the following holds:

$$\| F(\mathbf{x}_{t+1}) - F(\mathbf{x}_t) \| < \epsilon. \tag{3.32}$$

Given this conditions, the most popular error minimization techniques include

- gradient descent [131] (see next section)
- stochastic gradient descent [17] (see Section 3.4)
- Newton's method [131]

- Gauss-Newton (GN) [139] (see next section)
- Levenberg-Marquardt (LM) [101]
- Powell's Dogleg method [32, 119]
- Quasi-Newton methods (QN) [28]
- Fletcher-Reeves [49, 6]
- Polak-Ribière [119]

as well as variants or hybrids between the above [119, 139]. Note, that the first six methods are related to gradient descent, whereas the last two methods minimize the error based on conjugate gradient.

As can be seen, there exits many techniques for finding a local minimum for Equation (3.25). However, depending on the problem structure and the configuration space, some algorithms are better suited for optimization than others. In the context of robotic mapping we will see that path parametrized optimization (PPO) as described in Section 3.5 performs especially well. This approach is a hybrid between an approximation to Gauss-Newton and stochastic gradient descent. In the remainder of this chapter we will therefore first introduce preconditioned gradient descent (PGD) and focus on a specific instance of PGD, namely Gauss-Newton. Subsequently we describe stochastic gradient descent and present path parametrized optimization, PPO, which forms the basis to our tree-based network optimizer (TORO) in the next Chapter.

3.3 Preconditioned Gradient Descent-Based Approaches

In the previous section, we formulated the optimization problem in terms of least squares error minimization. Since the error function in general is non-linear, we cannot calculate the optimal configuration of the nodes within one step, i.e., calculate a least-square solution. It is therefore necessary to estimate the configuration in an iterative manner. Reformulating the optimization problem in terms of error minimization allows us to use a wide set of algorithms based on (preconditioned) gradient descent. Intuitively, by minimizing the error of the configuration we seek to find an $\mathbf{x}^*$ where the derivative of the error function is zero (otherwise this configuration would not be a local minimum). Note that the gradient of the error function is a vector pointing towards the steepest ascent. Moving the configuration along the negative of the gradient would lead to a new configuration with a smaller error than before. Repeating this process until no major difference in the configuration appears (and thus the length of the vector is close to zero) leads to a local minimum. In this case, we say that the algorithm is converged to a solution. In the following, we will discuss iterative error minimization techniques based on preconditioned gradient descent. Together with stochastic gradient descent (see next Section) this forms the basis of the path-parametrized optimization algorithm (PPO) presented in Section 3.5.

In its general form, preconditioned gradient descent can be formulated as

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \underbrace{K(\mathbf{x}^t)\frac{\partial \chi^2(\mathbf{x}^t)}{\partial \mathbf{x}}}_{-\Delta \mathbf{x}^t} \tag{3.33}$$

$$= \mathbf{x}^t - K(\mathbf{x}^t) \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{\frac{\partial \chi^2_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}}}_{-\Delta \mathbf{x}^t_{ij}} \tag{3.34}$$

$$= \mathbf{x}^t + K(\mathbf{x}^t) \sum_{\langle i,j \rangle \in \mathcal{C}} \Delta \mathbf{x}^t_{ij}. \tag{3.35}$$

The individual components of this equation are:

- $\mathbf{x}^t, \mathbf{x}^{t+1}$ is the configuration of the nodes at time (iteration) t and $t+1$ respectively.
- $K(\mathbf{x}^t)$ denotes the *preconditioning* matrix at time t. This matrix can in general can be chosen *arbitrarily* but mostly (and will be also in our case) is related to the Hessian of the residual.
- Finally, $\frac{\partial \chi^2_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}}$ is the gradient of the χ^2_{ij} error between node i and node j.

In most cases, an update after processing each constraint exactly once is called an *iteration*. Up to now, this is equal to a step in time, i.e., going from t to $t+1$. However, this must not hold in general. To prevent any ambiguity in the upcoming sections, we will refer to an iteration when all constraints have been processed exactly once. If an iteration is different from an update $t \rightarrow t+1$, we will refer to the latter as a *step*. It is important to keep in mind, that the design of the configuration space and the choice of the error function have a critical impact on the number of iterations needed until convergence (i.e. $\|\chi^2(\mathbf{x}^{t+1}) - \chi^2(\mathbf{x}^t)\| < \epsilon$, for a manually chosen $\epsilon > 0$) of the algorithm as well as how fast a single iteration can be calculated [59, 94]. To get an intuition why this is the case and how path parametrized optimization (PPO) in the context of robotic mapping is obtained, we need to have a more detailed look at the gradient of the χ^2_{ij} error and the preconditioning matrix. First, the gradient $\partial \chi^2_{ij}(\mathbf{x}^t)/\partial \mathbf{x}$ is reformulated as

$$\frac{\partial \chi^2_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}} = \frac{\partial \chi^2_{ij}(\mathbf{x}^t)}{\partial r_{ij}} \frac{\partial r_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}} \tag{3.36}$$

$$= \frac{\partial (r_{ij}(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t))}{\partial r_{ij}} \frac{\partial (\delta_{ij} - f_{ij}(\mathbf{x}^t))}{\partial \mathbf{x}} \tag{3.37}$$

$$= -2J^T_{ij}(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t), \quad \text{with } J_{ij}(\mathbf{x}^t) = \frac{\partial e_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}} = \frac{\partial f_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}} \tag{3.38}$$

Here, $J_{ij}(\mathbf{x}^t)$ is the *Jacobian* of the error $e_{ij}(\mathbf{x})$. Using this result (Eq. (3.38)), we can rewrite Equation (3.34) to

$$\mathbf{x}^{t+1} = \mathbf{x}^t + 2K(\mathbf{x}^t) \sum_{\langle i,j \rangle \in \mathcal{C}} J^T_{ij}(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t). \tag{3.39}$$

In the special case where we remove $K(\mathbf{x}^t)$ from the equation above we obtain the simplest form of gradient descent, namely

$$\mathbf{x}^{t+1} = \mathbf{x}^t + 2 \sum_{\langle i,j \rangle \in \mathcal{C}} J^T_{ij}(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t). \tag{3.40}$$

Reading Equation (3.40) from right to left allows us to explain the update rule in an intuitive fashion.

- The residual r_{ij} is the negative of the error. Thus, moving the node along the residual will decrease the error introduced by the corresponding constraint $\langle i, j \rangle$.

- The residual is modified according to the uncertainty $\Sigma_{ij} = \Omega_{ij}^{-1}$ associated with the constraint. In this case, the value is decreased, if the uncertainty is high (and thus the information gain is low) and vice versa.

- Finally, the Jacobian J_{ij} transforms the modification $\Omega_{ij} r_{ij}$ from error space into configuration space.

Unfortunately, removing $K(\mathbf{x}^t)$ in general leads to suboptimal convergence rates especially when the error surface is close to planar. Therefore many improved versions of gradient descent-based techniques include a preconditioning matrix speeding up the convergence rate. One possibility to obtain an appropriate $K(\mathbf{x}^t)$ is to calculate in each step an update $\Delta\mathbf{x}^t$ that zeroes the current gradient. Note that if the error function is linear, we converge within one step. In our case, however, the error function includes both translational as well as rotational components resulting in a non-linear error function. Therefore it is convenient to assume local linearity only (by zeroing the gradient at time t), resulting in the popular Gauss-Newton algorithm.

In order to calculate the step that zeroes the current gradient we need to start with the Taylor expansion of the gradient which is

$$\sum_{\langle i,j \rangle \in \mathcal{C}} \frac{\partial \chi_{ij}^2(\mathbf{x}^t + \Delta\mathbf{x}^t)}{\partial \mathbf{x}} = \sum_{\langle i,j \rangle \in \mathcal{C}} \left(-2 J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t) + \frac{\partial^2 \chi_{ij}^2(\mathbf{x}^t)}{\partial \mathbf{x}^2} \Delta\mathbf{x}^t + O\left((\Delta\mathbf{x}^t)^2\right) \right).$$

As stated above, we assume local linearity of the error e_{ij} which implies that we can approximate the χ^2 error locally by a polynom of degree 2. In other words, we assume the higher orders $O(\Delta\mathbf{x}^2)$ to be small and thus neglect their contribution in the above equation. Thus we get

$$\sum_{\langle i,j \rangle \in \mathcal{C}} \frac{\partial \chi_{ij}^2(\mathbf{x}^t + \Delta\mathbf{x}^t)}{\partial \mathbf{x}} \approx \sum_{\langle i,j \rangle \in \mathcal{C}} \left(-2 J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t) + \frac{\partial^2 \chi_{ij}^2(\mathbf{x}^t)}{\partial \mathbf{x}^2} \Delta\mathbf{x}^t \right) \tag{3.41}$$

$$= \sum_{\langle i,j \rangle \in \mathcal{C}} \left(-2 J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t) \right) + \sum_{\langle i,j \rangle \in \mathcal{C}} \left(\frac{\partial^2 \chi_{ij}^2(\mathbf{x}^t)}{\partial \mathbf{x}^2} \right) \Delta\mathbf{x}^t. \tag{3.42}$$

Setting this equation to zero yields the following expression for the increment $\Delta\mathbf{x}^t$ at time t:

$$\Delta\mathbf{x}^t = \left(\sum_{\langle i,j \rangle \in \mathcal{C}} \frac{\partial^2 \chi_{ij}^2(\mathbf{x}^t)}{\partial \mathbf{x}^2} \right)^{-1} \sum_{\langle i,j \rangle \in \mathcal{C}} 2 J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t) \tag{3.43}$$

$$= 2 H(\mathbf{x}^t)^{-1} \sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t), \tag{3.44}$$

with $H(\mathbf{x}^t)$ being the Hessian that we can further expand to

$$H(\mathbf{x}^t) = \sum_{\langle i,j\rangle\in\mathcal{C}} \frac{\partial^2 \chi_{ij}^2(\mathbf{x}^t)}{\partial \mathbf{x}^2} = \sum_{\langle i,j\rangle\in\mathcal{C}} \frac{-2J_{ij}^T(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}} \tag{3.45}$$

$$= \sum_{\langle i,j\rangle\in\mathcal{C}} -2J_{ij}^T(\mathbf{x}^t)\Omega_{ij}\frac{r_{ij}(\mathbf{x}^t)}{\partial \mathbf{x}} - \frac{\partial f_{ij}^T(\mathbf{x}^t)}{\partial \mathbf{x}^2}\Omega_{ij} r_{ij}(\mathbf{x}^t) \tag{3.46}$$

$$= \sum_{\langle i,j\rangle\in\mathcal{C}} 2J_{ij}^T(\mathbf{x}^t)\Omega_{ij} J_{ij}(\mathbf{x}^t) + \frac{r_{ij}^T(\mathbf{x}^t)}{\partial \mathbf{x}^2}\Omega_{ij} r_{ij}(\mathbf{x}^t). \tag{3.47}$$

Recall that we assume that we can sufficiently approximate the error $e_{ij}(\mathbf{x})$ (and therefore the residual $r_{ij}(\mathbf{x})$) through a linear function in the local neighborhood of $\mathbf{x}^t$. In this case, we can neglect the contribution of the second derivative (right term in Equation (3.47)) and approximate the Hessian $H(\mathbf{x}^t)$ with

$$H(\mathbf{x}^t) \simeq \sum_{\langle i,j\rangle\in\mathcal{C}} 2J_{ij}^T(\mathbf{x}^t)\Omega_{ij} J_{ij}(\mathbf{x}^t), \text{ and} \tag{3.48}$$

$$H(\mathbf{x}^t)^{-1} \simeq \frac{1}{2}\left(\sum_{\langle i,j\rangle\in\mathcal{C}} J_{ij}^T(\mathbf{x}^t)\Omega_{ij} J_{ij}(\mathbf{x}^t)\right)^{-1}. \tag{3.49}$$

This allows us to finally rewrite Equation (3.34) to

$$\mathbf{x}^{t+1} = \mathbf{x}^t - K(\mathbf{x}^t)\sum_{\langle i,j\rangle\in\mathcal{C}} \underbrace{\frac{\partial \chi_{ij}^2(\mathbf{x}^t)}{\partial \mathbf{x}}}_{-\Delta\mathbf{x}_{ij}^t} \tag{3.50}$$

$$\overset{\text{Eq. (3.44)}}{=} \mathbf{x}^t + H(\mathbf{x}^t)^{-1}\sum_{\langle i,j\rangle\in\mathcal{C}} 2J_{ij}^T(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t) \tag{3.51}$$

$$\overset{\text{Eq. (3.49)}}{=} \mathbf{x}^t + \left(\sum_{\langle i,j\rangle\in\mathcal{C}} J_{ij}^T(\mathbf{x}^t)\Omega_{ij} J_{ij}(\mathbf{x}^t)\right)^{-1}\sum_{\langle i,j\rangle\in\mathcal{C}} J_{ij}^T(\mathbf{x}^t)\Omega_{ij} r_{ij}(\mathbf{x}^t). \tag{3.52}$$

As already mentioned at the beginning of this chapter, the number of iterations needed until convergence is also dependent on the design of the configurations space and the error function. In other words: A constraint $\langle i,j\rangle$ can in general be dependent on more variables than $\mathbf{x}_i,\ldots,\mathbf{x}_j$. Let dep($\langle i,j\rangle$) denote this set of variables. Consider another constraint $\langle i',j'\rangle$ with $j'>i'>j$ and the corresponding set of variables the constraint is dependent on, dep($\langle i',j'\rangle$). One parametrization space could lead to an intersection set dep($\langle i,j\rangle$) $\cap$ dep($\langle i',j'\rangle$) containing more variables than in another parametrization space. However, the less variables a constraint is depending on the sparser is the Jacobian which in return allows us to use efficient algorithms for inverting the Hessian. This speeds up the calculation of a single iteration. On the other hand, the non-linearity of the error function has a direct influence of the regularity of the Jacobian (and thus the Hessian). A sub-optimal parametrization could lead to a highly irregular Jacobian [59, 94]. This in turn allows only for small steps, since the change $\Delta\mathbf{x}^t$ calculated in iteration t is based on a linearization of the error function that is then valid only in a small neighborhood of the current configuration $\mathbf{x}$.

In the case of robotic mapping, we will in general not be able to calculate a configuration that perfectly matches all the constraints. Our goal therefore is to minimize the overall observation error. If we have a closer look at Equation (3.52) we observe, that in each iteration t the change $\Delta\mathbf{x}^t$ is calculated using the inverse of the Hessian. In our case, however, this formulation bears several weak points. First of all, the Hessian is of size $n \times n$, with n being the number of nodes in the network and is re-calculated in each iteration. This calculation, however, is not feasible in practise scenarios except for very small graphs. Additionally if we calculate each step based on the gradient of *all* constraints we reduce the chance of escaping from a local minimum. This risk is increased in robotic mapping since it is more likely that constraints have contrary effects on each other forcing the algorithm to converge to a suboptimal solution. Indeed, Gauss-Newton is highly sensitive to the initial guess and it is not even guaranteed to converge. It is noteworthy, that the last remark is overcome by an improved variant known as Levenberg-Marquardt which has still the other problems discussed so far. These two points (computational complexity and escape from local minimum) are addressed by using *stochastic gradient descent* (SGD) which is described in the next section and can be seen as partially orthogonal to the least squares approach described above (Equation (3.52)). However, we will see in the next section, that this comes at the cost of an much higher number of iterations needed until convergence.

3.4 Stochastic Gradient Descent

When we introduced Gauss-Newton, a special formulation of preconditioned gradient descent, we saw, that in principle it bears two weak points. First, summing up the individual gradients can reduce our chance to escape out of a local minimum. Second, the calculation of the full Hessian is only feasible in the case of very small networks. Both points are addressed using stochastic gradient descent (SGD). As already mentioned in the previous section we can see this approach to be partially orthogonal to Gauss-Newton. In case of SGD, we start with pure gradient descent, i.e., removing $K(\mathbf{x}^t)$ from Equation (3.39) yielding

$$\mathbf{x}^{t+1} = \mathbf{x}^t + 2 \sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t) \tag{3.53}$$

$$\propto \mathbf{x}^t + \sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t). \tag{3.54}$$

The key difference to gradient descent, however, is that we do not sum up all gradients within one iteration but rather in each step choose *one* constraint at random and calculate the update given this gradient only. This leads to the following update rule:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t), \tag{3.55}$$

given the randomly selected constraint at time t is $\langle i, j \rangle$. Although this increases the chance to escape from a local minimum we cannot guarantee convergence anymore. Consider two constraints having an opposite effect on the same variable. Since one iteration contains the variation of a single constraint only (and no Hessian) this would lead to an infinity oscillation.

In order to ensure convergence, we use a learning rate λ^t (also called damping factor) to scale down the gradient over time. According to Bottou [22], we need to choose $\lambda(t)$ such that

$$\sum_{t=0}^{\infty} \lambda(t) = \infty, \qquad \sum_{t=0}^{\infty} \lambda(t)^2 < \infty. \tag{3.56}$$

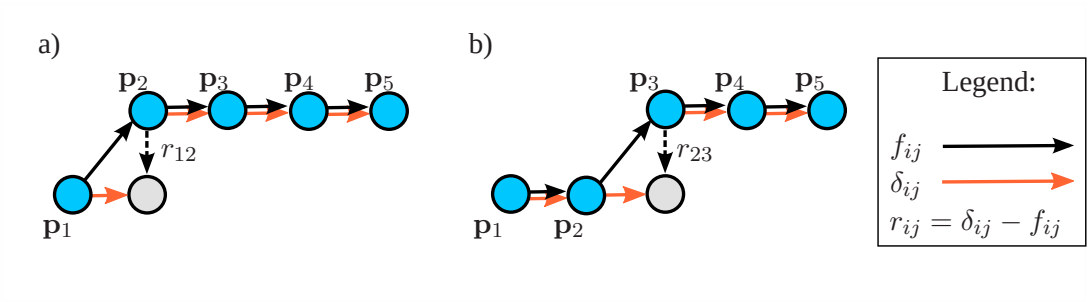

Figure 3.4: Choosing the configuration space can have critical impact on the performance of the optimization algorithm. Here, the nodes are parametrized in global poses $\mathbf{p}$. If initially (a) all constraints except $\langle 1, 2 \rangle$ are perfectly matched, moving $\mathbf{p}_2$ along the residual will introduce an error $e_{23} = -r_{23}$ in the next iteration (b). Therefore updating δ_{12} will slowly propagate through the network and the algorithm will need many iterations until convergence.

Note that this also implies $\lim_{t \to \infty} \lambda(t) \to 0$. Implicitly enforcing convergence by introducing a learning rate is also common within the machine learning community and leads to the update rule of stochastic gradient descent (also called on-line gradient descent):

$$\mathbf{x}^{t+1} \quad = \quad \mathbf{x}^t + \lambda(t) J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t), \tag{3.57}$$

Observe that no preconditioning of the gradient is applied which can result in many iterations if the error function at $\mathbf{x}^t$ is close to planar. Indeed, the combination of an approximation of Gauss-Newton and stochastic gradient descent together with a novel parametrization of the configuration space are the key ideas behind the PPO algorithm. This approach combines the strengths of both algorithms and is described in the next section.

Compared to Gauss-Newton, the missing preconditioning matrix makes SGD even more sensitive to the configuration space and the choice of the error function. Consider for example a parametrization in global poses, i.e., $\mathbf{x} = \mathbf{p}$. Figure 3.4 depicts a small example that will allow us to emphasize this problem. In the initial configuration (Fig. 3.4(a)) all but the constraint $\langle 1, 2 \rangle$ are perfectly matched resulting in all residuals to be zero except r_{12}. Since the nodes are parametrized in global coordinates, moving $\mathbf{p}_2$ along the residual r_{12} will indeed satisfy δ_{12} but on the other hand will result in $r_{23} \neq 0$ after this update as shown in (b). Thus, an update of a single constraint will slowly propagate through the network which will result in many iterations needed until convergence is reached. Here, the correlation between individual constraints is very high, since changing a variable locally (i.e., $\mathbf{p}_2$ in Figure 3.4) will have an effect on all subsequent variables.

On the other hand, if we parametrize the configuration space in such a way, that updating a constraint immediately propagates to all subsequent nodes involved, this would lead to an algorithm needing fewer iterations for converging. This effect is indicated in Figure 3.5. Starting from the same configuration as in Figure 3.4 all constraints are perfectly matched except δ_{12}. If we could immediately propagate this update to all subsequent nodes as indicated by the dashed gray arrows in (a) we would need fewer iterations for converging as indicated by Figure 3.5(b). This behavior indeed is the key result of the PPO algorithm which is described in the next section. This algorithm uses an approximation to Gauss-Newton in combination with stochastic gradient descent and forms the basis to our tree based network optimization algorithm (TORO) presented in Chapter 4.

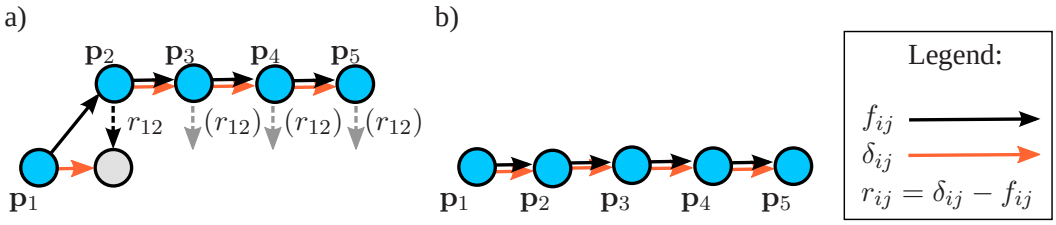

Figure 3.5: Choosing a configuration space where updating an constraint has the immediate effect of propagating through the network (indicated by the gray dashed arrows) yields an algorithm needing fewer iterations for convergence. This indeed is one of the key contributions of Olson's *et al.* PPO algorithm described in Section 3.5. This algorithm forms the basis of our improved tree based network optimizer presented in Chapter 4.

3.5 A Variant of SGD for Efficient 2D Optimization

In the context of 2D robotic mapping, Olson *et al.* [116] proposed path parametrized optimization (PPO), that can be seen as a variant of stochastic gradient descent (SGD). In more detail, it combines SGD with an approximation to Gauss-Newton and a novel parametrization of the configuration space. Similar to SGD, this approach minimizes Equation (3.20) by iteratively selecting *a single* constraint $\langle i, j \rangle$ and moving the nodes of the graph in order to decrease the error introduced by this constraint. Here, the stochastic element is given through a random permutation of the order of the constraints in each iteration, i.e., until all constraints have been used exactly once. Thus each iteration τ is equal to $|\mathcal{C}|$ time steps, with $|\mathcal{C}|$ being the number of constraints. In the remainder of this section we will first present the final update rule of this approach. Subsequently we will introduce the parametrization used in this approach, derive the corresponding Jacobian and discuss the remaining parts of the update rule in more detail.

Given a constraint $\langle i, j \rangle$ selected at time t belonging to iteration τ, the nodes of the network (graph) are updated according to the following equation:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \underbrace{\lambda(\tau) K_{ij}(\mathbf{x}^\tau) J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t)}_{\Delta \mathbf{x}_{ij}^t}. \tag{3.58}$$

Reading the term $\Delta \mathbf{x}_{ij}^t$ in Equation (3.58) from right to left gives an intuition about the sequential procedure of this approach:

- $r_{ij}(\mathbf{x})$ is the residual which is the opposite of the error vector. Changing the configuration of the nodes in the direction of the residual $r_{ij}(\mathbf{x})$ will decrease the error $e_{ij}(\mathbf{x})$.
- Ω_{ij} represents the information matrix of the constraint $\langle i, j \rangle$. Thus, $\Omega_{ij} r_{ij}(\mathbf{x})$ scales the residual components according to the uncertainty of the constraint.
- $J_{ij}(\mathbf{x})$ is the Jacobian of the error $e_{ij}(\mathbf{x})$ and maps the (scaled) residual term from error space into a variation of the nodes in configuration space.
- $K_{ij}(\mathbf{x}^\tau)$ is a preconditioning matrix that is calculated at the beginning of iteration τ for each constraint $\langle i, j \rangle$ given the actual configuration of the nodes, $\mathbf{x}^\tau$.
- Finally, $\lambda(\tau)$ is a learning rate with decreases with each iteration and makes the system to converge to a (local) minimum.

To understand how the individual components of Equation (3.58) look like, we first need to introduce the parametrization used within this approach. As stated at the end of Section 3.3 the choice of the parametrization has a high influence on the convergence speed of the algorithm. To address the limitations introduced by using global poses in the case of 2D robotic mapping, i.e., $\mathbf{p}_i = (x_i, y_i, \psi_i)$, Olson and colleagues propose to use a world representation based on pose differences that will allow us to propagate the update of a constraint $\langle i, j \rangle$ to all subsequent nodes.

In more detail, given the set of absolute poses $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$, with $\mathbf{p}_i = (x_i, y_i, \psi_i)$, we define a new parametrization $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\mathbf{x}_i = (\Delta x_i, \Delta y_i, \Delta \psi_i)$, based on pose differences with

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{p}_1 \\ \mathbf{x}_i &= \mathbf{p}_i - \mathbf{p}_{i-1}, \quad \text{for } i > 1. \end{aligned} \tag{3.59}$$

Note that $\mathbf{p}_i - \mathbf{p}_{i-1}$ is a pure vector subtraction, i.e, no motion composition is applied and that we can easily recover the absolute poses, since

$$\mathbf{p}_i = \sum_{k=1}^{i} \mathbf{x}_k = \sum_{k=2}^{i} \underbrace{\mathbf{p}_k - \mathbf{p}_{k-1}}_{\mathbf{x}_k} + \underbrace{\mathbf{p}_1}_{\mathbf{x}_1}. \tag{3.60}$$

Thus, moving node i in one iteration has the immediate effect of moving all nodes $j > i$ as well. It is therefore convenient to order the set of poses $\mathbf{p}$ (and thus $\mathbf{x}$) ascending in time, i.e., given $\mathbf{p}_i$ obtained at time t_1 and $\mathbf{p}_j$ obtained at time t_2, with $j > i$, we assume $t_1 < t_2$. This is easily achieved by using the incoming order of the state estimation process. For better readability we also assume without loss of generality that

- $j > i$ for each pair of indexes j, i, and
- all constraints refer to observations made from the corresponding node i about the corresponding node j, i.e., all constraints are of the form $\langle i, j \rangle$.

The latter case can be enforced by replacing each constraint $\langle j, i \rangle$ by a corresponding constraint $\langle i, j \rangle$. Given $\langle j, i \rangle$ is made up of two components, namely the information matrix Ω_{ji} and the relative transformation δ_{ji}, consisting of a translation $t_{ji} = (x_{ji}, y_{ji})$ and a rotation ψ_{ji} represented by the rotation matrix $\widetilde{R}_{ji}$. In more detail, given

$$\widetilde{R}_{ji} = \begin{pmatrix} \cos\psi_{ji} & -\sin\psi_{ji} \\ \sin\psi_{ji} & \cos\psi_{ji} \end{pmatrix} \quad \text{and} \quad J_{\ominus ji} = \begin{pmatrix} \widetilde{R}_{ji} & \begin{matrix} -y_{ji} \\ x_{ji} \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{pmatrix},$$

we can calculate $\langle i, j \rangle = (\delta_{ij}, \Omega_{ij})$ with $\delta_{ij} = (x_{ij}, y_{ij}, \psi_{ij})$ through (see Section 2.2)

$$\begin{aligned} \psi_{ij} &= -\psi_{ji}, \\ t_{ij} &= -\widetilde{R}_{ji} t_{ji}, \text{ and} \\ \Omega_{ij} &= J_{\ominus ji}^T \Omega_{ji} J_{\ominus ji}. \end{aligned}$$

Recall that each constraint $\langle i, j \rangle$ consists of a relative observation made about node j, seen from node i, δ_{ij}, and the corresponding information matrix Ω_{ij}, thus, each constraint is expressed in the local coordinate frame of node i. On the other side all nodes, though represented through

a relative sum, are in the global coordinate frame. To calculate the error of a constraint we therefore need to transform both into the same coordinate frame. In the following we will calculate the error in the local reference frame of node i. Given the motion composition operator $\oplus$ and its inverse $\ominus$ as introduced in Section 2.2 the error $e_{ij}(\mathbf{p})$ is

$$\begin{aligned} e_{ij}(\mathbf{p}) &= (\mathbf{p}_j \ominus \mathbf{p}_i) - \delta_{ij} && (3.61)\\ &= R_i^T(\mathbf{p}_j - \mathbf{p}_i) - \delta_{ij} && (3.62)\\ &= -r_{ij}(\mathbf{p}). && (3.63) \end{aligned}$$

Here, R_i is the homogeneous rotation matrix with

$$\begin{aligned} R_i &= \begin{pmatrix} \cos\psi_i & -\sin\psi_i & 0 \\ \sin\psi_i & \cos\psi_i & 0 \\ 0 & 0 & 1 \end{pmatrix} && (3.64)\\ &= \begin{pmatrix} \widetilde{R}_i & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{pmatrix} && (3.65) \end{aligned}$$

Using the definition of relative displacements (Eq. (3.59)) we can rewrite Equation (3.62) to

$$\begin{aligned} e_{ij}(\mathbf{x}) &= R_i^T(\mathbf{p}_j - \mathbf{p}_i) - \delta_{ij} && (3.66)\\ &\overset{\text{Eq. (3.59)}}{=} R_i^T \left(\sum_{k=1}^{j} \mathbf{x}_k - \sum_{k=1}^{i} \mathbf{x}_k \right) - \delta_{ij} && (3.67)\\ &= R_i^T \sum_{k=i+1}^{j} \mathbf{x}_k - \delta_{ij}. && (3.68) \end{aligned}$$

Recall that $\mathbf{x}_i = (\Delta x_i, \Delta y_i, \Delta\psi_i)^T$ and that R_i is the rotation of the i-th node in global coordinate frame. This rotation is now calculated given the relative orientations $R_{\Delta i}$ through

$$\begin{aligned} R_i &= \prod_{k=1}^{i} R_{\Delta k} =: R_{\Delta 1:\Delta i}, \text{ with} && (3.69)\\ R_{\Delta i} &= \begin{pmatrix} \cos\Delta\psi_i & -\sin\Delta\psi_i & 0 \\ \sin\Delta\psi_i & \cos\Delta\psi_i & 0 \\ 0 & 0 & 1 \end{pmatrix} && (3.70)\\ &= \begin{pmatrix} \widetilde{R}_{\Delta i} & & 0 \\ & & 0 \\ 0 & 0 & 1 \end{pmatrix} && (3.71) \end{aligned}$$

Again, as in Section 3.3, we assume that the constraints are independent of each other and that the observation likelihood can be modeled using a Gaussian probability density function. Additionally, by omitting the time index t for better readability the $\chi^2_{ij}(\mathbf{x})$ error of the constraint $\langle i, j \rangle$ is

$$\begin{aligned} \chi^2_{ij}(\mathbf{x}) &= r_{ij}(\mathbf{x})^T \Omega_{ij} r_{ij}(\mathbf{x}) && (3.72)\\ &= e_{ij}(\mathbf{x})^T \Omega_{ij} e_{ij}(\mathbf{x}) && (3.73)\\ &\overset{\text{Eq. (3.68)}}{=} \left(R_i^T \sum_{k=i+1}^{j} \mathbf{x}_k - \delta_{ij} \right)^T \Omega_{ij} \left(R_i^T \sum_{k=i+1}^{j} \mathbf{x}_k - \delta_{ij} \right). && (3.74) \end{aligned}$$

To derive the gradient of the $\chi^2_{ij}(\mathbf{x})$ error,

$$\frac{\partial \chi^2_{ij}(\mathbf{x})}{\partial \mathbf{x}} \propto -J_{ij}(\mathbf{x})\Omega_{ij} r_{ij}(\mathbf{x}), \tag{3.75}$$

that is part of the update term $\Delta\mathbf{x}_{ij}$ in Equation (3.58), we need to calculate the Jacobian $J_{ij}(\mathbf{x})$ of $e_{ij}(\mathbf{x})$. Observe that δ_{ij} is constant and R_i^T is *not* a function of the variables $\mathbf{x}_{i+1:k}$ but of the angles $\Delta\psi_{1:i}$ of $\mathbf{x}_{1:i}$, since $R_i = R_{\Delta 1:\Delta i}$. The Jacobian $J_{ij}(\mathbf{x})$ is therefore:

$$\begin{aligned}
J_{ij}(\mathbf{x}) &= \frac{\partial e_{ij}(\mathbf{x})}{\partial \mathbf{x}} && (3.76)\\
&= \frac{\partial \left(R_i^T \sum_{k=i+1}^{j} \mathbf{x}_k - \delta_{ij} \right)}{\partial \mathbf{x}} && (3.77)\\
&= \frac{\partial \left(R_i^T \sum_{k=i+1}^{j} \mathbf{x}_k \right)}{\partial \mathbf{x}} && (3.78)\\
&= \left(\underbrace{A_1 \cdots A_i}_{1,\ldots,i} \underbrace{R_i^T \cdots R_i^T}_{i+1,\ldots,j} \underbrace{\mathbf{0} \cdots \mathbf{0}}_{j+1,\ldots,n} \right). && (3.79)
\end{aligned}$$

Here, $\mathbf{0}$ is the 3×3 matrix containing only zero elements, i.e.,

$$\mathbf{0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{ and} \tag{3.80}$$

A_s, with $s \in (1, \ldots, i)$, is expanded to

$$\begin{aligned}
A_s &= \frac{\partial R_i^T \sum_{k=i+1}^{j} \mathbf{x}_k}{\partial \mathbf{x}_s} && (3.81)\\
&= \frac{\partial (\prod_{k=1}^{i} R_{\Delta k})^T \sum_{k=i+1}^{j} \mathbf{x}_k}{\partial \mathbf{x}_s} && (3.82)\\
&= \frac{\partial \prod_{k=i}^{1} R_{\Delta k}^T \sum_{k=i+1}^{j} \mathbf{x}_k}{\partial \mathbf{x}_s} && (3.83)\\
&= \begin{pmatrix} 0 & 0 & \\ 0 & 0 & \widetilde{R}^T_{\Delta i:\Delta s+1} \widetilde{R}_{\Delta s} \widetilde{R}^T_{\Delta s-1:\Delta 1} \sum_{k=i+1}^{j} (\Delta x_k, \Delta y_k)^T \\ 0 & 0 & \end{pmatrix}. && (3.84)
\end{aligned}$$

Note that the A_n only depend on the Δx and Δy component of the $\mathbf{x}_k$ and the contribution of the A_n is therefore proportional to translational part $\mathbf{p}_j - \mathbf{p}_i$. Assuming this translational part to be small, we can neglect the contribution of the A_n and approximate the Jacobian with

$$\begin{aligned}
J_{ij}(\mathbf{x}) &\simeq \left(\underbrace{\mathbf{0} \cdots \mathbf{0}}_{0,\ldots,i} \underbrace{R_i^T \cdots R_i^T}_{i+1,\ldots,j} \underbrace{\mathbf{0} \cdots \mathbf{0}}_{j+1,\ldots,n} \right) && (3.85)\\
&= R_i^T \left(\underbrace{\mathbf{0} \cdots \mathbf{0}}_{0,\ldots,i} \underbrace{I \cdots I}_{i+1,\ldots,j} \underbrace{\mathbf{0} \cdots \mathbf{0}}_{j+1,\ldots,n} \right) && (3.86)\\
&= R_i^T \sum_{k=i+1}^{j} \mathcal{I}_k && (3.87)
\end{aligned}$$

with $\mathcal{I}_k$ defined as

$$\mathcal{I}_k = \left(\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{0,\ldots,k-1}, \underbrace{I}_{k}, \underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{k+1,\ldots,n} \right), \text{ and} \tag{3.88}$$

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.89}$$

We can now see what the effects of choosing such a parametrization are. First of all, a constraint $\langle i, j \rangle$ will keep the node i fixed and distribute the residual along the variables $\mathbf{x}_{i+1}, \ldots, \mathbf{x}_j$ only. This update, however, will also immediately propagate to all nodes $k > j$ because of the special parametrization (see Equation (3.59)ff). Second, we also see that the Jacobian belonging to a constraint takes a very simple form (see Equation (3.87)) which allows us to calculate $J_{ij}^T(\mathbf{x}^t)\Omega_{ij}r_{ij}(\mathbf{x}^t)$ quite fast.

As stated in the beginning of this section, the update rule of PPO is

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \lambda(\tau) K_{ij}(\mathbf{x}^\tau) J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t), \tag{3.90}$$

thus it remains open how the preconditioning matrix $K_{ij}(\mathbf{x}^\tau)$ and the learning rate $\lambda(\tau)$ are chosen which will be discussed in the remainder of this section.

To ensure convergence of the algorithm, the authors propose to use a $\lambda(\tau)$ that decays in each iteration τ and fulfills the requirements as stated in Equation (3.56) with

$$\lambda(\tau + 1) = \frac{\lambda(\tau)}{\lambda(\tau) + 1}, \quad \text{for } \tau > 1, \text{ and} \tag{3.91}$$

$$\lambda(1) = 1/3, \text{ which results in} \tag{3.92}$$

$$\lambda(\tau) = 1/(\tau + 2), \quad \text{for } \tau \geq 1. \tag{3.93}$$

The preconditioning matrix $K_{ij}(\mathbf{x}^\tau)$ in Equation (3.90) is an approximation to the preconditioning matrix of Gauss-Newton which is (see Section 3.3)

$$H^{-1}(\mathbf{x}^t) \overset{\text{Eq. (3.49)}}{\propto} \left(\sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T(\mathbf{x}^t) \Omega_{ij} J_{ij}(\mathbf{x}^t) \right)^{-1}. \tag{3.94}$$

Looking closely at the equation above, we see that the matrix H is of size $3n \times 3n$, given $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and inverting such a matrix is computationally expensive if it is not very sparse. Therefore we calculate the Hessian only at the beginning of an iteration τ and approximate its inverse by inverting the diagonal elements of $H(\mathbf{x}^\tau)$ only. This results in

$$H^{-1}(\mathbf{x}^\tau) \approx \left[\text{diag} \left(\sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T(\mathbf{x}^\tau) \Omega_{ij} J_{ij}(\mathbf{x}^\tau) \right) \right]^{-1}. \tag{3.95}$$

Observe, that $H^{-1}(\mathbf{x}^\tau)$ is now a matrix of size $3n \times 3n$ having the only non-zero elements at the diagonal. We can further divide the diagonal into n blocks $D_k^{-1}, k = 1, \ldots, n$, each of size 3×3 (also with the only non-zero elements on the diagonal) such that

$$H^{-1}(\mathbf{x}^\tau) = \begin{pmatrix} D_1^{-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & D_2^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & D_n^{-1} \end{pmatrix}, \text{ with } \mathbf{0} := \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.96}$$

Since applying a constraint $\langle i, j \rangle$ yields in an update of the variables $\mathbf{x}_{i+1}$ to $\mathbf{x}_j$ only the corresponding preconditioning matrix $K_{ij}(\mathbf{x}^\tau)$ contains a total of $(j-i)$ matrices on the diagonal, each of size 3×3, and all other entries being zero. In more detail, $K_{ij}(\mathbf{x}^\tau)$ is of the following form:

$$K_{ij}(\mathbf{x}^\tau) = \begin{pmatrix} \mathbf{0} & & & & & & & \\ & \ddots & & & & \ddots & \vdots & \iddots \\ & & \mathbf{0} & & & \cdots & \mathbf{0} & \cdots \\ & & & W_{i+1}^{ij} & & \iddots & \vdots & \ddots \\ & & & & \ddots & & & \\ & \ddots & \vdots & \iddots & & W_j^{ij} & & \\ & \cdots & \mathbf{0} & \cdots & & & \mathbf{0} & \\ & \iddots & \vdots & \ddots & & & & \ddots \\ & & & & & & & \mathbf{0} \end{pmatrix}. \tag{3.97}$$

Here, the W_k^{ij} of $K_{ij}(\mathbf{x}^\tau)$, with $k = i+1, \ldots, j$, are computed using Equation (3.96) as

$$W_k^{ij} = (j-i) \left[\sum_{m=i+1}^{j} D_m^{-1} \right]^{-1} D_k^{-1}. \tag{3.98}$$

Intuitively, the individual W_k^{ij} weight the residual of the constraint $\langle i, j \rangle$ according to the ratio between the uncertainty of the current node $\mathbf{x}_k$ with respect to all other nodes involved in this update, scaled by the affected number of nodes, $(j-i)$. Here, the update of a node k will be proportional to the uncertainty affecting it. Consider the example where all constraints connected to node k have a low uncertainty (and therefore large numbers in the information matrix). According to Equation (3.95) the corresponding D_k^{-1} will contain small values only. Consider further a constraint $\langle i, j \rangle$ updating the nodes $i+1$ to j, with $i+1 \leq k \leq j$, whereas all the other nodes $\mathbf{x}_m$, with, $m = i+1, \ldots, j, m \neq k$ are connected to constraints (except $\langle i, j \rangle$) having a high uncertainty and therefore low values in the information matrix. In this case, the corresponding D_m^{-1} will be made up of high values. Thus, the corresponding weight W_k^{ij} will be small. In other words, nodes having a high uncertainty (expressed by connected constraints having a high uncertainty) will be affected by a stronger variation than nodes having a small uncertainty. It is noteworthy, that in the final step of this algorithm each weight W_k^{ij} is clamped to a maximum preventing the update from overshooting.

To summarize, PPO is an algorithm for 2D robotic mapping which is a combination of an approximation to Gauss-Newton and stochastic gradient descent. It uses a novel parametrization in the context of graph optimization based on pose differences that results in an update rule allowing us to compute a single iteration quite fast. This parametrization also significantly reduces the amount of iterations needed for convergence [114] since an update of a node i will immediately propagate to all subsequent nodes j, with $j > i$. Unfortunately, this algorithm still bears some drawbacks.

- PPO assumes that the nodes are ordered ascending in time. Therefore, PPO, cannot deal with arbitrarily connected networks which can be present in practice. Such a network could be for example constructed by two robots exploring an environment while sharing the same map observing each other once in a while.

- Furthermore, PPO assumes that the nodes are ordered according to poses along the trajectory. This results in adding nodes to the graph whenever the robot travels for an extended time in the same region resulting in an algorithm where the computational complexity is proportional to the time spent in the environment rather to the space. The reason for this is that the ordering of the nodes prevents PPO from merging multiple nodes into one. This, however, is a necessary precondition to perform life-long map learning.

- When updating a constraint $\langle i, j \rangle$ between the nodes i and j, the current parametrization requires to change $(j - i)$ nodes. As a result, each node is likely to be updated by several constraints, which leads to a high interaction between constraints and will typically reduce the convergence speed of this approach. For example, a node k will be updated by all constraints $\langle i', j' \rangle$ with $i' \leq k \leq j'$, which is common in practice when a robot re-traverses already known areas, i.e., performs loop closures. Note that by using an intelligent look-up structure [114], this operation can be carried out in $\mathcal{O}(\log n)$, where n is the number of nodes in the graph. Therefore, this is a problem of convergence speed of this algorithm and not a computational problem.

- The current parametrization is only valid for 2D robotic mapping, i.e., $\mathbf{p} = (x, y, \psi)$, although an extension towards (x, y, z, ψ) is straightforward, this algorithm cannot deal with full 3D rotations.

All these points are addressed by our novel tree-based network optimization algorithm for the 2D and 3D case presented in Chapter 4. We present a novel parametrization for graph-based error minimization. We will demonstrate that our algorithm yields accurate results for robotic mapping but needs substantially fewer iterations than the approach presented here, while the computational burden for a single iteration is comparable small. We will furthermore demonstrate, that our update rule allows us to deal with arbitrary networks and significantly reduces the oscillations of a node during an iteration. We will describe our node reduction technique resulting in an algorithm not depending on the length of the trajectory but on the area explored by the robot. Finally, we present an extension towards three-dimensional graph optimization and demonstrate the robustness in several simulated and real-world experiments.

Chapter 4

Tree-Based Graph Optimization

We present an extension to path parametrized optimization (PPO) by applying a novel parametrization of the nodes that significantly improves the performance in 2D. We present a further extension that enables us to deal with 3D data. Subsequently, we present a technique for node reduction yielding an approach whose computational time is depending on the size of the explored environment rather than on the time spent in it. We evaluate our approach on large simulated and real world data sets and demonstrate that we outperform current state-of-the-art techniques.

In the previous chapter we reviewed the basic principles about graph based optimization within the context of robotic mapping and saw that we can reformulate an graph-based optimization problem in terms of least squares error minimization. Subsequently, we described a technique commonly used within the robotics community, namely preconditioned gradient descent and focused on Gauss-Newton. Furthermore we described stochastic gradient descent, an approach containing orthogonal elements with respect to preconditioned gradient descend. We presented the path-parametrized optimization algorithm (PPO) which is a combination of both techniques. This approach is a fast and robust algorithm for 2D robotic mapping and belongs to the current state-of-the-art. However, the specific design of this algorithm still bears some disadvantages that result in a suboptimal behavior. This includes an increased number of iterations needed for convergence due to the parametrization as well as its computational dependence on the time a robot traveled in an environment rather than on the environment size. Finally, the algorithm works only on a configuration space containing at most one rotational axis, i.e., the yaw. In this chapter we will present our tree-based network optimization algorithm, an improved version of the PPO approach for both the 2D case and the full 3D case including 3D rotations. We demonstrate that our algorithm needs substantially fewer iterations until convergence for the 2D case (without any loss in accuracy). Furthermore, we show that our three-dimensional version can even optimize large networks in full 3D where many state-of-the-art techniques cannot be applied to.

This chapter is structured as follows. We will first introduce our novel tree-based parametrization in Section 4.1 and derive the corresponding update rule for optimization in 2D. Subsequently, we present our extension towards 3D optimization in Section 4.3 and analyze the rotational error distribution in Section 4.4. Furthermore, we present a technique for merging nodes and thus reducing the number of variables in the graph in Section 4.5 leading to an optimiza-

tion algorithm whose computational time is depending on the size of the explored environment rather than on the time spent in it. In Sections 4.6 and 4.7 we evaluate our approach on big simulated data sets as well as on real world data sets, both in 2D and 3D and compare our tree-based network optimizer to current state-of-the-art approaches, including a detailed comparison to the incremental approach (PPO). Finally, we discuss the relation of our algorithm to the literature in Section 4.8 and conclude in Section 4.9.

4.1 Tree Parametrization and 2D Graph Optimization

One of the key contributions of PPO is the novel parametrization of the configuration space in terms of pose differences. We saw that this resulted in a simple update rule that is easy to implement. In this section we propose a different parametrization that results in a slightly different update rule but is still compact and easy to implement. However, it reduces the number of times a node will be updated by different constraints, i.e., the correlation between different constraints is even lower than in the case of PPO. Especially, when closing a loop, we will see that our parametrization updates a smaller set of variables resulting in a faster convergence of the algorithm (to a correct solution). As in the case of PPO our update rule is of the form

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \underbrace{\lambda(\tau) K_{ij}(\mathbf{x}^\tau) J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t)}_{\Delta \mathbf{x}_{ij}^t}. \tag{4.1}$$

Here, we assume that we update the graph according to a constraint $\langle i, j \rangle$ selected at time t belonging to iteration τ. However, in our case we will not permute the set of constraints at each iteration τ (as it is the case in PPO) but select the constraints in a fixed order. We will see later, that this order is given by our tree structure and reduces the complexity of an iteration. For better readability, let us first recall from the previous chapter the individual components of this update rule:

- $r_{ij}(\mathbf{x})$ is the residual which is the opposite of the error vector. Changing the configuration of the nodes in the direction of the residual $r_{ij}(\mathbf{x})$ will decrease the error $e_{ij}(\mathbf{x})$.
- Ω_{ij} represents the information matrix of the constraint $\langle i, j \rangle$. Thus, $\Omega_{ij} r_{ij}(\mathbf{x})$ scales the residual components according to the uncertainty of the constraint.
- $J_{ij}(\mathbf{x})$ is the Jacobian of the error $e_{ij}(\mathbf{x})$ and maps the (scaled) residual term from error space into a variation of the nodes in configuration space.
- $K_{ij}(\mathbf{x}^\tau)$ is a preconditioning matrix. It is calculated at the beginning of iteration τ for each constraint $\langle i, j \rangle$ given the actual configuration of the nodes, $\mathbf{x}^\tau$.
- Finally, $\lambda(\tau)$ is a learning rate that decreases with each iteration and makes the system to converge.

In the remainder of this section we will first introduce our tree parametrization. Subsequently we will derive the individual components and show the differences compared to previous approaches.

Similar to PPO (see Section 3.5) we propose a parametrization based on pose differences. In our case, however, we relax the requirement that the poses are ordered along the trajectory but parametrize the configuration space as follows. Given the absolute poses $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)^T$,

with $\mathbf{p}_i = (x_i, y_i, \psi_i)^T$, the parametrization $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$, with $\mathbf{x}_i = (\Delta x_i, \Delta y_i, \Delta\psi_i)^T$ is obtained as:

$$\mathbf{x}_1 = \mathbf{p}_1 \quad \text{(we will call it the root node)} \tag{4.2}$$

$$\mathbf{x}_i = \mathbf{p}_i - \text{parent}(\mathbf{p}_i), \quad \text{for } i > 1. \tag{4.3}$$

Here, the parent of $\mathbf{p}_i$ is the predecessor with the smallest index (i.e., the "oldest" node) for which a constraint between both nodes exists, thus

$$\text{parent}(\mathbf{p}_i) := \mathbf{p}_k \text{ with } \exists\langle i,k\rangle \wedge (\forall\langle i,k'\rangle, k' \neq k : k' > k). \tag{4.4}$$

As in previous work [116], we also assume all constraints being of the form $\langle i,j\rangle$, with $j > i$ and that we can transform each constraint $\langle j,i\rangle$ into a corresponding constraint $\langle i,j\rangle$ as described in Section 2.2. We can also easily recover the global pose representation, since for each $\mathbf{p}_i$, there exists a *path* $\mathcal{P}_{1:i}$ connecting the root node $\mathbf{x}_1 = \mathbf{p}_1$ and $\mathbf{p}_i$ in the tree structure. Without loss of generality, let us assume this path to be $\mathcal{P}_{1:i} = (\mathbf{x}_{i_1} = \mathbf{x}_1, \mathbf{x}_{i_2}, \dots, \mathbf{x}_{i_n})$. We can then recover the pose $\mathbf{p}_i$ by summing up the individual components, i.e.,

$$\mathbf{p}_i = \sum_{k=i_1}^{i_n} \mathbf{x}_k. \tag{4.5}$$

To get an intuition about the structure consider the two examples shown in Figure 4.1. Each example consists of a trajectory and the corresponding tree structure given by our parametrization. Within each trajectory, black arrows indicate motion constraints and orange arrows visualize observations made about the corresponding node (i.e., loop closures). The second example (Figure 4.1(right)) displays a typical situation that is common in practice, namely re-traversing a loop several times. Especially in this case, the tree structure has the main advantage of keeping the number of nodes involved in an update of a constraint small. We shall see later, that an update indeed only involves the set of variables belonging to the loop *in the tree structure*. In more detail, consider for example the constraint $\langle 1, 12\rangle$ in the left image or alternatively $\langle 1, 9\rangle$ in the right image of Figure 4.1. Updating such a constraint using the incremental parametrization as proposed in the previous chapter would result in an error distribution among all nodes in this loop. When using our parametrization, however, updating this constraint yields in an update including the nodes $1, 2, 3, 10, 11, 12$ in the first example and $1, 5, 9$ in the second one. Again, updating the constraint $\langle 1, 9\rangle$ in the second example will only affect the nodes $1, 5, 9$ but leave all other nodes unchanged. Here, we can already see that this parametrization will have the effect that a node k, for which many constraints $\langle i,j\rangle$, with $i \leq k \leq j$, exists, will be updated less often through potentially counter effective constraints than in the case of the incremental parametrization. We will demonstrate in the experimental section, that this indeed will lead to an algorithm that needs less iterations until convergence. Note that in case no loop closures are present, our tree structure degenerates to a linear list yielding the same parametrization as in the case of PPO. To derive the individual components of the update rule given our parametrization let us start with the definition of the error of a constraint $\langle i,j\rangle$, which is

$$e_{ij}(\mathbf{p}) = (\mathbf{p}_j \ominus \mathbf{p}_i) - \delta_{ij} \tag{4.6}$$

$$= R_i^T(\mathbf{p}_j - \mathbf{p}_i) - \delta_{ij} \tag{4.7}$$

$$= -r_{ij}(\mathbf{p}). \tag{4.8}$$

Recall, that there exists a path $\mathcal{P}_{1:i} = (\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_n})$ from the root node, $\mathbf{x}_1 = \mathbf{x}_{i_1}$, to each node i present in the network. Thus the rotation of a node i, expressed through the homogeneous

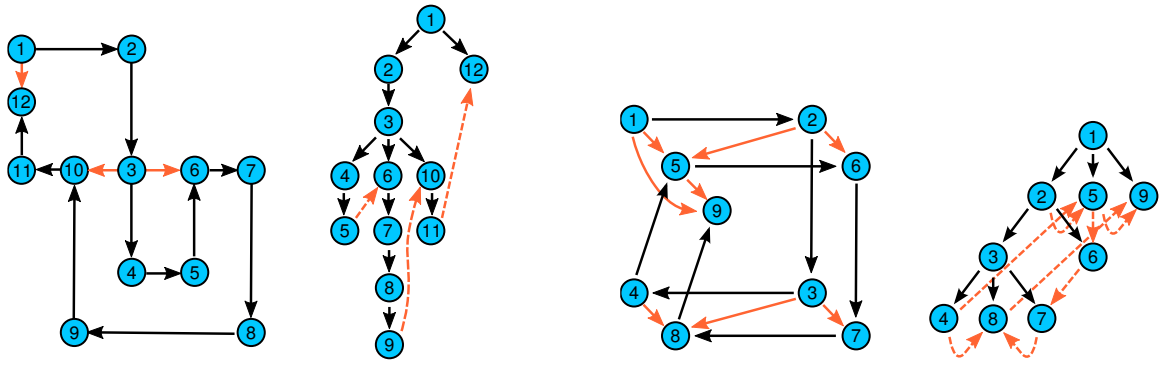

Figure 4.1: Two examples of a robot trajectory and the corresponding tree structures. The black arrows in the tree indicate the constraints between the node and its parent, i.e, the corresponding $\mathbf{x}_i$, whereas the dashed orange arrows visualize off-tree constraints, i.e., constraints that are present in the set $\mathcal{C}$ of the graph, but are not used in any $\mathbf{x}_i$, $i = 1, \ldots, n$. Observe, that for each node k, there exists a path between this node and the root node $\mathbf{x}_1$ by definition.

rotation matrix R_i (see also Equation (3.69)ff in Section 3.5) is

$$R_i = \begin{pmatrix} \cos\psi_i & -\sin\psi_i & 0 \\ \sin\psi_i & \cos\psi_i & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.9}$$

$$= \prod_{k=i_1}^{i_n} R_{\Delta k} =: R_{\Delta i_1:\Delta i_n}, \text{ with} \tag{4.10}$$

$$R_{\Delta k} = \begin{pmatrix} \cos\Delta\psi_k & -\sin\Delta\psi_k & 0 \\ \sin\Delta\psi_k & \cos\Delta\psi_k & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.11}$$

$$= \begin{pmatrix} \widetilde{R}_{\Delta k} & \begin{matrix} 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{pmatrix} \tag{4.12}$$

Using our tree parametrization, the error $e_{ij}(\mathbf{x})$ is then

$$e_{ij}(\mathbf{x}) = R_i^T(\mathbf{p}_j - \mathbf{p}_i) - \delta_{ij} \tag{4.13}$$

$$= R_i^T \left(\sum_{k=j_1}^{j_n} \mathbf{x}_k - \sum_{k=i_1}^{i_n} \mathbf{x}_k \right) - \delta_{ij} \tag{4.14}$$

Whereas Equation (4.14) looks more complex than in the PPO algorithm, we can also simplify it. Consider for example the constraint $\langle 5, 6 \rangle$ in Figure 4.1(left). To calculate the error of this constraint, we need to calculate $\mathbf{p}_6 - \mathbf{p}_5$. In this example we have $\mathbf{p}_6 = (\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_6)$, and $\mathbf{p}_5 = (\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5)$ and therefore $\mathbf{p}_6 - \mathbf{p}_5 = \mathbf{x}_6 - \mathbf{x}_5 - \mathbf{x}_4$. In other words, we need to climb up the tree from node i to the common parent node of node i and node j (which in the worst case is the root node) and then traverse the tree starting from this parent node downwards to node j. We refer to the nodes one has to traverse on the tree of a constraint as the *path* of that constraint. For example, $\mathcal{P}_{i:j} = (\mathbf{x}_{(i:j)_1}, \ldots, \mathbf{x}_{(i:j)_n})$ is the path from node i to node j given the constraint $\langle i, j \rangle$. We can divide such a path $\mathcal{P}_{i:j}$ into an ascending part $\mathcal{P}_{i:j}^{[-]}$ (from node i to the common ancestor) and into a descending part $\mathcal{P}_{i:j}^{[+]}$ starting from the common ancestor node

down to node j. In the example mentioned above, $\mathcal{P}_{5:6} = (-\mathbf{x}_5, -\mathbf{x}_4, \mathbf{x}_6)$, $\mathcal{P}_{5:6}^{[-]} = (\mathbf{x}_4, \mathbf{x}_5)$, and $\mathcal{P}_{5:6}^{[+]} = (\mathbf{x}_6)$. Note that formally $\mathbf{x}_k \in \mathcal{P}$ but we will also write $k \in \mathcal{P}$, which is an abbreviation for $k : \mathbf{x}_k \in \mathcal{P}$ in the remainder of this section. This allows us to formulate the error of a constraint as

$$
\begin{aligned}
e_{ij}(\mathbf{x}) &= (\mathbf{p}_j \ominus \mathbf{p}_i) - \delta_{ij} && (4.15)\\
&= R_i^T \left(\sum_{k=j_1}^{j_n} \mathbf{x}_k - \sum_{k=i_1}^{i_n} \mathbf{x}_k \right) - \delta_{ij} && (4.16)\\
&= R_i^T \left(\sum_{k^{[+]} \in \mathcal{P}_{i:j}^{[+]}} \mathbf{x}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{i:j}^{[-]}} \mathbf{x}_{k^{[-]}} \right) - \delta_{ij} && (4.17)\\
&= R_i^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j)\mathbf{x}_k - \delta_{ij}, && (4.18)
\end{aligned}
$$

and (for completeness) the χ^2 error of the constraint $\langle i, j \rangle$ as (see Section 3.1 for a derivation)

$$
\begin{aligned}
\chi_{ij}^2(\mathbf{x}) &= e_{ij}(\mathbf{x})^T \Omega_{ij} e_{ij}(\mathbf{x}) && (4.19)\\
&= \left(R_i^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j)\mathbf{x}_k - \delta_{ij} \right)^T \Omega_{ij} \left(R_i^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j)\mathbf{x}_k - \delta_{ij} \right). && (4.20)
\end{aligned}
$$

Here, we used $s(\mathbf{x}_k, i, j)$ to indicate to which part the node belongs to, given the path $\mathcal{P}_{i:j}$, i.e.,

$$
s(\mathbf{x}_k, i, j) = \begin{cases} +1 & \text{if } \mathbf{x}_k \in \mathcal{P}_{i:j}^{[+]} \\ -1 & \text{if } \mathbf{x}_k \in \mathcal{P}_{i:j}^{[-]} \end{cases} \tag{4.21}
$$

Given the error $e_{ij}(\mathbf{x})$ and keeping in mind that δ_{ij} is constant with respect to the configuration $\mathbf{x}$ we can now derive the Jacobian $J_{ij}(\mathbf{x})$ with

$$
\begin{aligned}
J_{ij}(\mathbf{x}) &= \frac{\partial e_{ij}(\mathbf{x})}{\partial \mathbf{x}} && (4.22)\\
&= \frac{\partial \left(R_i^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j)\mathbf{x}_k - \delta_{ij} \right)}{\partial \mathbf{x}} && (4.23)\\
&= \frac{\partial \left(R_i^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j)\mathbf{x}_k \right)}{\partial \mathbf{x}} && (4.24)\\
&= \frac{\partial \left[R_i^T \left(\sum_{k^{[+]} \in \mathcal{P}_{i:j}^{[+]}} \mathbf{x}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{i:j}^{[-]}} \mathbf{x}_{k^{[-]}} \right) \right]}{\partial \mathbf{x}}. && (4.25)
\end{aligned}
$$

Let $\mathcal{P}_{1:\mathrm{p}(i,j)}$ be the path from the root node $\mathbf{x}_1$ to the last (defined by their index) common parent of node i and j, $\mathrm{p}(i, j)$. The Jacobian $J_{ij}(\mathbf{x})$ is then build of four types of derivatives. The derivative of the error $e_{ij}(\mathbf{x})$ with respect to $\mathbf{x} \in \mathcal{P}_{1:\mathrm{p}(i,j)}$, $\mathbf{x} \in \mathcal{P}_{i:j}^{[-]}$, $\mathbf{x} \in \mathcal{P}_{i:j}^{[+]}$, and finally all

remaining $\mathbf{x}$. Thus, the Jacobian is of the form

$$J_{ij}(\mathbf{x}) = \frac{\partial \left[R_i^T \left(\sum_{k^{[+]} \in \mathcal{P}_{i:j}^{[+]}} \mathbf{x}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{i:j}^{[-]}} \mathbf{x}_{k^{[-]}} \right) \right]}{\partial \mathbf{x}} \tag{4.26}$$

$$= \left(\underbrace{A_1 \cdots A_n}_{1,\ldots,\mathrm{p}(i,j)} \underbrace{R_k^T \cdots R_k^T}_{\mathrm{p}(i,j)+1,\ldots,j} \underbrace{\mathbf{0} \cdots \mathbf{0}}_{(j+1),\ldots,n} \right) \text{ with } \mathbf{0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.27}$$

Here, the individual R_k^T belonging to the Jacobian $J_{ij}(\mathbf{x})$ are of the form

$$R_k^T = \begin{cases} -R_i^T & \text{iff } k \in \mathcal{P}_{i:j}^{[-]} \\ R_i^T & \text{iff } k \in \mathcal{P}_{i:j}^{[+]} \\ \mathbf{0} & \text{else.} \end{cases} \tag{4.28}$$

The $A_s, s \in 1, \ldots, \mathrm{p}(i,j)$ can be further expressed as

$$A_s = \frac{\partial R_i^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j) \mathbf{x}_k}{\partial \mathbf{x}_s} \tag{4.29}$$

$$\overset{\text{Eq. (4.10)}}{=} \frac{\partial (\prod_{k=i_1}^{i_n} R_{\Delta k})^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j) \mathbf{x}_k}{\partial \mathbf{x}_s} \tag{4.30}$$

$$= \frac{\partial \prod_{k=i_n}^{i_1} R_{\Delta k}^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j) \mathbf{x}_k}{\partial \mathbf{x}_s} \tag{4.31}$$

$$= \begin{pmatrix} 0 & 0 & \widetilde{R}_{\Delta i_n:\Delta s+1}^T \widetilde{R}_{\Delta s} \widetilde{R}_{\Delta s-1:\Delta i_1}^T \sum_{k \in \mathcal{P}_{i:j}} s(\mathbf{x}_k, i, j) (\Delta x_k, \Delta y_k)^T \\ 0 & 0 & \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.32}$$

Again, we see that the contribution of the A_s is proportional to the translational distance between node i and node j. Assuming this distance to be limited, we can neglect the effect of the individual A_s. Given the definition of $\mathcal{I}_k$ (see also Equation (3.85)ff), namely

$$\mathcal{I}_k = \left(\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{0,\ldots,k-1}, \underbrace{I}_{k}, \underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{k+1,\ldots,n} \right), \text{ with } I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{4.33}$$

we can finally approximate the Jacobian $J_{ij}(\mathbf{x})$ through

$$J_{ij}(\mathbf{x}) \approx R_i^T \left(\sum_{k^{[+]} \in \mathcal{P}_{i:j}^{[+]}} \mathcal{I}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{i:j}^{[-]}} \mathcal{I}_{k^{[-]}} \right). \tag{4.34}$$

In other words, the Jacobian is built of positive and negative rotational blocks, R_i^T, depending whether the corresponding node belongs to the ascending or the descending part of the path $\mathcal{P}_{i:j}$. In this case, the common parent node, $\mathrm{p}(i,j)$, is kept fixed and the error is distributed along the path $\mathcal{P}_{i:j}$. Recalling the update formula from the beginning of this section,

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \underbrace{\lambda(\tau) K_{ij}(\mathbf{x}^\tau) J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t)}_{\Delta \mathbf{x}_{ij}^t}, \tag{4.35}$$

we now need to address the remaining components, namely the preconditioning matrix, $K_{ij}(\mathbf{x}^\tau)$, the learning rate $\lambda(\tau)$ and the order of the constraints in each iteration τ. As in the case of the incremental parametrization we approximate the inverse of the Hessian, $H(\mathbf{x}^\tau)^{-1}$, at the beginning of iteration τ through

$$H^{-1}(\mathbf{x}^\tau) \approx \left[\text{diag}\left(\sum_{\langle i,j\rangle\in\mathcal{C}} J_{ij}^T(\mathbf{x}^\tau)\Omega_{ij}J_{ij}(\mathbf{x}^\tau)\right)\right]^{-1}. \tag{4.36}$$

Again, observe that $H^{-1}(\mathbf{x}^\tau)$ is now a matrix of size $3n\times 3n$ having the only non-zero elements at the diagonal. Thus we divide the diagonal into n blocks $D_k^{-1}, k=1,\ldots,n$, each of size 3×3 (also containing non-zero values on the diagonal only) such that

$$H^{-1}(\mathbf{x}^\tau) = \begin{pmatrix} D_1^{-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & D_2^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & D_n^{-1} \end{pmatrix}, \text{ with } \mathbf{0} := \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.37}$$

Updating the graph given the constraint $\langle i,j\rangle$ leads to a variation of *some* variables $\mathbf{x}_s$, with $\mathbf{x}_s \in \mathcal{P}_{i:j}$. Thus the preconditioning matrix $K_{ij}(\mathbf{x}^\tau)$ contains a total of $|\mathcal{P}_{i:j}|$ matrices on the diagonal, with $|\mathcal{P}_{i:j}|$ being the number of element of the path $\mathcal{P}_{i:j}$, which we will call the *length* of the corresponding path. These matrices are each of size 3×3, and all other entries in the matrix are zero. In more detail, $K_{ij}(\mathbf{x}^\tau)$ is of the following form:

$$K_{ij}(\mathbf{x}^\tau) = \begin{pmatrix} \mathbf{0} & & & & & & & \\ & \ddots & & & & \ddots & \vdots & \iddots \\ & & \mathbf{0} & & & \cdots & \mathbf{0} & \cdots \\ & & & W^{ij}_{\mathrm{p}(i,j)+1} & & \iddots & \vdots & \ddots \\ & & & & \ddots & & & \\ & \ddots & \vdots & \iddots & & W_j^{ij} & & \\ & \cdots & \mathbf{0} & \cdots & & & \mathbf{0} & \\ & \iddots & \vdots & \ddots & & & & \ddots \\ & & & & & & & \mathbf{0} \end{pmatrix}, \tag{4.38}$$

where the individual W_k^{ij} of $K_{ij}(\mathbf{x}^\tau)$ with $k\in[\mathrm{p}(i,j)+1,\ldots,j]$ are the only *possible* non-zero blocks. These are computed using Equation (4.37) as

$$W_k^{ij} = \begin{cases} \mathrm{s}(\mathbf{x}_k,i,j)|\mathcal{P}_{i:j}|\left[\sum_{m\in\mathcal{P}_{i:j}} D_m^{-1}\right]^{-1} D_k^{-1} & \text{if } k\in\mathcal{P}_{i:j} \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{4.39}$$

Note that our approach updates $|\mathcal{P}_{i:j}|$ variables rather than $(j-i)$ as in the case of PPO (see Equation (3.98)) since in PPO, all of the W_k^{ij} are non-zero. Again, $\mathrm{s}(\mathbf{x}_k,i,j)$ is defined as

$$\mathrm{s}(\mathbf{x}_k,i,j) = \begin{cases} +1 & \text{if } \mathbf{x}_k\in\mathcal{P}_{i:j}^{[+]} \\ -1 & \text{if } \mathbf{x}_k\in\mathcal{P}_{i:j}^{[-]}. \end{cases} \tag{4.40}$$

As proposed in previous work [114], we clamp the individual W_{ij}^k to a maximum (see Section 4.3) preventing an update from overshooting. Furthermore we choose a learning rate $\lambda(\tau)$ (see Equation (3.91)ff) as

$$\lambda(\tau) \;=\; 1/(\tau+2), \text{ for } \tau \geq 1. \tag{4.41}$$

Now that we have defined all components of the update rule (see Equation (4.35)) it remains open how to order the set of constraints, $\mathcal{C}$, in each iteration τ. In general, we could also permute the set $\mathcal{C}$ at the beginning of each iteration but our incremental tree provides us with a natural order which results in a reduced complexity. To understand why this is the case let us again have a look at the Jacobian, which is

$$J_{ij}(\mathbf{x}) \overset{\text{Eq. (4.34)}}{=} R_i^T \left(\sum_{k^{[+]} \in \mathcal{P}_{i:j}^{[+]}} \mathcal{I}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{i:j}^{[-]}} \mathcal{I}_{k^{[-]}} \right). \tag{4.42}$$

Thus we have to calculate $R_i = \prod_{k=i_1}^{i_n} R_{\Delta k}$ in order to update the variables, given the constraint $\langle i,j \rangle$ and the path $\mathcal{P}_{1:i} = (\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_n})$. Let the *level* of a node be the distance in the tree between the node itself and the root node (i.e., the depth in the tree), which implies levelOf$(\mathbf{x}_i) \leq i$. Let furthermore $\underline{\mathbf{x}}_{ij}$ be the node in the path $\mathcal{P}_{i:j}$ with the smallest level. The *level* of the constraint $\langle i,j \rangle$ is then defined as the level of $\underline{\mathbf{x}}_{ij}$. The common parent of the nodes i and j, p(i,j), is kept fixed and only nodes which have a bigger depth than the common parent are changed. In other words, our parametrization implies that updating a constraint will never change the configuration of a node with level smaller than the level of the constraint. Based on this knowledge, we can sort the constraints according to the level and process them in that order. As a result, it is sufficient to access the parent of $\underline{\mathbf{x}}_{ij}$ to compute R_i since all other nodes having a smaller level than $\underline{\mathbf{x}}_{ij}$ have already been corrected in this iteration. Otherwise, and it is the case in the previous approach, we would need to re-calculate R_i each time from scratch resulting in a higher complexity per iteration. Figure 4.2 illustrates such a situation. Each image consists of the trajectory (top) and the tree (bottom). Figure 4.2(a) shows the initial set up of the tree and the remaining images (b-f) visualize the processing order of the constraints, emphasized by the orange dashed arrows. Here, the processing order of the constraints is $\langle 8,9 \rangle, \langle 7,8 \rangle, \langle 11,12 \rangle, \langle 6,7 \rangle$, and $\langle 5,6 \rangle$. Note that we only provided the order of the off-tree constraints (i.e., of loops in the trajectory) and omitted to include constraints like $\langle 1,2 \rangle$ for simplicity. In Figure 4.2(b-f), the nodes of the path $\mathcal{P}_{i:j}$ and the corresponding common parent (which does not belong to the path) are shaded in the tree (bottom part). Within the trajectory (top part) the very same nodes are highlighted for better visualization. Optimizing such a network using the incremental parametrization rather than ours would lead to many updates in the nodes as will be shown in the experimental section of this chapter (Section 4.6). Consider for example the loop $1, 2, \ldots, 9$. Whereas in previous work, updating the constraint $\langle 1,9 \rangle$ results in a variation of the nodes $2, \ldots, 9$, our parametrization leads to a variation of the nodes 8 and 9 only.

The differences between the individual components of our incremental tree parametrization approach and the incremental path parametrization approach (PPO) (see Section 3.5) are summarized in Table 4.1. Here we can see, that our algorithm has a comparable update rule as well (and thus a comparable complexity per iteration). However, we will show in the next section, that our incremental tree parametrization leads to an algorithm needing substantially fewer iterations to converge than the previous approach based on the incremental parametrization only. Up to now, we have provided the update rule for the 2D case (i.e., $\mathbf{p}_i = (x_i, y_i, \psi_i)^T$)

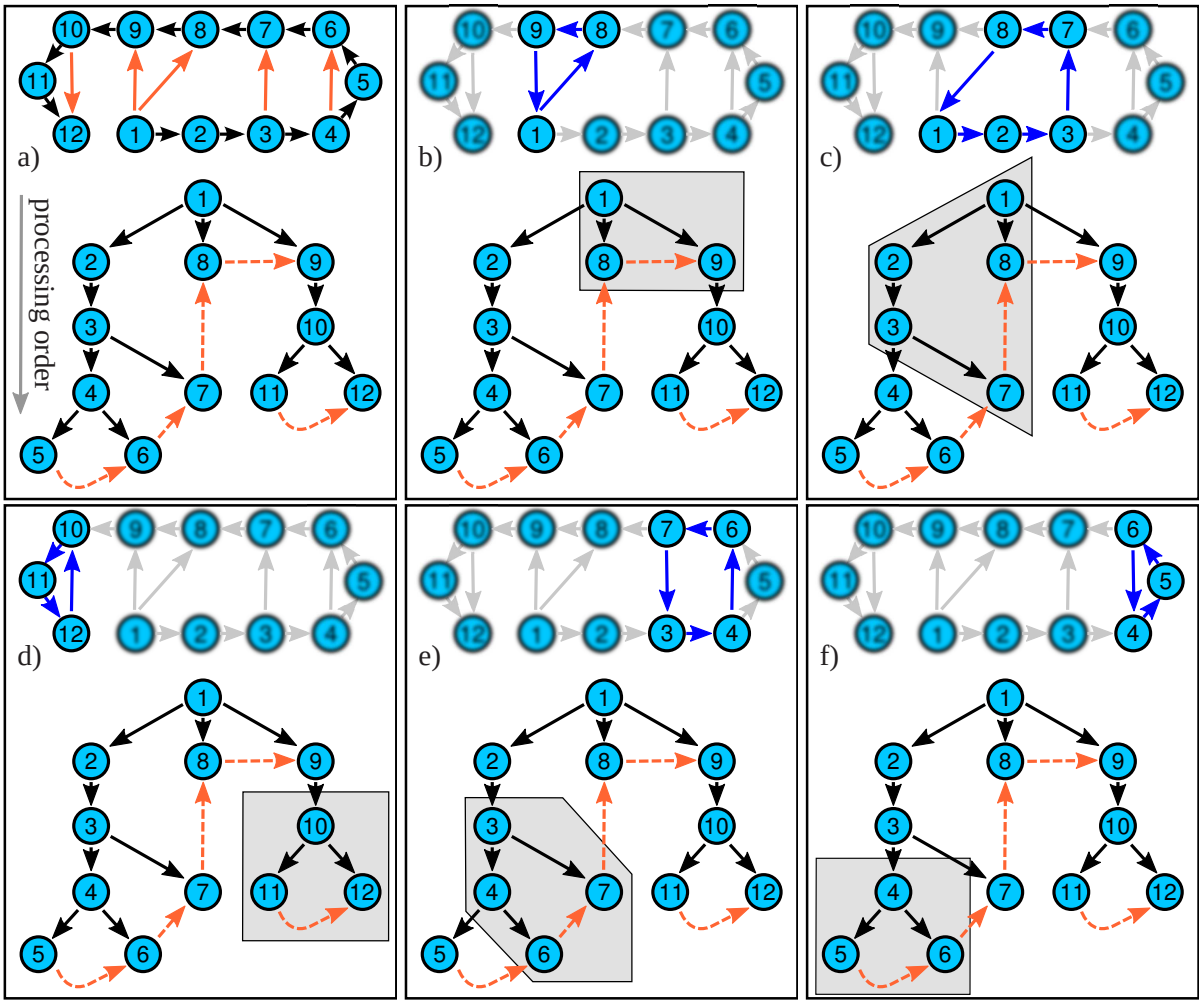

Figure 4.2: This example visualizes the processing order of the constraints. Each image (a-f) shows the trajectory (top part) and the tree (bottom part). The initial situation is shown in (a), whereas the images (b-f) show the processing of the constraints $\langle 8, 9\rangle, \langle 7, 8\rangle, \langle 11, 12\rangle, \langle 6, 7\rangle$, and $\langle 5, 6\rangle$. Here, the nodes affected by the constraint, including the common parent, are shaded within the tree and highlighted in the trajectory for better visualization. Note that this images show the processing of the off-tree constraints (dashed orange arrows) only but in general the tree-constraints, like $\langle 1, 2\rangle$ are also processed.

only. Although this approach can be extended towards 2.5D, i.e., $\mathbf{p}_i = (x_i, y_i, z_i, \psi_i)^T$ in a straightforward manner, the current form of the optimization algorithm can not deal with arbitrary 3D rotations. After providing an analysis of our current algorithm in the next section, we will therefore describe our update rule for the three-dimensional case in Section 4.3.

4.2 Analysis of the Algorithm

This section is designed to give a more detailed understanding on the effect of the incremental tree parametrization with respect to PPO as described in Section 3.5. We generated a data set by simulating a robot moving in a grid world. A schematic view of the trajectory is depicted in Figure 4.3(a). The robot starts at node 1 and subsequently moves to nodes $2, 3, \ldots, 23$. However, a path from one node to another in this description consists of total 6 nodes in our simulated data, leading to a network containing a total of $23 \cdot 6 = 138$ nodes (poses). The robot observes all other locations (nodes) in its local vicinity which results in 500 constraints. Both poses and edges are corrupted by a zero mean Gaussian noise having a standard deviation

	PPO	**Our approach**
$\mathbf{x}^{t+1}$	$= \mathbf{x}^t + \Delta\mathbf{x}_{ij}^t$, for a constraint $\langle i, j\rangle \in \mathcal{C}$	
$\Delta\mathbf{x}_{ij}^t$	$= \lambda(\tau) K_{ij}(\mathbf{x}^\tau) J_{ij}^T(\mathbf{x}^t) \Omega_{ij} r_{ij}(\mathbf{x}^t)$, for iteration τ	
$\mathbf{x}$	$\mathbf{x}_1 = \mathbf{p}_1$	
	$\mathbf{x}_i = \mathbf{p}_i - \mathbf{p}_{i-1},\ i > 1$	$\mathbf{x}_i = \mathbf{p}_i - \text{parent}(\mathbf{p}_i),\ i > 1$
$\mathcal{P}_{i:j}$	$= (\mathbf{x}_i, \ldots, \mathbf{x}_j)$	$= (\mathbf{x}_{(i:j)_1}, \ldots, \mathbf{x}_{(i:j)_{\lvert\mathcal{P}_{i:j}\rvert}}) = \mathcal{P}_{i:j}^{[+]} \cup \mathcal{P}_{i:j}^{[-]}$
$J_{ij}(\mathbf{x})$	$\approx R_i^T \sum_{k=i+1}^{j} \mathcal{I}_k$	$\approx R_i^T \left(\sum_{k^{[+]} \in \mathcal{P}_{i:j}^{[+]}} \mathcal{I}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{i:j}^{[-]}} \mathcal{I}_{k^{[-]}} \right)$
K_{ij}	build of diagonal blocks W_{ij}^k	
W_{ij}^k	$\begin{cases} (j-i) \left[\sum_{m=i+1}^{j} D_m^{-1} \right]^{-1} D_k^{-1} & , k \in \mathcal{P}_{i:j} \\ \mathbf{0} & , \text{otherwise} \end{cases}$	$\begin{cases} \lvert\mathcal{P}_{i:j}\rvert \left[\sum_{m \in \mathcal{P}_{i:j}} D_m^{-1} \right]^{-1} D_k^{-1} & , k \in \mathcal{P}_{i:j}^{[+]} \\ -\lvert\mathcal{P}_{i:j}\rvert \left[\sum_{m \in \mathcal{P}_{i:j}} D_m^{-1} \right]^{-1} D_k^{-1} & , k \in \mathcal{P}_{i:j}^{[-]} \\ \mathbf{0} & , \text{otherwise} \end{cases}$
$\lambda(\tau)$	$= 1/\lambda(\tau + 2),\ \tau > 1$	
$\mathcal{C}$	random permutation in each iteration τ	ordered wrt. tree parametrization

Table 4.1: Comparison of the optimization algorithms and their individual components. The individual components of PPO [116] (see previous chapter) are summarized in the left column whereas the right column states the components given our tree parametrization, respectively.

of 0.1 along all axes. The network obtained using these parameters is shown in Figure 4.3(b).

The evolution of the network after iteration $1, 5, 10$, and 30 for both approaches is shown in Figure 4.5. Comparing the result of PPO (left column) to our incremental tree approach (right column) we see that our approach leads visually to a more consistent graph already after five iterations compared to 30 iterations in the case of PPO. To quantitatively evaluate both approaches we calculated the χ^2 error per constraint. This number is obtained by dividing the overall χ^2 error (see Equation (3.17)) by the number of constraints, $|\mathcal{C}|$, and reflects the average error in the network. Figure 4.4 depicts the χ^2 error per constraint for PPO (dashed line) as well as ours (solid line) for the first 1000 iterations.

As stated in the beginning of this chapter, our incremental tree optimization algorithm needs fewer iterations until convergence (see also Figure 4.4). Again, the reason for this is that our approach typically updates fewer (but sufficiently many) nodes when distributing an error along a loop. This indeed has the effect that constraints having contrary effects on nodes will lead to less oscillations in the graph. To emphasize this let us focus on two nodes in Figure 4.3, namely node 3 and node 22. These nodes have been selected due to their different level of connectivity, i.e., the number of connected edges. The evolution of the nodes in each axis is shown in Figure 4.6 (top rows) and a scaled version of four intermediate iterations is shown in the bottom row of the same figure. Note that the $x-$axes of the plots show the number of steps (i.e., change of the network after updating one constraint) rather than iterations. Here, each iteration is equal to 500 steps, since $|\mathcal{C}| = 500$ and thus the four iterations are equal to 2000 steps. Observe that already node 3, although having the smallest connectivity in the network, is affected by a high variation in the case of PPO whereas this variation is significantly smaller using our approach. The oscillations and the difference in variation becomes even more severe when comparing nodes having a higher connectivity as indicated by node 22. Our incremental

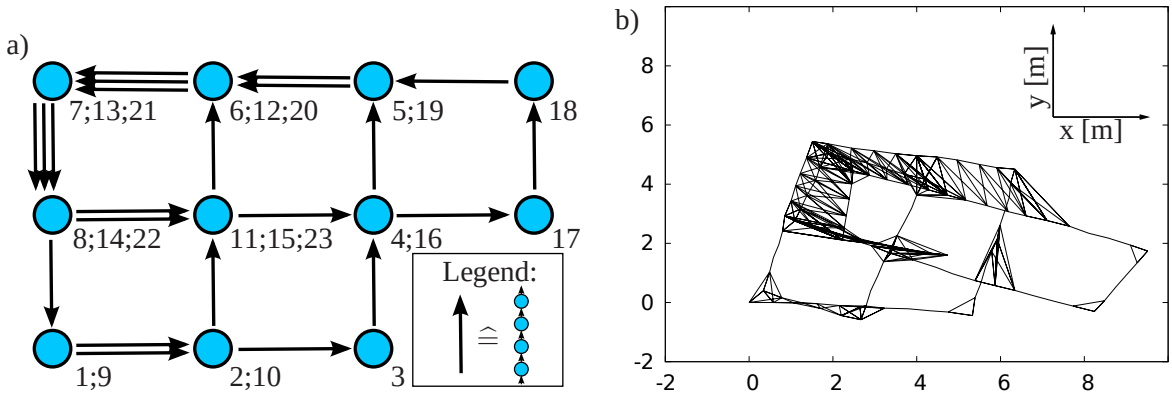

Figure 4.3: Schematic view of the trajectory (a) and the network (b) obtained by corrupting both the poses as well as the edges by zero mean Gaussian noise. The parameters of the Gaussian noise are found in the text.

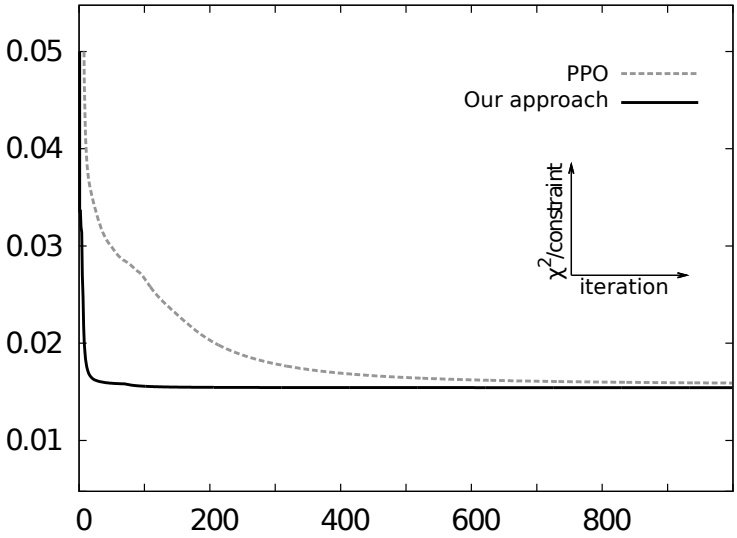

Figure 4.4: χ^2-error per constraint for the data set depicted in Figure 4.3 given our incremental tree approach (solid black line) and the path parametrized approach, PPO (dashed gray line).

tree structure allows us to decompose the optimization problem into a set of weakly interacting problems, namely each sub-graph in the tree structure. Thus, a node is less likely to be updated by other constraints. A good measure for evaluating the interaction between the constraints is the average path length l of updated nodes per constraint. For example, a network with a large value for l has typically a higher number of interacting constraints compared to networks with a low values of l. In all experiments, where we randomly generated graphs with a total number of constraints between around $4,000$ and 2 millions, our approach had a value for l between 3 and 7 (ignoring constraints between successor variables). In contrast to that, this value varies between 600 and $17,000$ in PPO on the same networks. This indeed reduces the convergence speed of PPO but does not introduce a higher complexity.

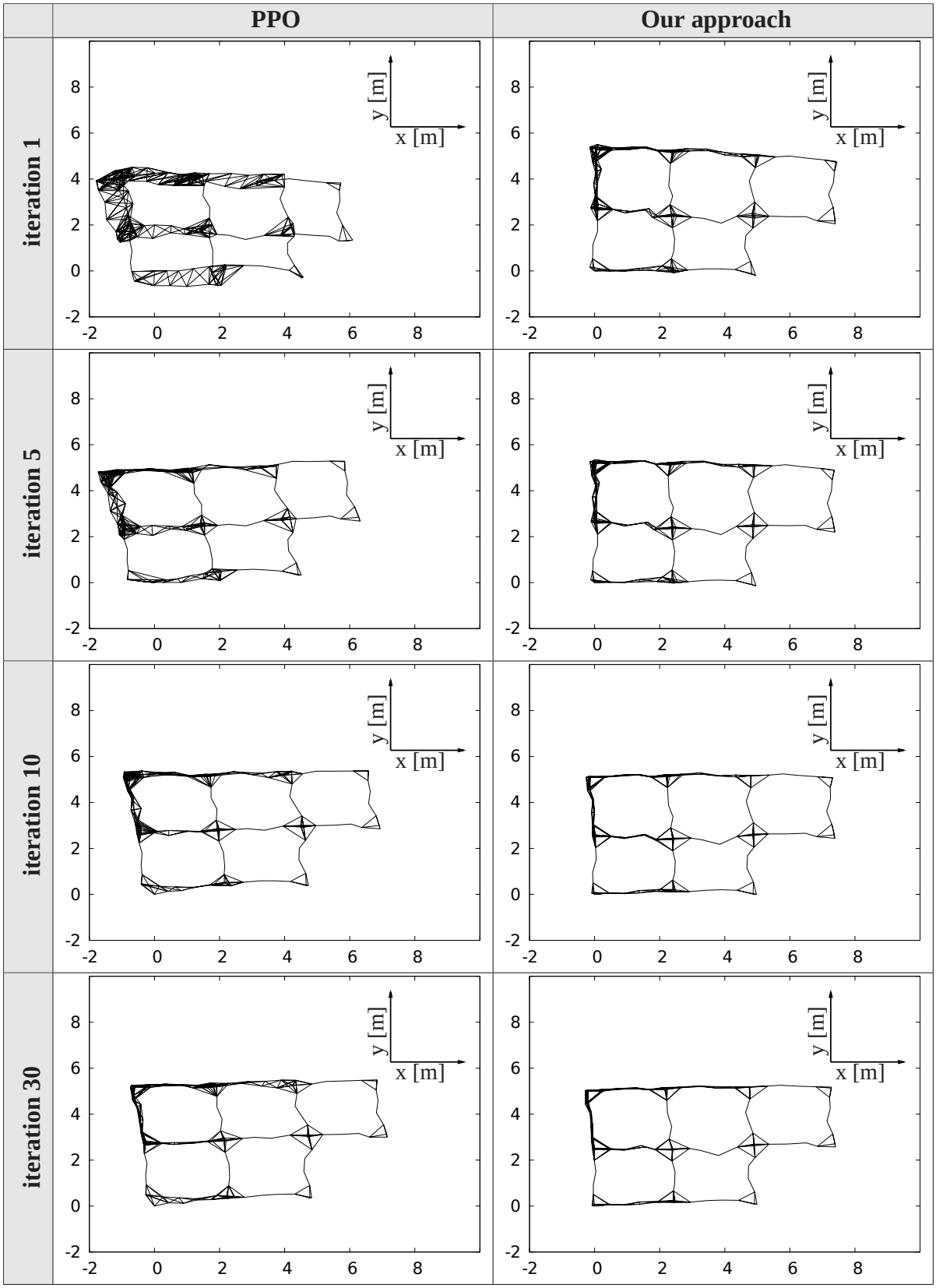

Figure 4.5: Evolution of the network depicted in Figure 4.3 given the incremental approach (left column) and ours (right column) for the iterations $1, 5, 10$, and 30. As can be seen, our approach leads visually to a more consistent network after 5 iterations than the incremental approach after 30 iterations.

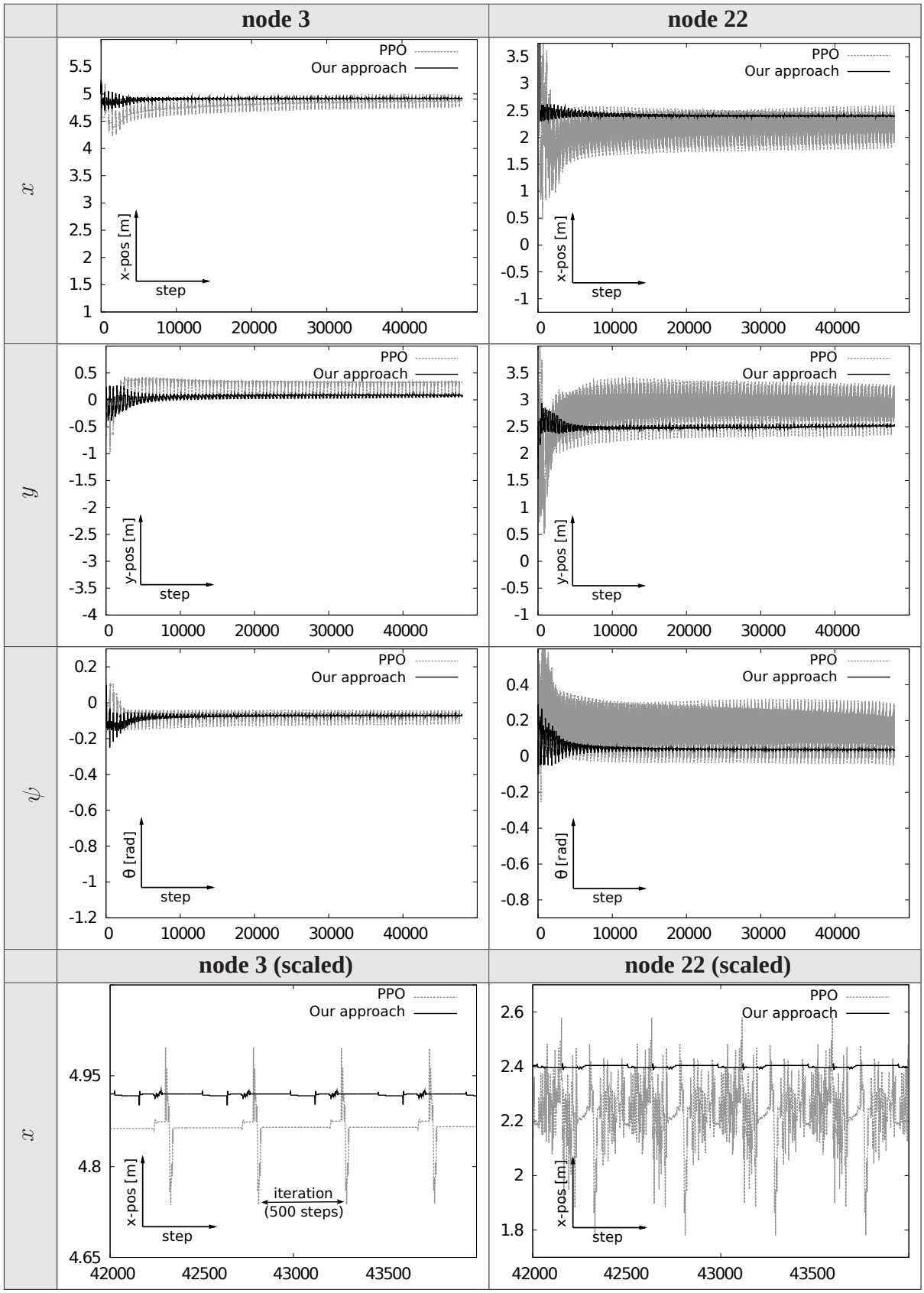

Figure 4.6: Evolution of typical nodes in a small data set. The first three rows depict the variation of the node 3 and 22 for all axis (i.e., x, y, ψ) whereas the bottom row depicts a scaled version of the x-axis over the period of 2000 steps, equal to four iterations. The outcome for PPO is shown using gray dashed lines, whereas the result of our incremental tree approach is visualized in solid black.

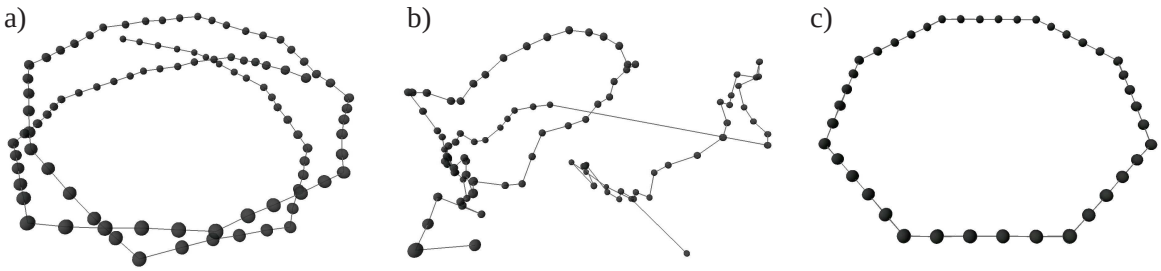

Figure 4.7: A small example that illustrates the problem of distributing the error in 3D. The input data that was obtained by moving a simulated robot over a hexagon twice with small Gaussian noise is shown in (a). The middle image (b) show the result obtained if the non-commutativity of the rotation angles is ignored, i.e., this image shows the result if applying the approach presented in the previous chapter. The result of our approach presented in this section is shown in (c) which is very close to the ground truth.

4.3 Distributing the Error in 3D

The reader might pose the question whether it is possible to apply the update rule of the previous section to full 3D rotations. Unfortunately, this is not possible. To understand the effect of this statement, consider the error distribution for the 2D case first. Here, we can distribute a residual $r^{\text{2D}} = (r_x, r_y, r_\psi)^T$ along a chain of n nodes by changing the pose of the i-th node by $(r_x/n, r_y/n, r_\psi/n)^T$, given the incremental (tree) parametrization. In the three-dimensional space, however, such a technique is not applicable due to the non-commutativity of the three rotations as visualized in Figure 4.7. In other words, comparing the 2D and the 3D case we have

$$R_{\text{2D}}(\psi) = \prod_{i=1}^{n} R_{\text{2D}}(\frac{\psi}{n}), \text{ but in general} \tag{4.43}$$

$$R_{\text{3D}}(\phi, \theta, \psi) \neq \prod_{i=1}^{n} R_{\text{3D}}(\frac{\phi}{n}, \frac{\theta}{n}, \frac{\psi}{n}), \tag{4.44}$$

where R_{2D} and R_{3D} are the rotation matrices for the two-dimensional and three-dimensional case. Here, the rotations along the $x, y,$ and z axis are *roll* (ϕ), *pitch* (θ), and *yaw* (ψ).

Given the notation of motion composition, $\oplus$, and its inverse, $\ominus$, as defined by Smith and Cheeseman [138] and Lu and Milios [100] (see Section 2.2) we define the incremental tree parametrization for the full 3D case (i.e., $\mathbf{p}_i = (x_i, y_i, z_i, \phi_i, \theta_i, \psi_i)^T$ as follows. The configuration space $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^T$ with $\mathbf{x}_i = (\Delta x_i, \Delta y_i, \Delta z_i, \Delta\phi_i, \Delta\theta_i, \Delta\psi_i)^T$ is calculated similar to the 2D case (see previous section) with:

$$\mathbf{x}_1 = \mathbf{p}_1 \quad \text{(the root node)} \tag{4.45}$$

$$\mathbf{x}_i = \mathbf{p}_i \ominus \text{parent}(\mathbf{p}_i), \quad \text{for } i > 1. \tag{4.46}$$

Similarly, the residual of a constraint $\langle i, j \rangle$ is

$$r_{ij}(\mathbf{p}) = (\mathbf{p}_i \oplus \delta_{ij}) \ominus \mathbf{p}_j. \tag{4.47}$$

For simplicity of notation, we will refer to the homogeneous transformation matrix of the vector $\mathbf{x}_i$ as X_i. This matrix of size 4×4 is build of a rotational component $R_{\Delta i}$ and a translational

component $\mathbf{t}_{\Delta i}$ with

$$X_i = \begin{pmatrix} R_{\Delta i} & \mathbf{t}_{\Delta i} \\ \mathbf{0} & 1 \end{pmatrix}, \mathbf{0} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^T, \mathbf{t}_{\Delta i} = \begin{pmatrix} \Delta x_i \\ \Delta y_i \\ \Delta z_i \end{pmatrix}, \text{and } X_i^{-1} = \begin{pmatrix} R_{\Delta i}^T & -R_{\Delta i}^T \mathbf{t}_{\Delta i} \\ \mathbf{0} & 1 \end{pmatrix}.$$

Using $\mathrm{c}(\alpha), \mathrm{s}(\alpha)$ short for $\cos\alpha$ and $\sin\alpha$ respectively the rotation matrix is calculated as

$$R_{\Delta i} = \begin{pmatrix} \mathrm{c}(\Delta\psi_i) & -\mathrm{s}(\Delta\psi_i) & 0 \\ \mathrm{s}(\Delta\psi_i) & \mathrm{c}(\Delta\psi_i) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{c}(\Delta\theta_i) & 0 & \mathrm{s}(\Delta\theta_i) \\ 0 & 1 & 0 \\ -\mathrm{s}(\Delta\theta_i) & 0 & \mathrm{c}(\Delta\theta_i) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathrm{c}(\Delta\phi_i) & -\mathrm{s}(\Delta\phi_i) \\ 0 & \mathrm{s}(\Delta\phi_i) & \mathrm{c}(\Delta\phi_i) \end{pmatrix}.$$

Similarly, we refer to the homogeneous transformation matrix of a vector $\mathbf{p}_i$ as P_i, and of the observation δ_{ij} as Δ_{ij}. Accordingly, we can compute the residual $r_{ij}(\mathbf{x})$ in the reference frame of node j (see Equation (4.47)) as

$$r_{ij}(\mathbf{x}) = P_j^{-1}(P_i \Delta_{ij}) \tag{4.48}$$

$$= \left(\prod_{k^{[+]} \in \mathcal{P}_{ij}^{[+]}} X_{k^{[+]}} \right)^{-1} \left(\prod_{k^{[-]} \in \mathcal{P}_{ij}^{[-]}} X_{k^{[-]}} \right) \Delta_{ij}. \tag{4.49}$$

At this point we can directly compute the Jacobian from the residual and apply Equation (4.35) to update the network. Although the resulting Jacobian has exactly $|\mathcal{P}_{i:j}|$ non zero blocks it does not have the simple form as the one in the previous section and can hardly be calculated manually. Even more, updating the network using this Jacobian would lead to a poor performance of the algorithm in case of large optimization problems. To understand this behavior, recall that the goal of the update rule is to iteratively reduce the (χ^2)-error introduced by a constraint. In Equation (4.35), the term $J_{ij}\Omega_{ij}$ maps the residual into a variation of the configuration space, $\mathbf{x}$. This mapping, however, is a *linear* function. As illustrated by Frese and Hirzinger [55], the error might increase when applying such a linear function in case of non-linear error surfaces. In the three-dimensional space, the three rotational components, roll, ϕ, pitch, θ, and yaw ,ψ, often lead to highly non-linear error surfaces. Therefore it is problematic to apply such an approach as well as similar minimization techniques directly to large mapping problems, especially in combination with a high noise in the observations. Furthermore, this also leads to a poor distribution of the error along the nodes, since the Jacobian is also used for approximating the Hessian, which in turn provides us with a distribution of the error along the nodes according to their uncertainty.

In our approach, we therefore choose a modified update rule. We apply a *non-linear* function to map the residual into a variation in the parameter space which is presented in the following. As in the linear case, the goal of this function is to compute a transformation of the nodes along the path $\mathcal{P}_{i:j}$ (of the tree) such that the error introduced by the corresponding constraint $\langle i, j \rangle$ is reduced. Intuitively, we decouple the rotation and the translation during optimization leading to an two step update per iteration. First, we keep the translational part fixed and update the network given the rotational error only, then we update the translational part, given the corrected rotations. In the latter case, the rotational part is independent of $\mathbf{x}$ (since it is kept fixed) and thus the corresponding Jacobian maintains its simple form similar to Equation (4.42).

To this end, we consider without loss of generality the origin ouf our reference system to be the origin $\mathbf{p}_i$ of the path $\mathcal{P}_{i:j} = (\mathbf{x}_{(i:j)_1}, \ldots, \mathbf{x}_{(i:j)_n})$. Thus, the orientation of $\mathbf{p}_j$ in the reference

system of $\mathbf{p}_i$ can be computed by multiplying the rotational matrices along the path $\mathcal{P}_{i:j}$. For better readability, we refer to this matrices as $R_{\Delta_1}, \ldots, R_{\Delta_n}$. Note that all rotational matrices of the form R_{Δ_k} indicate incremental rotations whereas we will omit the index Δ in the case of absolute values. Using this notation the orientation of $\mathbf{p}_j$ with respect to the reference system of $\mathbf{p}_i$ is

$$R_{\Delta_1:\Delta_n} \quad = \quad R_{\Delta_1} R_{\Delta_2} \cdots R_{\Delta_n}, \tag{4.50}$$

with n being the length of the path, i.e., $n = |\mathcal{P}_{i:j}|$. In case of 3D rotations we need to describe the error B as a set of increments that can be applied as intermediate rotations. In other words, we need to determine a set of increments for the intermediate rotations $R_{\Delta_1}, \ldots, R_{\Delta_n}$ of the chain so that the orientation of the last node (here node j) is $R_{\Delta_1:\Delta_n} B$, i.e., we seek a set of new rotation matrices $R'_{\Delta_1}, \ldots, R'_{\Delta_n}$ with

$$R_{\Delta_1:\Delta_n} B \quad = \quad \prod_{k=1}^{n} R'_{\Delta_k}. \tag{4.51}$$

To calculate the desired rotation matrices in the local reference frame of $\mathbf{p}_i$, we first have to transform the error B into the global reference frame, yielding Q with

$$Q \quad = \quad R_n B R_n^T, \text{ where} \tag{4.52}$$

R_n denotes the rotation of node $\mathbf{x}_{(i:j)_n}$ in the global reference frame. Now we can decompose the error Q into a set of incremental rotations

$$Q = Q_{\Delta_1:\Delta_n} \quad := \quad Q_{\Delta_1} Q_{\Delta_2} \cdots Q_{\Delta_n} \tag{4.53}$$

by using spherical linear interpolation (slerp) [135] for the individual Q_k. Here, given a parameter $w \in [0, 1]$, spherical linear interpolation is defined as $\text{slerp}(Q, w)$ with $\text{slerp}(Q, 0) = I$ and $\text{slerp}(Q, 1) = Q$ (see also Section 2.3). This allows us to calculate the incremental rotations as

$$Q_{\Delta_1} \quad = \quad \text{slerp}(Q, w_1^{ij}), \text{ and} \tag{4.54}$$

$$Q_{\Delta_k} \quad = \quad \left[\text{slerp}(Q, w_{k-1}^{ij})\right]^T \text{slerp}(Q, w_k^{ij}), \text{ for } k > 1 \tag{4.55}$$

This ensures a rotational error distribution, since

$$\begin{aligned} Q_{\Delta_1:\Delta_2} &= Q_{\Delta_1} Q_{\Delta_2} && (4.56)\\ &= \text{slerp}(Q, w_1^{ij}) \left[\text{slerp}(Q, w_1^{ij})\right]^T \text{slerp}(Q, w_2^{ij}) && (4.57)\\ &= \text{slerp}(Q, w_2^{ij}) && (4.58)\\ & && (4.59) \end{aligned}$$

and thus it follows by induction that

$$Q_{\Delta_1:\Delta_n} \quad = \quad \text{slerp}(Q, w_n^{ij}). \tag{4.60}$$

Now, given the incremental rotational error terms in the global reference frame we need to transform them into the local reference frame with the origin in $\mathbf{p}_i$. We therefore first compute the resulting orientation in the global reference frame

$$R'_k \quad = \quad Q_{\Delta_1:\Delta_k} R_k \tag{4.61}$$

with R_k being the rotation of node $\mathbf{x}_{(i:j)_k}$ in the global reference. Now, we can finally compute the desired R'_{Δ_k} (in the local reference frame) by

$$R'_{\Delta_k} \quad = \quad \left[R'_{\text{parent}(k)}\right]^T R'_k. \tag{4.62}$$

Note that in the equation above we implicitly use

$$R'_{\text{parent}(k)} \quad = \quad R_{\text{parent}(k)}, \text{ if parent}(k) \notin \mathcal{P}_{i:j}. \tag{4.63}$$

Although we can now calculate the intermediate rotations, R'_{Δ_k}, we left open how to obtain the corresponding w_k^{ij}, given the constraint $\langle i, j \rangle$. Since we do not want to overshoot when updating a constraint, i.e., the resulting rotation should not exceed the error B, we incorporate the learning rate and the path length when computing the values w_k^{ij} similar to the W_k^{ij} in the 2D case and clamp the resulting value to a maximum. In more detail, we compute the w_k^{ij} with $w_k^{ij} \in [0, 1]$ as

$$w_k^{ij} \quad = \quad \min(1, \lambda|\mathcal{P}_{i:j}|) \left[\sum_{m \in \mathcal{P}_{i:j}} d_m^{-1}\right]^{-1} \left[\sum_{m \in \mathcal{P}_{i:j} \wedge m \leq k} d_m^{-1}\right]. \tag{4.64}$$

Here, d_m is the sum of the smallest eigenvalues of the information matrices Ω_{im}, Ω_{mi} of all constraints connected to node m, thus

$$d_m \quad = \quad \sum_{\langle i,m \rangle \in \mathcal{C}} \min\left[\text{eigenvalues}(\Omega_{im})\right] + \sum_{\langle m,i \rangle \in \mathcal{C}} \min\left[\text{eigenvalues}(\Omega_{mi})\right]. \tag{4.65}$$

We found out that this approximation works well in practice for roughly spherical covariances. Note that we can compute the eigenvalues once in the beginning and store the values in the tree. To get a better intuition about the weights w_k^{ij}, compare Equation (4.64) to Equation (4.39), where the diagonal weight matrix W_k^{ij} is calculated for each $k \in \mathcal{P}_{i:j}$ through

$$W_k^{ij} \quad \overset{\text{Eq. (4.39)}}{=} \quad \text{s}(\mathbf{x}_k, i, j)|\mathcal{P}_{i:j}| \left[\sum_{m \in \mathcal{P}_{i:j}} D_m^{-1}\right]^{-1} D_k^{-1}. \tag{4.66}$$

The first part on the right hand side of Equation (4.66) scales the weight proportional to the path length $|\mathcal{P}_{i:j}|$ and sets the direction accordingly, given the node belongs to the ascending or descending part of the path. This is equivalent to the first term in Equation (4.64). Note that here the learning rate and the clamping is already done in one step whereas the very same operation is decoupled from the weight calculation in the 2D case. Comparing both equations in more detail, we observe that the difference lies within two points. First, the calculation of the individual d_m, D_m, and second in the rightmost part of both equations. While we calculate the sum of the d_m up to the current node k in the first equation (Eq. (4.64)), we use the individual D_k in the 2D case only (see Equation (4.66)). Indeed, this only looks like a difference, since they follow the same principle. The reason for the different look-a-like origins from the reference system the weights are calculated in. In the 2D case, the W_k are calculated in the relative reference frame implicitly containing the $D_m, m < k$. In contrary to this, the w_k are calculated in the global reference frame. Thus, the remaining difference between the weight calculation lies within the difference between d_m and D_m. First of all d_m is a scalar while D_m is a matrix. This results from the slerp where we have only one parameter. In other words, we average

the diagonal elements of D_k into the scalar d_k. Recall, that D_m is approximated through the diagonal elements of the Hessian as can be seen in Equation (4.36). In more detail, the D_k are calculated as

$$D_k = \text{diag}\left(\sum_{\langle i,m\rangle\in\mathcal{C}} J_{im}^T\Omega_{im}J_{im} + \sum_{\langle m,j\rangle\in\mathcal{C}} J_{mj}^T\Omega_{mj}J_{mj}\right), \tag{4.67}$$

and therefore, D_m is proportional to the uncertainty, which in turn, is proportional to the eigenvalues of the covariance matrix. Thus, we observe, that the weight calculation for our modified update rule originates from the same principle as the weight calculation for the 2D case.

As stated in the beginning of this section, we decouple each iteration into two steps. In the first step we update the rotational components only as described above. Now, given the corrected nodes (with respect to the rotation) we correct the translational part in the second step. To this end, we consider the rotation to be fixed when correcting the translational part and parametrize the tree as in the 2D case omitting the rotational part (see previous section), i.e. use the vector subtraction instead of the motion composition operator $\ominus$. Indeed, the Jacobians have now an even simpler form, since the rotational part is constant with respect to the configuration $\mathbf{x}$ and we therefore obtain

$$J_{ij}(\mathbf{x}) = \sum_{k^{[+]}\in\mathcal{P}_{i:j}^{[+]}} \mathcal{I}_{k^{[+]}} - \sum_{k^{[-]}\in\mathcal{P}_{i:j}^{[-]}} \mathcal{I}_{k^{[-]}}. \tag{4.68}$$

Note that in Equation (4.68) the Jacobian is not approximated by the right therm of this equation but is exactly the sum of the identity matrices, which is not the case in Equation (4.34). It is noteworthy, that this allows us to randomize the order of the constraints in each iteration, since we do not need to calculate R_i along the tree anymore.

Given the equations above one may ask, why we introduced the pure 2D optimization in the previous section first. The reason for this is the spherical linear interpolation needed in the three-dimensional case which is computationally expensive. Although the approach described in this section also works for the two-dimensional case it is about five times slower than our pure 2D version of the algorithm and we recommend the usage of the previous approach when dealing with two-dimensional problems. Using our approach we are now able to correct networks containing full 3D rotations. However, it is important that the angular change in the residual is limited when updating a constraint in order to prevent a flip along a rotational axis [58] that would lead to a sub-optimal configuration of the nodes (see for example Figure 4.8(top)). Such an error will affect any modules depending on the outcome of this algorithm, including mapping and path planning. We therefore first analyze our rotational error distribution in the next section and subsequently present our node reduction technique in Section 4.5.

4.4 Analysis of the Rotational Residual in 3D

When distributing the rotational error along a chain of nodes $i,\ldots,j$ one may increase the rotational value of the residual between two successive nodes $k-1$ and k, denoted as $r_{k-1,k}$. For the convergence of our approach however it is inevitable that the change of the error is bounded. Otherwise, competing constraints could result in a flip along a rotational axis as indicated in Figure 4.8. The top image displays the result if the requirement above does not hold [58, 62]. The errors in the optimized graph results from two competing constraints, i.e., rotation in the opposite direction, leading to a full rotation along the roll axis (i.e., 2π). The

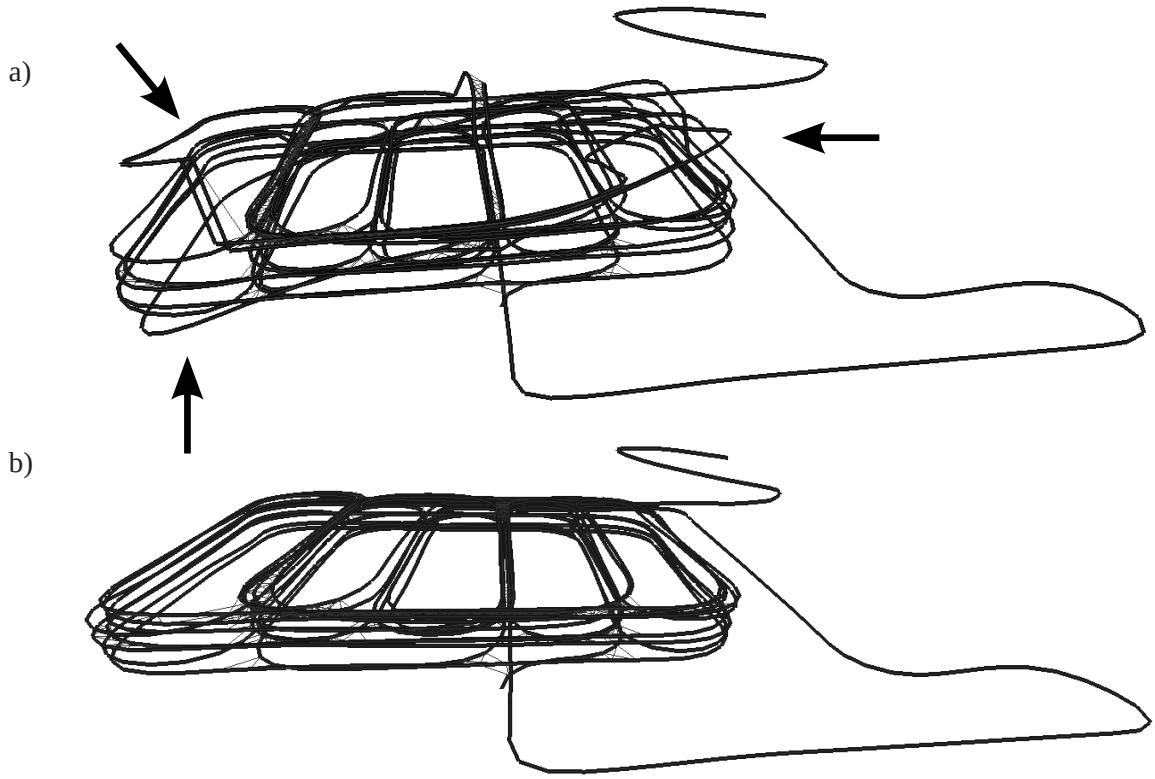

Figure 4.8: Graph obtained from a car driving multiple times through a parking lot covering three floors. Different error distributing techniques for the rotational error result in different optimized networks. The inconsistencies in the optimized graphs are marked by the arrows. Optimizing the network and distributing the rotational error given our previous work [58](a) and our approach described in this section (b). A detailed description of this experiment can be found in Section 4.7.

bottom image of Figure 4.8 shows the result of our approach as described in the previous section. In the following, we analyze the evolution of the rotational residual after distributing the error according to our approach.

A rotation can be described in manners of a rotational axis and the corresponding rotation angle. Given a three dimensional rotation matrix, R, we will refer to the axis of rotation as axisOf(R) and to the angle as angleOf(R). According to Barrera *et al.* [15], spherical linear interpolation (slerp) returns a set of rotations along the same axis, i.e., given a weight $w \in [0, 1]$, we obtain

$$R' = \text{slerp}(R, w) \tag{4.69}$$
$$\text{axisOf}(R') = \text{axisOf}(R) \tag{4.70}$$
$$\text{angleOf}(R') = w \cdot \text{angleOf}(R). \tag{4.71}$$

When distributing the rotation Q over a sequence of poses (see Equation (4.53)), we decompose it into a sequence of incremental rotations $Q = Q_{\Delta_1} Q_{\Delta_2} \cdots Q_{\Delta_n}$. Given the definition of $Q_{\Delta k}$

(see Equation (4.55)), we obtain

$$\begin{aligned}
\alpha_k &= \text{angleOf}(Q_{\Delta k}) && (4.72)\\
&\overset{\text{Eq. (4.55)}}{=} \text{angleOf}\left([\text{slerp}(Q, w_{k-1})]^T \,\text{slerp}(Q, w_k)\right) && (4.73)\\
&= \text{angleOf}\left(\text{slerp}(Q, w_k)\right) - \text{angleOf}\left(\text{slerp}(Q, w_{k-1})\right) && (4.74)\\
&\overset{\text{Eq. (4.71)}}{=} (w_k - w_{k-1}) \cdot \text{angleOf}(Q). && (4.75)
\end{aligned}$$

In the following, we show that when distributing the rotational error along a loop, the *angle* of the residual angleOf($r_{k-1,k}$) between two successive poses $k-1$ and k does not increase more than $|\alpha_k|$. According to Equation (4.49), the residual of a constraint $\langle k-1, k\rangle$ between the nodes $k-1$ and k is

$$r_{k-1,k} = X_k^{-1}\Delta_{k-1,k}. \quad (4.76)$$

Since the rotational part is important we focus on the rotational component of the residual only and ignore the translational part. Thus, we have

$$\begin{aligned}
\text{rot}(r_{k-1,k}) &= R_{\Delta_k}^T \Delta_{k-1,k}, \text{ which leads to} && (4.77)\\
R_{\Delta_k}^T &= \text{rot}(r_{k-1,k})\Delta_{k-1,k}^T. && (4.78)
\end{aligned}$$

After updating the rotations $R_{\Delta_1}, \ldots, R_{\Delta_n}$ by a constraint $\langle i, j\rangle$, with $k-1, k \in \mathcal{P}_{i:j}$, we obtain a new set of rotations in the global reference frame, namely $R_{\Delta_1}, \ldots, R_{\Delta_n}$ as shown in Equation (4.61). From these, we can recover the incremental rotation, R'_{Δ_k}, by using Equation (4.62):

$$\begin{aligned}
R'_{\Delta k} &\overset{\text{Eq. (4.62)}}{=} R_{k-1}'^T R'_k && (4.79)\\
&\overset{\text{Eq. (4.61)}}{=} \left[Q_{\Delta_1:\Delta_{k-1}} R_{k-1}\right]^T \left[Q_{\Delta_1:\Delta_k} R_k\right] && (4.80)\\
&= R_{k-1}^T Q_{\Delta_k} R_k && (4.81)\\
&= R_{k-1}^T Q_{\Delta_k} R_{k-1} R_{\Delta_k} && (4.82)
\end{aligned}$$

Combining this result with Equation (4.77), we can compute the new residual $r'_{k-1,k}$ after distributing the rotational error as

$$\begin{aligned}
\text{rot}(r'_{k-1,k}) &\overset{\text{Eq. (4.77)}}{=} R_{\Delta_k}'^T \Delta_{k-1,k} && (4.83)\\
&\overset{\text{Eq. (4.82)}}{=} \left(R_{k-1}^T Q_{\Delta_k} R_{k-1} R_{\Delta_k}\right)^T \Delta_{k-1,k} && (4.84)\\
&= R_{\Delta_k}^T R_{k-1}^T Q_{\Delta_k}^T R_{k-1} \Delta_{k-1,k} && (4.85)\\
&= \underbrace{R_{\Delta_k}^T \Delta_{k-1,k}}_{\text{rot}(r_{k-1,k})} \underbrace{\Delta_{k-1,k}^T R_{k-1}^T}_{=:Y^T} Q_{\Delta_k}^T \underbrace{R_{k-1}\Delta_{k-1,k}}_{=Y} && (4.86)\\
&= \text{rot}(r_{k-1,k}) Y^T Q_{\Delta_k}^T Y. && (4.87)
\end{aligned}$$

In the equation above (Eq. (4.87)), the term $Y^T Q_{\Delta_k}^T Y$ quantifies the increase in the rotational residual of the constraint $\langle k-1, k\rangle$ between the two consecutive nodes $k-1$ and k. Since both Y as well as Q_{Δ_k} are rotational matrices, and $Y^T Q_{\Delta_k}^T Y$ changes the rotational axis but not the angle, we finally see that the change in the residual is at most α_k and therefore limited, since

$$\begin{aligned}
|\text{angleOf}(Y^T Q_{\Delta_k}^T Y)| &= |\text{angleOf}(Q_{\Delta_k})| && (4.88)\\
&= |\alpha_k|. && (4.89)
\end{aligned}$$

To this end, we have described our algorithm for graph optimization in the 2D and 3D case and have shown that the change in the angular part of the residual is bounded when updating a constraint. However, the complexity of our approach grows with the number of nodes in the network. In other words, the complexity of our algorithm grows with the time the robot spends in the environment rather than the space the robot explored, which is critical for life-long map learning. This problem can be solved by merging nearby nodes to a single one which is explained in the next section.

4.5 Node Reduction

Due to the nature of the optimization technique, the complexity of our approach (per iteration) depends linearly on the number of constraints since each constraint is selected exactly once per iteration. For each constraint $\langle i, j, \rangle$, we need to modify exactly those nodes which belong to the path $\mathcal{P}_{i:j}$. However, the path of a constraint is defined given our tree parametrization. As a result, different constraints will have different path lengths and therefore a different complexity. Thus, we consider the average path length l to specify the overall complexity. It reflects the average number of operations needed to update a single constraint during one iteration. Given m constraints and an average path length l, this results in a complexity of $\mathcal{O}(m \cdot l)$. In our experiments we found that l is typically in the order of $\log n$, where n is the number of nodes in the network. However, the complexity of the current approach grows with the length of the trajectory of the robot rather than with the size of the environment the robot explored. These two quantities (i.e., length of trajectory versus time) grow different when the robot revisits already known areas and this effect is important in the context of life-long map learning, where the robot is bounded to a specific environment and has to update its map over time. Since our parametrization is not dependent on the sequence of the poses, i.e., the trajectory of the robot, we have the possibility of a further optimization. Intuitively, whenever the robot revisits a known area, we do not need to add a new node into the graph but rather want to propagate the information to already existing nodes of the network. In other words, we assign the current pose of the robot to an already existing node in the graph and update the constraints with respect to that node. Indeed, we can now even avoid adding new constraints to the network in case a constraint between the corresponding nodes already exists.

Given the constraint $\langle i, j \rangle^{(1)} = \langle \delta_{ij}^{(1)}, \Omega_{ij}^{(1)} \rangle$, already present in the network and a new constraint $\langle i, j \rangle^{(2)} = \langle \delta_{ij}^{(2)}, \Omega_{ij}^{(2)} \rangle$ between the same nodes i and j we can merge both into a new one, $\langle i, j \rangle$, made of δ_{ij}, Ω_{ij} with

$$\Omega_{ij} = \Omega_{ij}^{(1)} + \Omega_{ij}^{(2)} \tag{4.90}$$

$$\delta_{ij} = \Omega_{ij}^{-1} \left(\Omega_{ij}^{(1)} \delta_{ij}^{(1)} + \Omega_{ij}^{(2)} \delta_{ij}^{(2)} \right). \tag{4.91}$$

Note that this can be seen as an approximation similar to adding a rigid constraint between the already existing node in the network and a new one representing the current pose. This is especially useful if local maps (e.g. grid maps) are used as nodes since the robot can localize in the existing map quite accurately.

Using this technique, the size of the problem does not increase when the robot is revisiting already known areas but rather increases with the explored environment. Although the complexity stays the same, the number of nodes and edges (constraints) in the graph is reduced. As our experiments in the next section will demonstrate, this technique for node reduction leads to a faster convergence since less nodes and constraints need to be considered.

4.6 2D Experiments

This section is designed to evaluate the properties of our approaches described in Section 4.1 and Section 4.5. We first present the results of simulated experiments based on large 2D data sets and compare our approach to PPO and to Frese's multi-level relaxation [56] (MLR). Finally, we demonstrate that our method is also well suited to cope with the sensor and motion noise from different sources, including wheeled and aerial robots as well as humans carrying sensors around.

4.6.1 Simulated Experiments on Large Data Sets

This set of experiments is designed to measure the performance of our approach quantitatively. We compare our technique to PPO and Frese's *et al.* multi-level relaxation [56] (MLR). In these experiments we used two variants of our approach. First, we used our incremental tree parametrization while keeping all nodes in the network. Second, we used our node reduction technique as described in Section 4.5. In the following experiments we moved a virtual robot on a grid world. An observation is generated each time the current position of the robot was close to a previously visited one. We corrupted both nodes and edges, by zero mean Gaussian noise with different parameters and simulated datasets resulting in graphs with a number of constraints between around $4,000$ and 2 million. Figure 4.9 depicts intermediate graphs obtained by PPO and our approach at different iterations. Here, the network consists of $10,000$ nodes and $64,252$ constraints and the standard deviation of the Gaussian noise was set to 0.05 in both x, and y direction and 0.02 in the angular term. Although both approaches converge asymptotically to the same solution, our approach converges faster as can be seen in Figure 4.9. In all our experiments, the results of Frese's MLR strongly depend on the initial configuration of the nodes. Depending on the quality of the initial guess MLR converges to an accurate solution similar to our approach as shown in Figure 4.10 (left). Otherwise, it is likely to diverge (see Figure 4.10 (right)). PPO, as well as our technique are more robust and less independent of the initial poses of the nodes.

To evaluate our technique quantitatively, we first measured the average error (i.e., χ^2 error per constraint) in the network after each iteration. The left image of Figure 4.11 depicts a statistical experiment over 10 networks having the same topology but different noise realizations. As can be seen, our approach converges significantly faster than PPO. For small and medium size networks, both approaches converge asymptotically to approximately the same error value (see Figure 4.12). For large networks, however, the high number of iterations needed for PPO prevented us from demonstrating this convergence experimentally. Note that we omitted comparisons to EKF and Gauss Seidel relaxation because Olson *et al.* [114] already showed that PPO outperforms such techniques. Additionally, we evaluated the average computation time per iteration of the different approaches. These values are shown in Figure 4.11 (right). As a result of personal communication with Olson, we furthermore analyzed a variant of PPO which is restricted to spherical covariances. It yields similar execution time *per iteration* as our approach. However, this restricted variant has still the same convergence speed with respect to the number of iterations as Olson's unrestricted PPO technique. Note that in this experiments, our node reduction technique can speed up the computation up to a factor of 20.

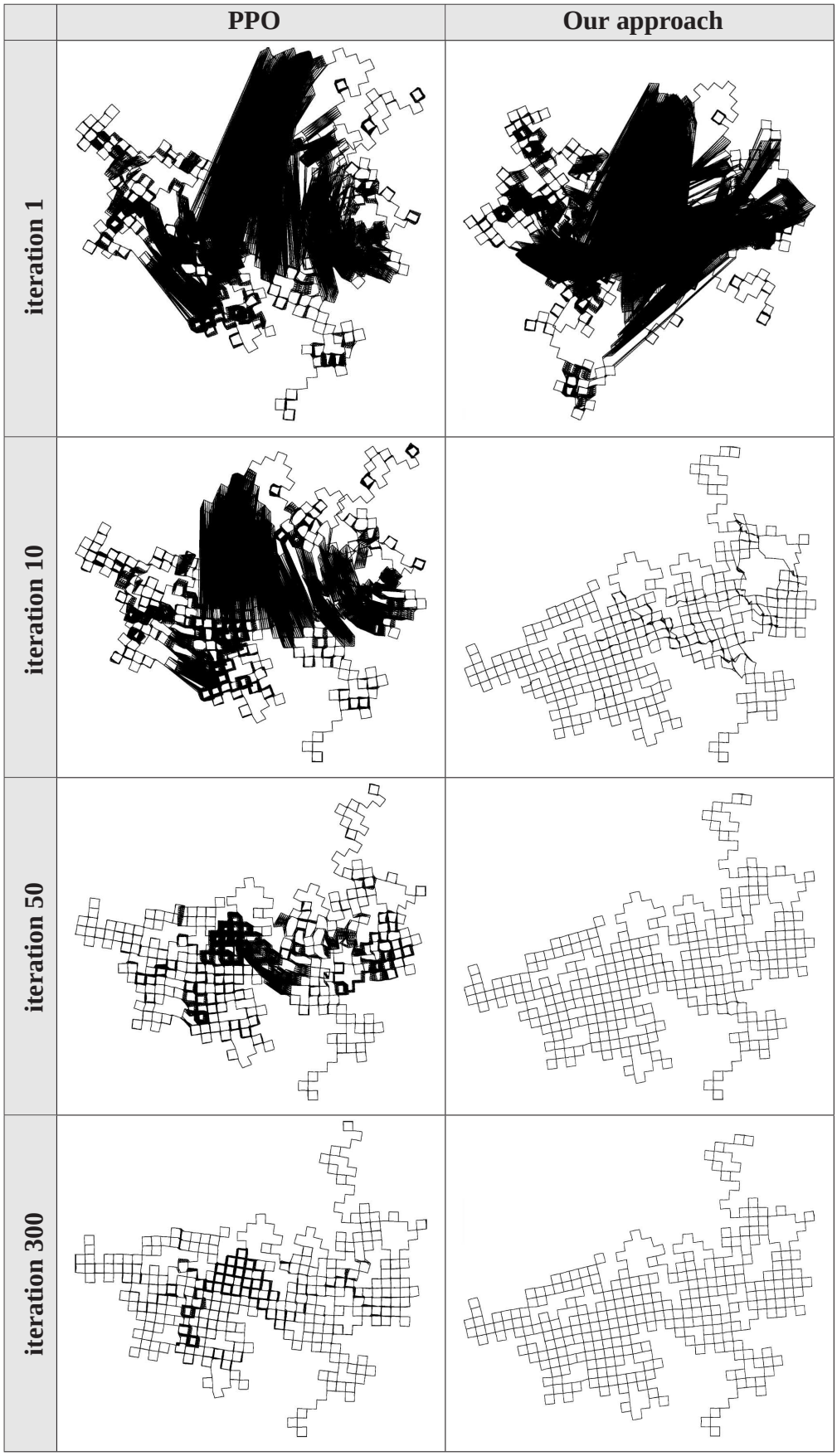

Figure 4.9: Result of PPO (left column) and our approach (right column) after $1, 10, 50$, and 300 iterations for a network consisting of $10,000$ nodes and approximately $64,000$ constraints. The black areas in the images originate from constraints between nodes which are not adequately corrected after the corresponding iteration.

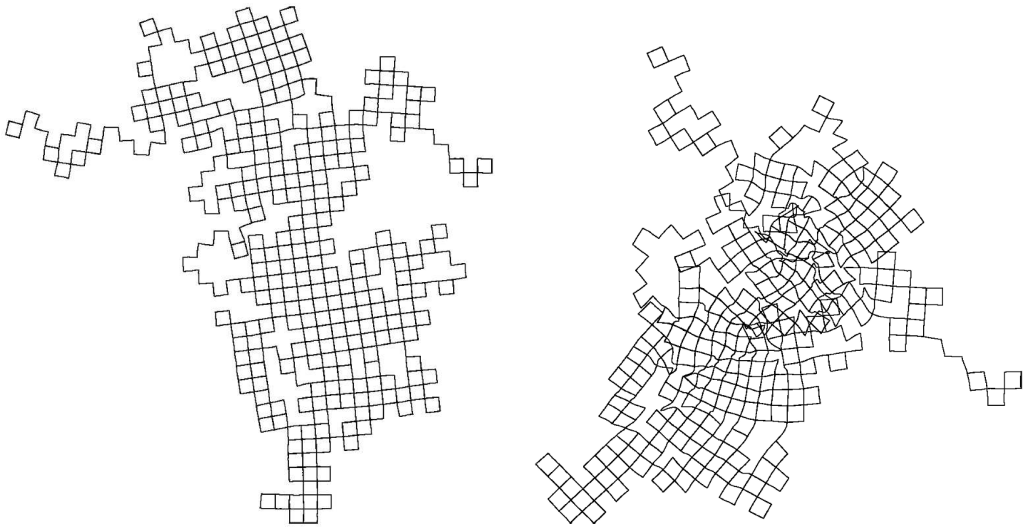

Figure 4.10: The outcome of Frese's multi-level relaxation is highly dependent on the initial configuration of the nodes. Left: small initial pose error, right: large initial pose error.

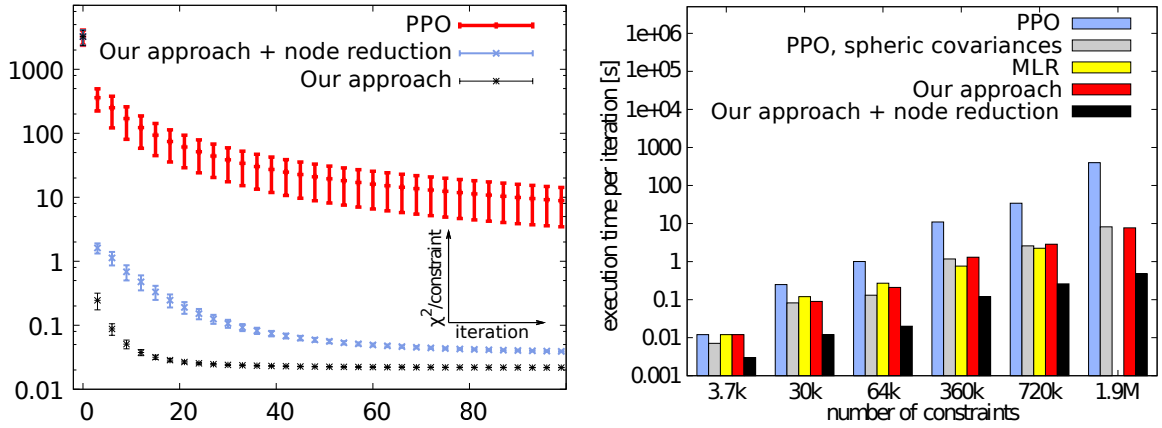

Figure 4.11: The left image shows the error of our approach and the error of PPO in a statistical experiment. The error bars have a width of 2σ. The right image depicts the average execution time *per iteration* for networks of different size. For the graph containing 1.9 million constraints, MLR required memory swapping and the result is therefore omitted.

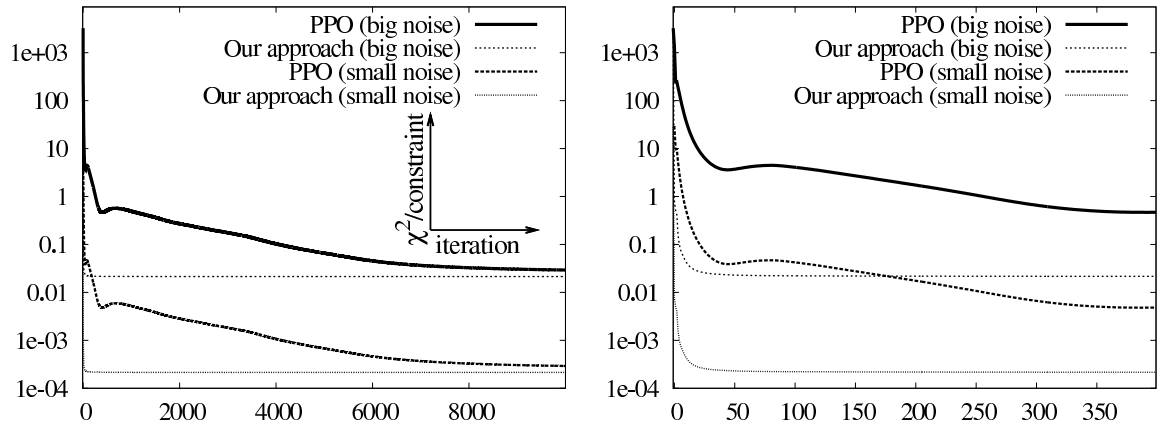

Figure 4.12: The left image shows that both techniques converge asymptotically to the same error. However, our approach converges significantly faster, as can be also seen in the right image, which is a scaling of the left one for the first 400 iterations.

Figure 4.13: Two maps of the Intel Research Lab in Seattle. The left map is constructed by using the raw odometry data only. The map shown in the right imaged is the result obtained by using our algorithm. Here, the graph consists of approximately $1,000$ nodes and about $1,800$ constraints. The execution time needed to converge was less than 1 second.

4.6.2 Real World Experiments

The experiments presented in this section are designed to illustrate that our approach can be used to build accurate maps from real 2D robot data.

In the first experiment, the robot collected data using its laser range scanner and the goal was to build an accurate map, given this data. The nodes in our graph correspond to the individual poses of the robot during data acquisition. The constraints are obtained in two ways. The first set of constraints between successor nodes is obtained from raw odometry. The second set of constraints is obtained by pair-wise matching of laser range scans. The latter also allows us to recognize previously visited locations (i.e., detect loop closures). Figure 4.13 depicts two maps of the Intel Research Lab in Seattle. The left one is constructed from raw odometry whereas the right one is the result obtained by our algorithm. As can be seen, the corrected map (Figure 4.13 (right)) shows no inconsistencies like double corridors. The network belonging to this data set contains about $1,000$ nodes and approximately $1,800$ constraints. Our approach needed less than 1 second to converge to the solution shown in Figure 4.13 (right) on a $2\,$GHz standard laptop computer.

The second experiments shows the result using the Bivosa data set from the Rawseeds [19] project. The graph was constructed using the technique described in the previous experiment. Compared to the previous data set, however, the raw odometry is already quite good. The map obtained from raw odometry is shown in Figure 4.14 (left) and the corrected map using our approach is shown in the right image of Figure 4.14. Here, the network consists of approximately $7,100$ poses and $8,100$ constraints, and our approach took less than four seconds to converge to shown solution.

Finally, we show the result of our approach using the CSAIL data set recorded at the MIT. The graph consists of approximately $1,800$ nodes and about $2,000$ constraints. Again, nodes and edges were constructed as described above. Figure 4.14 shows the map obtained from raw odometry (left) and after optimization using our approach (right). As in the first experiment, our approach took less than 1 second to converge to the solution shown in Figure 4.14 (right).

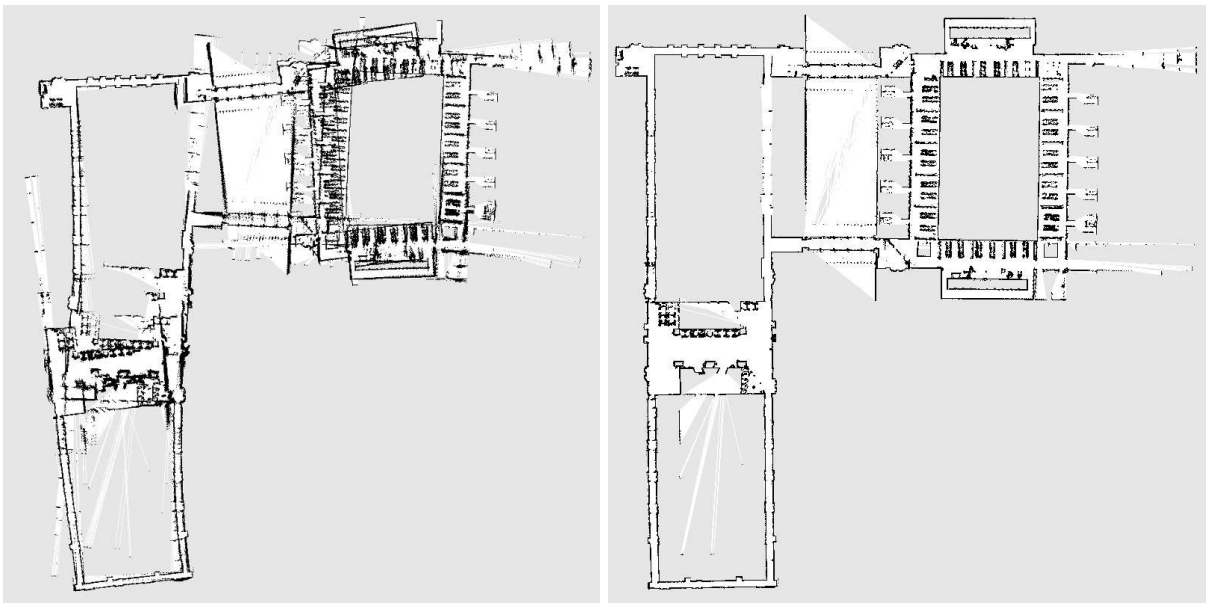

Figure 4.14: The Bovisa data set from the Rawseeds project. Left: map obtained from network without optimization. The right image shows the result after optimizing the graph for 100 iterations. Our approach needed less than four seconds to converge on the graph consisting of approximately $7,100$ nodes and $8,100$ constraints.

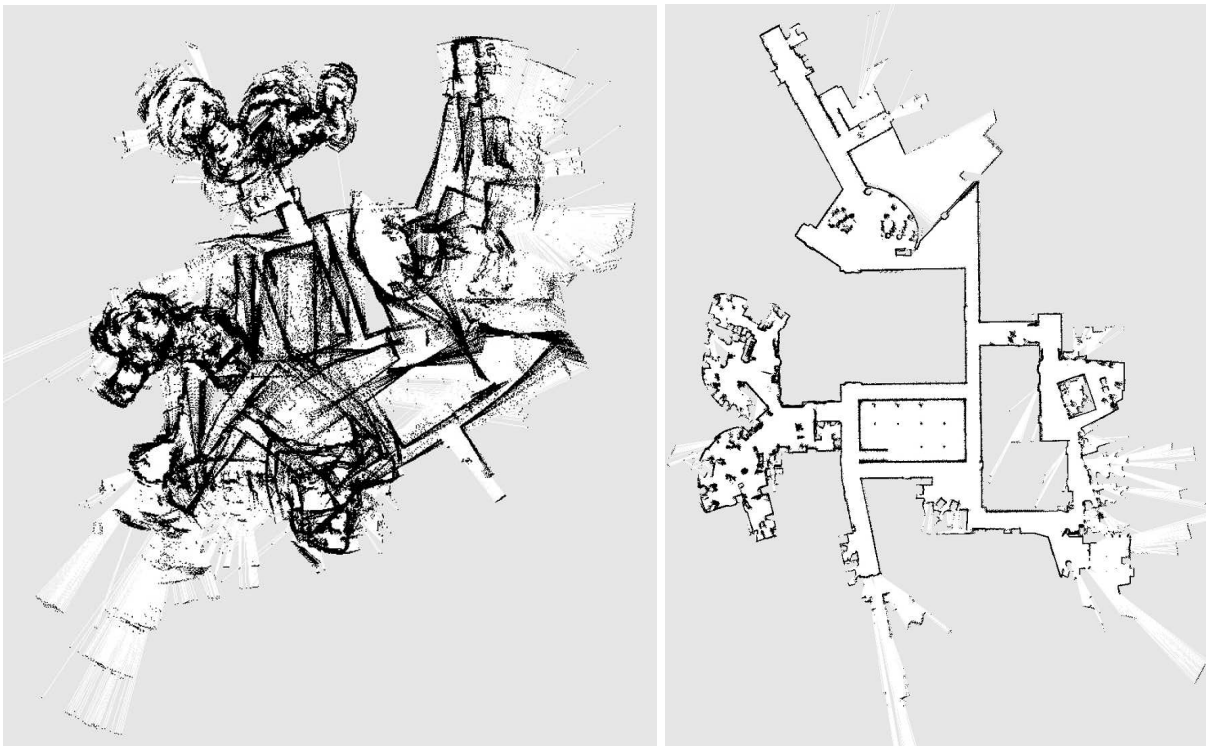

Figure 4.15: The CSail data set. The map obtained from raw odometry is shown on the left. The right image shows the result of our approach after 100 iterations. To converge to this solution, our approach needed less than 1 second. Here, the graph consists of approximately $1,800$ nodes and roughly $2,000$ constraints.

4.7 3D Experiments

This section is designed to evaluate the properties of our three dimensional optimization approach as presented in Section 4.3. We first present the results of simulated experiments on large 3D data sets and compare our approach to smoothing and mapping (SAM) [39, 88]. Subsequently, we present our results on partially real robot data by using real data for different floors of several simulated buildings. Finally, we present our results obtained using data recorded with a cars driving around a campus as well as a car driving multiple times through a parking lot with three floors.

4.7.1 Simulated Experiments on Large Data Sets

The following set of experiments is designed to demonstrate the robustness and usability for large 3D constraint networks using the approach proposed in Section 4.3. In these simulated experiments we moved a robot on the surface of a sphere and a box. An observation was generated each time a previous location was in the close vicinity of the current pose. All observations were corrupted with a variable amount of zero-mean Gaussian noise σ in each translational component (in m) and rotational component (in radians) for both nodes and edges.

The first experiment simulated a robot path along the surface of a sphere and is made of $2,200$ robot poses (nodes) and $8,647$ observations (constraints). Here, our approach took around 200 ms per iteration on a 2 GHz standard laptop computer. Figure 4.16 depict snapshots at iteration 0 (initial configuration), $10, 50$, and 300 for the noise parameters $\sigma = 0.05$, $\sigma = 0.1$, and $\sigma = 0.2$. As can be seen, our approach yield consistent results even in the presence of high observation noise and that even in the case of high observation noise, our approach converged in less than 300 iterations, which took around one minute.

In a subsequent experiment we simulated a path of a robot along the surface of a box. In contrast to the sphere experiment, less smooth constraints are available due to sharp maneuvers of the robot imposing an increased risk of an angle wrap-around (error distribution in wrong direction). The box data set consists of $2,401$ robot poses and 4762 observations. For optimizing this data set our approach took around 165 ms per iteration with a total of 50 s for 300 iterations. Intermediate snapshots as well as the initial configuration for different noise parameters are shown in Figure 4.17.

The corresponding error curves (i.e., average χ^2 error per constraint) of both data sets for the different noise levels are plotted in Figure 4.18. Here, the errors at iteration $10, 50$, and 300 corresponding to the snapshots in Figure 4.16 and Figure 4.17 are highlighted respectively.

We furthermore compared our approach to the smoothing and mapping (SAM) approach of Dellaert [39] using the sphere datasets. The SAM algorithm can operate in two modes. Either as a batch process which optimizes the entire network at once (like our approach) or in an incremental mode. The latter one only performs an optimization after a fixed number of nodes has been added. Note that this way of incrementally optimizing the network is more robust since the initial guess for the network configuration is computed based on the result of the previous optimization procedure. This has the effect that the risk of getting stuck in a local minima is typically reduced. However, this comes with a significant computational overhead. The comparison between our approach and SAM is summarized in Table 4.2. Observe, that the batch variant of SAM got stuck in a local minima for the sphere datasets with medium and large noise levels. In contrast to that, the incremental version always converged to a solution comparable to ours but still required substantially more computation time than our approach.

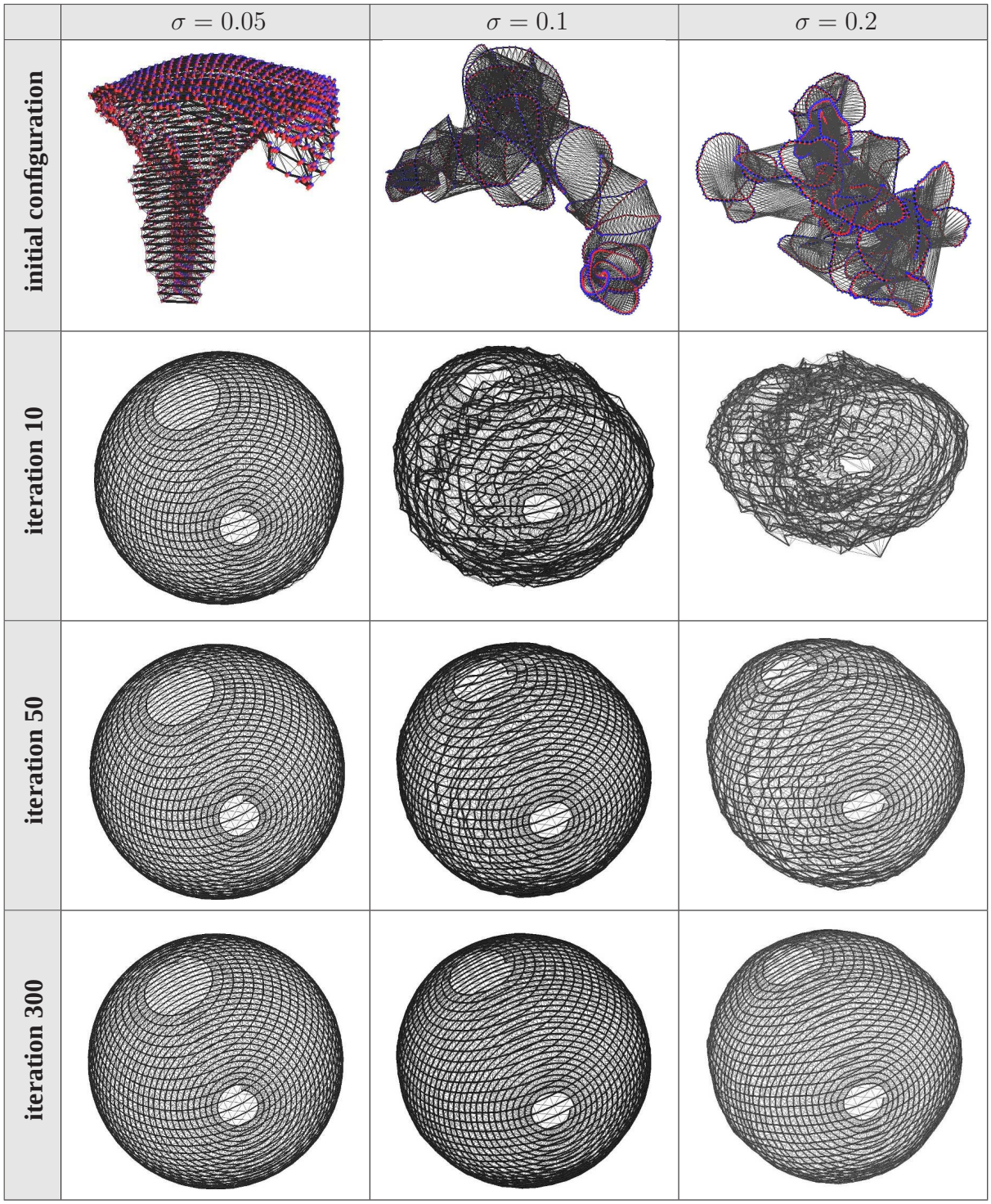

Figure 4.16: Snapshots for the initial configuration (iteration 0) and iterations $10, 50$, and 300 for the sphere data set. The three columns shows the results of our optimization approach given an observation noise of $\sigma = 0.05$ (left), $\sigma = 0.1$ (middle), and $\sigma = 0.2$ (right) in both translation (in m) and rotation (in radians).

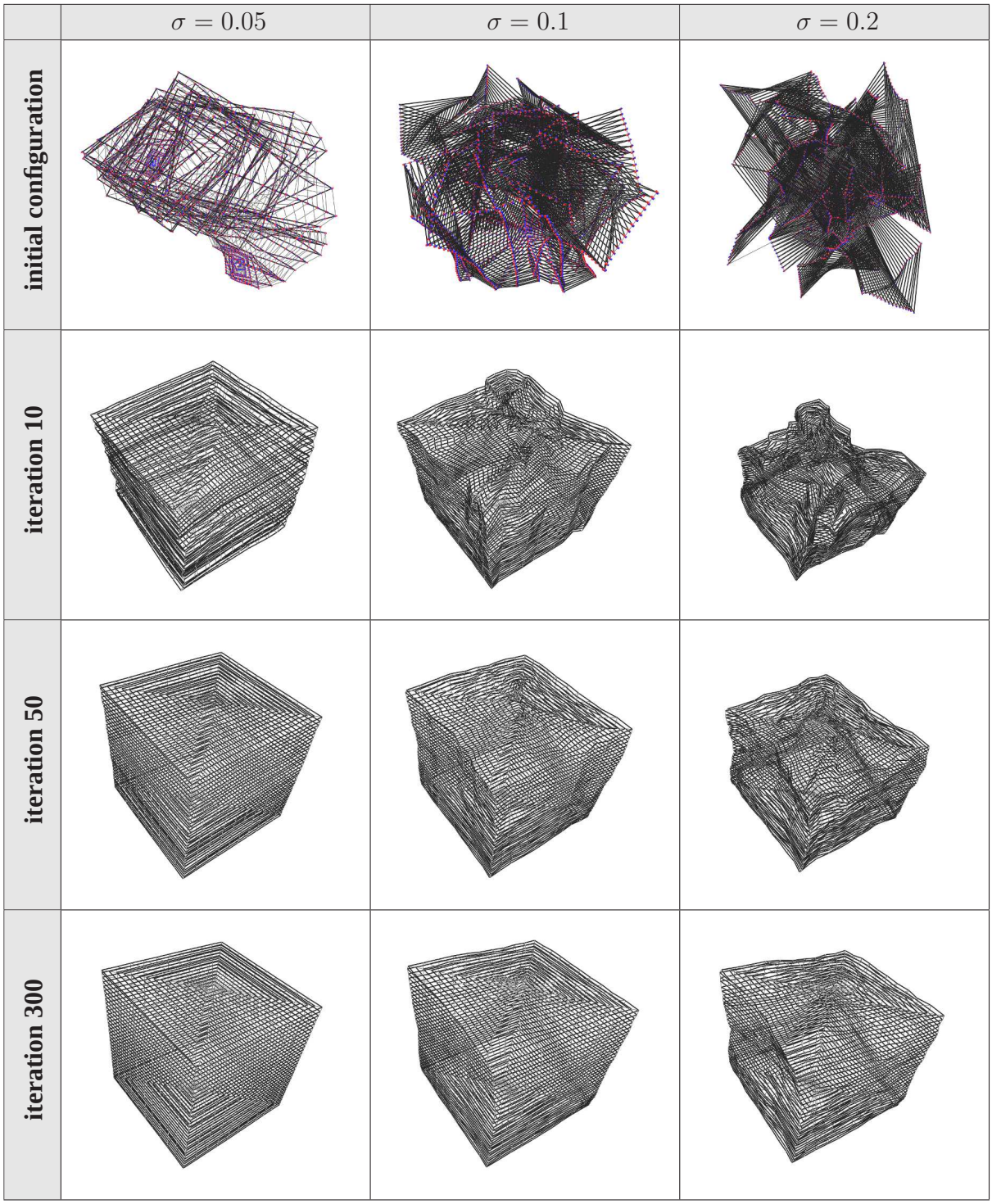

Figure 4.17: Snapshots for the initial configuration (iteration 0) and iterations $10, 50$, and 300 for the box data set. The three columns shows the results of our optimization approach given an observation noise of $\sigma = 0.05$ (left), $\sigma = 0.1$ (middle), and $\sigma = 0.2$ (right) in both translation (in m) and rotation (in radians).

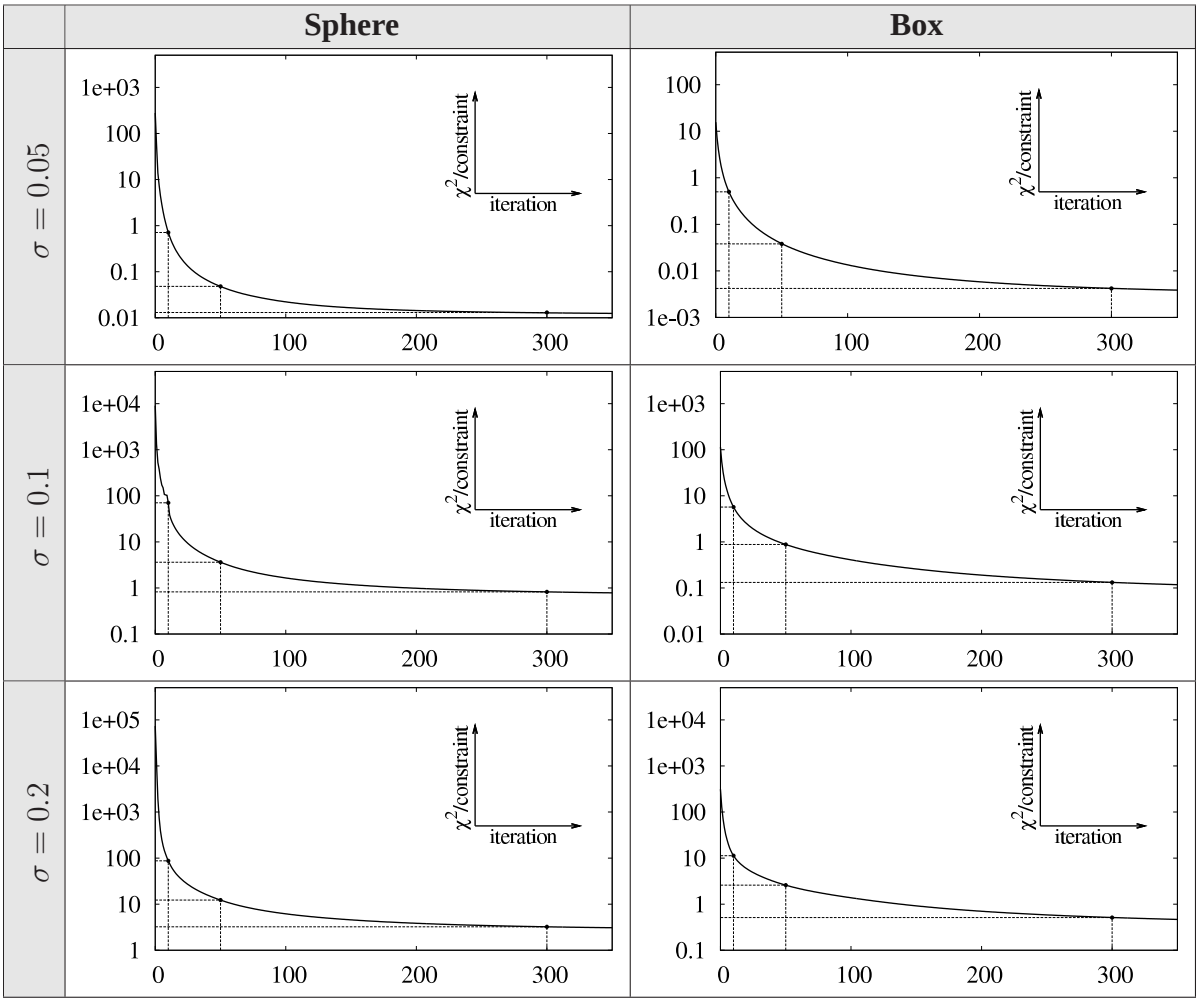

Figure 4.18: Evolution of the χ^2 error per constraint for the different data sets (columns) for different observation noise parameters (rows). The highlighted points mark the errors at iteration $10, 50$, and 300 for the corresponding snapshots in Figure 4.16 and Figure 4.17.

Noise level	SAM (batch)	SAM (incremental)	Our approach (batch)
$\sigma = 0.05$	119 s	not tested (see batch)	30 s (150 iterations)
$\sigma = 0.1$	diverged	270 s (optimized each 100 nodes)	50 s (250 iterations)
$\sigma = 0.2$	diverged	510 s (optimized each 50 nodes)	50 s (250 iterations)

Table 4.2: Comparison to SAM for the sphere datasets with different noise realizations.

4.7.2 Real World Experiments

The first experiment is obtained by extending data from a robot equipped with a 2D laser range finder into three dimensions. We used the 2D real world dataset of the Intel Research Lab (see previous section) and constructed virtual buildings with multiple floors. The constraints between buildings and floors are manually added but all other data originated from a real robot. The resulting dataset consists of roughly $15,000$ nodes and approximately $72,000$ constraints. We introduced a high error in the initial configuration of the poses in all dimensions which results in no visible structure in the initial configuration as can be seen in Figure 4.20(top). Applying our approach however yields an accurate result, as shown in the bottom of the same image. The image in the middle shows the resulting map after 10 iterations. The final result was obtained after 80 iterations and needed around 3 minutes calculation time.

In the second experiment we used data recorded with an instrumented car at the EPFL campus in Lausanne. Using cars as robots became popular in the robotics community [27, 118, 152, 158]. Here, the Smart car was equipped with 5 SICK laser scanners, an inertial measurement unit (IMU), cameras, and various other pose estimation sensors. The robot constructs local three dimensional maps, so-called multi-level surface maps [156] and builds a network of constraints where each node represents such a local map. The constraints between nodes were generated by odometry (i.e., GPS and IMU data) and laser scan matching in the case of recognizing previously visited locations. The dataset contains a trajectory which is approximately 10 km long and contains several loops. Furthermore, it includes multiple levels such as an underground parking garage and a bridge with an underpass. The corresponding constraint network given this data is shown in Figure 4.19 (top left) and the result obtained using our algorithm is shown in Figure 4.19 (top right). The corrected network is plotted on an aerial image of the same area. Note that the corrected network visually fits accurately into the aerial map of the environment. We also used this dataset to compare our algorithm to the approach of Triebel *et al.* [156] that iteratively applies LU decomposition on dense matrices, i.e., no sparsification is applied which causes a substantial increased runtime. Here, both approaches converge to more or less the same solution. The time needed to achieve this correction, however, is several orders of magnitude smaller when using our approach. This is visualized in Figure 4.19 (bottom), where the error per constraint is plotted versus the execution time. Note that the bottom right image of this figure shows a magnified view for the first 400 ms.

We also present an experiment using an autonomous car driving multiple times through a parking lot in Stanford recorded by Kümmerle *et al.* [95]. The trajectory covers three different levels, from the bottom up to the roof. The car is equipped with several laser scanners, GPS, cameras and an inertial navigation system for pose estimation. The corrected trajectory obtained as a result of our approach is shown in Figure 4.8(b) on page 55. Figure 4.21 illustrates a multi-level surface map created from the corrected constraint network as well as an aerial image of the parking lot for comparison.

Up to now, all data sets were obtained using laser range scanners. In the following, we present a set of experiments in order to emphasize that our approach is a general framework for robotic mapping and can be used with a variety of platforms equipped with different sensors. In the next experiment, a blimp was equipped with a down looking camera, a down looking sonar and an IMU. Here, the limited payload of this highly embedded system prevented us from mounting a second camera or another additional sensors. The aerial vehicle and typical images obtained from the analog video link are shown in Figure 4.22 (top). The network obtained given the raw data estimated by matching visual features (SURF)[16] in combination with a variant of PROSAC[34] is shown in Figure 4.22 (bottom left). The corresponding corrected network is shown in Figure 4.22 (bottom right). Black lines between different nodes in the

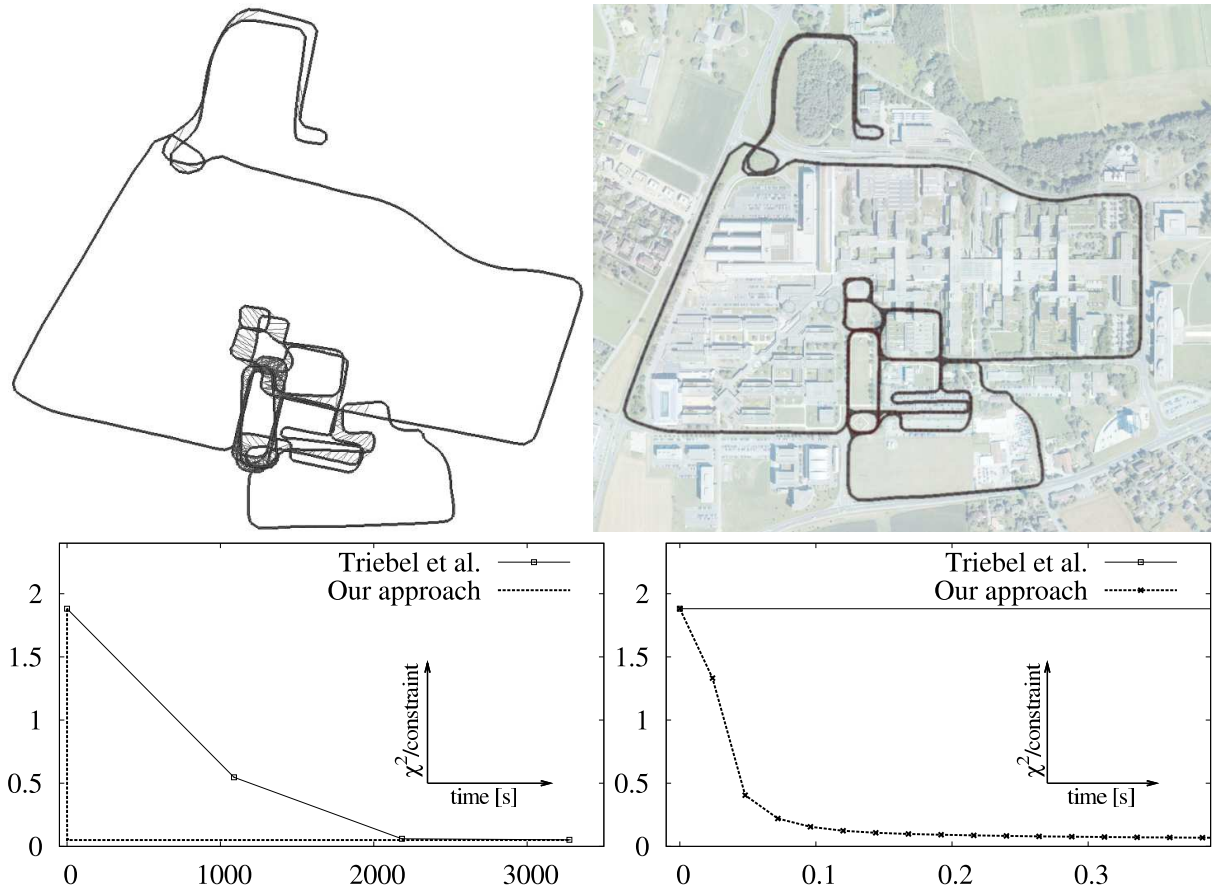

Figure 4.19: The constraint network corresponding to a dataset recorded with an instrumented car at the EPFL campus in Lausanne. Top left: Trajectory of the car before optimization. Top right: Corrected trajectory obtained using our algorithm. The corrected network is overlayed with an aerial image of the same area for better visibility. Bottom: The evolution of the average error per constraint versus the execution time for this data set using our approach and the approach of Triebel *et al.* . The bottom right image shows a magnified view of the first 400 ms.

graph correspond to constraints between those. As can be seen, our system can also be used within embedded systems and performs satisfactory well.

In a final experiment we equipped a human with two down-looking cameras and an IMU mounted on a stick (see Figure 4.23 (top)) and let him walk around the campus. As in the previous experiment, the transformations (and thus the constraints) between images were estimated using visual features (SURF). The raw trajectory obtained using this setup for an outdoor data set where the human walked around a building is shown in Figure 4.23 (middle left), and the corresponding corrected network using our approach is shown in the middle right image of the same Figure. The real trajectory has a length of approximately 190 m (estimated via Google Earth) and the corresponding network contains approximately $1,400$ nodes and $1,600$ constraints. Again, given the network, the convergence took less than 1 second. In each node, we also store the corresponding stereo image recorded by the cameras. Thus, we are able to reconstruct the map, given the corrected poses. A perspective view of the corrected network is shown in the bottom of Figure 4.23. The trajectory after applying our optimization algorithm is 208 m long. However, given the low cost stereo system used for acquiring the data has an uncertainty of around 10 cm at 1 m distance to the ground, the overestimation of approximately 9 % to the true length is within the bounds of a consistent map. More details about this work as well as additional experiments can be found in our paper about learning 3D maps using attitude and noisy vision sensors [142].

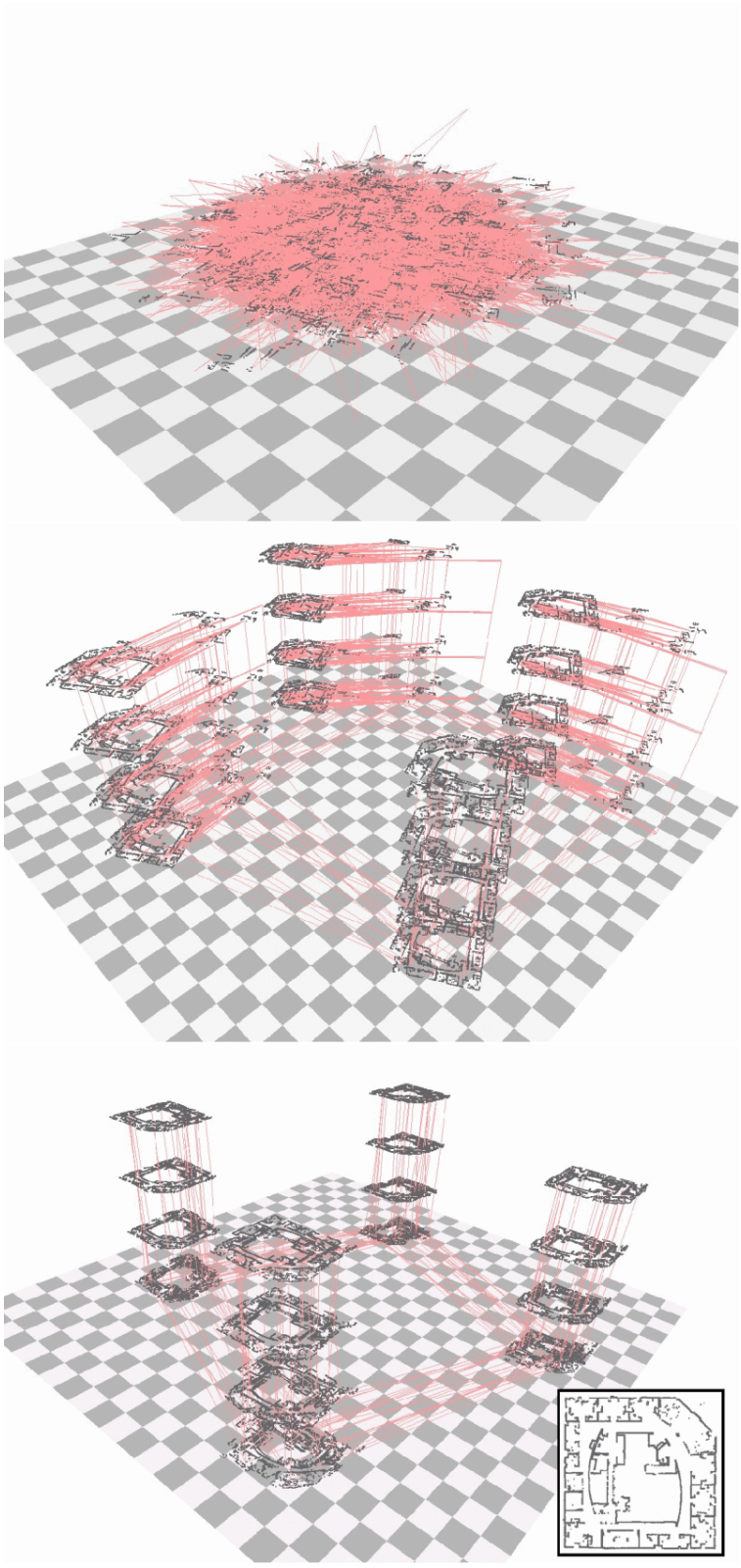

Figure 4.20: The real world dataset of the Intel Research Lab recorded in 2D is used to generate a large 3D dataset. Each of the four buildings consists of four identical floors. The top image depicts the initial configuration. The image in the middle depicts an intermediate results and the right one the corrected map after 80 iterations of our approach. Note that we also plotted constraints (gray / red) between individual floors and buildings but omitted to plot constraints from a single floor for better visibility. The corresponding lased data is shown in black. The small image in the bottom right corner shows the corrected map of the two dimensional laser range data.

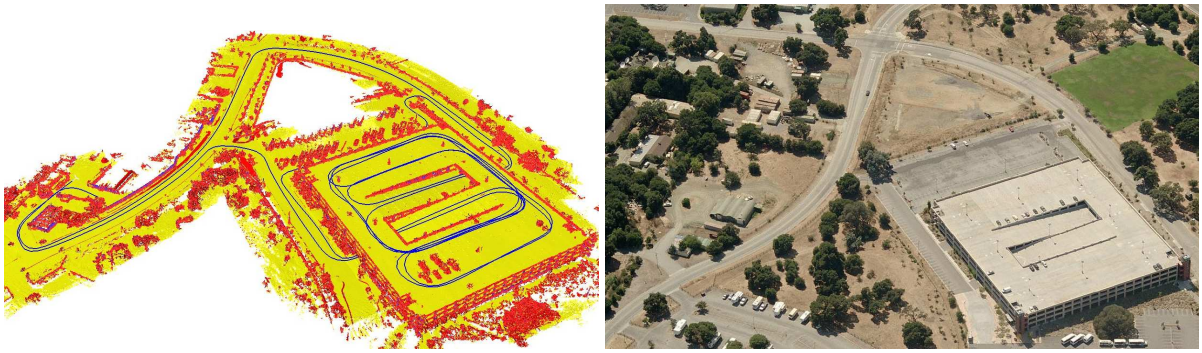

Figure 4.21: Corrected network from a car driving in a parking lot containing three floors multiple times (see also Figure 4.8). A multi-level surface map created from the corrected constraint network is shown in the left image. An aerial image of the same environment is shown in the right image. The images are a courtesy of Rainer Kümmerle.

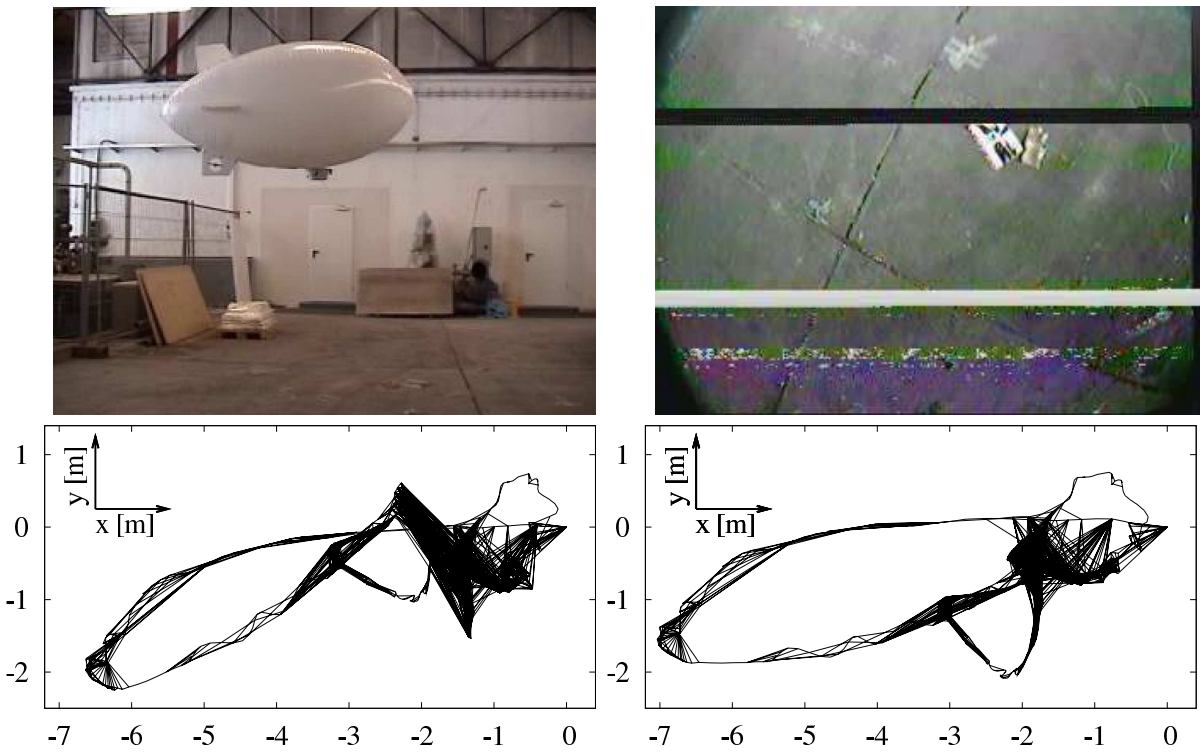

Figure 4.22: The blimp (top left) and an example image obtained through the analog video link (top right). Bottom left: Raw odometry estimated by tracking visual features (left). The small loops and the discontinuities in the trajectory result from the tracking of visual features in all frames given the limited sensor setup. The right image shows the trajectory obtained after applying our optimization algorithm using this embedded system.

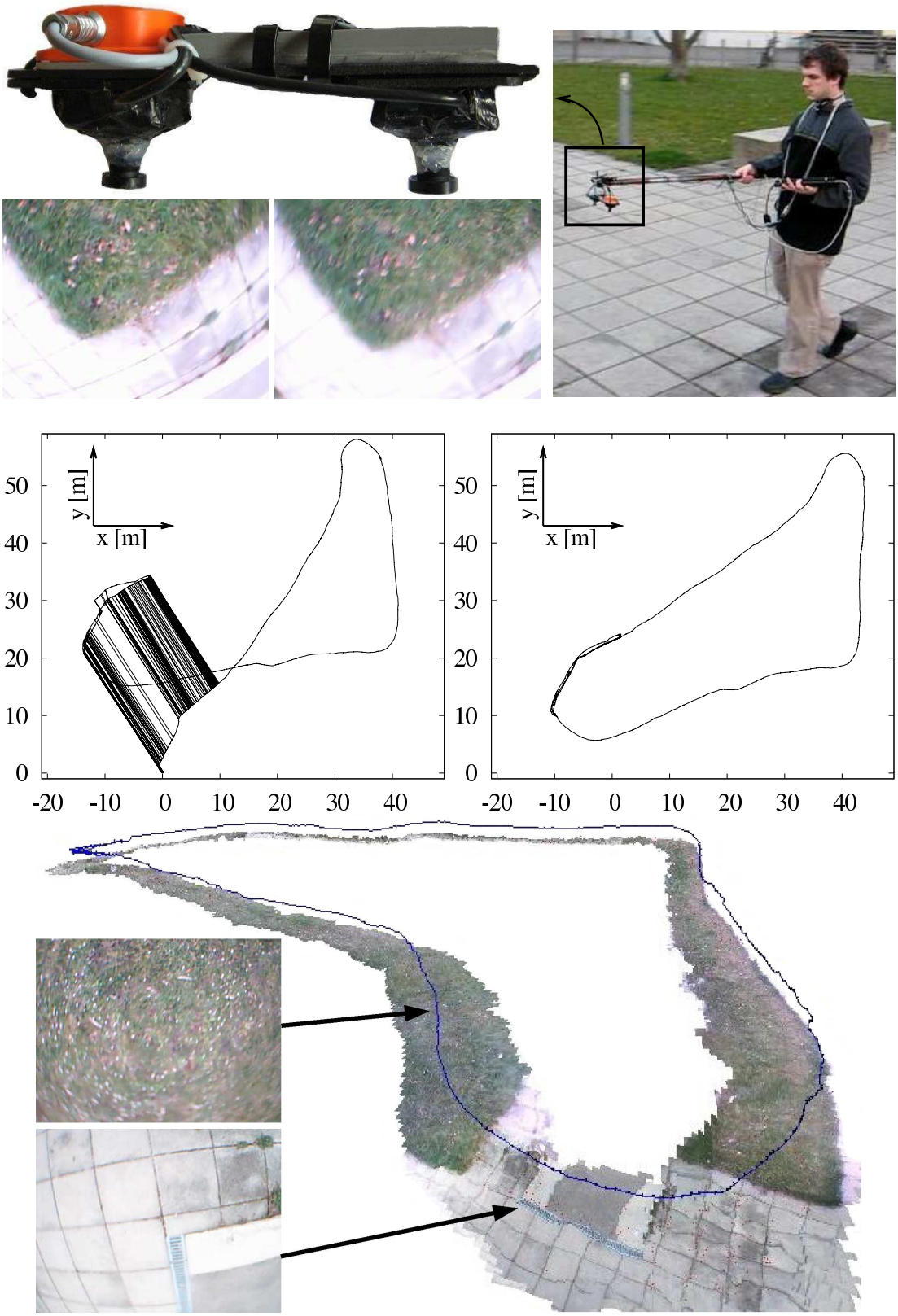

Figure 4.23: Top: the sensors used for testing our approach. We assembled two cheap USB web-cams as a stereo pair and combined it with a Xsens MTi IMU. We mounted the IMU together with the cameras looking downwards on a stick and walked around the campus. Middle: Raw odometry obtained by walking around a campus building (left) and the corrected trajectory obtained as a result of our optimization algorithm. Bottom: Perspective view of the textured elevation map of the outdoor experiment together with two camera images recorded at the corresponding locations.

4.8 Related Work

In the context of robotic simultaneous localization and mapping (SLAM) we can classify mapping techniques according to the underlying estimation approach. The most popular ones are extended Kalman filters (EKF) [97, 137], sparse extended information filters (SEIF) [45, 151], particle filters (PF) [106], and least squares error minimization techniques [100, 56, 74]. It has been shown, that for some applications it can be also sufficient to learn local maps of the environment only [78, 152, 166].

Using extended Kalman filters for robotic mapping definitely belongs to the oldest and most popular attempts [138, 137, 11, 107, 98]. Here, the constraints are modeled as linear functions and noise is assumed to be white, i.e., Gaussian. In this case, the effectiveness of these approaches originates from the fact that they estimate a full correlated posterior about robot poses and landmarks. They maintain a mean μ and a covariance Σ of the full posterior and access of this information is performed in constant time. However, incorporating a constraint is expensive ($\mathcal{O}(N^2)$, with N being the number of poses and landmark) and thus these approaches are not very effective except for small N [136].

In contrary to this the dual representation of the extended Kalman filter, namely the extended Information filter (EIF), maintains the information matrix $\Omega = \Sigma^{-1}$ and the information vector $\eta = \Sigma^{-1}\mu$. Although incorporating a constraint can be done in constant time, recovering the posterior (covariance matrix and mean) is now in $\mathcal{O}(N^2)$ [134]. Unfortunately, this information is constantly needed during the estimation process why in the current described from the complexity of EKF versus EIF is comparable. However, the information form of the filter bears a main advantage: the advantage of sparsity. The information matrix can be seen as the adjacency graph of the network. Here, an element is non-zero if an edge is present in the corresponding network [45]. Since the perception range of the robot is limited, the information matrix contains many zeroes. This is not the case in the corresponding covariance matrix. Even more, the sparsity allows for faster recovery of the posterior mean and covariance. Thrun *et al.* [151] propose to truncate small weights resulting from marginalization [53] in the information matrix to speed up computation resulting in the sparse extended information filter (SEIF). However, is bears the risk that the covariance estimate can become overly confident. This problem is addressed by Eustice *et al.* [45, 160]. Here, the error bounds within the SEIF framework are computed more accurately reducing the risk mentioned above.

Unfortunately, EKF based approaches as well as EIF based approaches approximate the constraints through a linear function and thus suffer from linearization errors. If these errors are too big, the approaches are likely to diverge [87, 157].

Particle filter based approaches approximate a distribution through a set of samples (particles). In the most popular approaches, particles represent different map realizations [103, 61, 140]. These differences originate from the uncertainty in the motion model. In other words, each particle represents a specific realization of the motion noise. Thus, these approaches do not suffer from linearization errors like the ones discussed so far and have been widely used in the past years. However, their computational complexity grows with the number of particles. To keep this number as small as possible, Fox *et al.* [50] compute the Kullback-Leibler distance between the true distribution and the one approximated through the particles. This allows them to adapt the number of particles in each step. However, convergence of this approaches to the optimal solution is only guaranteed, if the number of particles goes to infinity [149].

An alternative approach is to find maximum likelihood maps by least squares error minimization. The idea is to compute a network of relations (constraints) given a sequence of sensor readings. These constraints represent the spatial configuration between the robot's poses. In the

work presented in this chapter, we also follow this way to formulate the SLAM problem.

Lu and Milios [100] were the first ones applying least squares error minimization in context of robotic SLAM using a kind of brute force method. Their approach aims to optimize the whole network at once. Konolige *et al.* [92] use preconditioned conjugate gradient descent for minimizing the error in the constraints. An algorithm based on conjugate gradients was also proposed by Montemerlo and Thrun [104]. In their work, they utilize them to efficiently invert the sparse information matrix of the system. This allowed them to learn large maps using a Segway robot. Gutmann and Konolige [74] proposed an effective way for constructing a graph and for detecting loop closures while running an incremental estimation algorithm. Their general four steps, namely, incremental motion estimation, consistent pose estimation, map correlation, and optimization are still the basic ideas behind many SLAM algorithms.

Howard *et al.* [81] apply relaxation to localize the robot and build a map. In contrast to the technique described in this chapter, these approaches optimize in a step a single pose based on all connected constraints. This can be seen as dual to approaches like ours, where in each step all (relevant) poses are optimized with respect to a single constraint. Duckett *et al.* [42] propose the usage of Gauss-Seidel relaxation to minimize the error in the network of constraints. To make the problem linear, they assume knowledge about the orientation of the robot. Frese *et al.* [56] propose a variant of Gauss-Seidel relaxation called multi-level relaxation (MLR). It applies relaxation at different resolutions. MLR is reported to provide very good results in flat environments especially if the error in the initial guess is limited. Recently, Olson *et al.* [116, 114] presented a novel technique for 2D robotic mapping based on stochastic gradient descent (PPO).

Frese's Tree Map [54] assumes a topological tree structure of the graphical model and enforces this assumption by pruning edges and ignoring small entries in the information matrix. Thus, only an approximation to the true map is calculated. However, since the posterior model has a tree structure, Frese is able to perform an update in $\mathcal{O}(\log n)$, with n being the number of features [86]. This assumption is also made in the case of Paskin's Thin Junction Tree Filter (TJTF) [117]. As in the case of Frese's Tree Map algorithm, the topological tree structure reduces the complexity compared to EKF approaches since filtering using junction trees can be performed in linear time.

However, techniques such as PPO, Frese's multi level relaxation or our tree incremental network optimizer focus on calculating the most likely map and assume that the nodes and constraints are known. Since constraints reflect data associations we can get those using for example the ATLAS [21] framework, the approach of Nüchter *et al.* [113] or hierarchical SLAM [44]. Note that these methods also globally optimize the network but this step can be replaced by our optimization algorithm to make them more efficient. In case of visual SLAM, we can obtain such constraints by matching visual features as proposed by Steder *et al.* [142].

In the context of three-dimensional mapping, only a few techniques have been proposed so far [104, 88]. Howard *et al.* [82] propose to map 3D urban environments by optimizing in 2D only. Here, roll and pitch are assumed to be measured accurately enough using an IMU. Therefore, they avoid the problem of distributing the error in three dimensions but correct (x, y, z) and the yaw-orientation only. Nüchter *et al.* [113] describe a mobile robot which is able to build accurate 3D models. In their work, the error in a loop is uniformly distributed along the poses obtained from odometry. Although this technique provides good estimates it can not deal with multiple as well as nested loops.

Triebel *et al.* [156] described an approach for correcting the poses in 3D. In their approach, the problem is linearized in each iteration and solved using LU decomposition. This yields to accurate maps, both for small and medium size environments especially when the rotational error is small. However, as illustrated in the experimental section (see Section 4.7.2), our ap-

proach is orders of magnitude faster than this method and thus their method is not well suited to build maps for large networks.

Dellaert and colleagues proposed a smoothing method called square root smoothing an mapping [39, 41, 88, 125] (SAM). Here, the poses and landmark locations are smoothed and the information matrix is factorized by using Cholesky decomposition. This approach has several advantages compared to EKF-based solutions. First of all it better covers the non-linearities of the constraints. Second, it is faster to compute. In contrast to SEIFs, it furthermore provides an exactly sparse factorization of the information matrix. Note that the major speed-up of this method comes from a good ordering of the variables in the information matrix. Finally, SAM can be applied in an incremental way [88] and additionally is one of the few techniques which can be used in 2D as well as in 3D.

Grisetti *et al.* [59] described a general approach for optimization using Gauss-Newton on Manifolds together with sparse Cholesky factorization. They also propose a hierarchical variant of their approach able to minimize the error on different levels of abstraction [60]. In joint work, Konolige, Grisetti, and colleagues [93] described a special variant for 2D mapping. Recently, Kümmerle *et al.* [94] presented g^{2}o, a general framework for graph optimization which is also able to use bearing information typically obtained when using vision systems.

As mentioned earlier, the work closest to our approach is the work of Olson *et al.* [116] (see Section 3.5). They apply a variant of stochastic gradient descent with a novel parametrization of the configuration space. In contrast to their approach, we apply a different parametrization of the nodes that better reflects the topology of the environment. This, however, leads to a faster convergence of the algorithm. Furthermore, our approach is able to correct three-dimensional networks and our node reduction technique allows us to prevent adding new nodes into the network when revisiting an already known part of the environment. In return, the complexity of our approach grows with the size of the explored environment rather than the length of the robot's trajectory. In addition to the faster convergence this is also an advantage compared to other approaches such as PPO or MLR since it allows for life-long map learning.

4.9 Conclusion

We presented a highly efficient solution to the problem of learning maximum likelihood maps for mobile robots for both the two dimensional as well as the three dimensional case. Our technique is based on the graph-formulation of the simultaneous localization and mapping problem and applies a gradient descent based optimization scheme. Our approach extends the PPO algorithm by introducing a tree parametrization for the nodes in the graph for the two dimensional case. We furthermore presented an update rule for distributing the rotational error in 3D. The tree parametrization has a significant influence on the convergence speed and execution time of the method. It furthermore enables us to correct arbitrary graphs and not only a list of sequential poses. In this way, the complexity of our method depends on the size of the explored environment rather than on the length of the input trajectory. This indeed is an important precondition to allow a robot lifelong map learning in its environment.

Our method has been implemented and exhaustively tested in 2D and 3D using simulated as well as real robot data. We furthermore compared our method to other existing state-of-the-art solutions, namely PPO and multi-level-relaxation in the two dimensional case and SAM as well as the approach of Triebel *et al.* for the 3D datasets. In all cases, our approach converges significantly faster than the other approaches and leads to accurate maps with low errors. Note that we omitted to compare our approach to EKF and Gauss Seidel relaxation because Olson *et al.* [114] already showed that their approach outperforms such techniques.

Part II

State Estimation, Navigation, and Mapping for Embedded Sensor Systems

Chapter 5

Autonomous Indoor Flying using a Quadrotor Robot

We present a general navigation system that enables a small-sized quadrotor to autonomously operate in indoor environments. To achieve this, we systematically extend and adapt techniques that have been successfully applied on ground robots. We describe all developed methods and present an extensive set of experiments illustrating that they enable a quadrotor to reliably and autonomously navigate in indoor environments.

In the previous part of this work, we described an algorithm for efficient robotic mapping. We modeled the world as a graph where each node represents a pose of the robot. These nodes can also contain additional information like the current view made about the environment. Edges between nodes, also called constraints, represent the spatial relation between two robot poses. In the following two chapters we describe two embedded systems envisioned to assist human personnel. First, we describe our enabling technology for autonomous indoor flying using a quadrotor. Subsequently we describe our algorithm for indoor mapping based on human activity in Chapter 6. In both systems, we employ a graph as the basic data structure for mapping and utilize our tree network optimizer for estimating the most likely map and trajectory, given the sensor observations.

In recent years, the robotics community has shown an increasing interest in autonomous aerial vehicles, especially quadrotors. Low-cost and small-size flying platforms are becoming broadly available and some of these platforms are able to lift relatively high payloads and provide an increasingly broad set of basic functionalities [29, 112, 1]. This directly raises the question of how to equip them with autonomous navigation abilities. Whereas most of the proposed approaches for autonomous flying [150, 36] focus on systems for outdoor operation, vehicles that can autonomously operate in indoor environments are envisioned to be useful for a variety of applications including surveillance and search and rescue tasks [24]. Compared to ground vehicles, the main advantage of flying devices is their increased mobility.

As for ground vehicles, the main task for an autonomous flying robot consists in reaching a desired location in an unsupervised manner, i.e., without human interference. In the literature, this task is known as *navigation* or *guidance*. To address the general task of navigation, one is required to tackle a set of problems ranging from state estimation and world modeling to trajectory planning.

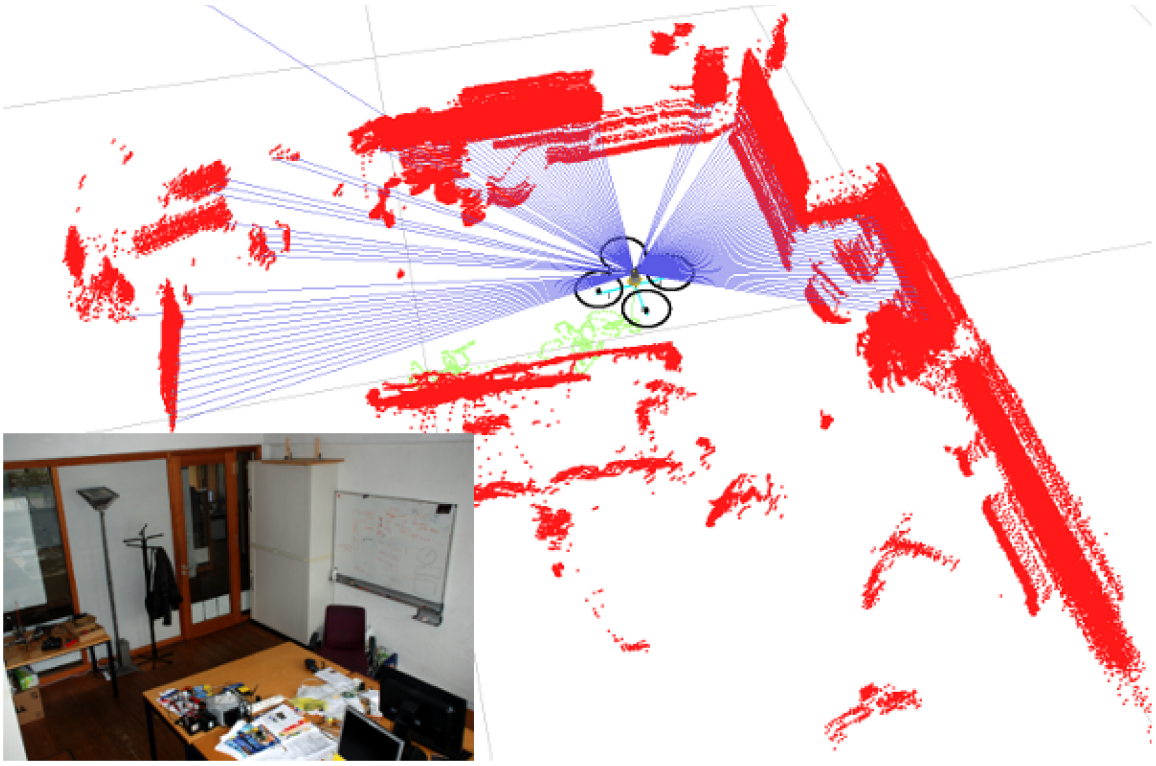

Figure 5.1: Autonomous flight of our quadrotor in a cluttered office room. The free space around the robot is seriously confined, imposing high demands on pose stability, state estimation, and control. The quadrotor uses a laser scanner (blue lines) to sense the environment. The map already acquired by the robot is shown in red. The image in the bottom left shows the office room from a similar view point as the snapshot of our navigation system.

The general principles of navigation algorithms, that have been successfully applied on wheel-based robots, could in principle be transferred to flying vehicles. However, this transfer is not straightforward for several reasons. Wheeled-based robots are inherently stable. In other words, issuing a zero velocity command results in a smooth deceleration until the robot stops. The same does not hold for flying robots that need to be actively stabilized even when they are already at the desired location. Obviously, turning off the rotors in this situation would result in a crash. Even more, due to the fast dynamics of a flying vehicle compared to a ground-based one all the quantities necessary to stabilize the vehicle should be computed within a short time and with an adequate level of accuracy. Thus, porting existing navigation systems for ground robots to aerial vehicles requires to fulfill more stringent constraints on both accuracy and efficiency.

In this chapter, we present the enabling technology for autonomous quadrotor navigation in indoor environments and describe a navigation system including key functionalities, namely localization, planning, surface estimation, map-learning, control, and obstacle avoidance. Although a flying vehicle moves in 3D there is usually enough structure present indoors (i.e., walls, cupboards, ...) to describe the environment in 2D. Instead of using a full 3D representation we therefore rely on a 2D one consisting of vertical structures like walls augmented with the elevation of the floor. The advantage of this choice compared to the full 3D representation is that we can operate in a large class of indoor environments by using efficient variants of 2D algorithms that work on dense grid maps instead of space and time consuming 3D methods. Having these functionalities adapted for the 3D case would be either too slow or not accurate enough given the limited time constraints to make the system stable. To correct for odometry errors, we embed the robots pose into a graph. This allows us to use our tree network optimization algorithm described in the previous chapter for correcting the map when closing loops.

Our system is a result of an integrated hardware/software design that meets several of the

challenging constraints imposed by small-sized flying vehicles while preserving a large degree of flexibility. Furthermore, it can be operated at different levels of autonomy. It can be used to assist a pilot by keeping the position of the vehicle when no commands are given or it can be instructed to autonomously reach given locations in a known map. It can additionally construct a map on-line and correct for errors when closing a loop while flying in an unknown environment. We evaluated our system on an open source quadrotor, the so-called Mikrokopter [29] (see Section 5.2). Figure 5.1 visualizes our quadrotor system and its internal state while autonomously flying within a highly cluttered office room.

This chapter is structured as follows. We first discuss the requirements of a navigation system for an indoor flying quadrotor followed by a description of our system architecture in Section 5.2. Subsequently, we present our navigation system in Section 5.3 and present an extensive set of experiments demonstrating the capabilities of our algorithms using an open-source quadrotor in Section 5.4. Finally, we discuss the relation of our system to the literature in Section 5.5 and conclude in Section 5.6.

5.1 Requirements for Autonomous Indoor Flying

In this section, we present the general problems in robot navigation and discuss the additional issues introduced by a flying platform for indoor environments.

To autonomously reach a desired location, a mobile robot has to be able to determine a collision-free path connecting the starting and the goal location. This task is known as *path planning* and requires a map of the environment to be known. Usually, this map has to be acquired by the robot itself by processing the sensor measurements obtained during an exploration mission. The task of generating the map is known as *simultaneous localization and mapping* (SLAM). In some cases it is sufficient to perform SLAM off-line on a recorded sequence of measurements. If such a map has been obtained, the robot needs to be aware of its position at any point in time in order to follow the path with a sufficient accuracy. This task is known as *localization*. A further fundamental component of a navigation system is the *control module* which aims to move the vehicle along the desired trajectory, given the pose estimated by the localization or SLAM module. For all the tasks described above, we also need to know the current movement of the robot. This is known as *incremental motion estimation*.

Several authors proposed effective control strategies to accurately steer ground vehicles with complex kinematics. Most of these approaches rely on high frequency estimates of the relative movements of the vehicle obtained by integrating the wheel encoders. The localization module, however, does not need to run at a high frequency due to the accuracy of the odometry within short time intervals. Unfortunately, odometry estimates are often not available on flying vehicles. In principle, one could obtain a dead reckoning estimate by integrating the inertial sensors. However, the limited payload typically requires designers to use only lightweight micro-electro-mechanical (MEMS) devices which are affected by a considerable drift. For these reasons, one needs frequent localization updates to implement effective control strategies.

In outdoor scenarios one can estimate the pose of the vehicle by fusing information obtained from a global positioning system (GPS) and an inertial measurement unit (IMU). Unfortunately, a reliable GPS signal is not available indoors. In the case where the building is equipped with "indoor" GPS (e.g., based on ultra wide band (UWB) or indoor motion capturing), the robot is restricted to operate in these previously prepared environments only. Thus, in order to assure maximum autonomy, the robot is required to localize itself and build a map with the on-board sensors only. To detect and avoid obstacles, these sensors should also reliably reveal the surrounding obstacles.

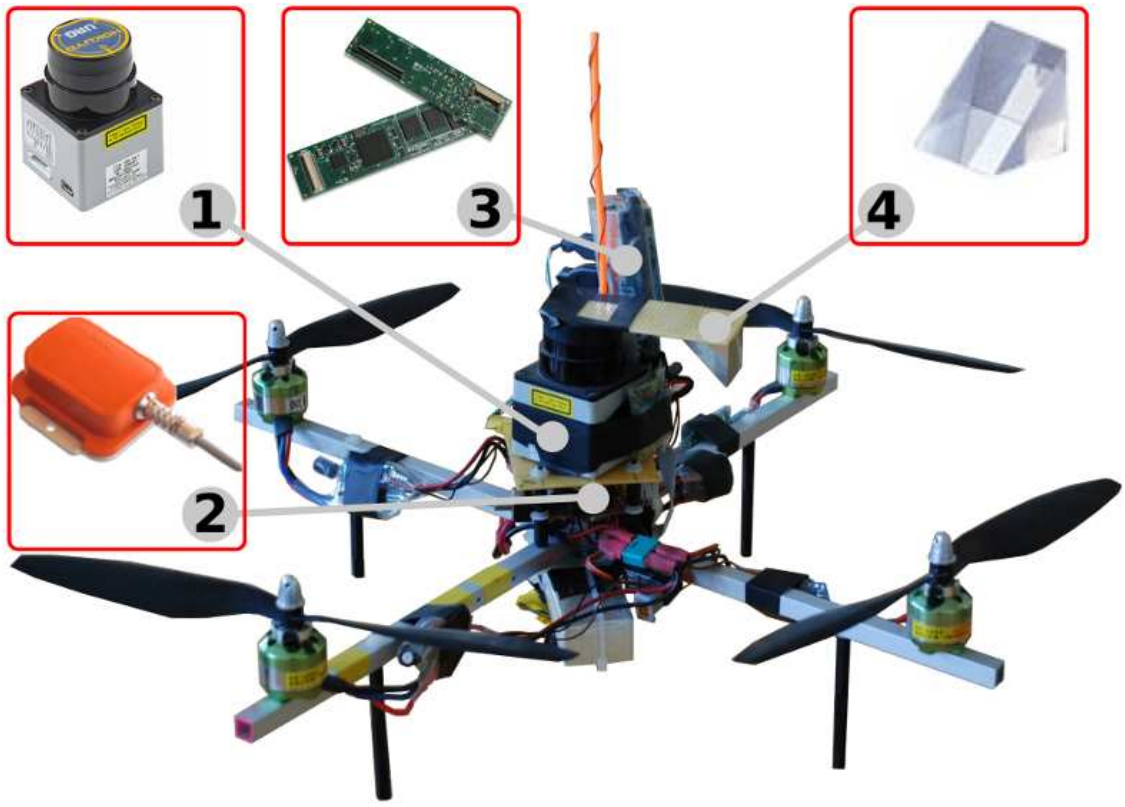

Figure 5.2: The quadrotor used to evaluate the navigation system is based on a Mikrokopter. We equipped the platform with a Hokuyo laser range finder (1), an Xsens IMU (2), a Gumstix computer (3), and a laser mirror (4).

Due to the increased risk of damaging the flying platform during testing, the user should have the possibility to take over the control of the platform at any point in time. Finally, the more complex dynamics of a flying platform pose substantially higher requirements on the accuracy of the state estimation process than for typical ground-based vehicles. Although positioning errors up to 1 m might be acceptable in outdoors scenarios, this is not the case indoors, as the free-space around the robot is substantially more confined. Before presenting our algorithms meeting these requirements, we first describe our hardware platform and general system architecture in the next section.

5.2 System Architecture

Our platform is shown in Figure 5.2. It shows a Mikrokopter [29] open source quadrotor equipped with additional sensors and computational devices. The Mikrokopter comes with a low level controller for roll, pitch, and yaw. Additionally, we equipped it with the following components:

1. a Hokuyo-URG miniature laser sensor for SLAM and obstacle avoidance,
2. an Xsens MTi-G MEMS inertial measurement unit (IMU) for estimating the attitude of the vehicle,
3. a Linux-based Gumstix embedded PC with USB interfaces and a WiFi network card, and
4. a laser mirror used to deflect some of the laser beams along the vertical (z) direction to measure the distance to the ground.

The Hokuyo-URG is a laser range finder able to measure distances up to $5.6\,\text{m}$ at a frequency of $10\,\text{Hz}$. Yet, the ability to detect an obstacle in range depends on the color of the sensed material and the impact angle. We use the laser range scanner for both measuring the distances to obstacles in the surrounding of the robot and measuring the distance to the ground via the laser mirror. The IMU provides accurate estimates of roll and pitch up to 1° at a rate of $100\,\text{Hz}$, that are directly used within our localization and mapping modules. The Gumstix communicates with the micro-controller on the quadrotor via an RS-232 interface and reads all the sensors. It furthermore communicates with an off-board PC over WLAN. Since the embedded PC provides a Linux operating system, we can develop our algorithms off-board on standard PCs and execute them on-board. All on-board sensing and computation devices together weigh about 300 grams and drain approximately 7.5 watts of power. The quadrotor itself consumes about 120 watts while hovering at $50\,\text{cm}$ and is equipped with a $11.1\,\text{V},\ 3,300\,\text{mAh}$ Lithium-Polymer battery (LiPo). The current system has a weight of approximately $1050\,\text{g}$ and is able to fly up to 16 minutes. Including the extends of the blades, the total span of the quadrotor is $0.65\,\text{m}$.

Our navigation system is based on a modular architecture in which different modules (i.e., incremental motion estimation, SLAM, path planning, ...) communicate via the network using a publish-subscribe mechanism. In our current system all device drivers are executed on-board while the more computationally intensive algorithms run on a remote PC communicating wireless with the platform.

Due to the combination of the laser, the IMU, and the laser mirror we are able to generate two additional sets of *virtual* laser measurements. The first set consists of the laser measurements projected onto 2D, given the IMU estimates about roll (ϕ) and pitch (θ). The second set consists of all beams deflected by the laser mirror. Again, the estimate about roll and pitch are used to compute the current distance to the ground. More formally, the laser range scanner measures a set of distances r_i^t along the x-y plane in its own reference frame at time t. Each of these distances can be represented by a homogeneous vector $\widetilde{\mathbf{b}}_i^t$ in 3D space, $\widetilde{\mathbf{b}}_i^t = (r_i^t \cos\alpha_i, r_i^t \sin\alpha_i, 0, 1)^T$, with α_i being the angle of the individual (i.e., $i-$th) laser beam. Let $T_{\text{IMU,laser}}$ be the homogeneous transformation matrix from the IMU reference frame to the laser reference frame, known from an initial calibration procedure, i.e.,

$$T_{\text{IMU,laser}} = \begin{pmatrix} R_{\text{IMU,laser}} & \mathbf{t}_{\text{COR,laser}} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix}, \tag{5.1}$$

with $R_{\text{IMU,laser}}$ denoting the 3×3 rotational matrix (i.e., rotational offset between IMU and laser) and $\mathbf{t}_{\text{COR,laser}}$ being the 3×1 translational vector from the center of rotation (COR) to the laser, respectively. Note that the translational offset between the IMU and the laser is not needed, since both the IMU and the laser are attached to the robot's frame which is a rigid body. Thus, the orientation estimate of the IMU is independent of the location where it is mounted.

Given the estimate about roll (ϕ^t) and pitch (θ^t) at time t, let $T^t_{\text{world,IMU}}$ be the time dependent transformation from the world reference frame to the IMU reference frame, i.e.,

$$T^t_{\text{world,IMU}} = \begin{pmatrix} R_{\theta^t} R_{\phi^t} & \mathbf{0} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix}, \text{ with } \mathbf{0} = (0,0,0)^T. \tag{5.2}$$

Using $c(\alpha), s(\alpha)$ as an abbreviation for $\cos(\alpha)$ and $\sin(\alpha)$ allows us to write $R^t_\theta R^t_\phi$ as

$$R^t_\theta R^t_\phi = \begin{pmatrix} c(\theta^t) & 0 & s(\theta^t) \\ 0 & 1 & 0 \\ -s(\theta^t) & 0 & c(\theta^t) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c(\phi^t) & -s(\phi^t) \\ 0 & s(\phi^t) & c(\phi^t) \end{pmatrix}. \tag{5.3}$$

Given these transformations we can compute the position of the individual laser beam $\mathbf{b}_i^t$ at time t *not* deflected by the laser mirror by:

$$\mathbf{b}_i^t = T^t_{\text{world,IMU}} \cdot T_{\text{IMU,laser}} \cdot \widetilde{\mathbf{b}}_i^t. \tag{5.4}$$

Consequently, the point $\mathbf{h}_i^t$, of a beam $\widetilde{\mathbf{b}}_i^t$ *deflected* by the mirror is calculated by the following chain of transformations:

$$\mathbf{h}_i^t = T^t_{\text{world,IMU}} \cdot T_{\text{IMU,mirror}} \cdot \widetilde{\mathbf{b}}_i^t, \tag{5.5}$$

with $T_{\text{IMU,mirror}}$ representing the transformation from the IMU to the *virtual* laser position that accounts for the effect of the mirror.

5.3 Navigation System

Since roll (ϕ) and pitch (θ) measured by the IMU are in general accurate up to 1°, we can directly use this information within our navigation system. This allows us to reduce the state estimation problem from 6 degrees of freedom (DOF) namely $(x, y, z, \phi, \theta, \psi)^T$ to 4DOF, consisting of the 3D position $(x, y, z)^T$ and the yaw angle ψ. However, the only sensor used to estimate these 4DOF and to detect obstacles is the laser range scanner.

In short, based on known initial calibration parameters and on the current attitude $(\phi, \theta)^T$ estimated by the IMU, we project the endpoints of the laser into the global coordinate frame. Given the projected laser beams, we estimate the $(x, y, z, \psi)^T$ of the vehicle in a 2D map containing multiple levels per cell. To compensate for the lack of odometry measurements, we estimate the incremental movements in $(x, y, \psi)^T$ by matching subsequent 2D laser scans. Finally, we control the altitude of the vehicle and simultaneously estimate the elevation of the underlying surface by fusing the IMU accelerometers and the distance from the ground measured by the laser. Accordingly, we track and map multiple levels underneath the robot within an environment. This enables our robot to correctly maintain its height even when flying over obstacles like tables or chairs. To calculate a path from the current location to a goal we use a variant of D^* lite [91]. While flying towards a goal, we detect dynamic obstacles by comparing the current laser reading to the map which allows the robot to re-plan the trajectory or stop if no valid plan can be calculated anymore.

In the remainder of this section, we first discuss our approach for incremental motion estimation. Subsequently, we present our algorithms for localization in a known map, SLAM, altitude estimation, and pose and altitude control. Finally, we present our algorithms for path planning and obstacle avoidance.

5.3.1 Incremental Motion Estimation

Some tasks, like pose stabilization, rely on an accurate local pose estimate of the vehicle in its surroundings. To this end, we can estimate the relative movement of the robot between two subsequent scans by using a scan-matching algorithm. Since the attitude is known from the IMU, this procedure can be carried out in 2D, assuming structured indoor environments. A scan-matching algorithm estimates the most likely pose of the vehicle $\widehat{\mathbf{x}}^t$ at time t given the previous k poses $\mathbf{x}^{t-k:t-1}$ and the corresponding $(k+1)$ laser measurements $\mathbf{b}^{t-k:t}$, as follows

$$\widehat{\mathbf{x}}^t = \underset{\mathbf{x}:=(x,y,\psi)}{\operatorname{argmax}}\ p(\mathbf{x}^t \mid \mathbf{x}^{t-k:t-1}, \mathbf{b}^{t-k:t}), \tag{5.6}$$

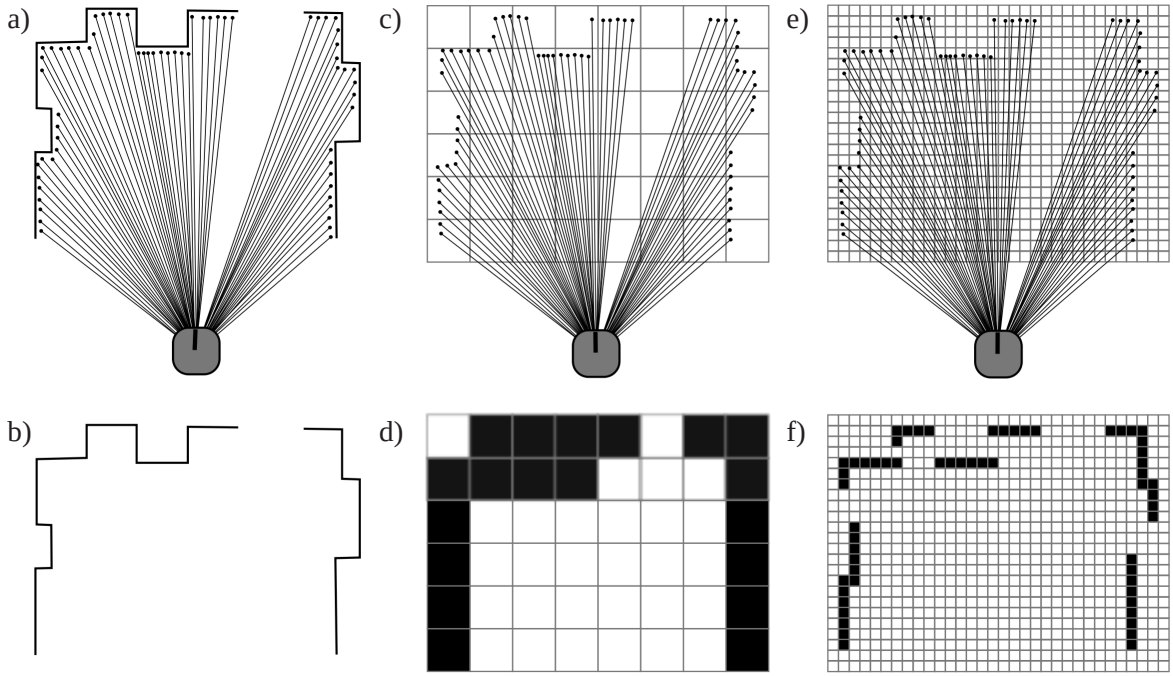

Figure 5.3: The laser range scanner measures a set of distances to the closest obstacles (a). The measurements can be interpreted as a sampled approximation to the surrounding environment (b). A grid map discretizes the world into a set of cells of the same size. The value of a cell reflects the probability that this cell is occupied. Here, the color is proportional to this value with black standing for "occupied" and white for "free". The smaller the size of an individual cell, the higher the resolution of the corresponding map as can be seen by comparing (c) and (d) versus (e) and (f). However, the storage requirements are in $\mathcal{O}(n^d)$, with n being the number of cells per dimension d of the gridmap.

with $\mathbf{b}^t = (\mathbf{b}_1^t, \ldots, \mathbf{b}_m^t)$ denoting the individual laser beams at time t not deflected by the mirror as discussed in Section 5.2. The key idea of a scan-matching algorithm is the following. Assuming the motion between successive laser readings to be small with respect to the field of view, the current laser reading should have a high overlap with the previous one(s). However, each overlap is the result of a relative transformation between the previous scan and the current one. Thus, the transformation which results in the "best" overlap is said to be the relative movement of the robot. In this case we need to address two questions. How to get transformation candidates and how to estimate the quality of their overlap. To solve this, and therefore Equation (5.6), we use correlative scan-matching on multiple resolutions, similar to the approach proposed by Olson [115]. The idea behind a correlative scan-matcher is to discretize the search space $\mathbf{x}^t = (x^t, y^t, \psi^t)^T$ and to perform an exhaustive search in the parameter space around a given initial guess. To efficiently evaluate the likelihood $p(\mathbf{x}^t \mid \mathbf{x}^{t-k:t-1}, \mathbf{b}^{t-k:t})$ of a given solution $\mathbf{x}^t$, we use likelihood fields [149] obtained by the most likely map generated from the last observations $\mathbf{b}^{t-k:t-1}$. In more detail, we build a grid map with a specific cell size given the previous observations. One cell in this grid map is of size $s \times s$ and its value reflects the probability of an obstacle in it. Here, s is the resolution of the grid map, e.g., in cm. A synthetic example of a laser reading and the corresponding grid maps for different cell sizes is shown in Figure 5.3, whereas Figure 5.4 illustrates an example of a likelihood field computed from a map. In this example, the likelihood field is calculated through a convolution of the grid map with a Gaussian kernel K_5 of size 5×5. Note that a Gaussian kernel of size $k \times k$ can be approximated by multiplying the normalized k-th line of Pascal's triangle (transposed) by itself

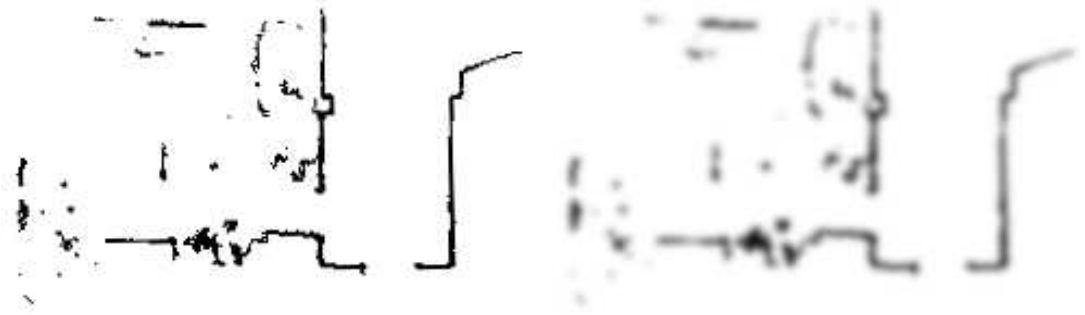

Figure 5.4: This figure illustrates a grid map (left image) and the corresponding likelihood field (right image). The latter is the result of a convolution of the grid map with a Gaussian kernel of size 5×5.

(i.e., the outer product), which in our example is:

$$K_5 = \left(1/16 \cdot (1,4,6,4,1)^T\right) \cdot \left(1/16 \cdot (1,4,6,4,1)\right), \tag{5.7}$$

$$= 1/256 \cdot \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}. \tag{5.8}$$

Recall that our scan-matching algorithm discretizes the search space and performs an exhaustive search within a search radius. For each of these candidate transformations we can now calculate a score (likelihood) using the values from the cells the endpoints of the beams fall into. Since the values represent the negative log likelihood this score is obtained by summing up the individual values.

Thus, the complexity of a correlative scan-matcher depends linearly (per dimension) on the resolution at which the parameters are discretized and on the search range. A naive implementation of this algorithm is not adequate for our application that demands both high accuracy and efficient computation. To overcome this problem, we employ a multi-resolution approach. The idea is to perform the search at different resolutions, from coarse to fine. The solutions found at a coarse level are then used to restrict the search at a higher resolution.

In our implementation (see also Algorithm 2) we use a constant velocity model to compute the initial guess for the search. We furthermore perform the correlative scan-matching at three different resolutions (i.e., $4\,\text{cm} \times 4\,\text{cm} \times 0.4°$, $2\,\text{cm} \times 2\,\text{cm} \times 0.2°$, and $1\,\text{cm} \times 1\,\text{cm} \times 0.1°$). We set the search area a for the first level depending on the maximum speed $v_{\max}$ of the vehicle and on the frequency f of the scanner as $v_{\max}/f$ for each dimension. The search areas of the subsequent hierarchies are calculated as the sum of the resolution of the preceding level plus the current resolution. Consider as an example a maximum velocity of $1.2\,\text{m}$ per second. Since we obtain laser observations at a rate of $10\,\text{Hz}$, this results in a search area of $0.12\,\text{m}$ at the first level in the x-y plane. The search space along the yaw rotation is depending on the aggressiveness of the controller. In our case, we assume a maximum rotational speed of approximately 70 degrees per second. Thus, we obtain a search radius of $0.12\,\text{m} \times 0.12\,\text{m} \times 0.12°$. Given the most likely solutions of this level, we restrict the search area around this initial guess to $0.06\,\text{m} \times 0.06\,\text{m} \times 0.06°$ for the second level and so on. Let N be the number of beams. Then we need to access the memory a total of $3 \cdot 7^3 \cdot N$ times ($7^3 \cdot N$ per level). Without the multi-resolution approach however, we would need $25^3 \cdot N$ look-ups to cover the same search area. Due to the nature of the measurement process, we obtain a more dense sampling of objects (i.e., more laser beams that

Algorithm 2 Multi-Resolution Correlative Scan-Matcher

Input: current projected laser reading not deflected by mirror $\mathbf{b}^t = (\mathbf{b}_1^t, \ldots, \mathbf{b}_m^t)$
Input: number of scan-matchers, n, and the individual resolutions $\mathbf{r}_{1:n}$
Input: likelihood fields for the corresponding resolutions $\mathbf{m}_{1:n}$ // built from last $k-1$ scans
Input: the search area $\mathbf{a} = (a_x, a_y, a_\psi)$ and the initial guess $\widehat{\mathbf{x}}^t = (\widehat{x}^t, \widehat{y}^t, \widehat{\psi}^t)$

$\mathbf{r}_{n+1} = (0, 0, 0)$
for $i = 1 \ldots n$ **do**
  $\mathbf{r} = \mathbf{r}_i$ // $= (r_x, r_y, r_\psi)$
  $T_i = \emptyset$ // candidates found so far
  s_i^* = minimumLikelihood // current best negative log likelihood
  $I_x = (-a_x, -a_x + r_x, -a_x + 2r_x, \ldots, +a_x)$ //search space in x
  $I_y = (-a_y, -a_y + r_y, -a_y + 2r_y, \ldots, +a_y)$ //search space in y
  $I_\psi = (-a_\psi, -a_\psi + r_\psi, -a_\psi + 2r_\psi, \ldots, +a_\psi)$ //search space in ψ
  for $\Delta\mathbf{x} \in I_x \times I_y \times I_\psi$ **do**
    s = calculateNegativeLogLikelihood($\mathbf{b}_1^t, \ldots, \mathbf{b}_{m/2}^t, \mathbf{m}_i, \widehat{\mathbf{x}}^t + \Delta\mathbf{x}$)
    $\widehat{s} = s + (m/2) \cdot$ maxNegativeLogLikelihood($\mathbf{m}_i$).
    if $\widehat{s} < s_i^*$ **then**
      continue
    end if
    $s = s +$ calculateNegativeLogLikelihood($\mathbf{b}_{m/2+1}^t, \ldots, \mathbf{b}_m^t, \mathbf{m}_i, \widehat{\mathbf{x}}^t + \Delta\mathbf{x}$)
    $s_i^* = \max(s_i^*, s)$
    $T_i = T_i \cup \langle s, \Delta\mathbf{x} \rangle$
  end for
  $\langle s_i^*, \widehat{\mathbf{x}}_i^* \rangle$ = getBestEstimate(T_i)
  // if the best solution is below a minimum likelihood, no solution was found
  // in this case we use the initial guess for this level as the result.
  // otherwise the initial guess for the next level is the calculated best solution
  if $s^* >$ minLikelihood **then**
    $\widehat{\mathbf{x}}^t = \widehat{\mathbf{x}}_i^*$
  else
    $\widehat{\mathbf{x}}_i^* = \widehat{\mathbf{x}}^t$
  end if
  $\mathbf{a} = \mathbf{r}_i + \mathbf{r}_{i+1}$ // search area for the next level
end for
return weightedMean($\mathbf{x}_1^*, \ldots, \mathbf{x}_n^*$, GaussianKernel)

are reflected by the object) that are close to the laser scanner. Using all laser beams therefore would lead to a solution biased towards these densely sampled parts of the environment. To limit this problem, we use only a subset of all laser beams. This subset is obtained by choosing for all grid cells only one from all beams that ends in this specific cell.

We can further reduce the overall complexity by including a heuristic which tells us if we can get a better likelihood than the current best one. In theory, when summing up the negative log likelihoods of the beams, we can calculate in each step (i.e., after each beam) the maximum additional score we can get from the remaining beams. If we will not get a higher likelihood than the current best solution in this best case, we can immediately stop processing the beams for this candidate transformation and proceed with the next one. Unfortunately, performing this check after each beam is even more expensive in practice due to the additional calculation.

approach →	**4 cm**	**2 cm**	**1 cm**	**weighted mean**	**unit**
mean(x, y)	$0.107, -0.045$	$0.105, 0.060$	$0.149, -0.040$	$0.066, -0.050$	[m]
std(x, y)	$0.145,\ 0.081$	$0.148, 0.088$	$0.165,\ 0.087$	$0.123,\ 0.076$	[m]
mean($\|v_x\|, \|v_y\|$)	$0.146,\ 0.159$	$0.095, 0.106$	$0.084,\ 0.090$	$0.075,\ 0.072$	[m/s]
std($\|v_x\|, \|v_y\|$)	$0.118,\ 0.117$	$0.071, 0.083$	$0.065,\ 0.072$	$0.058,\ 0.057$	[m/s]

Table 5.1: Effect of the scan-matching algorithm on the pose stability of the flying robot

However, we observed that making this calculation once after half of the beams have been already processed results in an average speed-up of approximately 35%. More quantitatively, our algorithm estimates the incremental motion in less than 5 ms on average using a standard laptop computer. In contrast to this, a single correlative scan-matcher at the same resolution needs about 100 ms.

We control the position of the vehicle based on the velocities estimated by the scan-matcher. Accordingly, the performances of the scan-matcher play a major role in the stability of the robot. In particular, we want to have a fast, accurate, and smooth (i.e., less oscillations) estimate. To get an intuition about the desired accuracy, consider an error in the position estimate of ± 2 cm. Assuming a sensor frequency of 10 Hz this error leads to a variation of $20\,\mathrm{m/s}$ in the velocity estimate between two laser scans. This in turn can generate wrong commands by the controller which reduces stability.

In our hierarchical scan-matcher, the high-resolution estimate (i.e., 0.01 cm grid size) is affected by frequent oscillations due to the limited resolution of the likelihood field. Although these oscillations could in general be filtered out by a low-pass filter, this type of filtering would introduce a phase shift in the pose and velocity estimate (the estimated pose is in the past). Choosing the estimate of the low resolution instead leads to a slow and "choppy" reaction of the robot, since the robot moved already for a certain distance until a correction is executed.

To obtain both an accurate position estimate and a smooth signal, we compute the final solution as the weighted mean of the estimates of all scan-matchers in the hierarchy. The weights of the sum lie on a Gaussian centered at the finest resolution estimate. In several experiments we found that the weighted average of the estimates is better for control than each single estimate. Typical outcomes of hovering experiments are shown in Table 5.1. The table contains experimental results comparing the effect on the pose stability using the estimate of the individual scan-matchers versus our weighted mean approach. All runs reflect experiments where the goal of the quadrotor was to hover at the same spot at 0.5 m height for 15 minutes. To quantitatively evaluate our approach, we compare the mean and standard deviation in both position (x, y), and absolute velocity $(|v_x|, |v_y|)$ obtained from the navigation system. To be able to compare the values over different resolutions, we compute the outcome of our weighted estimate in all runs, but only the outcome of the specific scan-matcher is used for control. It is important to note that the mean of the pose is highly dependent on the initial calibration of the gyroscopes. Nevertheless, we included these values for completeness.

As can be seen, using a weighted average of the different resolutions has a positive effect on the control loop. This originates from the fact that the weighted averaging has a smoothing effect on the pose and velocity estimate but does not include any phase shift into the system. Since we use a simplistic model of our quadrotor (constant velocity), using the output of the weighted mean (with the prediction used as the initial guess for the search) is equal to run a Kalman filter with a large uncertainty on the prediction. Whereas including a more sophisticaded model for the prediction would lead to better estimates, using this simplistic strategy was sufficient for our purposes.

5.3.2 Localization

If a map of the environment is known a-priori, pure localization (in contrast to simultaneous localization and mapping) is sufficient for estimating the remaining 4DOF of the quadrotor. We estimate the 2D position $(x, y, \psi)^T$ of the robot in a given grid map by Monte-Carlo Localization (MCL) [40]. The idea is to use a particle filter to track the position of the robot. Each particle represents a possible pose of the robot. Thus, we approximate the true belief of the robot about its pose via a set of samples, namely the particles. We sample the next generation of particles given the relative movement estimated by the scan-matcher. The update is applied when the robot moved a certain distance ($0.1\,\mathrm{m}$ in our experiments), which prevents the filter from degenerating towards a wrong solution. In short, particle filter-based localization can be summarized through the following algorithm:

Algorithm 3 Monte-Carlo Localization

Input: $\{\mathbf{x}_1^{t-1}, \ldots, \mathbf{x}_n^{t-1}\}$ // set of particles at time $t-1$
Input: $\mathbf{b}^t$ //current measurement
Input: $\mathbf{m}$ //map of the environment
Input: $\mathbf{v}^{t-1}, \Delta\mathbf{x}^t$ // velocity and relative displacement estimate
Output: $\{\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t\}$

1: **for** $i = 1, \ldots, n$ **do**
2: //prediction step
3: $\hat{\mathbf{x}}_i^t$ = motionmodel.sampleMotion($\mathbf{x}_i^{t-1}, \mathbf{v}^{t-1}, \Delta\mathbf{x}^t$)
4: //measurement update
5: w_i = observationModel.calculateObservationLikelihood($\hat{\mathbf{x}}_i^t, \mathbf{b}^t, \mathbf{m}$)
6: **end for**
7: normalizeWeights($\{w_1, \ldots, w_n\}$)
8: $\{\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t\}$ = $\{\hat{\mathbf{x}}_1^t, \ldots, \hat{\mathbf{x}}_n^t\}$.sampleNewSetProportionalTo($\{w_1, \ldots, w_n\}$)
9: **return** $\{\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t\}$

First, we sample a new generation based on a proposal distribution (the motion model) according to

$$\hat{\mathbf{x}}_i^t \sim p(\hat{\mathbf{x}}^t | \mathbf{x}_i^{t-1}, \mathbf{v}^t, \Delta\mathbf{x}^t) \tag{5.9}$$

where $\hat{\mathbf{x}}_i^t$ is the i-th particle, generated from its predecessor $\mathbf{x}_i^{t-1}$, $\mathbf{v}_t$ are the velocities estimated by differentiation, and $\Delta\mathbf{x}^t$ is the relative movement estimated by the scan matcher at time t. This step is indicated by line 3 of Algorithm 3. Subsequently, we calculate for each particle the observation likelihood, given the predicted pose. This calculation is shown in line 5 and the weights are proportional to the likelihood

$$p(\mathbf{b}^t | \hat{\mathbf{x}}_i^t, \mathbf{m}) \tag{5.10}$$

of the measurement. Recall that $\mathbf{b}^t = (\mathbf{b}_1^t, \ldots, \mathbf{b}_m^t)$ is the current projected laser measurement, $\hat{\mathbf{x}}_i^t$ is the pose of the i-th particle, and $\mathbf{m}$ is the known map (which could have also been acquired by a different robot). Again, we calculate this value using likelihood fields as in the case of the incremental motion estimation. Finally, we sample a new set of particles proportional to the normalized weights as shown in lines 7–8 by employing low variance sampling as described in [149].

5.3.3 Simultaneous Localization and Mapping

Our system can acquire models of unknown environments during autonomous or manual flights by simultaneous localizing and mapping the environment (SLAM). The goal of a SLAM algorithm is to estimate both the vehicle position and the map of the environment by processing a sequence of measurements acquired while moving in the environment. Note that a local map is needed until the robot is localized if the robot is running autonomously even when a map is known a-priori. In our system we use the graph-based SLAM algorithm described in Chapter 4. The idea of this algorithm is to construct a graph from the measurements of the vehicle. Each node in the graph represents a position of the vehicle in the environment and a measurement taken at that position. Measurements are connected by pairwise constraints encoding the spatial relations between nearby robot poses. These relations are determined by matching pairs of measurements acquired at nearby locations. Whenever the robot reenters a known region after traveling for a long time in an unknown area, the errors accumulated along the trajectory become evident. These errors are modeled by constraints connecting parts of the environment that have been observed during distant time intervals and are known in the SLAM community as *loop closures* (i.e., previously visited parts of the environment). To recover a consistent map we use our tree network optimization algorithm that finds the positions of the nodes that maximize the likelihood of the edges. Figure 5.5 illustrates a typical pose-graph computed by our algorithm. Here, the quadrotor flies a loop and re-localizes in the previously visited environment. Without the explicit search for loop closures, we can not correct the error accumulated over time as shown Figure 5.5 (left). Using our approach together with our tree network optimization algorithm we are able to correct the map as shown in Figure 5.5 (right). In the remainder of this section we explain how we construct the graph from a sequence of laser scans and IMU readings. The optimization approach itself is discussed in detail in Chapter 4.

Again, we restrict our estimation problem to 4DOF, since the attitude provided by the IMU is sufficiently accurate for our mapping purposes. Furthermore, we assume that the vehicle flies over a piecewise constant surface and that the indoor environment is characterized by vertical structures, like walls, doors, and so on. Although trash bins, office tools on a table or the table itself are violating this assumption using a 2D map is still sufficient for accurate mapping and localization as will be shown in the experimental Section 5.4.2. This arises from the fact that clutter in general is only visible in a small portion of the current measurement, whereas mapping, for example, a desk improves localization since there is a clear difference in x-y between a desk and a nearby wall. Thus we restrict our approach to estimate a 2D map and a 2D robot trajectory spanning over 3DOF, $(x, y, \psi)^T$, i.e., we map all objects as if they had an infinite extend along z. The estimate of the trajectory is the projection of the 6DOF robot motion onto the ground plane, along the z axis. We estimate the altitude of the platform once the 2D position and the attitude are known, based on the procedure described in the next section.

We construct the graph incrementally, by adding one node $\mathbf{x}^t$ at a time. We connect the newly added node and the previous one $\mathbf{x}^{t-1}$ with an edge $\langle t-1, t\rangle$. This edge is labeled with the relative transformation between the two measurements computed from the scan matcher $\mathbf{x}^t \ominus \mathbf{x}^{t-1}$, and can be regarded as an odometry measurement between the current and the previous pose. Whenever the robot reenters a known region, we compute an approximation of the conditional covariances of all nodes in that region using a Dijkstra [153] expansion starting from the current node backwards to all previous poses. We then attempt to match the current scan with all nodes whose uncertainty intersects the current field of view. Finally, we add a new loop closure edge to the graph between the current position and each past node where the matching succeeds, i.e. exceeds a minimum likelihood.

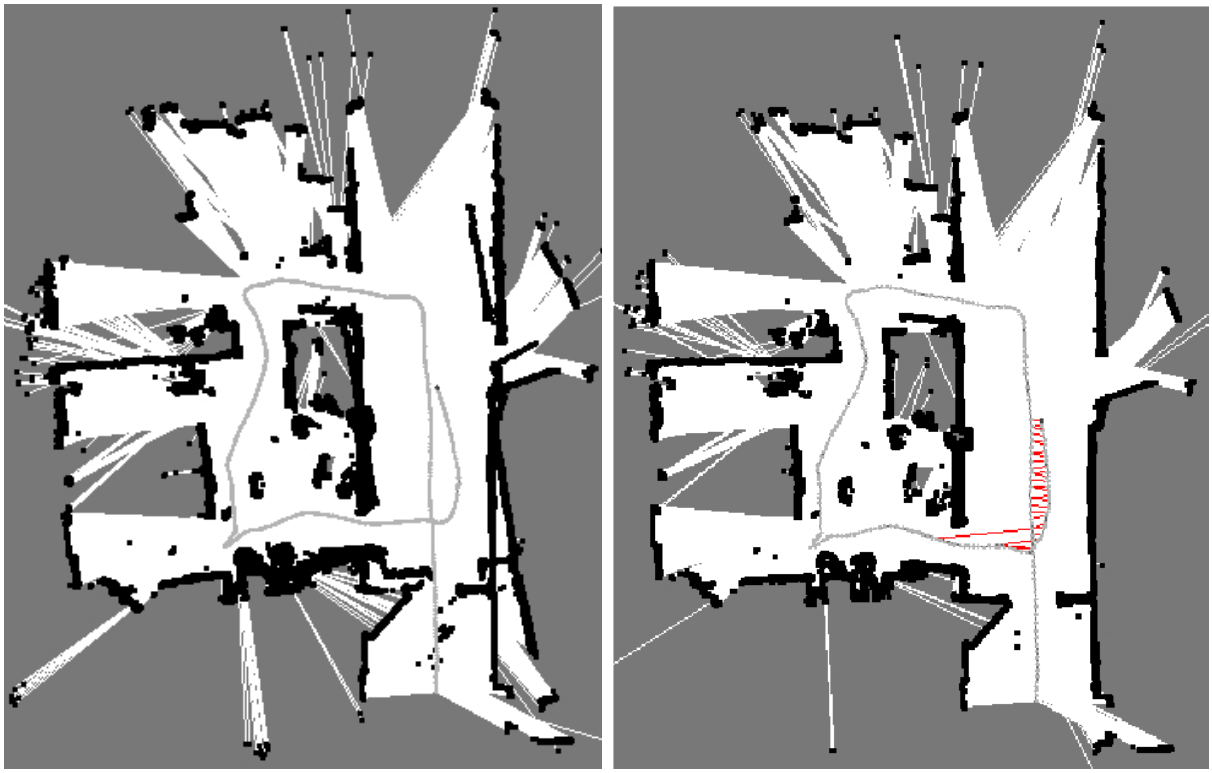

Figure 5.5: This figure illustrates our graph-based SLAM algorithm. The nodes of the graph (robot poses) and the incremental constraints between them are shown in gray. Both images illustrate a map when the robot reenters a known region after traveling for some time in an unknown area. Without searching for loop closures and optimizing the map, the accumulated error results in map inconsistencies as shown in the left image. Our system re-localizes the robot in the previously known part of the environment and inserts the loop closure constraints in the graph (additional red lines in the right image). However, these loop closures are not satisfied by the actual configuration of the nodes and we therefore use our tree network optimization algorithm to calculate a consistent map as shown in the right image.

5.3.4 Altitude Estimation

Estimating the altitude of the vehicle in an indoor environment means determining the global height with respect to a fixed reference frame. Since the vehicle can move over non-flat ground, we cannot directly use the z component of the beams $\mathbf{h}_i$ deflected by the mirror. Otherwise, the vehicle would change its global altitude when flying for example over a table by the height of that table. Our approach therefore concurrently estimates the altitude of the vehicle and the elevation of the ground under the robot. In our estimation process, we assume that the $(x, y, \psi)^T$ position of the robot in the environment is known from the SLAM module described above. We furthermore assume that the elevation of the surface under the robot is piecewise constant. We call each of these connected surface regions having constant altitude a “level”. The extent of each level is represented by a set of cells in a 2D grid sharing the same altitude.

Since our system lacks global altitude sensors like barometers or GPS to determine the altitude of the vehicle, we need to estimate the elevation and the extension of the level under the robot. To this end, we track the altitude of the vehicle over the ground and map different elevations using a two-staged system of Kalman filters. Algorithm 4 describes our approach in detail. The first Kalman filter is used to track the altitude and the vertical velocity of the vehicle by combining inertial measurements, altitude measurements, and already mapped levels under the robot. The second set of filters is used to estimate the elevation of the levels currently measured by the robot. To prevent drifts in the elevation estimate, we update the altitude of a level only

when the robot measures the level for the first time or whenever the robot reenters it (i.e., enters or leaves that particular level). Otherwise, we would constantly add measurement errors in the system leading to a divergence of both the levels altitude estimation and the quadrotor's height estimate.

In detail, the first Kalman filter has a state consisting of the altitude of the robot z, its vertical velocity v_z, and the corresponding covariance matrix Σ. Given the previous state and current measurements from the IMU, we first predict the current altitude of the quadrotor, as indicated in line 2 of Algorithm 4 whenever a new measurement from the mirror is available. The predicted state $(\hat{z}^t, \hat{v}^t)^T$ of the filter is computed as follows:

$$\begin{pmatrix} \hat{z}^t \\ \hat{v}_z^t \end{pmatrix} = A^t \begin{pmatrix} z^{t-1} \\ v_z^{t-1} \end{pmatrix} + B^t a_z^t, \text{ with} \tag{5.11}$$

$$A^t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}, \quad B^t = \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix}, \text{ and} \tag{5.12}$$

$$\widehat{\Sigma}^t = A^t \Sigma^{t-1} (A^t)^T + R^t \tag{5.13}$$

Here, a_z^t denotes the acceleration in z-direction measured by the IMU at time t and Δt is the time elapsed between the current and the last iteration. Furthermore, R denotes the covariance matrix of the prediction. To obtain this matrix, we measured the standard deviation of the IMU's acceleration estimate along the z-direction, σ_z, and computed R^t as

$$R^t = B^t (B^t)^T \sigma_z^2 \tag{5.14}$$

$$= \begin{pmatrix} 0.25\Delta t^4 & 0.5\Delta t^3 \\ 0.5\Delta t^3 & \Delta t^2 \end{pmatrix} \sigma_z^2. \tag{5.15}$$

Although the mirror measures only a single level most of the time it can also happen that during level transitions more levels are sensed. For instance, when flying over a table it can happen that one fraction of the beams is fully reflected by the tabletop, some beams are partially reflected by the tabletop and partially by the floor (which results in a measured value somewhere between the two objects), whereas the remaining beams are fully reflected by the floor. Not taking into account these effects would result in wrong altitude estimates for both the robot and the levels.

Since the beams deflected by the mirror can measure more than one level simultaneously, we need to cluster them into neighboring sets. Each of those sets is assumed to originate from a single level and is parametrized by the mean and the covariance matrix of the beams in the set. Let $\widehat{\mathbf{h}}$ be the set of parameters of these measurements. Based on the prediction of the vehicle pose $\widehat{\mathbf{x}}^{t+1}$ and the measurements $\widehat{\mathbf{h}}$, we compute the expected elevations of the levels underneath the robot (line 5), $\widehat{\mathbf{L}} = (\widehat{L}_1, \ldots, \widehat{L}_k)$. We then match the projected measurements of the level altitude with the levels already present in the map. Since the current 2D pose $(x^t, y^t, \psi^t)^T$ is known, we search in the current neighborhood of the map for a level L' whose elevation is closer than a threshold δ_1 to one of the predicted elevations (lines 7–13).

If such a level is found in the local neighborhood of the current pose, i.e., if the current measurement falls into a confidence region of the prediction, we assume no change in the floor level and can calculate the updated state (z^t, v_z^t) for the altitude and the vertical velocity. Let $L^* = \mathcal{N}(x; \mu_{L^*}, \sigma_{L^*})$ be the Gaussian pdf representing a level consiting of the altitude estimate μ_L^* (mean) and standard deviation σ_L^*. Let furthermore $\mathbf{h}^* = \mathcal{N}(x; \mu_{h^*}, \sigma_{h^*})$ denote the Gaussian

Algorithm 4 Multilevel-SLAM

Input: beams deflected by mirror at time t: $\mathbf{h}^t = (\mathbf{h}_1^t, \ldots, \mathbf{h}_p^t)$
Input: current multilevel map: $\mathbf{M}$
Input: current state: $\mathbf{x}^t = (x^t, y^t, z^t, v^t, a^t, \Sigma^t)^T$

// ——————————— update height estimate and measurements ———————————
$\widehat{\mathbf{x}}^{t+1} =$ predictState($\mathbf{x}^t$)
$\widehat{\mathbf{h}} =$ clusterHeightBeams($\mathbf{h}^t$)
// predict possible level measurements
$\widehat{\mathbf{L}} = (\widehat{L}_1, \ldots, \widehat{L}_k) =$ predictMeasuredLevels($\widehat{\mathbf{x}}^{t+1}, \widehat{\mathbf{h}}$)
$\mathbf{C}^t = \emptyset$ // the set of candidates
for $\widehat{L} \in \widehat{\mathbf{L}}$ **do**
    // does the measured level already exist at the current location or in the local
    // neighborhood $(\pm\Delta x, \pm\Delta y)$?
    if $\exists L' \in \mathbf{M}(x^t \pm \Delta x, y^t \pm \Delta y) : |\widehat{L} - L'| < \delta_1$ **then**
        $\mathbf{C}^t = \mathbf{C}^t \cup \langle \widehat{L}, L' \rangle$
    end if
end for
if $\mathbf{C}^t \neq \emptyset$ **then**
    // measurement update of the filter
    $\langle \mathbf{x}^{t+1}, \mathbf{L}, \mathbf{C}^t \rangle =$ updateStateAndLevels($\widehat{\mathbf{x}}^{t+1}, \widehat{\mathbf{L}}, \mathbf{C}^t, \mathbf{M}$)
else
    $\mathbf{x}^{t+1} = \widehat{\mathbf{x}}^{t+1}$
    $\mathbf{L} = \widehat{\mathbf{L}}$
end if
// ——————————————————————— update map ———————————————————————
// new levels
for $L \in \mathbf{L}, \langle L, \cdot \rangle \notin \mathbf{C}^t$ **do**
    $\mathbf{M}(x^t, y^t) = \mathbf{M}(x^t, y^t) \cup L$
end for
// level exists but was not found in previous time step $\rightarrow$ measurement update for this
// levels in the map
for $\langle L, L' \rangle \in \mathbf{C}^t, \langle \cdot, L' \rangle \notin \mathbf{C}^{t-1}$ **do**
    updateLevelInMap($\mathbf{M}, \langle L, L' \rangle$)
end for
// level was found in the neighborhood, but is not present at the current location
// $\rightarrow$ extend level to current pose in the map
for $\langle L, L' \rangle \in \mathbf{C}^t, L' \notin \mathbf{M}(x^t, y^t)$ **do**
    $\mathbf{M}(x^t, y^t) = \mathbf{M}(x^t, y^t) \cup L'$
end for
// search for loop closures and update map
for $L'_j \in \mathbf{M}(x^t, y^t), L'_k \in \mathbf{M}(x^t \pm \Delta x, y^t \pm \Delta y)$ **do**
    if $L'_j \neq L'_k$ and $|L'_j - L'_k| < \delta_2$ **then**
        mergeLevels($\mathbf{M}, L'_j, L'_k$) // see Equation (5.21)
    end if
end for
return $\mathbf{x}^{t+1}$

of a virtual measurement. This virtual measurement is obtained from the laser beams assumed to be reflected by the level L^* and the uncterainty of the laser sensor. Then, we can compute the *Kalman Gain,* K^t as

$$K^t = \widehat{\Sigma}^t D^T (D\widehat{\Sigma}^t D^T + Q^t)^{-1}, \text{ with} \tag{5.16}$$

$$D = (1, 0), \text{ and} \tag{5.17}$$

$$Q^t = \sigma_{L^*}^2 + \sigma_{h^*}^2. \tag{5.18}$$

Subsequently, we can compute the update of the state by

$$\begin{pmatrix} z^t \\ v_z^t \end{pmatrix} = \begin{pmatrix} \hat{z}^t \\ \hat{v}_z^t \end{pmatrix} + K^t \left((\mu_L^* + \mu_h^*) - D \begin{pmatrix} \hat{z}^t \\ \hat{v}_z^t \end{pmatrix} \right), \tag{5.19}$$

$$\Sigma_t = (I - K^t D)\widehat{\Sigma}^t, \text{ with } I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{5.20}$$

If more then one level was found, we first merge the level's altitude estimates and corresponding beams into one single measurement (similar to Equation (5.21) at the end of this section) and update the state with the corresponding virtual measurement reading. Since the estimate of the global altitude was updated, we also propagate this information to the estimate of the current measured levels, $\widehat{\mathbf{L}}$, underneath the robot.

However, if no level was found, i.e., $\mathbf{C} = \emptyset$ in Algorithm 4, the gap between the current estimate and the measurement is assumed to be generated by a at least one new floor level and thus, the prediction of the filter is the best estimate of the current altitude (lines 14–20).

Once we have an updated estimate of the altitude and the vertical velocity of the vehicle, we can update the elevations of the levels in the map. This is done by a second filtering stage where we use the current estimate for the altitude and the measurements of the current levels to update the multi-level map (lines 21–43). We assume measurements not falling into a confidence region to be generated by a new floor level. These new floor levels can be directly included into the map, as shown in lines 22–25 in Algorithm 4. For all measurements falling into the confidence region of a level in the map, there exist two possibilities. Either this level was already found in the previous time-step, i.e., the robot is (1) flying over the table and thus observed the table already before, or (2) it is currently entering or leaving this particular level. Unfortunately, we cannot update the level's state in the first case, since this would lead to a divergence of the filter as already mentioned in the beginning of this section. In the second case, however, we can use the current altitude estimate in order to update the corresponding altitude of the level in the map (lines 26–30). Here, each elevation of a level is tracked by an individual Kalman filter and the update equations are similar to the Equations 5.11– 5.20.

Since we explicitly store objects in 2D with an extent in x-y rather than by individual levels per cell, we seek for levels present in the neighborhood of the map, explained by one of the measurements currently obtained. If such a level is found (and not present at the current location), we extend this level to the current cell, as shown in lines 33–35.

Note that the robot observes only a limited portion of the underlying surface. Thus it may also happen that the robot "joins" the surfaces of different levels to form a new one. Figure 5.6 illustrates this situation. Initially two levels corresponding to a chair (Level 1) and a table (Level 2) are identified (a). The robot then leaves the table behind, makes a turn, and flies over a different area of the same table. Since Level 2 is not mapped in the neighborhood of the current pose, our system creates a new level (for the same table), noted as Level 3 in (b). Finally, the quadrotor continues to the originally covered area of the table that introduces an

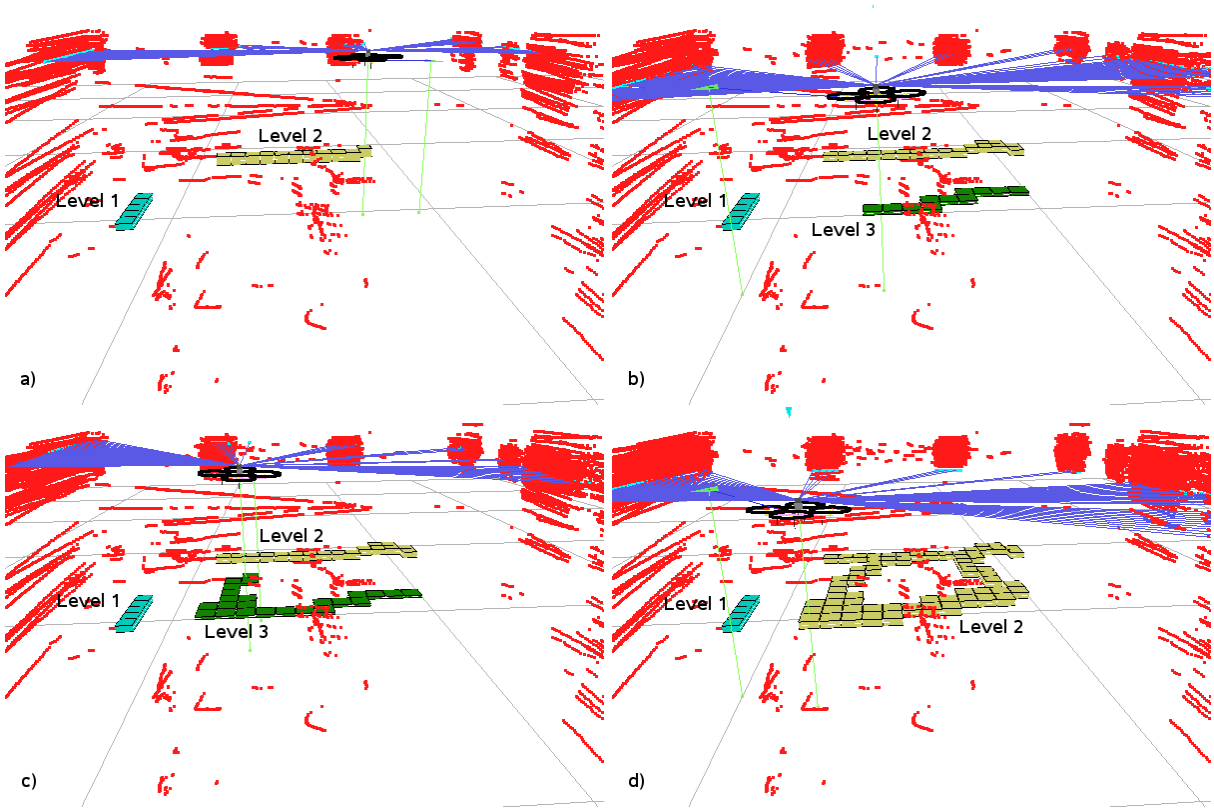

Figure 5.6: Example of level joining during the estimation of the altitude of the vehicle and of the elevation of the underlying surfaces. Each level is represented as a set of contiguous cells in the 2D grid that share the same elevation. The robot starts exploring an office environment. Initially it recognizes two levels (Level 1 and Level 2), corresponding to a chair and a table (a). Subsequently, it flies away from the table, turns back, and flies over a different region of the same table (b). This results in the creation of the new Level 3. Then the robot keeps on hovering over the table until it approaches the extent of Level 2 which has the same elevation as Level 3, originating from the same table. This situation is shown in (c). Finally, the robot enters Level 2 from Level 3. Our system recognizes these two Levels to have the same elevation. Accordingly, it merges them and updates the common elevation estimate (d).

intersection of the current Level 3 and the previously generated Level 2. As a consequence, it joins Levels 2 and 3 (see (c) and (d)).
When two levels, L'_j and L'_k, having altitudes μ_j and μ_k and covariances σ_j^2 and σ_k^2 are merged (lines 37–41), the Gaussian estimate $\mathcal{N}(\mathbf{x}; \mu, \sigma)$ of the joint level has the following values:

$$\mu = \frac{\sigma_k^2 \mu_j + \sigma_j^2 \mu_k}{\sigma_j^2 + \sigma_k^2}, \quad \sigma^2 = \frac{\sigma_j^2 \sigma_k^2}{\sigma_j^2 + \sigma_k^2}. \tag{5.21}$$

To summarize, we store a level as a set of 2D grid cells representing the area covered by the corresponding object. First, we estimate the current height of the robot given the known levels in the multi-level map. In a second step we update the map, given the estimated altitude of the robot. Here, a level is constantly re-estimated whenever the vehicle enters or leaves this specific level, and the data association is resolved by the known $(x, y, \psi)^T$ position of the vehicle. Finally, measurements not explained by any level present in the map are assumed to be generated by new levels that are then included in the map. Given the techniques described so far we are able to estimate the current pose of the robot up to an accuracy of the finest resolution of the grid map (0.01 m in our case) and can now take care of the high-level control enabling autonomous flights.

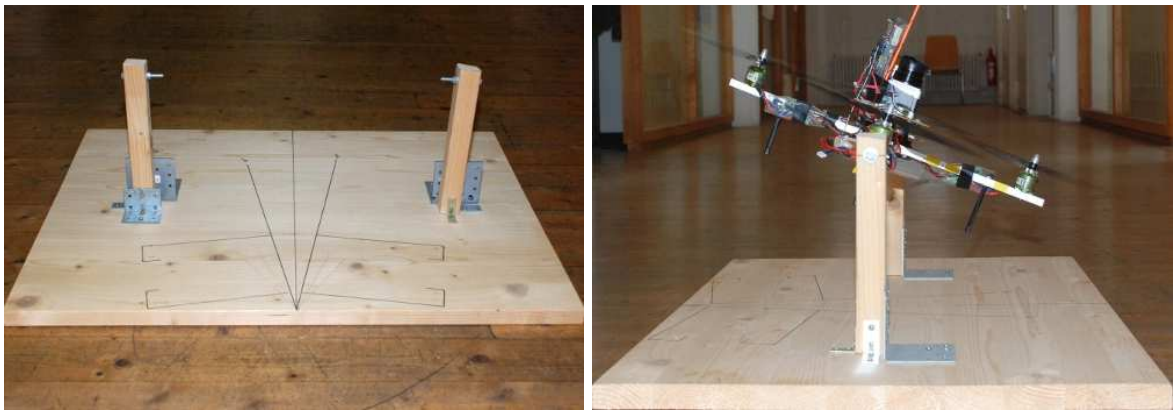

Figure 5.7: Our test bench for learning a mapping between a command and the corresponding angle. This simple device allows for fixing one axis of the quadrotor and monitoring the other one using the IMU.

5.3.5 High-Level Control for Pose and Altitude

The high level control algorithm is used to keep the vehicle in the current position. The outputs of the control algorithm are variations in the roll, pitch, yaw, and thrust, denoted respectively as u_ϕ, u_θ, u_ψ, and u_z. The inputs are the position and the velocity estimates coming from incremental scan matching. A variation of the roll results in a variation of the acceleration a_y which in return translates into a motion along the y-axis. Analogously, a variation in the pitch finally results in a motion along the x-axis and a variation of the thrust results in a change in the vertical acceleration. We separately control the individual variables via proportional-integral-differential (PID) or proportional (P) controllers. Since in our case all control commands are dependent on the current pose estimate, our high level control module runs at $10\,\text{Hz}$, since the laser scanner provides measurements at this rate.

Note that the Mikrokopter (as most commercially available platforms) is equipped with a low level controller for roll, pitch, and yaw. Thus we do not have to take care of the control of the individual motors. Instead, we need to calculate appropriate control commands resulting in a desired angle or thrust. In our particular case, the low level controller of the Mikrokopter quadrotor runs at $500\,\text{Hz}$. Since commands for the yaw on common platforms determine how fast the quadrotor should turn and not how far, these parameters reflect the users wish of the robots aggressiveness with respect to the yaw rotation. In contrast to this, commands for roll and pitch result in a desired angle for which independent mapping functions must be learned. In order to learn the mapping for our quadrotor, we fixed one axis of the vehicle to an external frame allowing the vehicle to rotate along the other axis only. We learned the mapping function by monitoring the current angle measured by the IMU compared to the sent command. Our test bench for learning this mapping is shown in Figure 5.7.

During an autonomous flight, the computed commands are sent directly to the micro-controller via RS 232 which is in charge of the low level control (roll, pitch, and yaw) of the platform. For safety reasons, the user can always control the vehicle via a remote control and our system fuses the commands from the user and from the program. During our experiments, we allow the programs to perturb the user commands by $\pm 20\%$. In this way, if one of the control modules fails the user still has the possibility to safely land the vehicle immediately without needing to press a button first.

In particular, we control the pitch and the roll by two independent PID controllers that are fed with the x and the y coordinates of the robot pose. The control function in x is the following:

$$u_\phi = k_p \cdot (x - x^*) + k_i \cdot e_x + k_d \cdot v_x. \tag{5.22}$$

Here x and x^* are the measured and the desired x-positions, v_x is the corresponding velocity, and e_x denotes the error integrated over time. The control in y is analogous to the control in x. Note that the integral part could be omitted (i.e., $k_i = 0$), but we have observed an improved hovering behavior if a small k_i is used. This originates from the fact that in our case only integer values can be transmitted to the micro controller although the desired command is a float value. The corresponding values k_p, k_i, and k_d were estimated by searching in a predefined parameter space in an extensive set of real world experiments. The relation between the proportional part k_p (acceleration) and the differential part k_d (deceleration) was manually adjusted in such a way that the robot's maximum velocity does not exceed $1\,\mathrm{m/s}$. We also took special care during parameter fitting in order to not overshoot while approaching the desired goal location. We will show in the experimental Section 5.4.4 that using this simplistic model is sufficient for keeping the desired position in x and y direction up to $\pm 0.2\,\mathrm{m}$.
We control the yaw, ψ, by the following P controller:

$$u_\psi = k_p \cdot (\psi - \psi^*). \tag{5.23}$$

Here ψ and ψ^* are the measured and desired yaw and u_ψ is the control input, whereas k_p reflects the desired rotational speed of the robot. Even without an incremental and a differential part, this control is able to rotate the quadrotor to the desired angle with an error of less than 2°.

The altitude is controlled by a PID controller which utilizes the current height estimate z, the velocity v_z, and the current battery voltage U_t respectively. The control u_z is defined as

$$u_z = \text{offset}(V_{\text{bat}}(t)) + k_p \cdot (z - z^*) + k_i \cdot e_z + k_d \cdot v_z, \tag{5.24}$$

with k_p, k_i and k_d being the constants for the P, I, and D part respectively, and $\text{offset}(V_{\text{bat}}(t))$ denotes the thrust command offset given the current battery voltage $V_{\text{bat}}(t)$ at time t. Here z^* denotes the desired height and e_z denotes the integrated error. Including a thrust command offset allows us to treat the system as stationary, and therefore to use constant coefficients for the PID controller. We learned the function $\text{offset}(V_{\text{bat}}(t))$ by monitoring the thrust and the battery level of the vehicle in an expectation-maximization fashion. We started with a PID control without $\text{offset}(V_{\text{bat}}(t))$ and recorded the computed thrust command required to keep the current altitude during several test flights. For each battery level $V_{\text{bat}}(t)$ we then computed the average thrust command used to keep the current altitude. In subsequent flights we used this offset as an initial guess for $\text{offset}(V_{\text{bat}}(t))$ and repeated the experiments resulting in an refinement for $\text{offset}(V_{\text{bat}}(t))$ until no major change in the estimated offset appeared.

To sum up, we can now create maps of the environment during a mission and use the control algorithms described above to let the quadrotor autonomously fly to a predefined goal location. However, up to now we assume a direct path of flight towards the goal. Since this assumption does not hold in indoor environments (e.g. flying around a corner), we need path planning techniques. This allows us to compute a valid trajectory the quadrotor has to follow in order to reach the desired goal location. Furthermore, this module should be able to react quickly to appearing dynamic obstacles and force the robot to either avoid them (if possible), or to stop when no valid plan can be computed anymore. In the next section, we describe our modified version of the popular D* lite algorithm used within our navigation system.

5.3.6 Path Planning and Obstacle Avoidance

The goal of the path planning module is to compute a path from the current location to a user specified goal location. This path should satisfy one or more optimality criteria and should

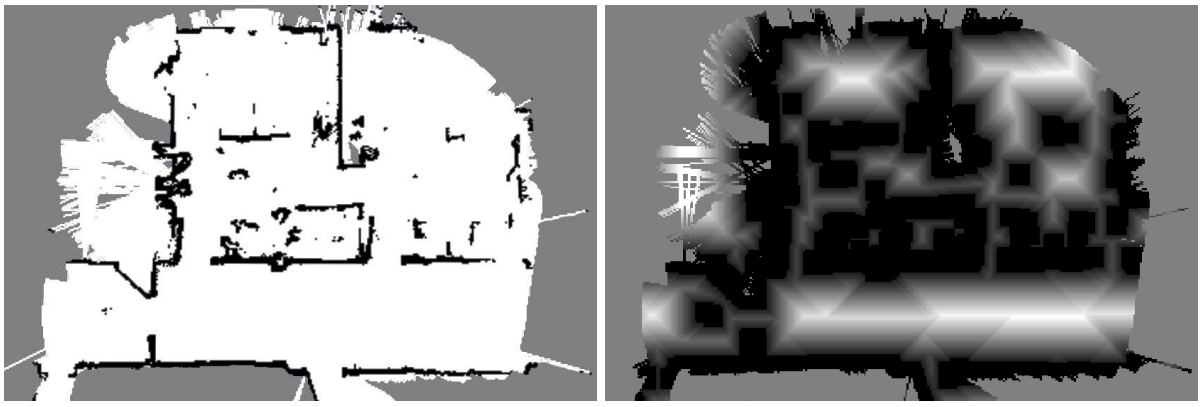

Figure 5.8: Grid map of the environment (left) and the corresponding cost map (right). The darker a cell the higher the cost for traversing it. Black cells indicate obstacles with infinite traversability costs. By using this cost function the robot prefers to traverse regions with a high clearance (brighter cells). We model the quadrotor as a point mass and extend all obstacles in the gridmap by half of the quadrotor's size plus a safety margin reflecting the robots uncertainty in pose stabilization. Therefore, the obstacles in the right image are thicker than in the left one.

be safe enough to prevent collisions even in the case of small disturbances. Safety is usually enforced by choosing a path that is sufficiently distant from the obstacles in the map. Finally, due to the increased degrees of freedom of a flying vehicle compared to a ground robot, the path should be planned in 4DOF space (i.e., $(x, y, z, \psi)^T$) instead of 3DOF (i.e., $(x, y, \psi)^T$). In our system we use D* lite [91], a variant of the A^* algorithm that can reuse previous solutions to correct an invalid plan rather than recomputing it from scratch. Since directly planning in 4DOF is too expensive for our system, we compute the path in two consecutive steps. First, we use D* lite to compute a path in the x–y–z space, but we only consider actions that move the robot in 2D space x–y. For each $(x, y)^T$ location we know the elevation of the surface underneath the robot from the multi-level map. This known elevation is used to determine a possible change in altitude the robot would have to take when moving to a nearby cell. A change in altitude is reflected by increased traversability costs proportional to the distance in the z-direction. In other words, we only allow the robot to fly over obstacles but not underneath. Furthermore, the cost function of a state $(x, y, z)^T$ of the robot depends on the distance of that location to the closest vertical obstacle in the map. The quadrotor has a quadratic shape. To simplify path planning, we model it as a point mass in the cost map but extend all obstacles by half of the quadrotor's size and an additional safety margin reflecting the pose uncertainty during autonomous flight. For reasons of computational complexity, we use the same resolution for the distance grid map as in the first level of the scan-matcher (i.e., $0.04\,\mathrm{m} \times 0.04\,\mathrm{m}$). An example of a map and the corresponding cost map is shown in Figure 5.8.

If there exists a valid path from the current location to the goal, the technique described above will return the optimal path with respect to the cost map. The computed trajectory is a sequence of neighboring grid cells and we could in principle follow this trajectory independent of the current yaw angle. However, since the laser scanner is heading forwards, it is desirable that the robot turns towards the direction of flight first which allows us to detect dynamic obstacles. On the other hand, we want the quadrotor to perform small maneuvers, like flying $10\,\mathrm{cm}$ backwards, without performing a rotation first. Therefore we augment each cell of the trajectory with a computed ψ component. To achieve the behavior mentioned above, we first compute the desired angle that would result in flying forwards with respect to the local frame of the robot. In contrast to this, we compute penalty costs for not turning towards the next cell in the trajectory, proportional to the difference of the angle. Trading off the costs of rotation versus costs of moving to the desired cell without rotating first allows the robot to perform pure sidewards or even backwards movements and thus prevents the vehicle from performing unnatural maneu-

vers. We furthermore re-use the already existing solution (plan) whenever possible. Although we can compute a new plan after a new state estimation is available (i.e., each 100 ms), we do this only when the previous one has been labeled as invalid. This can happen for two reasons. Either the robot detected dynamic obstacles blocking his current path or the previous plan was already used for a certain period of time ($\Delta t = 500$ ms in our implementation). The latter constraint enables us to correct for detours in the trajectory that have been introduced to avoid obstacles that are no longer present. As stated above, we use a grid resolution of 4 cm in our implementation. With these settings, the planner requires about 50-80 ms to compute a typical 10 m path from scratch. Re-planning can be done in less than 10 ms.

Dynamic obstacles are detected by considering the endpoints of the laser beams that are not explained by the known map. This allows us to detect dynamic obstacles very fast and is known in the literature as background subtraction. Each detected dynamic obstacle is enlarged with a safety margin of 1.5 m and the cells in the map are augmented with infinite traversability costs. In this case, our planner either computes a detour or returns no plan if there does not exist a valid trajectory to the goal anymore. In the latter case this forces the quadrotor to stop moving and hover at the current location. On the other side, we re-compute the costs for traversing a cell when it is not occupied by a dynamic obstacle anymore. This allows us to recover a trajectory and continue the mission.

5.4 Experiments

In this section we present experiments that show the performances of each of the modules presented in the previous section, namely: localization, SLAM, multi-level mapping, autonomous pose stabilization, path planning, and obstacle avoidance. All modules have been intensively tested under real world conditions and most of the modules were active during all of the numerous (> 50) live demonstration. However, for better readability, we will describe the outcome of a typical experiment for each module only. At the end of this section, we will also show the results of an experiment where the sensors of the quadrotor were powered by a fuel-cell prototype. Videos of a series of different flights can be found on the Web [123].

5.4.1 Localization

Using 2D grid maps for localization enables our system to operate with maps acquired by different kinds of robots and not necessarily built by the flying vehicle itself. In this section we present an experiment in which we perform global localization of the flying quadrotor in a map acquired with a ground-based robot. This robot is equipped with a Sick LMS laser scanner mounted at a height of approximately 80 cm. In this experiment, the robot autonomously kept a height of 50 cm ± 10 cm and we employed $5,000$ particles in the filter for global localization. Given this number of particles, our current implementation requires 10 ms per iteration in total on a standard 2 GHz laptop, including the incremental scan-matching that takes about 5 ms on average. Figure 5.9 shows three snapshots of the localization process at three different points in time. In each of the snapshots the green (small) circle indicates the current true pose. In this case, the true pose was obtained by manually matching the scan to the map. The blue circle indicates the current estimate of the filter and the highlighted (pink) part of the environment is the projection of the current laser measurement with respect to the location of the robot as it is estimated by the filter. Each small black dot surrounded by (white) free space represents a particle. The top image depicts the initial situation, in which the particles were sampled uniformly over the free space. After approximately 1 m of flight, the particles start to focus around the true

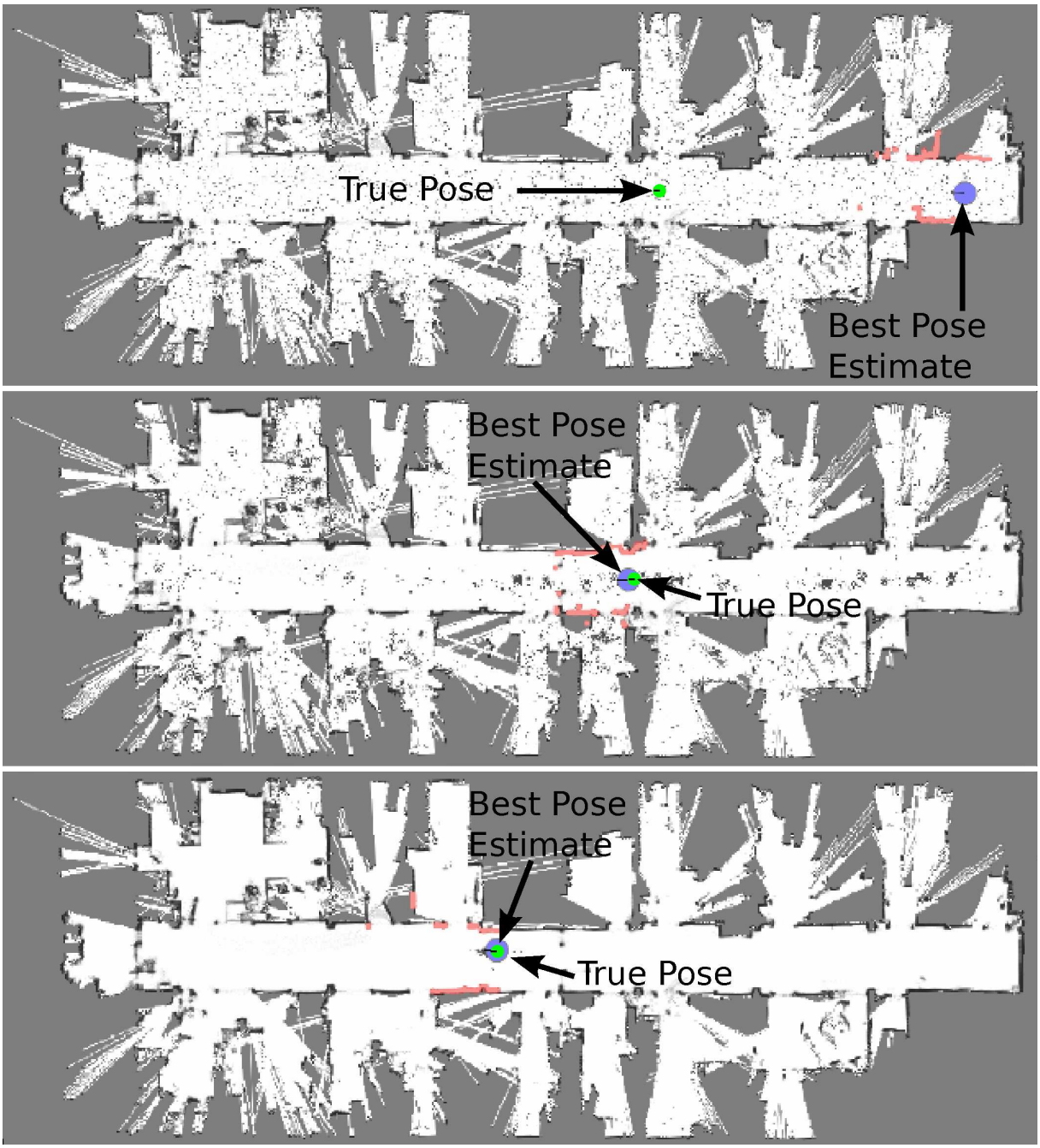

Figure 5.9: Global localization of our quadrotor in a map previously acquired by a ground-based platform equipped with a SICK LMS laser scanner mounted at a height of $0.8\,\text{m}$. Here, the quadrotor kept an altitude of $0.5\,\text{m}\pm0.1\,\text{m}$. The blue and the green circle highlight the current estimate of the particle filter and the true pose respectively. The highlighted part of the environment (pink) reflects the laser measurement projected at the pose estimated by the filter. Particles are shown as black dots within the free space. Top: initial situation. Here, $5,000$ particles were employed and sampled uniformly over the free space. Middle: after approximately $1\,\text{m}$ of flight the particles start to focus around the true pose. Bottom: after approximately $5\,\text{m}$ of flight the quadrotor is localized.

pose of the robot (see middle image). Here, already most of the particles originally located in an office room have been replaced by particles within the corridor. The bottom image depicts the situation after approximately $5\,\text{m}$ of flight. By flying this distance, the robot collected enough distinct information in this experiment to globally localize itself. Throughout all experiments carried out in this environment, it took in average about $7.1\,\text{m}\pm2.3\,\text{m}$ until the filter converged to a solution with an average error of $\pm0.05\,\text{m}$ (due to the discretization of the map).

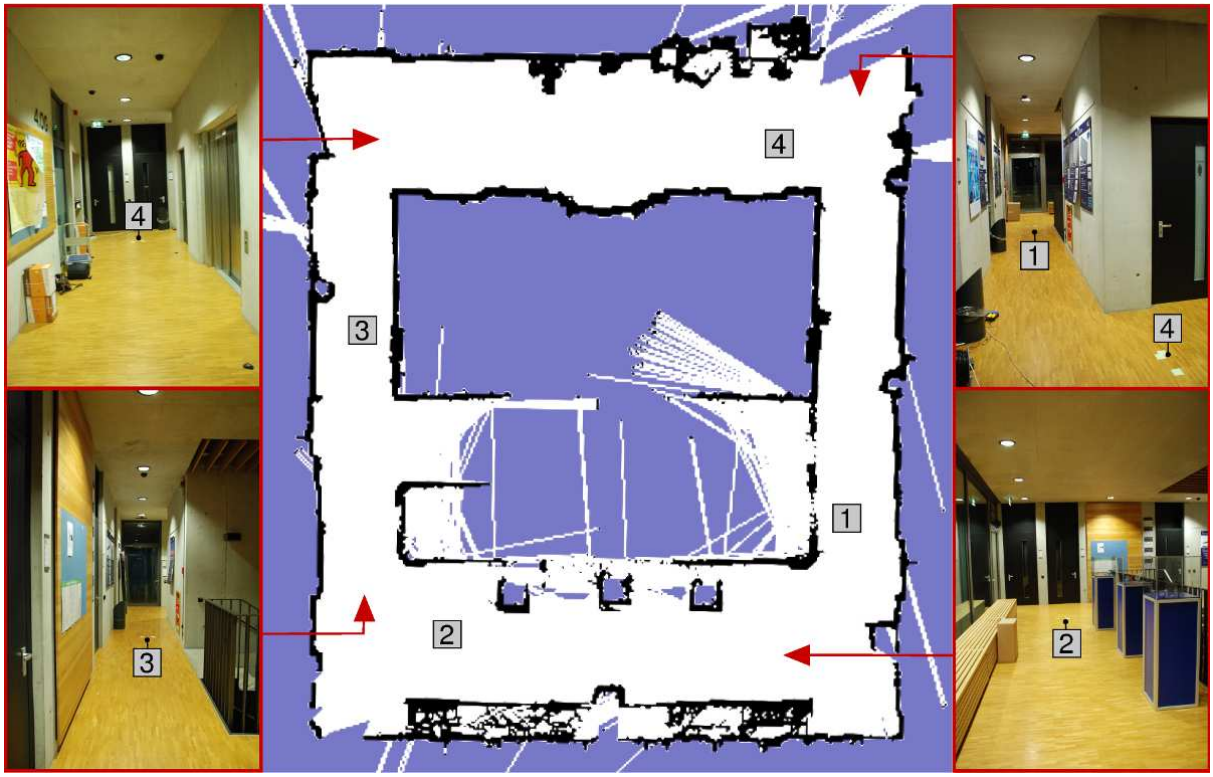

Figure 5.10: Map of on office building built with our approach using the quadrotor. The labels 1–4 reflect the locations of individual markers used for evaluating the accuracy of our mapping approach. Red arrows indicate the pose of the corresponding camera images. The clutter in the bottom of the map originates from the seating containing horizontal slots (see bottom right image).

5.4.2 Simultaneous Localization and Mapping

We also evaluated the mapping system by letting the quadrotor fly four loops (approximately $41\,\mathrm{m}$ each) in a rectangular shaped building with an approximate corridor size of $10\,\mathrm{m} \times 12\,\mathrm{m}$ (outer walls). The result of our SLAM algorithm is shown in Figure 5.10. To quantitatively evaluate the accuracy of our mapping system we placed markers on the floor (labeled $1, \ldots, 4$) and manually landed the quadrotor close to the markers. The map in Figure 5.10 also contains the locations of the markers and is annotated with four red arrows. They reflect the approximate origin of the taken camera images in order to give an impression of the environment.

Since we never perfectly landed on the predefined markers we manually moved the quadrotor the remaining centimeters to match the predefined spots. This enables us to measure three types of errors: the re-localization error, the absolute positioning error, and the error in open-loop. The re-localization error is the difference between the current estimate and the estimate of the same real world pose in the previous loop. The error in open-loop is the re-localization error without enabling graph optimization. The absolute error is the difference between the estimated pose and the ground truth. To measure the absolute error we manually measured the locations (with respect to the origin set to marker 4) of the markers and compared it to the positions estimated by the robot when landing on the corresponding spots. Table 5.2 shows the manually measured and the estimated poses of the markers for all loops. As can be seen, both the relative error between the individual loops and the global pose estimation with respect to the manually measured ground-truth have a maximum error of $1\,\mathrm{cm}$. In this experiment, the incremental mapping during the first loop was accurate enough ($< 1\,\mathrm{cm}$ error) thus the optimization did not improve the map at all. However, when ignoring the first loop, our optimization algorithm leads to a corrected map similar to the one of the first loop. Note that all subsequent loops were also

marker	loop 1	loop 2	loop 3	loop 4	ground-truth
x_1	1.11 m	1.11 m	1.11 m	1.10 m	1.11 m
y_1	-7.50 m	-7.51 m	-7.50 m	-7.50 m	-7.50 m
x_2	-6.21 m	-6.21 m	-6.21 m	-6.21 m	-6.21 m
y_2	-9.21 m	-9.21 m	-9.21 m	-9.21 m	-9.21 m
x_3	-7.85 m	-7.85 m	-7.85 m	-7.85 m	-7.85 m
y_3	-3.83 m	-3.83 m	-3.83 m	-3.82 m	-3.82 m
x_4	-0.01 m	-0.01 m	-0.01 m	-0.01 m	0.00 m
y_4	-0.00 m	-0.00 m	-0.00 m	-0.00 m	0.00 m

Table 5.2: Estimated and manually measured locations of the markers for the flight containing four loops in total. Note that the quadrotor re-localized in the existing map build of the previous loop(s) during the subsequent ones.

marker	loop 1	loop 2	loop 3	loop 4	finest resolution
x_4	-0.01 m	-0.35 m	-0.08 m	-0.17 m	0.01 m
y_4	-0.00 m	0.12 m	-0.07 m	0.04 m	
x_4	-0.42 m	-0.59 m	-0.36 m	-0.64 m	0.02 m
y_4	0.20 m	0.23 m	0.11 m	0.33 m	
x_4	-0.91 m	-0.59 m	-0.54 m	-0.60 m	0.04 m
y_4	0.28 m	0.38 m	0.29 m	0.29 m	

Table 5.3: Comparison of our incremental SLAM (without map optimization, i.e., incremental SLAM) for each loop using different grid resolutions at the finest level of our hierarchical scan matcher. Each scan-matcher consists of three levels, with the second level having twice the grid resolution as the finest level. The top level has a grid size equal to four times the finest level.

re-localized in the existing map. We therefore also evaluated each loop independently of each other without optimization. The results of the individual loop flights for marker 4 (origin) are shown in Table 5.3 (first row). The worst flight (2nd loop) resulted in an error of approximately 0.37 m total distance to the origin. To get an additional intuition about the effect of the incremental SLAM algorithm we also evaluated the effect of using different grid resolutions at the finest level of our hierarchical mapping approach on the accuracy of the individual loops. The outcome of this experiments can be seen in the second and third row of Table 5.3. As can be seen, using a cell size of 0.01 m yields the best accuracy throughout this experiment.

Recall that we assume the robot is operating in indoor environments build of vertical structures like walls, cupboards, and so on. This simplification allows us to treat the SLAM problem in 2D and estimate the incremental motion in less than 5 ms. One may now imply, that this simplification allows us to fly in highly restricted indoor environments only, since the SLAM accuracy could drop rapidly when flying at different levels of altitude. As already mentioned earlier, our approach will not work satisfactory in extremely cluttered environments, like, for example, in a forest, but indeed provides accurate results in typical indoor environments. However, although the presented SLAM algorithm is carried out in 2D, we can always re-project the data into 3D given the estimated 2D pose. To emphasize that our simplification yields accurate 3D maps we performed additional experiments. In the first experiment, we let the quadrotor hover around the origin and occasionally changed the altitude resulting in laser measurements taken at different heights ranging from 0.2 m up to 1.70 m. The accumulated 3D point cloud given the estimated pose of the SLAM approach is shown in Figure 5.11 (top). Here we can already see, that our motion estimation algorithm yields accurate results since there are no inconsistencies present in the point cloud like, for example, double walls. Even more, we can use

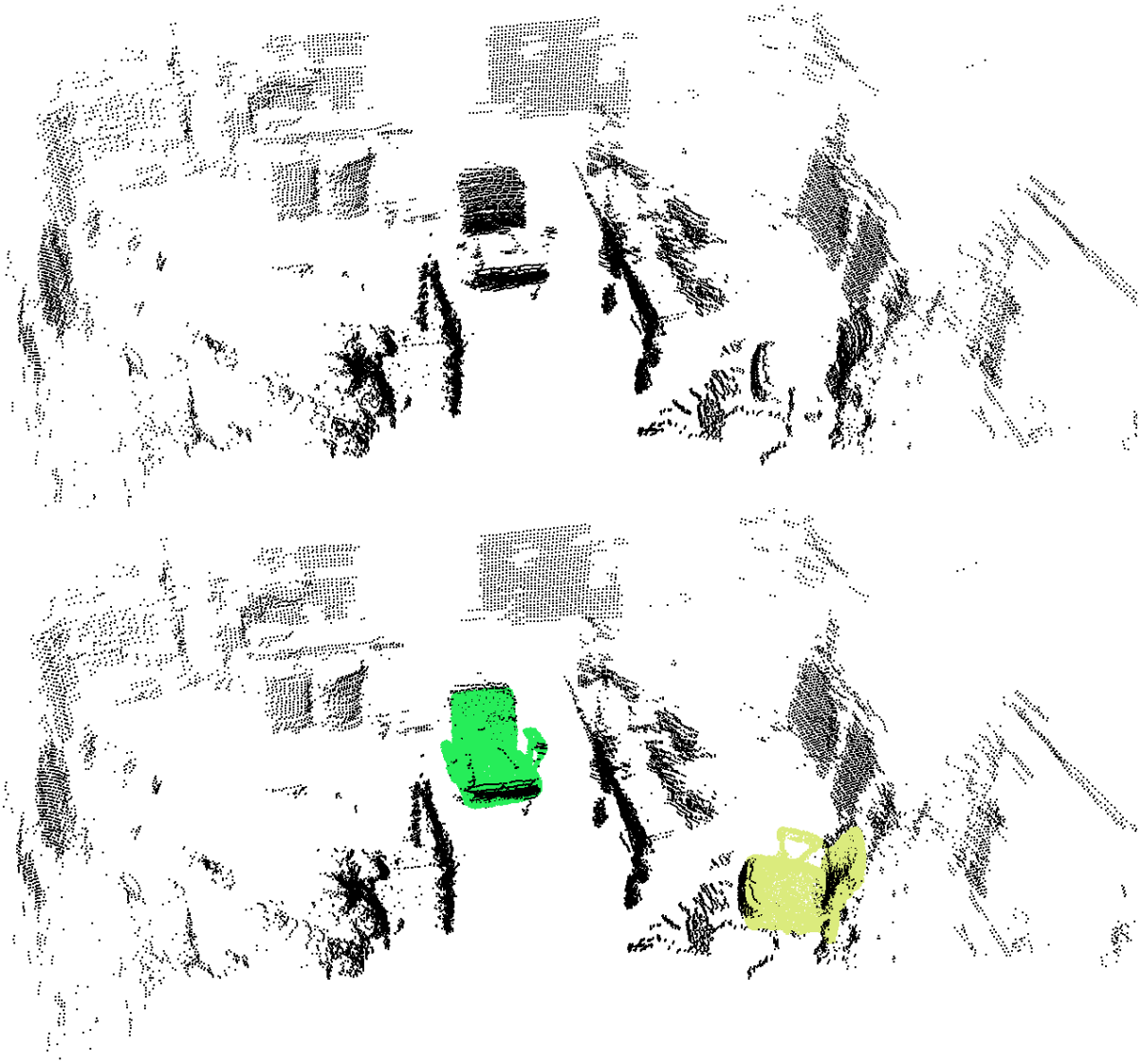

Figure 5.11: This experiment was performed to illustrate the accuracy of our SLAM algorithm. Although we assume structured indoor environments and thus project the data into 2D, the estimated pose is accurate enough to reconstruct the three dimensional environment (top) without any inconsistencies like double walls. Even more, we can detect typical objects in the environment previously build of data gathered by ground robots. The bottom image depicts the situation for detecting the object "chair" using the approach of Steder *et al.* [144].

this 3D data to detect objects in the environment. To achieve this, we applied the object detection algorithm proposed by Steder *et al.* [144, 72]. In their work, individual objects-models like a chair are learned from high density laser data. This data was obtained by a wheeled robot equipped with a Sick LMS laser scanner mounted on a pan-tilt unit. A 3D scan is obtained when the wheeled robot stays at the spot and uses the pan-tilt unit to cover as much 3D space as possible. Since our quadrotor is not equipped with a pan-tilt unit, we simulated the data acquisition by changing the current altitude when hovering around the same spot. The bottom image shown in Figure 5.11 depicts the outcome of the matching for the object "chair". Here, both chairs in the environment were correctly detected and the corresponding measurements are overlayed with the high density point clouds of the chair object. As can be seen, our pose estimation is accurate enough for object detection even though the assumption of how the data was obtained is highly violated (i.e., fixed position and pan-tilt unit versus freely floating quadrotor).

The second experiment was performed in order to test if the accumulated 3D point clouds are accurate enough for place recognition. We manually flew the quadrotor within an office

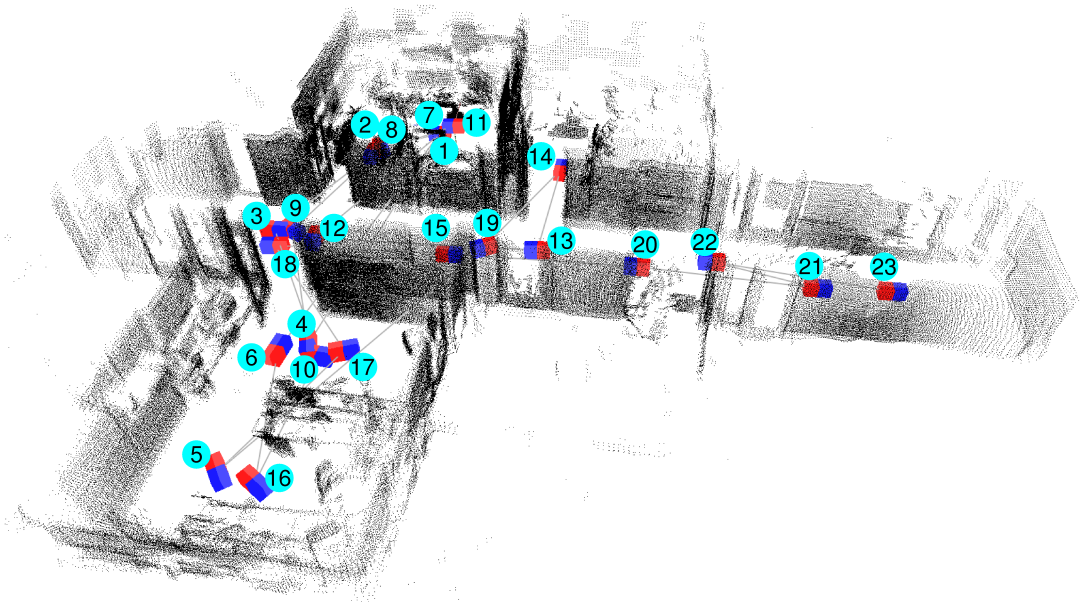

Figure 5.12: 3D map of our office environment acquired by the quadrotor. The rectangles labeled $1, \ldots, 23$ mark the individual locations where the quadrotor recorded a 3D scan by occasionally changing its altitude. The blue part of the rectangle reflects the front and the red part represents the rear of the scan position, respectively.

environment. Again, once in a while we occasionally changed the altitude while hovering around the same spot. We also turned the robot by 180 degrees in order to obtain a 360 degree scan. The whole data set consists of 23 scans, each recorded around one spot. The map obtained using our SLAM module is shown in Figure 5.12 whereas Figure 5.13 and Figure 5.14 show the 3D scans acquired at the individual locations.

To detect if two scans were recorded in the same area, we first extract normal-aligned radial features (NARF) [146] for each 3D measurement and match those against the features from each other scan. Since each NARF encodes a full 3D transformation we also estimate the relative transformation between the two scans. Based on an observation model of the laser scanner we now compute a similarity value between different scans, given the matched NARF's and the estimated transformation. Intuitively, the computed value reflects the confidence that both measurements were recorded in a local vicinity. Note that this is a brief description of the whole algorithm and we refer to [145, 72] for a detailed description.

Each scan-pair yields a confidence score between 0 (no similarity) and 1 (perfect match). The ground truth confusion matrix representing this values is shown in Figure 5.15 (top left) together with some examples of matches. The confusion matrix computed by our approach is plotted in Figure 5.16 (left). In both plots, white areas reflect a confidence of 0 whilst black areas represent the value 1. Dark cells not located at the diagonal describe estimated loop closures, i.e., different scans acquired in the same area similar to each other. Figure 5.16 (right) plots the recall rate, the number of true positives and the number of false negatives with respect to the maximum distance between individual scan. For example, we correctly recognized 39 out of 46 loop closures (i.e., recall rate of 0.75) for which the distance between the scan-pairs is at most $1.5\,\text{m}$ (as measured by our SLAM system). Due to the limited range of the laser scanner the overlap between individual scan drops rapidly the further away the scans are from each other. This effect can be seen in the plot starting from a distance of $2\,\text{m}$. The recall rates, however, are similar to those where data was obtained with wheeled robots equipped with a SICK LMS laser scanner mounted on a pan-tilt unit (see [145]).

As can be seen from these experiments, our SLAM module yields accurate results in indoor environments although the algorithm is carried out in 2D only.

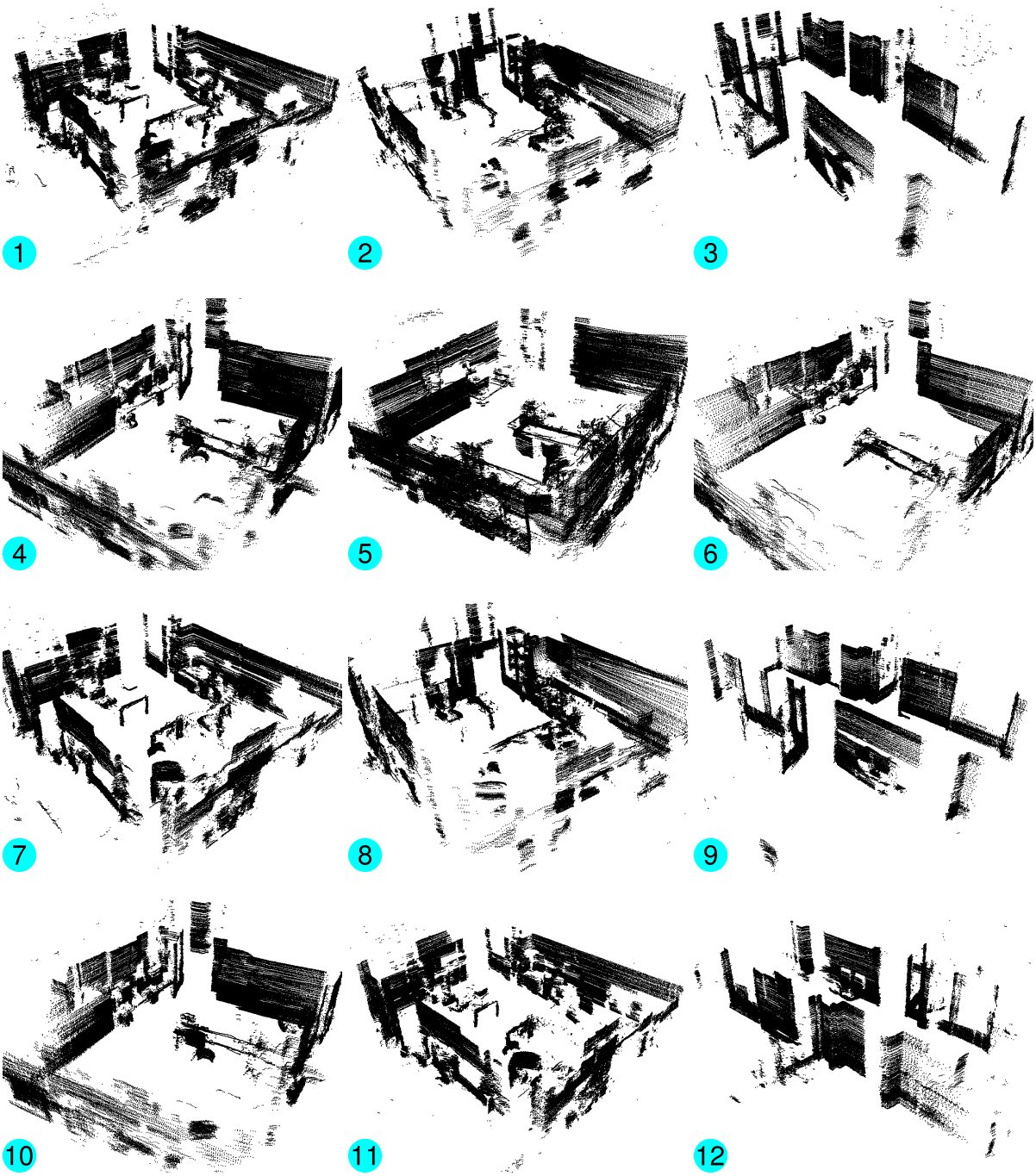

Figure 5.13: Individual 3D scan taken at positions 1 to 12 (see Figure 5.12).

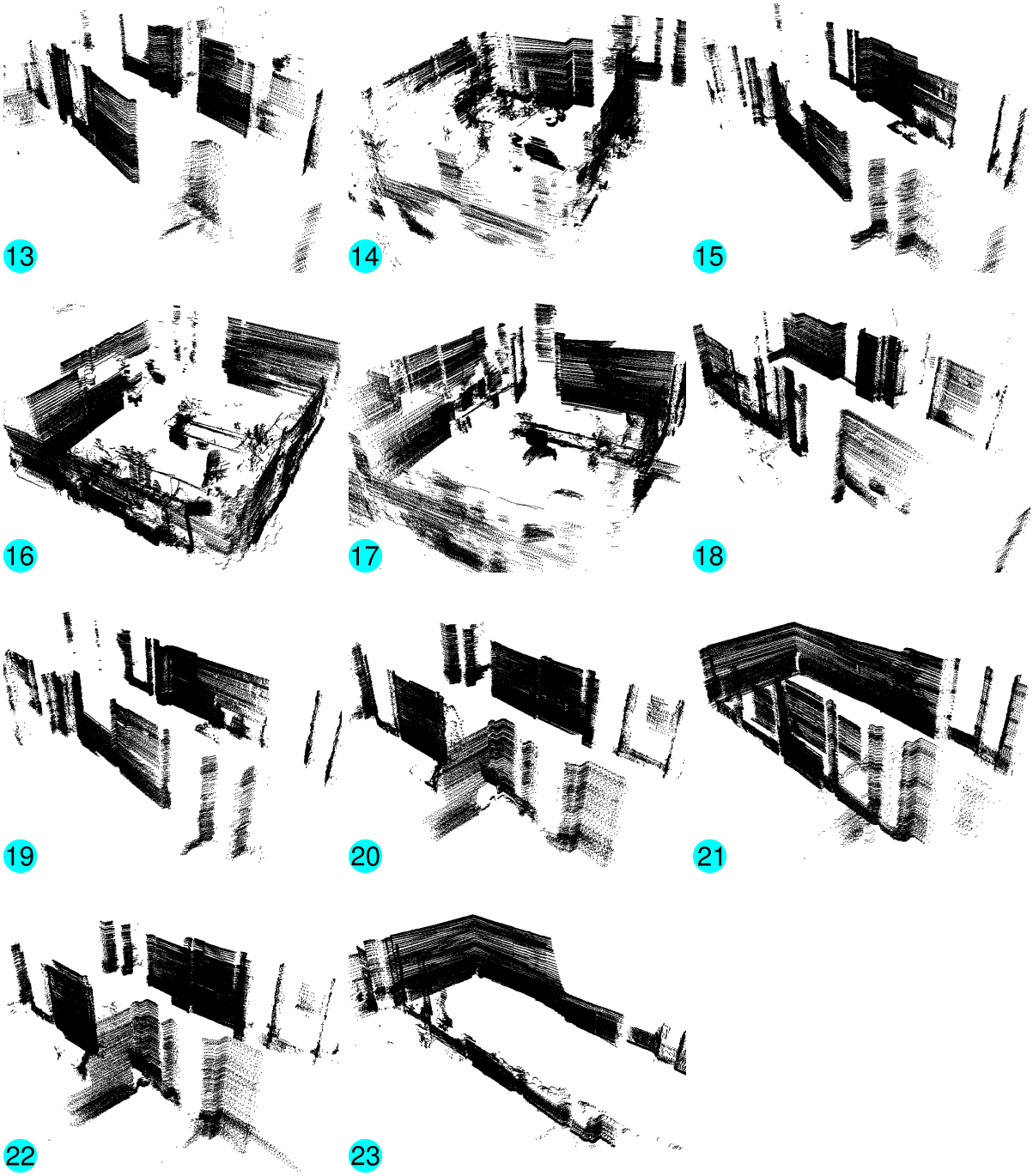

Figure 5.14: Individual 3D scan taken at positions 13 to 23 (see Figure 5.12).

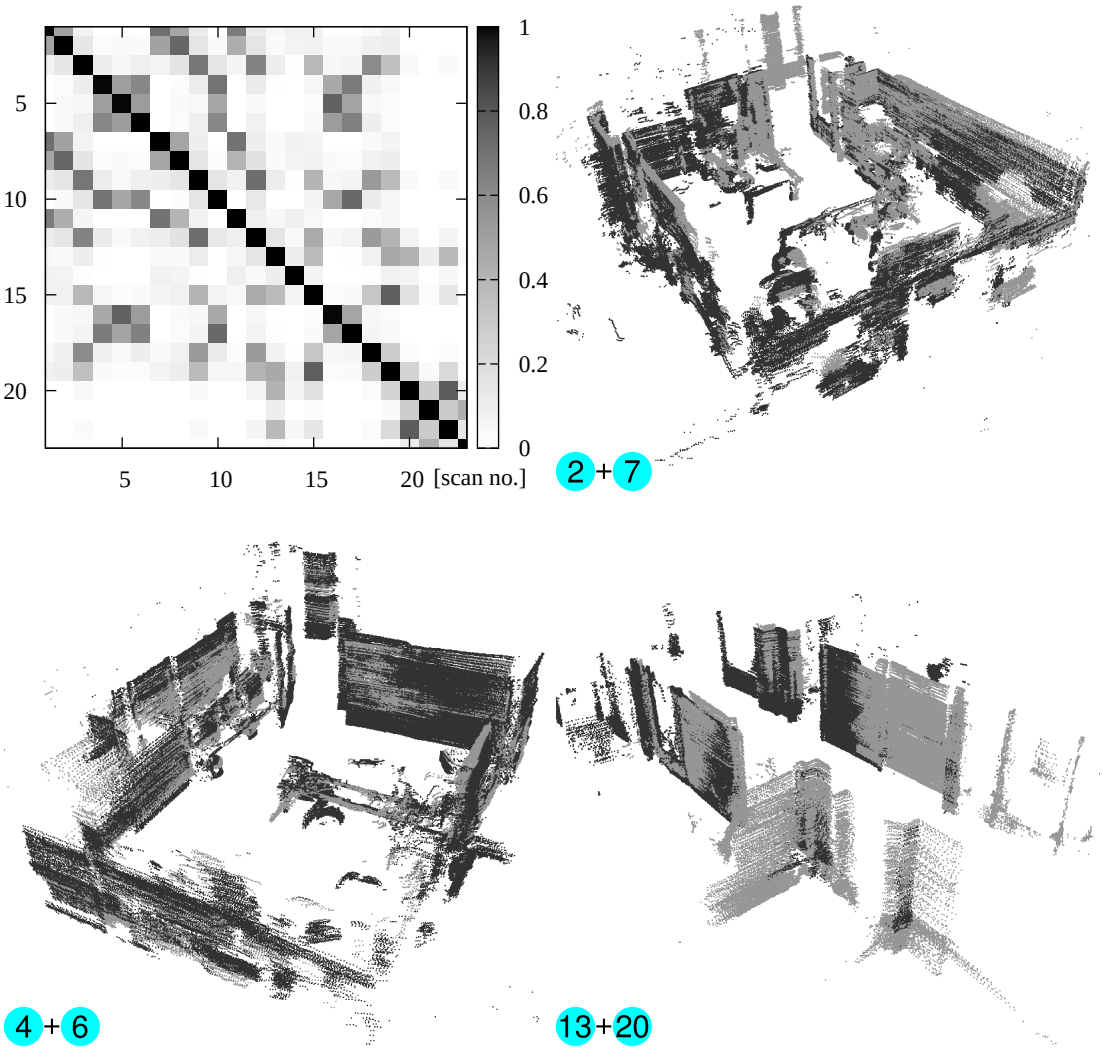

Figure 5.15: Ground truth confusion matrix for the scans $1, \ldots, 23$ (top left). The remaining images show examples of matches between scan 2 and 7 (top right), 4 and 6 (bottom left), and 13 and 20 (bottom right).

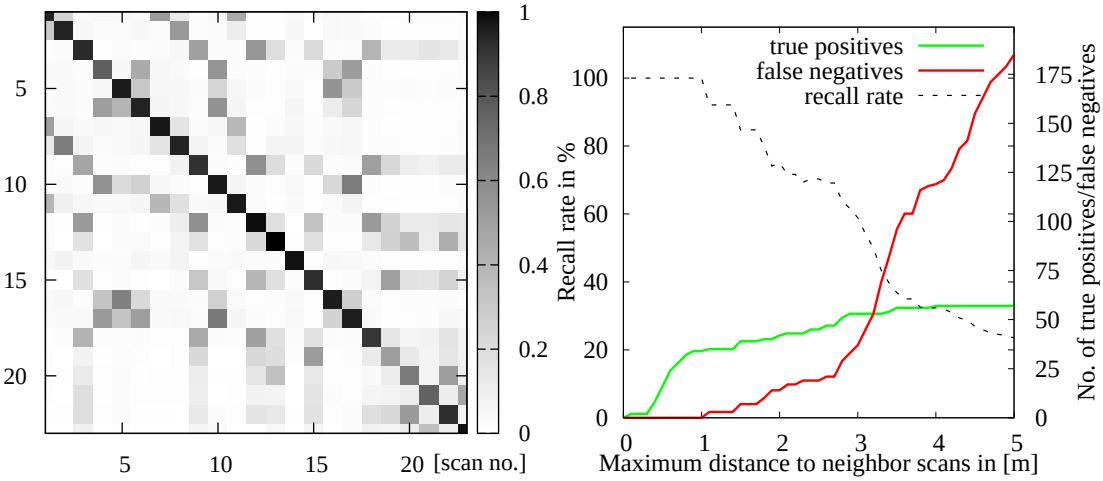

Figure 5.16: Computed confusion matrix (left) and the results obtained using our approach [145] for place recognition (right). Due to the limited range of the laser scanner the recall rate starts to drop rapidly for scans more than 2 m apart.

5.4.3 Multi-Level SLAM and Altitude Estimation

In the following, we show the typical behavior of our altitude estimation module. In this experiment, we let the robot fly autonomously in a typical office containing chairs, tables, and a high amount of clutter. The chairs have a height of 48 cm and the tables are arranged next to each other having a height of 77 cm. During this mission the system flew once over the chair and several times over the tables where it also flew a loop. Figure 5.17 (top images) shows an image of the office environment the robot operated in and a snapshot of our multi-level mapping system during this mission. As can be seen from this figure, our algorithm correctly detected the objects at corresponding levels. The estimated height of the chair was 48.6 cm± 2.7 cm and the estimated height of the tables was 74.9 cm± 2.8 cm, respectively. Each multi-level cell has an extend of 10 cm $\times$ 10 cm. The raw height measurement and the estimated height given our multi-level SLAM are shown in Figure 5.17 (bottom left) whereas the estimation of the different levels underneath are depicted in Figure 5.17 (bottom right). Note the loop closure at time 53. Here, two different realizations of the tables (level 2 and level 3) were merged into one. Intermediate snapshots of this experiment can also be seen in Figure 5.6 on page 93. We have also performed several tests by flying over different types of objects, including objects stacked on each other (for example, a box placed on top of a table). Throughout all experiments, we correctly detected the objects underneath the robot with an average altitude estimation error of 2.1 cm ± 1.3 cm. The outcomes of additional experiments can be found in [65].

5.4.4 Pose Control

Since the system is stabilized by independent controllers, we discuss the result of each individual controller separately. Again, all modules were extensively tested under real world conditions in over 50 live demonstrations and were active during the previous experiments as well. The following experiments are therefore included for completeness.

Altitude Control For testing the altitude control, we set the desired altitude to 1.50 m. In the beginning the vehicle was hovering over the ground. After enabling the stabilization the vehicle started climbing to the desired altitude. The desired height was kept by the vehicle up to an error of ± 12 cm. The results are shown in Figure 5.17. This experiment was performed whilst flying over different elevations. Throughout all experiments, the quadrotor is in general able to keep the desired altitude up to an average error of approximately ± 10 cm.

Yaw Control Similar to the experiment regarding the altitude, we ran an experiment to assess the behavior of the yaw control. In this test we set a desired yaw of 0° and once in a while, we turned the helicopter via the remote control. When the user released the remote control, the vehicle always returned back to its desired yaw with an error of $\pm 2^\circ$. Figure 5.18(a) plots the outcome of a typical run for yaw stabilization.

x, y **Control** Finally, we show an experiment for the pose stabilization only. The pose stability is strongly affected by the latency of the system (i.e., the time needed to calculate the command given the laser data). Although incremental motion estimation takes less than 5 ms in average (with a maximum of 15 ms) we have to deal with a latency of around 120 ms in average due to the wireless transmission and decoding of the wireless signal on the Gumstix processor. A typical run including autonomous pose stabilization is shown in Figure 5.18(b). Here, the quadrotor was set to keep the initial pose of $(0, 0)$ and once in a while, the user used the remote

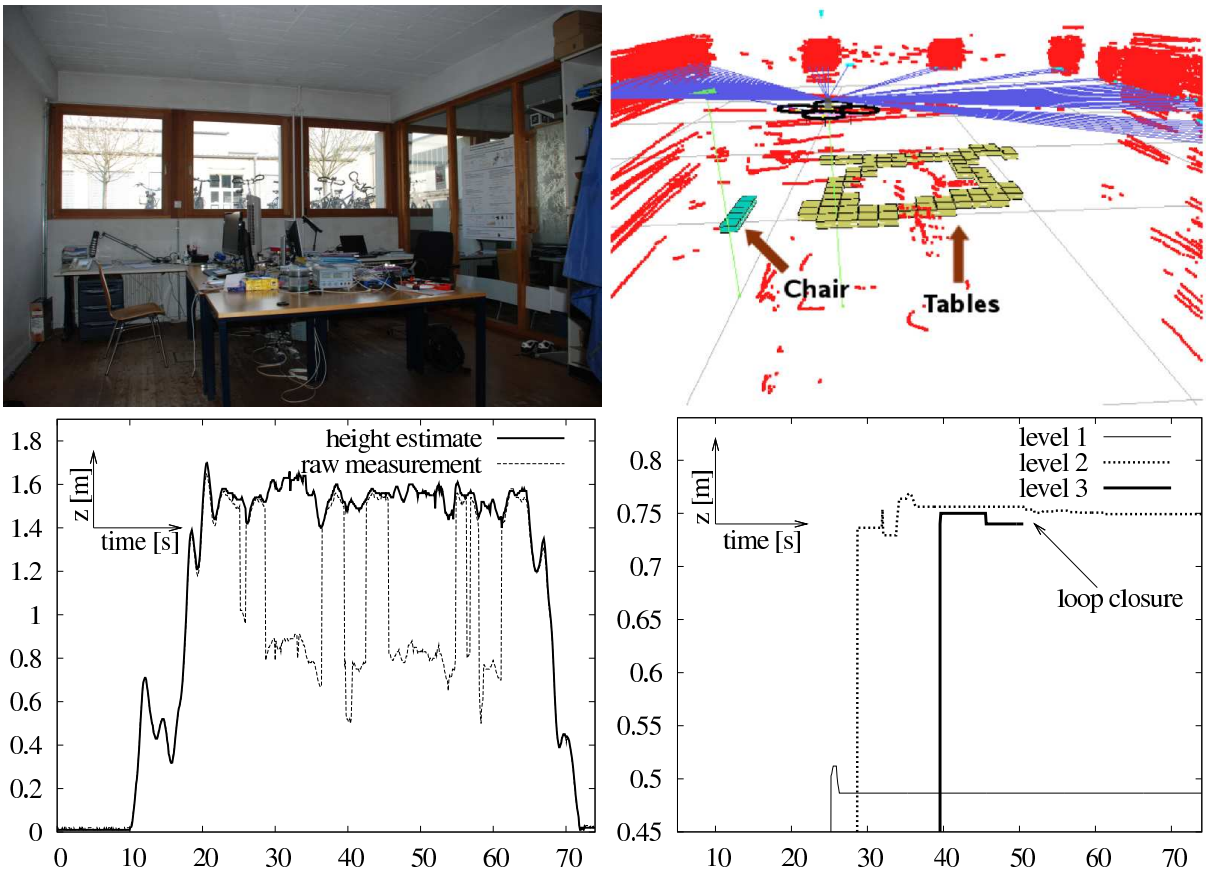

Figure 5.17: Estimation of the global height of the vehicle and the underneath floor level. Whenever the quadrotor moves over a new level, the innovation is used to determine a level transition. The estimate of the height of each level is refined whenever the robot reenters that particular level. Top left: The office environment our robot operated in. This image is recorded from a view point close to the one shown in the top right. The latter shows a visualization of our multi-level SLAM system during the mission. The cyan and dark yellow colored level corresponds to the chair and the tables detected underneath. The blue lines represent the current laser measurement not deflected by the mirror in the local reference frame of the quadrotor robot (black). Bottom Left: A plot showing the estimated altitude of the vehicle over time versus the raw measurement. The corresponding estimated levels are depicted in the bottom right plot. Here, level 1 is the chair, and the levels 2 and 3 reflect the individual tables. Note that Level 3 is merged with Level 2 after the loop closure at time index 53.

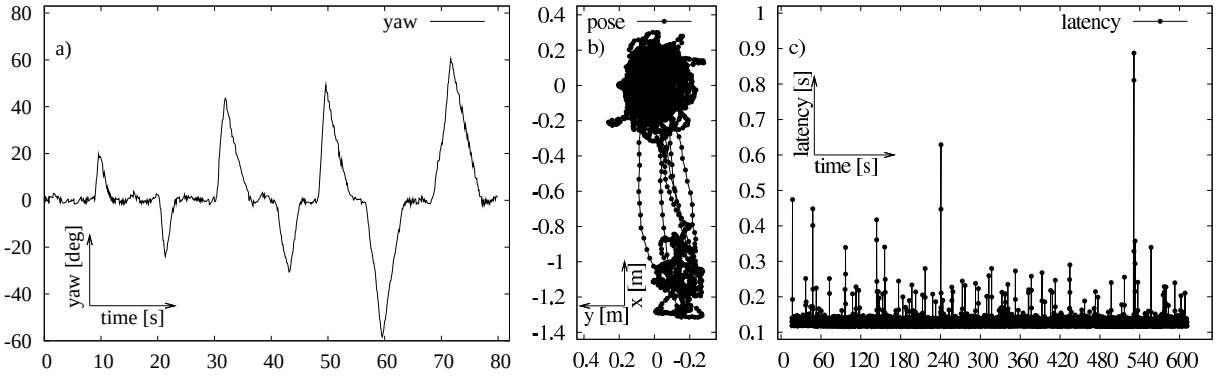

Figure 5.18: Experiments for the autonomous stabilization of yaw (a) and pose (b). During the yaw stabilization experiment, the quadrotor was required to rotate to $0°$, while the user manually turned the robot once in a while to a random orientation. Within the pose stability experiment the quadrotor was set to hover at $(0, 0)$, but was manually moved backwards once in a while and required to fly back to the initial pose autonomously. The latency of the system is shown in (c).

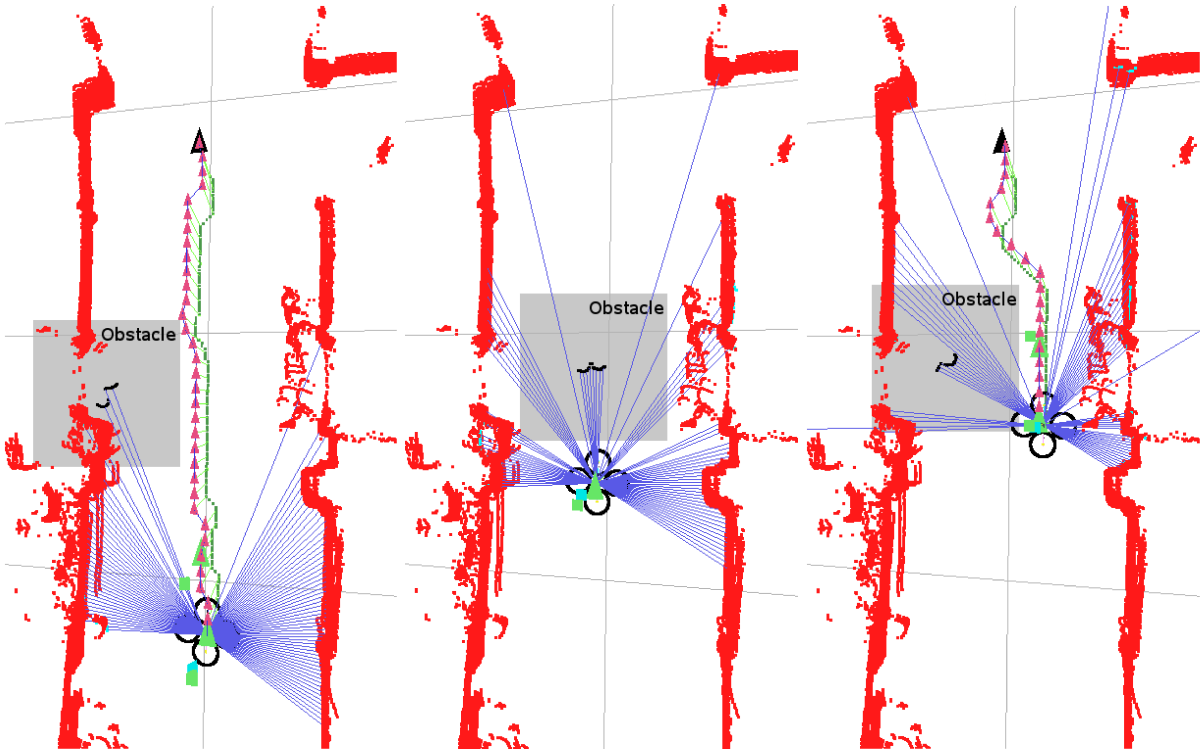

Figure 5.19: Experiment for path planning and dynamic obstacle avoidance. The quadrotor is given a goal point approximately 5 m in front of it. The goal point and the planned trajectory are shown in the left image. Here, the black triangle represents the final goal whereas the sequence of red triangles reflect the planned path. While the quadrotor approaches the desired goal location, a person enters the corridor and blocks the robot's path. The person (including the safety margin of 1.5 m) is visualized as a shaded box. Since the human is blocking the robot's path, there is no valid plan to the goal anymore. The quadrotor therefore hovers around the last valid way point as illustrated in the middle image. In the third image the person moved back, leaving the quadrotor enough space for a detour.

control to move the quadrotor around 1 m backwards. The quadrotor then autonomously moved back to the desired position. Depending on the latency in the system the pose oscillations are typically around ± 20 cm (around the desired location). The latency of the system from a typical experiment is depicted in Figure 5.18(c). With our current setup, the quadrotor is able to autonomously keep the desired pose up to a latency of approximately 350 ms. In this case, the oscillations are around ± 40 cm. However, we observed in several experiments a high risk of a crash with nearby walls when the latency grows beyond 400 ms over an extended period of time.

5.4.5 Path Planning and Obstacle Avoidance

In this section, we present an experiment demonstrating our algorithms for path planning and dynamic obstacle avoidance. The quadrotor was given a goal point approximately 5 m in front of it (i.e., along the x direction). Figure 5.19 (left) demonstrates this situation. The final goal $(x, y, z, \psi)^T$ is visualized by a black triangle and the planned trajectory is shown via a sequence of red triangles. In the beginning, a person was standing on the left (see the shaded area in Figure 5.19) entering the corridor and stopping in front of the quadrotor while the robot moved to its desired goal. Although only the upper part of the human legs are detected, the dynamic obstacle is enlarged by a safety margin of 1.5 m meters which is visualized by the shaded box. The situation where the human is entering the corridor is depicted in the left and middle image.

Figure 5.20: Top left: Fuel-cell prototype mounted on our quadrotor robot. The stack is build of six generators and provides up to 11.5 Watt of power. This is enough to power all on-board modules of the robot except the motors. The reactor for providing the necessary hydrogen is shown in the bottom left image. The right image depicts a snapshot of an experiment. Here, the author of this work was controlling the quadrotor while the hydrogen for the fuel-cell was provided by a supply line. The power generation of the fuel-cell while being cooled and dried by the wind present next to the propellers is shown in Figure 5.21.

In the latter one, the person is completely blocking the robot's path. In this case the quadrotor hovered around the last valid way point since there was no valid plan to the goal anymore. When the person moved to the left again, the quadrotor was able to follow a detour as shown in the right image of Figure 5.19.
The snapshots show the endpoints of the laser only. Although it looks like the quadrotor might have the space to fly around the person in the middle image, there is no valid plan since in the planning approach the quadrotor is modeled as a point. Consequently, each laser measurement is enlarged by the robots dimensions (see also Figure 5.8 on page 96).

5.4.6 On-Board Power Generation using a Fuel-Cell Prototype

Up to now, we used a fixed hardware setup and presented different software modules that allowed autonomous indoor flights. However, our system is robust enough even in the presence of additional payload. For the sake of completeness, we will therefore describe an experiment performed with an experimental fuel-cell in the next section. It was used to power the navigation system of the robot (i.e., all components except the motors).

In an attempt to test new power sources regarding usability in the context of small flying vehicles, we tested a fuel-cell prototype which was developed by the group of Robert Hahn at

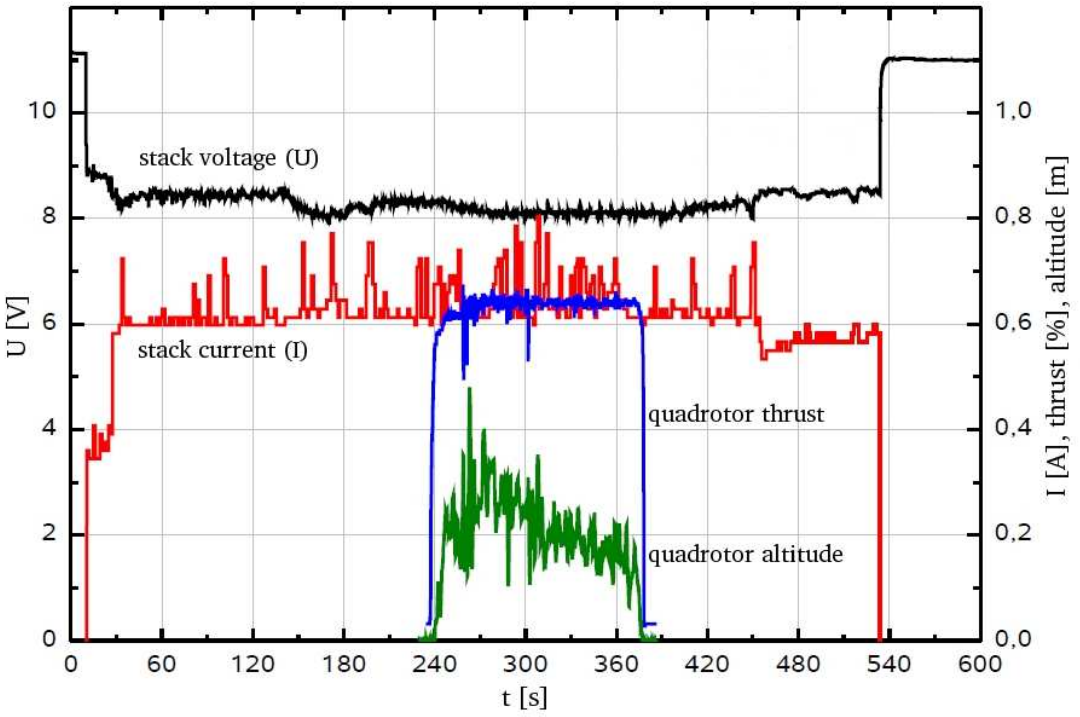

Figure 5.21: Outcome of the experiment shown in the previous figure. The quadrotor was flying between seconds 240 and 380. As can be seen from the plot, the voltage and the drained current did not drop during the whole mission,i.e., the fuel-cell reliably provided the necessary power for the on-board modules of the quadrotor.

TU-Berlin within the muFly project [108]. In cooperation with them, we tested their prototype for on-board power supply using our quadrotor. The overall fuel-cell is a stack built of six generators, each generating power at 1.5 V. The whole stack is able to provide up to 11.5 Watt of power and has a total weight of approximately 80 g. We used our quadrotor to test the effect of environmental conditions on the power generation. In detail, we tested if the fuel-cell is able to reliably provide the necessary power, given the fuel stack is constantly cooled and dried by the air that is present underneath or close to the propellers. Here, the fuel-cell provided power for all components of the quadrotor except the motors. The stack (fuel-cell) mounted to our quadrotor robot is shown in Figure 5.20 (top left). Figure 5.20 (right) shows an experiment where the necessary hydrogen was provided via a supply line. The outcome of this experiment is shown in Figure 5.21. As can be seen, even though the stack was constantly cooled by the thrust generated from the propeller, the supply voltage as well as the current did not drop (see seconds 240 up to 380). We furthermore tested on-board hydrogen generation for the fuel-cell using a reactor where 3 ml H_2O heated up to 60 degrees were combined with 0.8 g $NaBH_4$, and performed full autonomous indoor flights. A snapshot of a flight is shown in Figure 5.22. The whole video can be found on the Web [123]. The running reactor is also shown separately in Figure 5.20 (bottom left). Here, the total of 4 g of fuel were enough to power the on-board sensors up to 7 minutes. Although in the current configuration (80 g fuel stack, 6 g reactor, and 4 g fuel) the whole system is clearly outperformed by standard Lithium Polymer batteries, it shows that this technology can in principle be used in small flying vehicles. Note that this prototype is several orders of magnitude lighter than other current state-of-the-art fuel-cells. However, the current technology is still too heavy for small unmanned aerial vehicles (UAV's) but we are optimistic that there will be a new generation suitable for this size within the next decades.

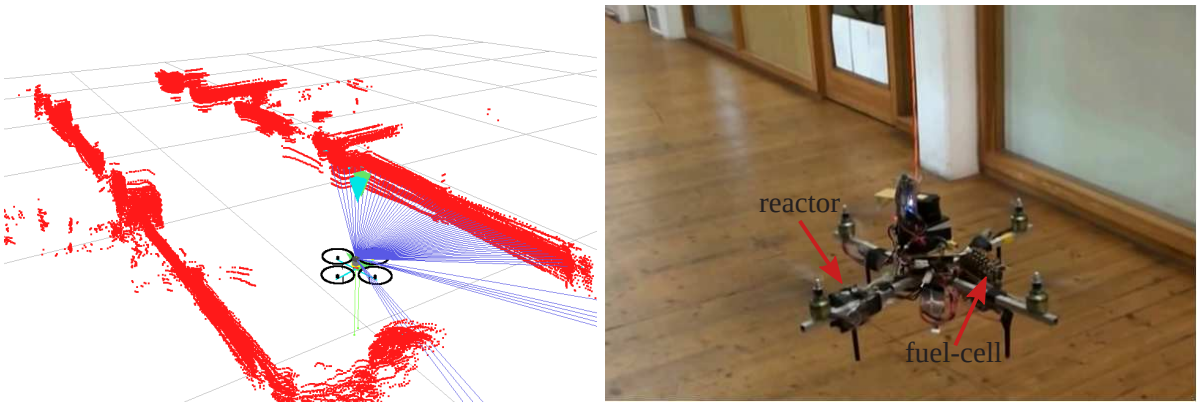

Figure 5.22: Snapshot from an autonomous indoor flight using the fuel-cell for powering the on-board modules. The whole video can be found on the web [123]. The left image depicts the outcome of our SLAM algorithm after flying the corridor back and forth. The right image displays the quadrotor at the same time. The location of the reactor and the fuel-cell are highlighted. Despite the presence of additional payload the robot reliably performed autonomous navigation.

5.5 Related Work

In the last decade, flying platforms received an increasing attention from the research community. Indeed, IEEE recently elected unmanned aerial vehicle technology to one of the top eleven technologies of the last decade [84]. However, one of the major reasons for the increased interest is the availability of enabling technology at low cost. Starting from electrical motors able to generate the desired thrust up to micro controllers which have sufficient processing power for on-board stabilization. In the context of helicopter-like UAV's, most authors focused on modeling and control of these vehicles in the beginning with a special focus on roll, pitch, and yaw stabilization [121, 147, 7, 8, 25, 43, 30, 80, 162]. The available payload is one of the key difficulties for autonomous flights. In other words, the more payload available, the more and better sensors can be carried. In the field of micro and small air vehicles only very limited sensing and processing power is available. Bouabdallah *et al.* [24] present a micro coaxial helicopter which is able to stabilize along roll and pitch. It is furthermore equipped with reactive obstacle avoidance using a miniature omnidirectional camera and eight laser pointers. Although lot of research has been done in context of learning and modeling the low level control, we also need capabilities to obtain the global pose to allow autonomous navigation. Hoffmann *et al.* [79] presented a model-based algorithm for autonomous flying with their STARMAC-quadrotor. Their system is able to fly autonomously in outdoor environments. Here, the IMU is used for stabilizing the individual axes and GPS measurements allow for autonomous outdoor flights. Ng and colleagues [111, 36, 2] have developed algorithms for learning controllers for autonomous helicopter navigation. Their approach allows helicopters to perform impressive and aerobatic maneuvers in outdoor environments, including flying upside down. In context of outdoor autonomous flying, Scherer *et al.* [132] describe an algorithm for flying fast among obstacles like buildings, trees and wires. They use a big helicopter which is able to carry a SICK laser scanner and a desktop computer and employ 3D grid maps for registering objects detected by the laser scanner. Subsequently, they use a two step planning mechanism for obstacle avoidance. The first level (called global planning) is a multi-resolution Laplacian planning algorithm and calculates the desired trajectory towards the goal. The trajectory planned by this system is locally adapted by a potential field-like algorithm in order to avoid obstacles.

Most of the work addressing navigation for UAV's is based on vision [73, 76, 12, 89, 4, 18, 90]. Templeton *et al.* [148] demonstrate how to use vision for outdoor terrain mapping

and autonomous landing. In their work, they use geo-referenced images from a single camera and utilize a recursive multi-frame planar parallax algorithm [57] for terrain mapping. Landing areas are detected by scoring different elevations of an area candidate within the estimated map. Tournier *et al.* [155] and Bourquardez *et al.* [26] use monocular vision to estimate and stabilize the current pose of a quadrotor. Whereas Tournier and colleagues estimate the current state based on Moire-Patterns, Bourquardez *et al.* use zero and first order moments extracted from images for control. Johnson *et al.* [85] use vision in combination with ultrasound for autonomous flights in corridor-like environments. They use a Sobel filter to detect edges and extract line features from the image. Parallel lines are assumed to represent a bounding box of the corridor which is used for autonomous flying. The same principle in combination with a similar approach for edge detection was also used later by Celik *et al.* [31].

Thrun *et al.* [150] used a remotely controlled helicopter to learn large-scale outdoor 3D models. They employ a downwards facing SICK laser scanner and align the measurements using scan-matching in order to generate three-dimensional maps. However, this information is not used for autonomous control. Steder *et al.* [143, 142] also employ a downwards facing camera for building accurate maps of the environment. They track SURF features over a sequence of images and perform graph-based optimization whenever a loop-closure has been detected. Blösch *et al.* [18] also use a down-looking monocular camera. They learn an accurate model of their quadrotor which allows autonomous indoor flights. Cheviron *et al.* [33] combine information from an IMU and a down-looking camera to estimate the current pose and velocity of the quadrotor during manually controlled flights.

Ahrens *et al.* [5] use an external tracking system to estimate the current state of the flying robot. Here, the robot is equipped with visual markers which are accurately tracked via Vicon camera tracking system [159]. Such a system is also used by Mellinger and colleagues for performing aggressive maneuvers [102]. Huang *et al.* [83] developed a detailed model of their STARMAC II quadrotor in order to fly difficult maneuvers, while Purwin *et al.* [122] use learning by iteratively solving a linear least squares problem to achieve a similar performance.

There has also been some work addressing the navigation of flying vehicles in indoor environments in absence of GPS. Several authors used vision to control or assist the control of an indoor quadrotor [85, 89, 5]. Roberts *et al.* [129] used ultrasound sensors for controlling a flying vehicle in a structured testing environment, while He *et al.* [77] presented a system for navigating a small-size quadrotor without GPS using laser. Here, the pose of the vehicle is estimated by an unscented Kalman filter. Whenever the robot has to reach a given location, a path which ensures a good observation density is computed from a predefined map. These highly dense observations minimize the risk of localization failures. Achtelika *et al.* [3] developed an indoor autonomous quadrotor equipped with a laser range scanner and cameras enabling autonomous hovering in a constraint indoor environment. The work closest (although orthogonal) to ours is a recent work of Bachrach *et al.* [13]. Here, the authors present a system for performing autonomous exploration and map acquisition in indoor environments. They extend the 2D robot navigation toolkit CARMEN [130] by adding a Rao-Blackwellized particle filter for SLAM and an algorithm for frontier-based autonomous exploration. However, they do not provide localization, map optimization, obstacle avoidance or mutli-level SLAM. In contrast to that, we utilize the more robust graph-based SLAM algorithm in our system allowing for map optimization and thus correcting previous poses as well. In more detail, graph-based SLAM addresses the full SLAM problem while other filtering techniques address the on-line variant of SLAM only. We also presented our algorithm for estimating the altitude of the surface underlying the robot. This enables a quadrotor equipped with our system to fly over surfaces with heights that are piecewise constant.

5.6 Conclusion

We presented a navigation system for autonomous indoor flying utilizing an open-hardware quadrotor platform. We described a complete navigation solution that approaches the different aspects of incremental motion estimation, localization, (multi-level) mapping, path-planning, obstacle avoidance, height estimation, and control. Since we do not rely on special characteristics of the flying platform like the system dynamics, we believe that our system can easily be adapted to different flying vehicles. All modules in our system run on-line. However, due to the relatively high computational cost of some algorithms only a part of the software runs on-board on the Gumstix processor whereas the other part runs off-board on a laptop computer. Preliminary tests make us confident that the whole system will run on-board using the next generation of embedded computers based on the Intel Atom processor. We provided a wide range of extensive experiments and videos [123] that highlight the effectiveness of our system. Although we assume structured indoor environments, our mapping algorithm provides accurate 3D results which is also suitable for object recognition as well as place recognition.

Chapter 6

Activity-Based Indoor Mapping and Estimation of Human Trajectories

We present a novel approach to incrementally determining the trajectory of a person in 3D utilizing human motion and activity in real-time. In our algorithm, we estimate the motions and activities of the user given the data obtained from a motion capture suit which is equipped with several inertial measurement units (IMUs). These activities include walking up and down staircases as well as door opening and closing events. We interpret the first two types of activities as motion constraints and door handling events as landmark detections in a graph-based simultaneous localization and mapping (SLAM) framework. Since we cannot distinguish between individual doors, we employ a multi-hypothesis tracking approach on top of the SLAM procedure to deal with the high data-association uncertainty. As a result, we are able to accurately and robustly recover the trajectory of the person. Additionally, we present an approach to build approximate maps of structured environments using this type of information. We take advantage of the fact that people traverse free space and that doors separate rooms to recover the geometrical and the topological structure of the environment after the graph optimization. We evaluate our approach in several experiments carried out by different humans in various environments.

In the previous chapter we described the navigation system enabling a quadrotor to fly autonomous indoors. The world was encoded using a graph structure. This allowed us to use our tree network optimization algorithm (see Chapter 4) for finding the most likely map, given the observations. This data structure is also used in the following work allowing us to simultaneously localize a human and map the indoor environment based on human activities only.

The problem of localizing and tracking people has recently received substantial attention in the robotics community as knowledge about the current position of its users can help a robot to improve its services. Especially in emergency situations, like after earthquakes or during fire fighting, the knowledge about the location of people can greatly support search and rescue missions. Consider, for example, firefighters in a building enclosed by smoke and fire. If a map of the environment can be constructed while the firefighters are within the building, an operator

Figure 6.1: Left: The author wearing the Xsens MVN data suit. Right: Typical data obtained from the suit when a person opens a door and enters a room.

or automated system can re-route the people to the exit in case of an emergency. Alternatively, one can use the map of the environment to more intelligently coordinate the actions of the rescue workers to more efficiently search the environment for potential victims and contemporaneously reduce the time the rescue workers are exposed to potential threats and hazards.

In this chapter, we consider the problem of simultaneously estimating the trajectory of a person walking through an indoor environment and the map of the environment based on data obtained with an Xsens MVN data suit [165] by treating activities as landmarks. The MVN data suit records full body postures of a human, by using a set of inertial measurement units (IMUs) and a biomechanical human model. Figure 6.1 (left) shows the author wearing the MVN data suit and Figure 6.1 (right) depicts typical data obtained when a person opens a door.

Figure 6.2 (left) depicts the raw odometry estimated by the suit when walking in a typical university building. The outcome of our proposed approach is shown in Figure 6.2 (right). To correct odometry errors, our approach applies supervised learning for classification of different types of activities such as stair climbing and door handling. It then utilizes the learned classifiers to detect doors and stairs and applies a graph-based formulation of the SLAM problem to recover the full 3D trajectory of the person. In this formulation, the odometry estimated by the IMUs and the estimated heights of the steps are regarded as (chain-)links between detected doors, which are the landmarks of our system. To deal with the high data association uncertainty in the landmark detection, our algorithm applies a multi-hypothesis tracking scheme. After calculating the path of the person, our algorithm renders a map containing the individual stairs, the estimated doors, and approximate locations of walls.

This chapter is structured as follows. First, we describe the hardware system in the next section. Subsequently, we present our algorithms for learning door handling events and detecting stair steps. Section 6.3 introduces the multi-hypothesis tracking technique for sensors providing only positive feedback and especially the expressions needed to calculate the probabilities of individual world hypotheses. In Section 6.4, we describe how we detect potential loop closure candidates. This is followed by the description of our overall system in Section 6.6. In Section 6.7 we present our experimental results based on real data recorded with people walking inside various buildings. The experimental section includes trajectories covering single as well as multiple floor levels. We furthermore present our results on approximate mapping and compare the estimated maps with floor plans of the same building. Finally, we discuss related work in Section 6.8 and conclude our presented work in Section 6.9.

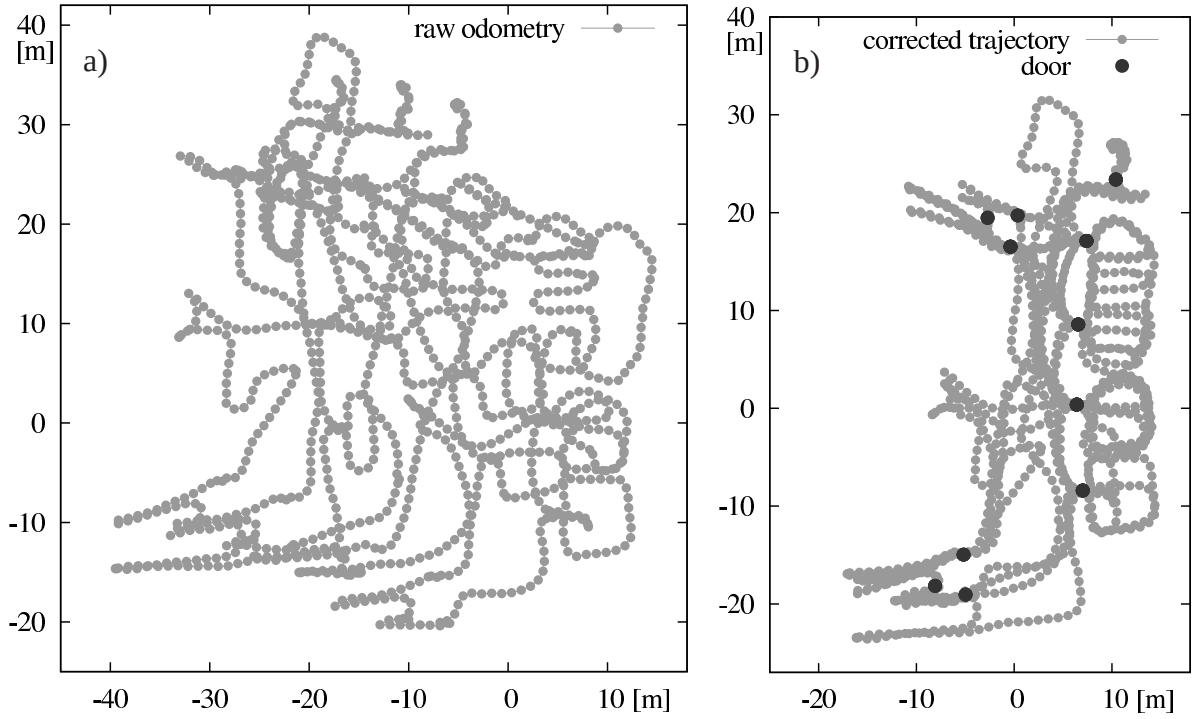

Figure 6.2: Our approach uses human motions to detect door handling events. These events are used as landmarks in a graph-based formulation of the SLAM problem for recovering the full trajectory of the person. The raw odometry data provided by the data suit is shown in (a). The corrected trajectory after applying our approach is visualized in (b).

6.1 Hardware Architecture

The Xsens MVN data suit used within this work is shown in Figure 6.3. It has been used with the software *MVN Studio 2.6* from Xsens and has the following properties:

- 17 hardware IMU's
- 23 IMU's in total (hardware and software emulated)
- measurements up to 120Hz for each IMU consisting of
- pose, orientation, velocity, and acceleration

The 17 hardware IMUs are visualized as blue circles and blue stars in Figure 6.3, depending if they are localized on the back (circles) or in the front (star). The software, however, emulates additional six sensors by interpolation. The pink circles indicate the location of these emulated sensors. The most important ones are also labeled with names reflecting their position. In total, the Xsens software processes the raw data and we get filtered measurements from 23 inertial measurement units at a frequency up to 120Hz. Due to an underlying human biomechanical model and the corresponding kinematic chain, the Xsens software calculates a full 6D position (i.e., $(x, y, z, \phi, \theta, \psi)^T$) for each of the sensors. Additionally, we obtain filtered velocity and acceleration estimates. However, the individual IMUs are affected by magnetic disturbances in the environment, since parts of the IMUs orientation estimation is based on the measurement of the earth-magnetic field.

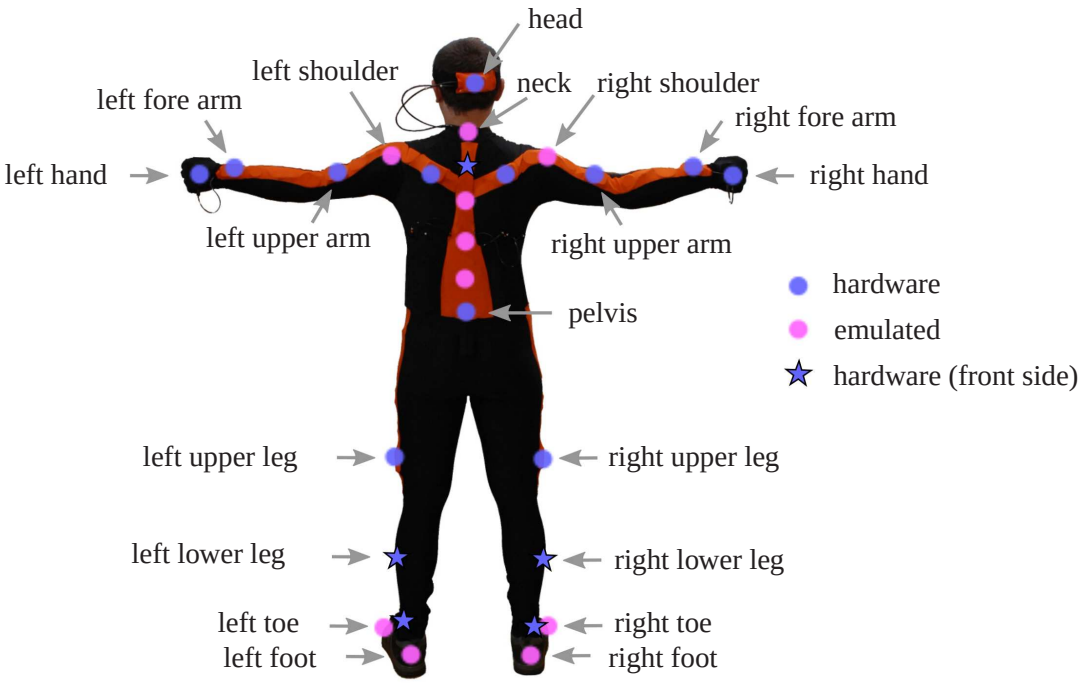

Figure 6.3: The Xsens MVN data suit is equipped with 17 MTi IMUs. Together with an underlying human body model a total of 23 IMUs are emulated. The data provided for each of these sensors includes the position (x, y, z), orientation (ϕ, θ, ψ), acceleration, and velocity. The blue circles highlight the position of the hardware IMUs visible from the back, whereas blue stars reflect the hardware IMUs located in front. The pink circles show the position of additional emulated IMUs.

6.2 Feature Detection

The MVN software filters the raw data of the IMUs in the data suit and estimates an odometry of the body segments consisting of the (filtered) 6D pose, velocity, and acceleration. A dead reckoning estimate of the trajectory typically leads to an inconsistent map due to the accumulation of small errors over time as shown in Figure 6.2 (left). Therefore, we need to keep track of other specific events or features. Without this additional information we cannot detect loop closures and thus cannot correct the raw odometry from the data suit.

Within this work, we restrict ourselves to structured environments such as office buildings. To allow us to correct the odometry within such buildings, we propose to use information about human activities as landmarks. We extract two different types of activities: *opening or closing of a door* and *walking up or going down a stair*. We use motion templates to detect door opening or closing events and a neural network to detect steps. In the next sections, we will briefly describe both approaches.

6.2.1 Door Handling Events

To learn the typical motion used for handling a door we use motion templates (MT) as proposed by Müller *et al.* [109]. The key idea of this work is to use simple Boolean features like *right hand is above head* to create more expressive features (motion templates) by combining the simple ones. Given n_f of those features and a motion sequence of length K, this leads to a matrix of size $n_f \times K$. Each entry of this matrix is either 1 or 0 indicating this feature being active or not at the specific time and that the sequence length K can in general be different for

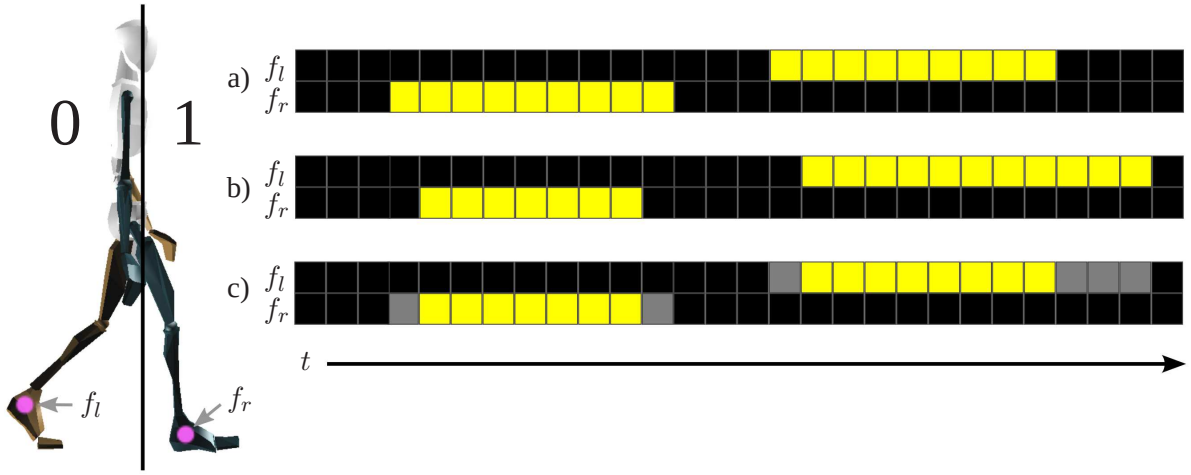

Figure 6.4: A synthetic example: Given two examples (a) and (b) of the same motion *walking*. The features f_l, f_r are 1 (yellow) iff the left/right foot is in front of the body and 0 otherwise. The resulting merged template is depicted in (c). Here, gray areas indicate the value 0.5, meaning *don't care*. Intuitively, the matrix can be interpreted as: *feet parallel, right foot in front, feet parallel, left foot in front, feet parallel*.

each motion sequence. Consider for example two features $f_l,\ f_r$ with f_l indicating the left foot being in front of the body and f_r being 1 if and only if the right foot is in front of the body. Given this set of features, a typical walking template for two different sequences of the same length look likes Figure 6.4 (a) and (b). If we generalize to a common motion template (also called *class template*) given the two examples, we would learn a motion sequence as shown in Figure 6.4 (c). Here, black and yellow cells reflect the value 0 and 1 respectively. However, we obtain a new value 0.5 visualized by the gray shaded boxes. This value represents the flag *don't care*. The value 0.5 is obtained since exactly one feature at this time is 0 and the other is 1. In the following, we will briefly describe the algorithm for learning a class template $\mathcal{C}_\mathcal{A}$ for a single activity $\mathcal{A}$ from a set of training examples $\mathcal{D}$. The algorithm for learning a class template for a single activity can be summarized through the following steps (see also Algorithm 5):

1. Calculate the motion templates for all examples of this activity.
2. Take one of the motion templates, call it reference template, and align all remaining to this one using dynamic time warping [124]. This procedure ensures that all other templates have now the same length as the reference template.
3. Compute a new template as the average of all and store it.
4. Repeat the previous two steps for each motion template being exactly once the reference template.
5. Replace the training data by the outcome of the calculated templates from the previous step.
6. Repeat the whole process until no major difference between the calculated templates exists.

Note that the averaging of the templates includes more complicated steps, but we refer to the original work of Müller *et al.* [109] for more details about learning a motion template.

Algorithm 5 Learn Class Template

Input: $\mathcal{D} = (D_1, \ldots, D_n)$ // the set of training examples for activity $\mathcal{A}$
Output: $\mathcal{C}_\mathcal{A}$ // class template for activity $\mathcal{A}$

$\mathcal{M}_0 = (M_1, \ldots, M_n) =$ calculateMotionTemplates$(\mathcal{D})$ // initial motion templates.
$t = 0$ // iteration
repeat
  $i = i + 1$
  for $i = 1, \ldots, n$ **do**
    $\mathcal{T}_i = \mathcal{M}_{t-1}[i]$ // current reference template
    for $j = 1, \ldots, n; j \neq i$ **do**
      $\mathcal{T}_j = \mathcal{M}_{t-1}[j]$.alignToReference$(\mathcal{T}_i)$
    end for
    $\mathcal{M}_t = \mathcal{M}_t \cup$ averageTemplates$(\mathcal{T}_1, \ldots, \mathcal{T}_n)$
  end for
until (differenceBetweenTemplatesIn$(\mathcal{M}_t) < \epsilon$) || $(t > t_{\text{max}})$
$\mathcal{C}_\mathcal{A} =$ averageTemplates$(\mathcal{M}_t)$
return $\mathcal{C}_\mathcal{A}$

Now, given the learned class template for each activity and a new motion sequence, we can calculate a similarity between both. To do so, we compute a motion template of the actual sequence and align it to each class template utilizing dynamic time warping. We furthermore compute a distance score for each pair of templates. This score varies between 0 and 1. Intuitively, the value 0 reflects a perfect match whereas a 1 indicates a disparity for each feature at each time between both templates. However, if this score is below a threshold τ, the actual motion sequence is said to belong to the same motion class as the class template.

Since we are only interested in the motion used for handling a door with either the left or the right hand we use features based on the pose and velocity of the hands only. More precisely, we use a set of features describing whether the hand is at the level of the door handle, whether it is raising, hold still or lowered, and finally whether the hand is moving towards the body or away from it. An example of such a sequence is visualized in Figure 6.5. It shows a typical motion while the user is opening a door (pulling towards him) using his right hand. The figures labeled (a) trough (d) show intermediate snapshots of the motion at the highlighted time index. In detail, Figure 6.5 (a) shows the initial situation where the user is approaching the door's handle using his right hand. In this case, the pattern in the features $1, 2,$ and 3 reflect the vertical position of the right hand, whereas features $7, 8,$ and 9 describe the horizontal motion of the hand away from the body towards the handle. The explicit handling of the door is visualized in (b). The corresponding pattern in the vertical velocity space (features $4, \ldots, 6$) reflect the situation where the hand's vertical velocity is close to zero (since the user touches the handle) followed by pushing the handle downwards in order to open the door. Subsequently, Figure 6.5 (c) shows the part of the motion where the user pulls the door towards him. This can be seen in the features $7, \ldots, 9$. Finally, the user releases the door handle, resulting in a moving the hand downwards again, as visualized in (d) through the features $1, \ldots, 3$.

We learned the template for the activity *handling a door*, which consists of the four subclasses *open left, close left, open right, close right*, using 10 examples from a training data set for each subclass. Based on a second validation data set, we selected the threshold $\tau = 0.25$ for detecting the motion. Intuitively, we require a match in at least 75% of the whole sequence between the learned template and the actual sequence. Using this threshold, we did not encounter

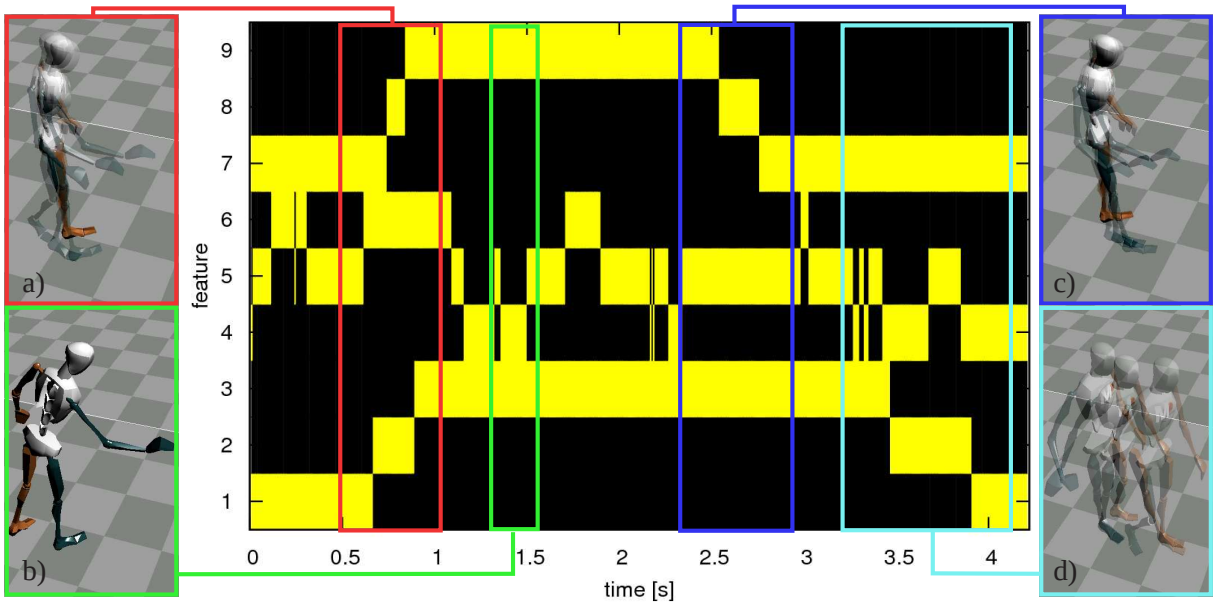

Figure 6.5: Motion Template for a door opening sequence (middle image). Yellow and black blocks reflect the corresponding feature being 1 and 0 respectively. The first three features describe if the right hand is at the level of a typical door handle. Subsequently, the next three features describe if the right hand is being lowered, hold still, or raised. Finally the last three features describe if the right hand is moving towards the body or way from it. The highlighted parts are visualized by the images located at the left and at the right hand side of the motion template. (a) approaching the door. (b) handling the door. (c) opening the door (pull) and (d) releasing the door handle.

any *false positives* on the validation data set. Within this process, we used data recorded by three subjects. The motion of two subjects was used for training, whereas the motion of the third one was used for validation. Although the features used for detecting a door are quite simple, we can reliably detect the point in time when the door handle was touched within 1.5 seconds of the true event (we evaluated this using manually labeled ground truth). Therefore, we can use the pose of the hand as an approximation of the location of the door.

Although one could utilize features based on the geometry of the feet during the motion sequence as well, we realized in our experiments, that humans have a very high disparity in the walking pattern when opening a door. Either the human stopped in front of the door, walked constantly while opening it, moved forth and back to avoid the door and so on. One could still utilize this information but this would lead to a separation of the classes into additional subclasses based on the number of walking patterns. However, our intention was to use a small set of simple features.

6.2.2 Stair Detection

To be able to reconstruct 3D trajectories within buildings, it is inevitable to detect vertical movements of the user. Due to the high uncertainty in the height estimate of IMUs, the manufacturer's software assumes an environment consisting of a single floor. When walking up or down a staircase, the software "snaps" the human to the ground as indicated by Figure 6.6. Therefore, one needs additional means for determining changes in the z coordinate. In our approach, we achieve this by identifying stair stepping motions carried out whenever the user walks up or down staircases. In principle, we could have employed the same motion template approach as for the door handling events. However, motion templates are especially useful for detecting complex activities (like door opening) at a coarse time resolution. In practical experiments we found that during typical stair-climbing people need approximately 0.5 seconds for

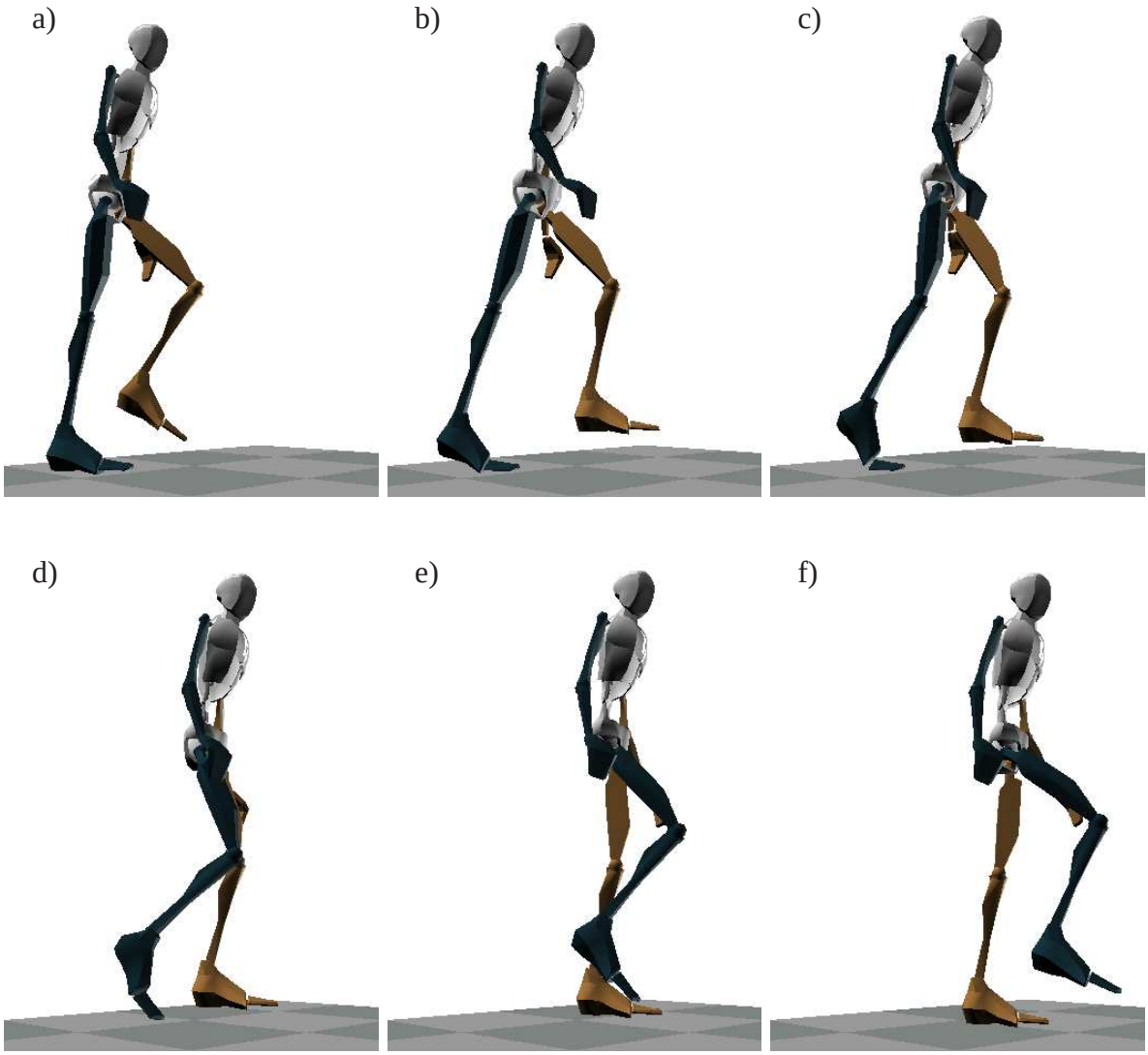

Figure 6.6: Typical data obtained from the data suit while climbing up a stair. The left (brown) foot is placed on the next step in (a), whereas the subject then moves his right (dark blue) foot onto the next step. The images labeled (b)-(f) show how the software "snaps" the human to the ground while climbing up the next step of the staircase.

each stair so that the motion templates described above, which detect doors with an accuracy of 1.5 seconds, were not accurate enough to exactly determine the point in time when the foot is placed onto a stair. Unfortunately, increasing the time resolution of the MT accordingly leads to a high computational complexity due to the dynamic time warping. We therefore developed an efficient and temporal substantially more accurate classifier for detecting the individual stairs based on neural networks.

The goal of the following approach is to detect *stair* events, consisting of two subclasses namely *stair up* and *stair down*. To achieve this, our method employs a sliding window consisting of 5 frames that correspond to 40.7 milliseconds. Within this window, we extract features from the suit's data. In more detail, we use the relative position of the feet and the toes as well as the minimum and maximum acceleration resulting in a total of 19 input features. We trained the neural network using manually labeled training data employing SNNS [167] and RProp [128] as learning functions. The neural network consists of 21 nodes in the first layer (see Table 6.1), 12 nodes in the hidden layer, and 3 nodes in the output layer. The latter nodes represent the three classes "step up", "step down", and "other", whereas the amount of neurons in the hidden layer

F. No.	Value	F. No.	Value
1	RF.z_t - LF.z_t	**12**	$\min$(RF.$z_{t-2:t+2}$, RT.$z_{t-2:t+2}$, LF.$z_{t-2:t+2}$, LT.$z_{t-2:t+2}$)
2	RF.z_t - LT.z_t	**13**	$\max$(RF.$z_{t-2:t+2}$, RT.$z_{t-2:t+2}$, LF.$z_{t-2:t+2}$, LT.$z_{t-2:t+2}$)
3	RT.z_t - LF.z_t	**14**	$\min$(RF.Acc.$x_{t-2:t+2}$, RT.Acc.$x_{t-2:t+2}$)
4	RT.z_t - LT.z_t	**15**	$\max$(RF.Acc.$x_{t-2:t+2}$, RT.Acc.$x_{t-2:t+2}$)
5	RF.Acc.x_t	**16**	$\min$(RF.Acc.$z_{t-2:t+2}$, RT.Acc.$z_{t-2:t+2}$)
6	RF.Acc.y_t	**17**	$\max$(RF.Acc.$z_{t-2:t+2}$, RT.Acc.$z_{t-2:t+2}$)
7	RF.Acc.z_t	**18**	$\min$(LF.Acc.$x_{t-2:t+2}$, LT.Acc.$x_{t-2:t+2}$)
8	LF.Acc.x_t	**19**	$\max$(LF.Acc.$x_{t-2:t+2}$, LT.Acc.$x_{t-2:t+2}$)
9	LF.Acc.y_t	**20**	$\min$(LF.Acc.$z_{t-2:t+2}$, LT.Acc.$z_{t-2:t+2}$)
10	LF.Acc.z_t	**21**	$\max$(LF.Acc.$z_{t-2:t+2}$, LT.Acc.$z_{t-2:t+2}$)
11	$\|$(RT.x_t-LT.x_t)$\|_2$		

Table 6.1: Features employed in the neural network for step detection. Within this table "F. No". is short for "Feature Number". Subsequently, "RF", "RT", "LF", and "LT" is short for "Right Foot", "Right Toe", "Left Foot", and "Left Toe". Finally, "Acc" is short for "Acceleration" and $(t-2:t+2)$ denotes a window of 5 frames.

was chosen as $(21+3)/2$. The training data was recorded by a person walking up and down two different staircases twice and contains a total of 56 stair events, covering slightly more than two minutes. Once our predictor has detected a stair event, we estimate the height of each stair, by calculating the difference between the two feet along the z-axis given the pose estimates obtained from the data suit. Using this approach, we are able to detect step events with an error of 1.5 frames (≈ 12 ms) with respect to a manually labeled ground truth.

Note that one could employ the neural network approach also for detecting door handling events. We tested this in several experiments using the same data sets also used for the motion template approach. In all runs, the recall rate using the neural network was around 60%. However, the worst recall rate using motion templates was approximately 88% as will be shown in the experimental section.

Up to now, we are able to detect when the user climbed up or down a staircase, and employing the motion templates, we are able to detect when the user touched *a* door. However, we do not possess any information of which door was handled. We therefore have to take care of possible data associations, which we deal with by employing a multi-hypothesis-tracker as described in the next section.

6.3 Multi Hypothesis Tracking

In this section we briefly review the Multi Hypothesis Tracker (MHT) as described by Reid [127] for sensors providing only positive feedback. Subsequently, we derive the expressions needed to compute the probabilities for a data association given detected door handling events. If the user handles a door, we gain information about this door only and not about any other door in the users neighborhood, which is different from tracking multiple targets with a laser scanner for example. In the original paper by Reid, sensors providing only this kind of positive feedback are called type 2 sensors. There, any measurement can be either detected (assigned to an existing track), marked as a false alarm, or be a new track. Since in our particular case the tracks are static doors, we will call them doors in the remainder of this section, rather than tracks. As described in Section 6.2.1 we select a threshold for detection in such a way, that we do not have to model false positives. Note that we will discuss at the very end of the next section why

including a model for handling false positives does not necessary improve the data association in our case.

Since we do not model false positives, a measurement can only be interpreted as *detected* (when matched to an existing door) or as a *new door*. In the following, we assume that a hypothesis consists of the current trajectory, the estimated locations of doors and the data association between different doors. To derive the probabilities of individual measurement assignments we start by reviewing the formulation of the Multi Hypothesis Tracker for type 2 sensors.
Let Ω_j^k be the $j-$th hypothesis at time k and $\Omega_{p(j)}^{k-1}$ the parent hypothesis from which Ω_j^k was derived. Let further $\Psi_j(k)$ denote an assignment that, based on the parent hypothesis $\Omega_{p(j)}^{k-1}$ and the current measurement z_k, gives rise to Ω_j^k. The assignment $\Psi_j(k)$ associates the current measurement either to an existing door or a new door. Given the probability of an assignment and the probability of the parent hypothesis $\Omega_{p(k)}^{k-1}$, we can calculate the probability of each child hypotheses that has been created through $\Psi_j(k)$. This calculation is carried out recursively [127]:

$$\begin{aligned} p(\Omega_j^k|z_k) &= p(\Psi_j(k), \Omega_{p(j)}^{k-1}|z_k) \\ &\overset{Bayes+}{\underset{Markov}{=}} \eta\, p(z_k|\Psi_j(k), \Omega_{p(j)}^{k-1}) \cdot p(\Psi_j(k)|\Omega_{p(j)}^{k-1}) \cdot p(\Omega_{p(j)}^{k-1}), \end{aligned} \tag{6.1}$$

with $p(\Omega_{p(j)}^{k-1})$ being the recursive term, i.e., the probability of its parent. Here, the factor η is a normalizer. The leftmost term on the right-hand side after the normalizer is the measurement likelihood. In our case of mapping indoor environments using human motion and activity, we assume that a measurement z_k associated with a door j has a Gaussian probability density function (pdf) centered around the measurement prediction $\hat{z}_k^j$ with innovation covariance matrix $\mathcal{S}_k^j$, and

$$\mathcal{N}(z_k) := \mathcal{N}(z_k\,; \hat{z}_k^j, \mathcal{S}_k^j). \tag{6.2}$$

Here, the innovation covariance matrix is the uncertainty of the door with respect to the current trajectory and its derivation is described later in Section 6.4. We further assume the pdf of a measurement belonging to a new door to be uniform in the observation volume V with probability V^{-1}. Hence, we have

$$p(z_k|\Psi_j(k), \Omega_{p(j)}^{k-1}) \;=\; \mathcal{N}(z_k)^{\delta} V^{\delta-1}\,, \tag{6.3}$$

with δ being 1 if and only if the measurement has been associated with an existing door and 0 otherwise. The central term on the right-hand side of Equation (6.1) is the probability of an assignment set, $p(\Psi_j(k)|\Omega_{p(j)}^{k-1})$, which is composed of the following two terms: the probability of detection $p_{det_j^k}$ and the probability of a new door. In our case the probability of a detection is equal to choosing one of the current candidate doors, i.e., all doors within an uncertainty ellipsoid. Therefore,

$$p_{det_j^k} \;:=\; \mathrm{NC}(\mathbf{x}_{1:k}, \Omega_{p(j)}^{k-1})^{-1}, \quad \text{with} \tag{6.4}$$

$\mathrm{NC}(\mathbf{x}_{1:k}, \Omega_{p(j)}^{k-1})$ being the number of door candidates, assuming the trajectory $\mathbf{x}_{1:k}$ within the world $\Omega_{p(j)}^{k-1}$. Assuming the number of new doors following a Poisson distribution with expected number of doors λ_{new} in the observation volume V we obtain

$$p(\Psi_j(k)|\Omega_{p(j)}^{k-1}) \;=\; p_{det_j^k}^{\delta} \cdot \mu(1-\delta; \lambda_{new} V), \tag{6.5}$$

where

$$\mu(n;\lambda V) := \frac{(\lambda V)^n \exp(-\lambda V)}{n!} \tag{6.6}$$

is the Poisson distribution for n events given the average rate of events is λ in the volume V. Therefore, Equation (6.1) can be reformulated as

$$\begin{aligned} p(\Omega_j^k|z_k) \quad &= \quad p(\Psi_j(k), \Omega_{p(j)}^{k-1}|z_k) \\ &\overset{\substack{Bayes+\\ Markov}}{=} \quad \eta\, p(z_k|\Psi_j(k), \Omega_{p(j)}^{k-1}) \cdot p(\Psi_j(k)|\Omega_{p(j)}^{k-1}) \cdot p(\Omega_{p(j)}^{k-1}) \\ &\overset{\text{Eq. (6.3)-Eq. (6.6)}}{=} \quad \eta\, \mathcal{N}(z_k)^{\delta} V^{\delta-1} p_{det_j^k}^{\delta} (\lambda_{new} V)^{1-\delta} \cdot \\ &\qquad\qquad \exp(-\lambda_{new} V)(1-\delta)!^{-1} p(\Omega_{p(j)}^{k-1}). \end{aligned} \tag{6.7}$$

Observing that $(1-\delta)!$ is always 1 (since δ is $\in \{0,1\}$) and noting that $\exp(-\lambda_{new} V)$ can be taken into the normalizer η (since it is constant for all hypotheses), we can finally rewrite Equation (6.7) as

$$p(\Omega_j^k|z_k) \quad = \quad \eta \left(\mathcal{N}(z_k)\, p_{det_j^k}\right)^{\delta} \cdot \lambda_{new}^{1-\delta} \cdot p(\Omega_{p(j)}^{k-1}). \tag{6.8}$$

So far, we can detect doors and stair steps and calculate the probability of a data association. In the next section we address the remaining questions of our SLAM procedure, namely the detection of possible door candidates (i.e., loop closures), the calculation of the innovation covariance, and the algorithms which are utilized to correct the trajectory.

6.4 Simultaneous Localization and Mapping

We address the simultaneous localization and mapping problem by its graph based formulation. A node in the graph represents either a pose of the human (i.e., represented by the center of the hip) or a location of a door whereas an edge between two nodes models a spatial constraint between them. These spatial constraints arise either from incremental odometry, potentially adjusted according to the stair heights estimated from stair climbing events, or by closing a loop which corresponds to establishing a data association between two doors. Thus, the edges are labeled with the relative motion between two nodes. To compute the spatial configuration of the nodes which best satisfies the constraints encoded in the edges of the graph, we utilize a variant of our optimization algorithm (see Chapter 4). Since the door handling activities do not give us information about roll and pitch, we restrict our optimization problem to $(x, y, z, \psi)^T$, with ψ being the yaw. This allow us to adapt the fast 2D ($(x, y, \psi)^T$) version of our tree-based network optimizer towards $(x, y, z, \psi)^T$ optimization and still maintain its computational properties. By repeatedly performing this optimization whenever a new door has been detected and a new data association has been established we can incrementally reduce the uncertainty in the current pose estimate while processing the data.

Since we are only able to detect the fact that there is a door, we have to track different possibilities of data association, namely whether the current detected door is one of the already mapped doors or whether the door has not been perceived before. As already mentioned in the previous section, we utilize multi-hypothesis tracking for keeping track of all possible outcomes. To detect a potential loop closure (i.e., recognize a previously seen door), we identify all formerly detected doors which are within the uncertainty ellipsoid of the current pose by a

Dijkstra projection of the node covariances starting from the current position. The innovation covariance is directly used for calculating the likelihood of the door as described in Equation (6.8). All doors being within 3σ confidence of the current pose are considered as potential loop closure candidates, and together with the possibility of the current detected door being a *new door*, lead to $n+1$ different outcomes, given the number of loop closure candidates is n.

For each of these association possibilities we create a separate graph, encode the selected constraint and optimize it. The multi-hypothesis tree therefore grows exponentially in time and pruning of this tree is mandatory to keep computational costs reasonable. In our case, we utilize *N-scan-back* pruning as proposed by Cox and Hingorani [38], which works as follows: it considers an ancestor hypothesis at time $k-N$ and looks ahead in time to all its children at the current time k (the leaf nodes). The probabilities of the children are summed up and propagated to the parent node at time $k-N$. Given the probabilities of the possible outcomes at time $k-N$, the branch with the highest probability at time k is maintained whereas all others are discarded. Since in our case, a step in the MHT only arises when a door has been detected, this is identical to localize N steps ahead in time (at door level). In our implementation, we do not count a data association (step) in time if the only child of each hypothesis is the association with a *new door* or if the trajectory between two subsequent handling events was smaller than 1 m, reflecting the immediate closing of the same door after passing it. Thus we ensure that at least one combination of N data associations in time reflect an N step localization among different and already partially mapped doors.

An example of the N-scan-back MHT algorithm is visualized in Figure 6.7. This example is a snapshot from one of our experiments described in detail in Section 6.7. At the specific time t, the human walked around the building leaving at the top exit and entered the building through the main entry labeled $A0$ in 6.7 (a). Starting from the pose $\mathbf{x}$, where the current door was detected, the uncertainty of the pose was back-propagated utilizing Dijkstra expansion. Since we used the same uncertainty for x and y, the resulting ellipsoid is a circle. Note that due to the back-propagation of the uncertainty the current pose is in the uncertainty region of the door $A0$. For better visibility, only the doors being considered as candidates are shown with their uncertainty regions. Therefore, only two data associations are possible in this case, namely matching the current door with $A0$, which in this case is the correct association, or marking it as a new door. Calculating the posterior probability of each association leads to $p = 0.597$ for the case *new door* and $p = 0.403$ for the correct association. Note that in this situation, a maximum likelihood approach selects the wrong association. However, as the human enters the building and opens another door, given the previous association, different outcomes are possible. Figure 6.7 (b) depicts the situation for the case that the previous decision was *new door* and Figure 6.7 (c) shows the situation for the decision *match with* $A0$. Given this sequence of doors, the full posterior of the branch *new door* at time t sums up to 0.3683 while the probability for the branch *match with* $A0$ sums up to 0.6317 (see Figure 6.7 (bottom row)). Here, an N-scan-back of 2 would be already sufficient to keep track of the correct data association, since the MHT can decide to keep *match with* $A0$ at time t and discard the other branch.

As stated earlier, we selected a threshold τ within the motion templates approach in such a way that we do not need to model false positives. Note that modeling false positives would "only" introduce another term in the MHT, namely a λ_{FP} similar to λ_{new}. To understand why including this model is suboptimal in our case consider the following. First of all, we will never be able to distinguish between a single measurement of a door and a false positive. To resolve this ambiguity, the human has to handle a door at least twice (assuming that both events are detected). Since a human typically does not handle the same door multiple times in a short period of time, we would need to allow for an infinite N-Scan-back, which is computationally

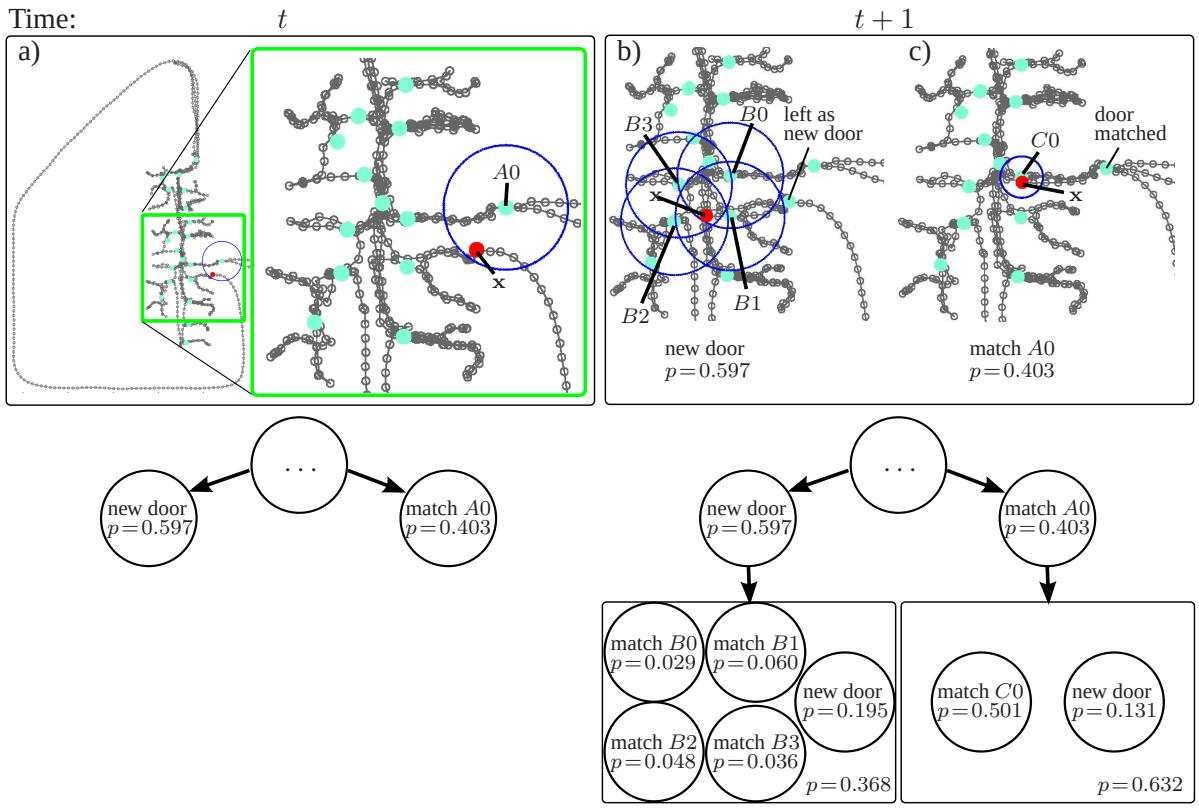

Figure 6.7: A snapshot from one of our experiments. The human re-enters the building through door $A0$ (a). Based on the MHT decision *new door* and *match with* $A0$ different hypothesis are generated as shown in (b) and (c). The probability for a match with $A0$ is lower than for the new door, which would be the wrong data association. However, comparing the probabilities of all possible world evolutions given the previous decision we see that the probability of the branch "previously matched with $A0$" is now higher than "previously mapped to a new door". Thus, postponing the data association by only one step is (in this example) already sufficient to keep track of the correct data association.

infeasible. However, given any finite N-Scan-back, consider the three possible cases (excluding the possibility of matching a measurement with an existing door).

1. $\lambda_{new} > \lambda_{FP}$: Regardless of upcoming measurements, every false positive is labeled as a new door.

2. $\lambda_{new} = \lambda_{FP}$: If we detect the same door within the N-Scan-back period at least once again, we would choose the hypothesis that labeled it as "new door" first. If not, we would choose one of both hypotheses by chance.

3. $\lambda_{new} < \lambda_{FP}$: Only doors, that were handled multiple times (depending on the ratio between both values) *within* the N-Scan-back period would be included in the map. All others would be labeled as false positives and therefore discarded. In the worst case, the N-Scan-back could be smaller than the minimum amount of times a door needs to be handled. However, given any finite N-Scan-back, this would substantially reduce the chance to detect loop closures and thus substantially decrease the robustness of the approach.

As can be seen, none of the cases would generally improve the data association in the multi hypothesis tracker but increase the computational complexity due to the increased amount of generated child hypothesis.

Since we do not model false positives, each of those events will be either labeled as a new door or matched to an already mapped one. However, the latter only happens, if the false positive is close to an existing door and the subsequent handling events match the current map quite well. This, in return, would only impose a small error in the overall map. A door which was observed only once, however, will have no effect on the trajectory optimization since no loop closures are present. We will see in the experimental section, that this also had no effect on the room segmentation.

6.5 Room Segmentation and Approximate Mapping

The output of the multi-hypothesis tracking can be used to generate an approximate map of the environment. Assuming that doors separate rooms, we first cut the whole trajectory based on the locations of individual doors. Thus, even if a door was not always detected or the user moved through an open door, the trajectories are segmented into different rooms, given the specific door was detected at least once. Since steps have no effect on the segmentation, we also obtain segments covering multiple floors (i.e., the hallway). The process of room segmentation is also visualized in Figure 6.8 and Figure 6.9. The raw trajectory and the outcome of the MHT process is shown in (a) and (b) respectively. In order to get all trajectories belonging to the same room, we cluster the data in the following way: Each segment is augmented with a segment id, depending to which door this segmented trajectory is connected to, and on which side of the door it belongs to. This step is visualized in Figure 6.8 (c). Since this id is calculated incrementally for each point starting from those directly connected to a door, a trajectory between two doors is typically cut in half. Given the orientation of a door, we now merge subsequent segments which are connected to the same door and on the same side. We repeat the last step until no change in the segment ids occur, i.e., until convergence (see Figure 6.8 (d)-(e)). Finally, we merge segments which intersect with each other in order to cope with the situation that a room has more than one door. The outcome of this process is shown in Figure 6.8 (f). In order to seek for walls, we incrementally enlarge each segment one after the other until it touches the extend of a segment belonging to another room or up to a threshold d, which was set to 1.5 m in all experiments yielding an approximate map as shown in Figure 6.9 (a). The floor plan of the same building is shown in Figure 6.9 (b).

To sum up, we build a modified Voronoi diagram with respect to the segmented trajectory. We place an obstacle (wall) at every location having the same distance to the closest point of a trajectory belonging to a neighboring room. The difference to the general Voronoi diagram is that we also place an obstacle when we exceed a maximum distance to the trajectory. Note that since we segment the trajectory according to different rooms, we also obtain a topological map of the environment at the same time when calculating the approximate map.

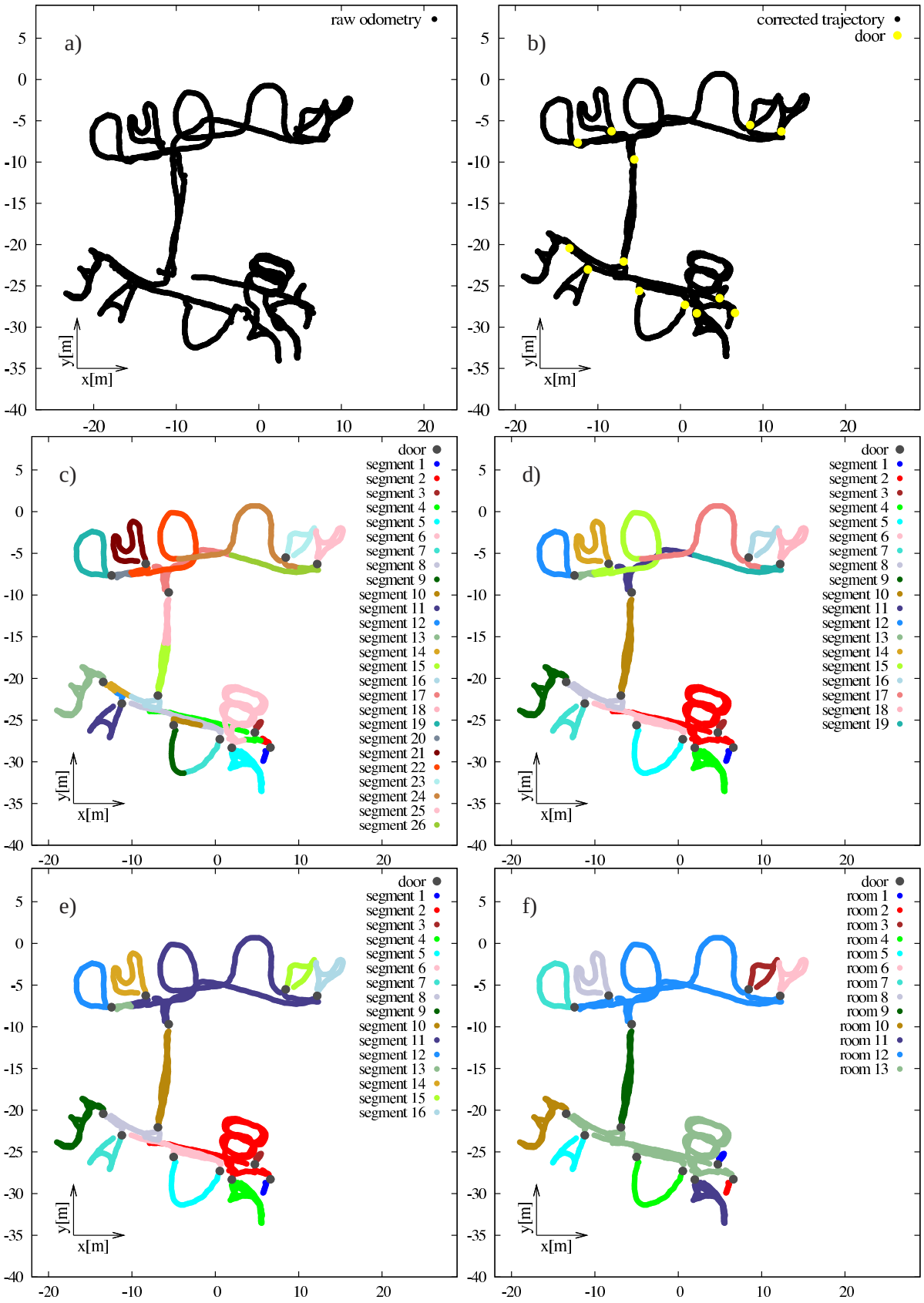

Figure 6.8: Approximate map generation: The raw odometry is shown in (a) and the corrected trajectory is shown in (b). The corrected trajectory is segmented based on the location of the individual doors. For each segment, we obtain an individual id based on the door and its orientation (c). Segments connected to the same door and on the same side are then merged. This process is repeated until convergence (d)-(f). Note that we also obtain a topological representation of the environment. The estimated map given the segmentation plotted in (f) is shown in Figure 6.9 (a).

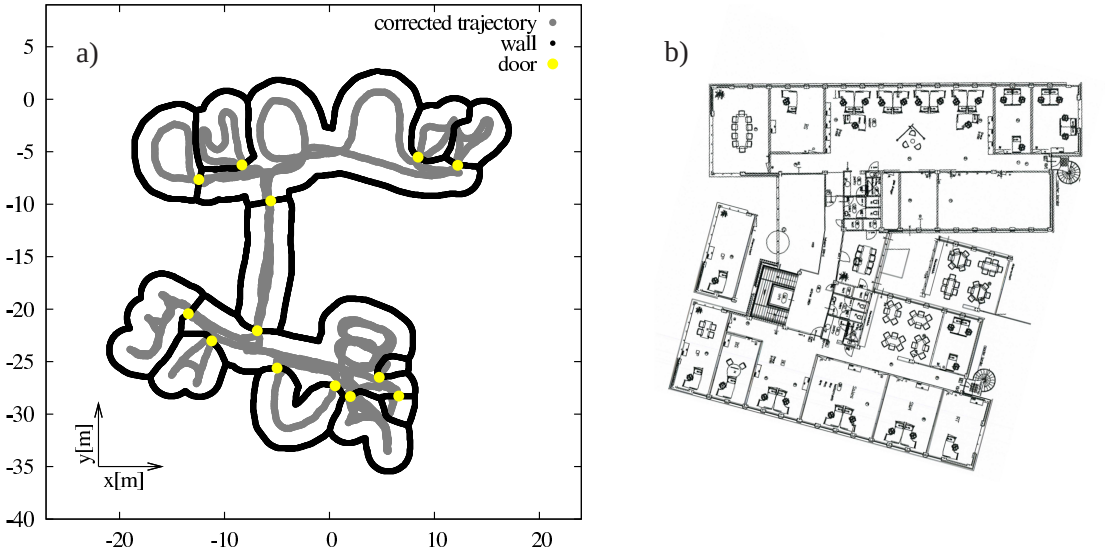

Figure 6.9: Continuation from Figure 6.8. The outcome of the iterated segmentation is shown in Figure 6.8 (f). We finally enlarge each segment incrementally and place a wall whenever it hits the extend of another segment or exceeds a maximum distance (a). The floor plan of the same building is shown in (b) for comparison.

6.6 Overall System

Our approach is summarized by the pseudo-code in Algorithm 6. Given the odometry up to the current point in time t, $\mathbf{x}^{1:t}$, the N-scan-back size n and the current multi-hypothesis tree $\Omega^{1:k} = \{\Omega^1, \ldots, \Omega^k\}$, with $\Omega^j = \{\Omega^j_{j_1}, \ldots, \Omega^j_{j_n}\}$, the algorithm works as follows. Note that k is the current depth of the hypothesis tree and is increased only if there is ambiguity in the data association of a door. First, we add a node (current pose of the hip) and an edge into each graph of the current hypothesis at the current depth k and detect the activities at time t in line 1-3. This is performed by using motion templates for detecting door handling events and neural networks for detecting step activities as described in Section 6.2. If an activity is detected and this activity is a stair step, we augment the odometry information of the recently added nodes with our height estimate (lines 4-8). This height estimate is obtained by estimating the height difference between the left and the right foot and incorporating if the current detected step is a "step up" or a "step down". If a currently detected activity is a door handling event, we calculate for each hypothesis Ω^k_j at depth k potential loop closure candidates C^k_j using a Dijkstra expansion starting from the corresponding current pose. If for all hypotheses no potential loop closure candidate exists, the only explanation of this measurement is that it originates from a previously unseen door, thus each of the current hypotheses can only include a *new door* as described by lines 16-19. In this case it is obsolete to adjust the hypotheses probabilities since all probabilities are multiplied by the same factor λ_{new} which would be normalized out later on. In the case that at least one hypothesis at depth k has one potential loop closure candidate we create a new set of children for all hypotheses (lines 21-22). The number of each set is equal to the number of loop closure candidates plus the additional one reflecting the association "new door". The latter (a *new door*) is added to one child of each hypothesis whereas the graphs of the remaining children are augmented with the loop closure edges. The probabilities of the individual hypotheses are calculated according to Equation 6.8 (lines 23-30). Subsequently, we normalize the probabilities and perform the N-scan-back pruning as described in the Section 6.4. Finally, we optimize the remaining hypotheses at depth $k+1$ using our tree network optimizer (see Chapter 4) and calculate the approximate map of the environment as specified by lines 31-35 and described in the previous section.

Algorithm 6 Human Indoor Mapping

Input: measurements up to current time t: $\mathbf{x}^{1:t}$
Input: N-scan-back size: n
Input: hypothesis tree: $\Omega^{1:k}$

addNodeToEachHypothesis($\mathbf{x}^t$)
addEdgeToEachHypothesis($\mathbf{x}^{t-1}$,$\mathbf{x}^t$)
$\mathcal{A}$ = detectCurrentActivities($\mathbf{x}^{1:t}$)
if stepActivity $\in \mathcal{A}$ **then**
  $\mathbf{x}^t$ = estimateHeight($\mathbf{x}^t$)
  updateLastAddedNodeInEachHypothesis($\mathbf{x}^t$)
  updateLastAddedEdgeInEachHypothesis($\mathbf{x}^{t-1}$,$\mathbf{x}^t$)
end if
if doorActivity $\in \mathcal{A}$ **then**
  $k_n = |\Omega^k|$ // number of hypothesis at depth k
  $v = 0$ // number of all loop closure candidates
  for $j = 1, \ldots, k_n$ **do**
    C_j^k = calculateLoopClosureCandidates(Ω_j^k)
    $v = v + |C_j^k|$
  end for
  // no candidates → *new door* for all hypothesis
  if $v == 0$ **then**
    addDoorNodeToEachHypothesis(doorActivity.hand($\mathbf{x}^t$))
    addEdgeToEachHypothesis($\mathbf{x}^{t-1}$, doorActivity.hand($\mathbf{x}^t$))
  else
    for $j = 1, \ldots, k_n$ **do**
      $v_j = |C_j^k|$ // current number of candidates
      $\{\Omega_1^{k+1}, \ldots, \Omega_{v_j+1}^{k+1}\}$ = createChildren(Ω_j^k, $v_j + 1$)
      // *new door*
      $\Omega_{v_j+1}^{k+1}$.addDoorNode(doorActivity.hand($\mathbf{x}^t$))
      $\Omega_{v_j+1}^{k+1}$.addEdge($\mathbf{x}^{t-1}$, doorActivity.hand($\mathbf{x}^t$))
      calculateProbability($\Omega_{v_j+1}^{k+1}$)
      // *loop closures*
      for $i = 1, \ldots, v_j$ **do**
        Ω_i^{k+1}.addLoopClosureEdges($C_j^k(i)$)
        calculateProbability(Ω_i^{k+1})
      end for
    end for
    $k = k + 1$
    normalizeProbabilities(Ω^{k+1})
    nScanBackPruning($\Omega^{k+1-n:k+1}, n$)
    optimizeEachHypothesis(Ω^{k+1}, numIterations)
    calculateApproximateMapForEachHypothesis(Ω^{k+1})
  end if
end if

6.7 Experiments

The following sections show the results obtained with our currently implemented system. First, we will present our results on trajectory estimation based on human motion and activity and evaluate the error of our estimated door locations with respect to a manually measured ground truth. We calculate the error by first estimating the best transformation between the estimated map and the ground truth throughout all floors. This transformation is then used to calculate the error (mean and std) between the estimated door locations and the ground truth map. In Section 6.7.2, we finally present our results on approximate and topological mapping. Videos of the experiments can be found on the Web [110]. They show the incremental update of the final best hypothesis. Our current system, though not fully optimized, is able to perform an incremental update at a rate of 10Hz on an Intel i7 1.7 GHz laptop.

We evaluated the approach described in this chapter on different data sets in which different people walked in various buildings. The first set of experiments was performed covering multiple floor levels while the second set of the experiments contains data recorded by different humans covering a single floor level partially in the same buildings. All experiments were performed using an N-scan-back of 3 and $\lambda_{new} = 0.03$, which is approximately the number of doors relative to the area covered by the building. In general, λ_{new} depends on the type of building. For example, in a hotel λ_{new} should be significantly higher than in a warehouse. However, we found that small changes to this parameter do not lead to substantially different results. Thus, the remaining free parameter is the covariance matrix of the odometry used for the Dijkstra expansion. Recall that we have no information about the current magnetic field. The covariance matrix, therefore, also reflects the magnetic disturbances present in the building, since high magnetic field errors result in a high pose error estimation from the data suit. We will show the outcome of the maximum likelihood hypothesis in the upcoming experiments.

6.7.1 Trajectory Estimation

In this section, we present the results of several experiments covering single as well as multiple floors of different buildings. Note, that all upcoming plots of single levels of the buildings also contain all points up to the middle of the next and the previous floor respectively. Please also note that the raw data (without the step detection) contains no information along the z-axis with respect to different floors, i.e., only a single floor level is present.

The first experiment contains a trajectory of approximately 2.2 km including 222 door handling actions and is shown in Figure 6.10 and in Figure 6.11. The building has three floor levels, namely the first floor, an intermediate floor level containing the main entrance, and the second floor. Since the intermediate level contains only the main entrance door, we omitted to plot this floor separately for better readability. We used a variance of 0.03 m per meter in x and y and a variance of 0.1 m per meter along the z axis. Our approach reliably detected 215 out of the 222 door handling events with one false alarm. The average error of the estimated door locations is $0.31\,\text{m}\pm0.17\,\text{m}$ wrt. a manually measured ground truth. We detected 106 out of 116 stairs, missing 7 stairs down and 3 stair up and had one false alarm. The difference in the calculated stair size between up and down is approximately 3.5 cm. The raw odometry trajectory is depicted in Figure 6.10 (a). Although no floor level information is present in the raw data, the raw odometry trajectory is already quite accurate. This results from the fact that the building contains less metal structure compared to modern buildings so that we obtained only small magnetic disturbances. As can be seen in the next experiments, larger disturbances typically lead to higher pose errors. The raw trajectory including our step detection is plotted in Figure 6.11 (a).

The maximum-likelihood map estimated by our approach is depicted in Figure 6.11 (b). For better comparison, we also segmented the trajectory for different floor levels and compare them to floor plans generated by the architect of the same building as shown in Figure 6.10(b)-(e). The alignment for all floors was performed based on the three middle doors of the first floor.

The data for the second experiment was recorded in a typical university building containing several floors and including small seminar rooms as well as big lecture rooms. The trajectory is approximately 2.85 km long covering three floor levels. This experiment is challenging for two reasons. First, disturbances rising from the metal structure of the building itself and from walking closely to chairs and tables lead to a high pose error as can be seen in the raw data depicted in Figure 6.12 (a). Second, the first and the second floor are nearly identical on one side of the building which results in many potential loop closure candidates. Compared to the first experiment, this building contains in total five different staircases. Two staircases are present in each of the two lecture halls (see Figure 6.13 (b) left part) connecting the first floor and the second floor, whereas the main staircase connects all three floor levels. In this experiment, we used a variance of 0.1 m per meter in all directions, i.e., x, y, and z. The raw trajectory including the steps detected by our algorithm is plotted in Figure 6.13 (b). The result of our approach compared to the floor plans of this building are shown in Figure 6.12 (b)-(d). Finally, the maximum likelihood estimation of the whole building is depicted in Figure 6.13 (f). In this experiment we detected 175 out of 178 door handling events with an average error of $1\,\text{m}\pm0.41\,\text{m}$. We also have one false alarm at the third floor which originates from the user moving a chair away in the library which was blocking his path. Regarding the stair detection we missed 62 out of 473 stairs (42 stairs up and 20 stairs down). The average difference between the calculated stair heights is 1.3 cm.

The third experiment was recorded in a university building consisting of five floors and containing a substantial amount of metal structures. Here, the magnetic disturbances did not even allow for a proper initial calibration of the data suit. This had a severe influence on the estimated raw odometry trajectory. We intentionally included this experiment to show the robustness of the current approach even in the context of substantial disturbances. Since our assumption of a Gaussian error in all degrees of freedom is highly violated (for example, one staircase is rotated by 45 degrees in the raw odometry data) we still were able to approximately recover the true trajectory although with one misaligned door (see Figure 6.14 (b)). This door, which is marked by an arrow in the figure, is wrongly labeled as a new door. As in the previous experiment, we used an innovation of 0.1 m per meter along all axis. The total distance traveled in this building is approximately 1.46 km and contains 135 door handling events from which our approach detected 126. It furthermore resulted in one false alarm in the lower left corner of the ground level. The average error of our estimated door locations is $0.67\,\text{m}\pm0.40\,\text{m}$. Regarding the step detection, we were able to detect 271 out of 280 stairs, missing 7 stairs up and 2 stairs down. The calculated stair size for the class "stair down" was on average 4 cm higher than for the class "stair up". The raw trajectory is depicted in Figure 6.14 (a) and Figure 6.15 (a) together with the raw steps and doors detected by our algorithm. The resulting map estimated by our approach is depicted in Figure 6.15 (b). The individual floors plotted on top of the floor plan are shown in Figure 6.14 (b)-(f). Note that the estimate of the first floor is slightly suboptimal due to the severe error in the raw data yielding a small drift along the x-axis between the different levels. Since some of the doors were locked, we were not able to enter all rooms. The corresponding doors appear to be visually not connected to the trajectory in Figure 6.14(b)-(f). This originates from the fact, that the user was not able to pass through the corresponding doorways, i.e., door positions are obtained by the hand pose handling the door whereas the trajectory is given by the position of the user's hip.

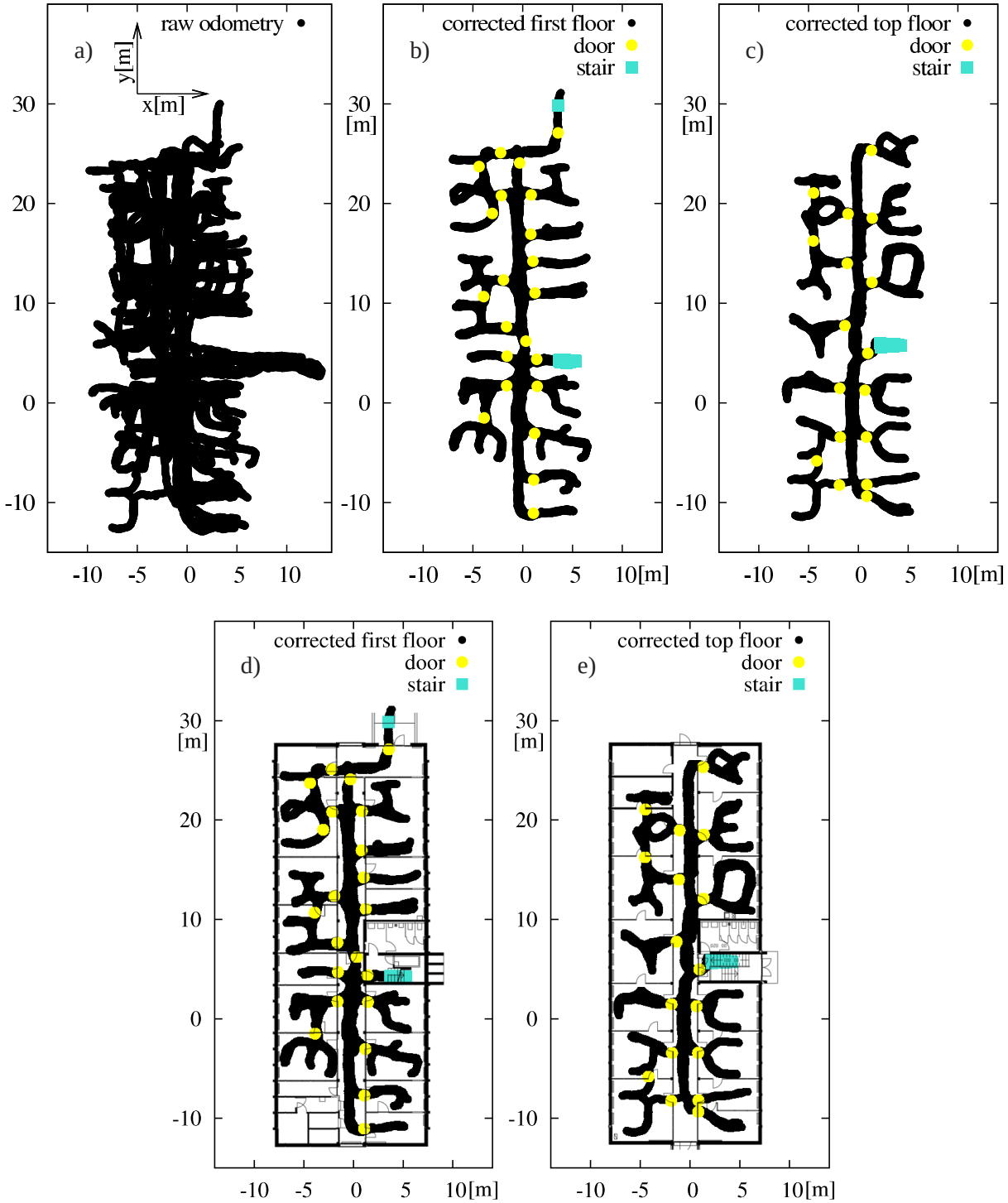

Figure 6.10: Outcome of the first experiment: (a) the raw odometry trajectory estimated by the data suit. The maximum likelihood estimation of the first floor and the top floor using our approach are shown in (b) and (c), and aligned to a floor plan in (d) and (e) respectively. A perspective view of the data is shown in Figure 6.11. Note that we omit to draw the labels of the axes in figures (b)-(e) for better readability.

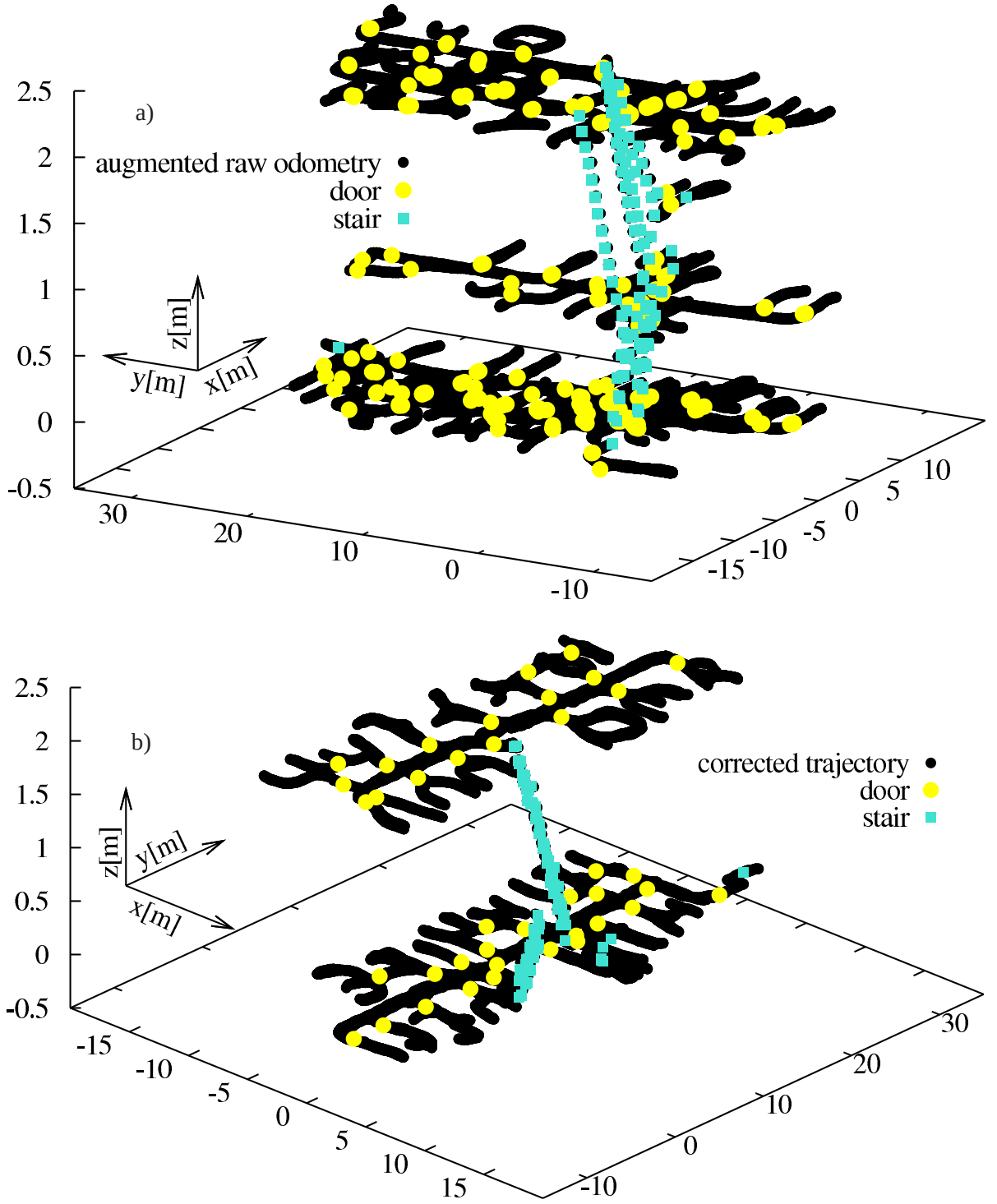

Figure 6.11: Perspective view of the outcome for the first experiment. The raw odometry trajectory augmented with the raw detection of stairs and doors is shown in (a). Note that the elevation in (a) is obtained as the difference between the altitude of the feet in the raw data given our classifier detected a step event (i.e., the suit provides only 2D data a shown in Figure 6.10 (a). A 3D plot of the corrected trajectory estimated by our approach is shown in (b).

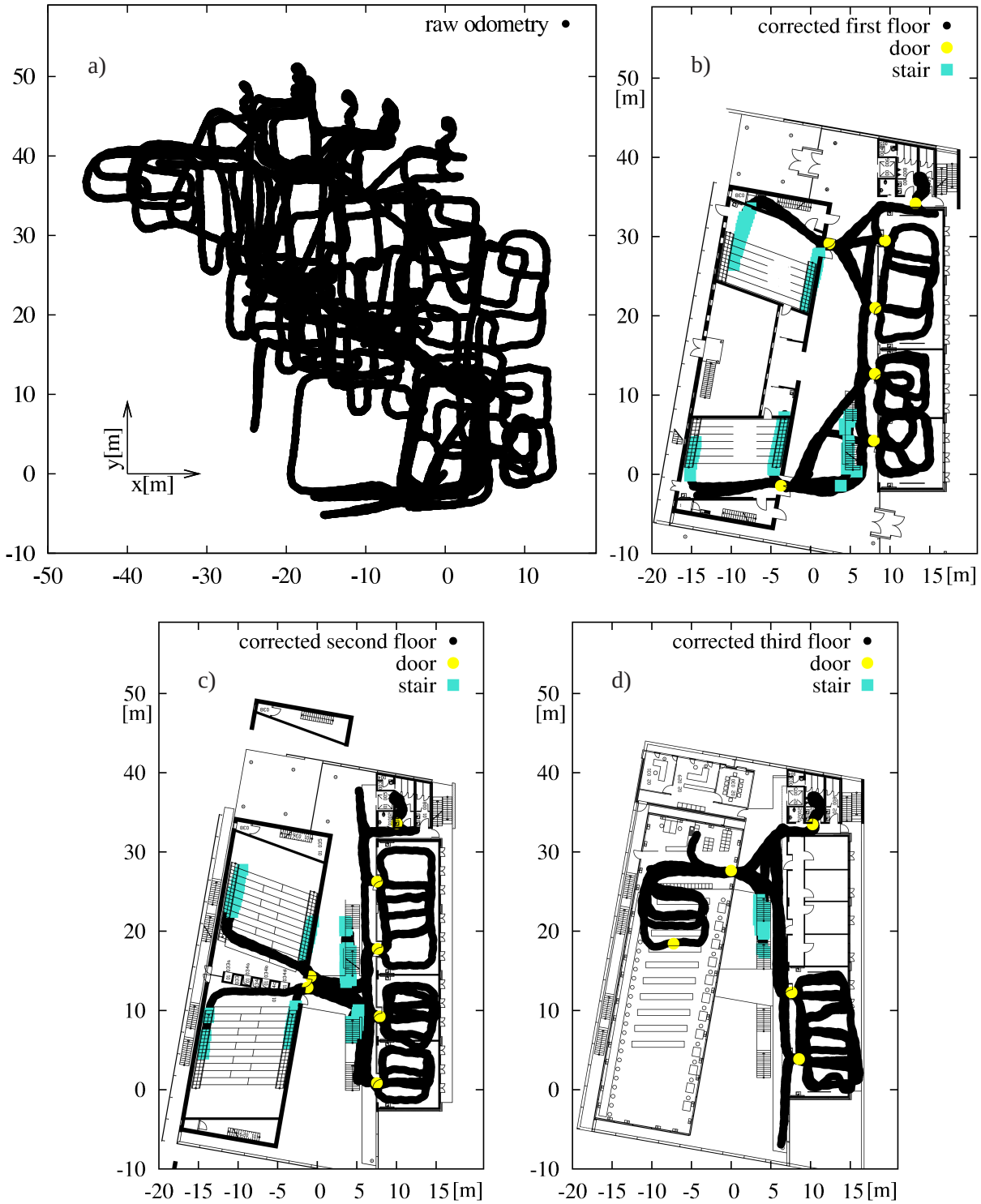

Figure 6.12: The second experiment was performed in a typical University building containing several small seminar rooms as well as two big lecture rooms. The raw odometry data is depicted in (a), whereas the different floor levels of our maximum likelihood estimate plotted on top of the floor-plans of the building are shown in (b)-(d). A perspective view of the data is shown in Figure 6.13. Again, we omit to draw the labels of the axes in figures (b)-(d) for better readability.

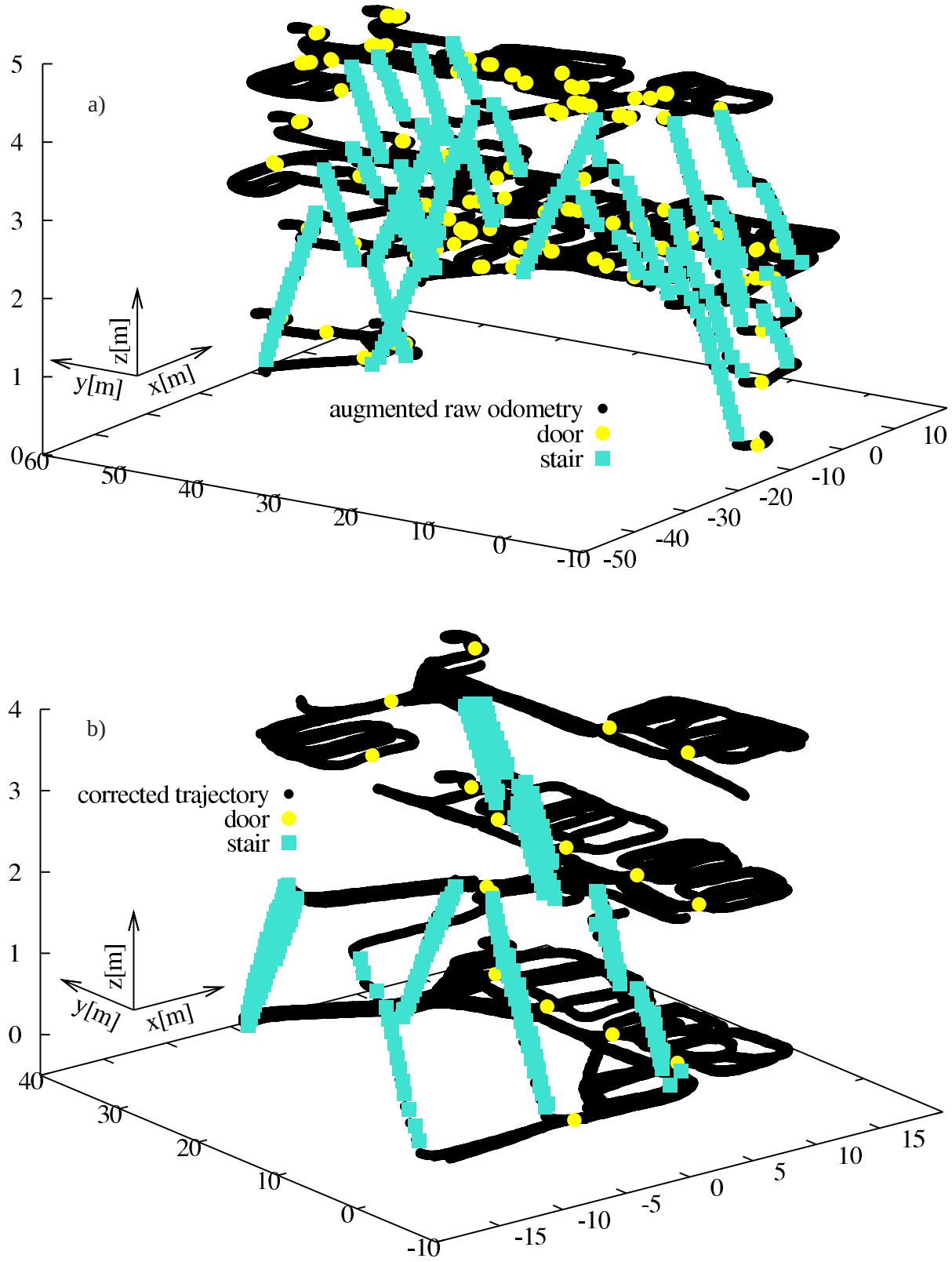

Figure 6.13: Perspective view of the outcome for the second experiment. The raw odometry trajectory including the uncorrected location of stairs and doors is depicted in (a). The maximum likelihood estimate of the whole building using our approach is shown in (b).

Figure 6.14: The third experiment was performed in a building containing a substantial amount of metal structure. This introduced severe errors in the odometry estimate provided by the data suit, especially when walking up and down the staircase between the first and the second floor. The raw odometry is depicted in (a). Our maximum likelihood solution of the individual floors is shown in (b)-(f). The high errors in the raw data led to a wrong data association in the first floor (a), where the left door marked with the arrow was wrongly labeled as a *new door*.

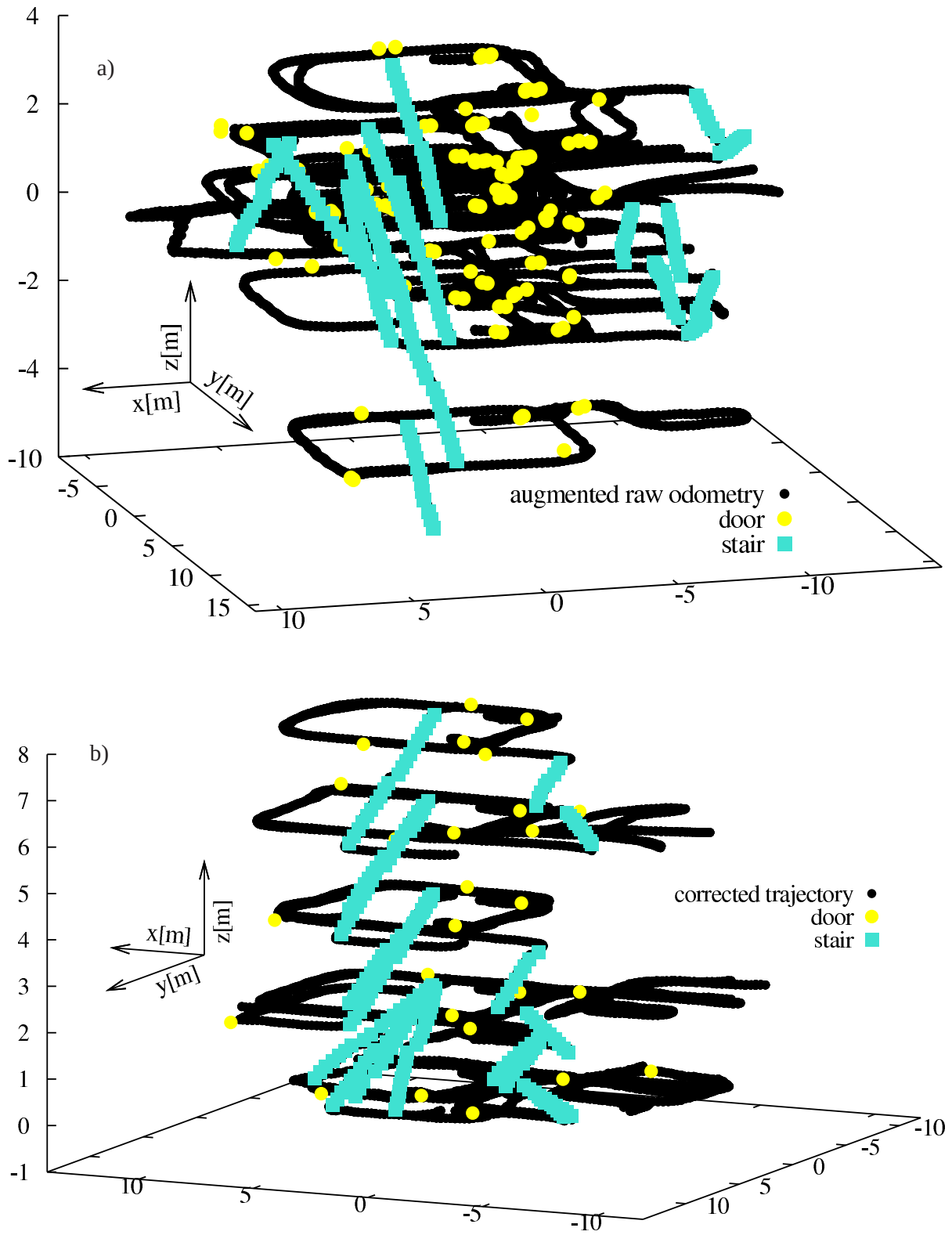

Figure 6.15: Perspective view of the outcome for the third experiment. The raw odometry trajectory including the uncorrected location of detected stairs and doors is depicted in (a). Note the two instances of the staircases which are rotated by approximately 45 and -40 degrees in the raw odometry trajectory estimated by the suit (a) due to the high disturbances in the magnetic field. The corresponding maximum likelihood estimate of the whole building using our approach is shown in (b).

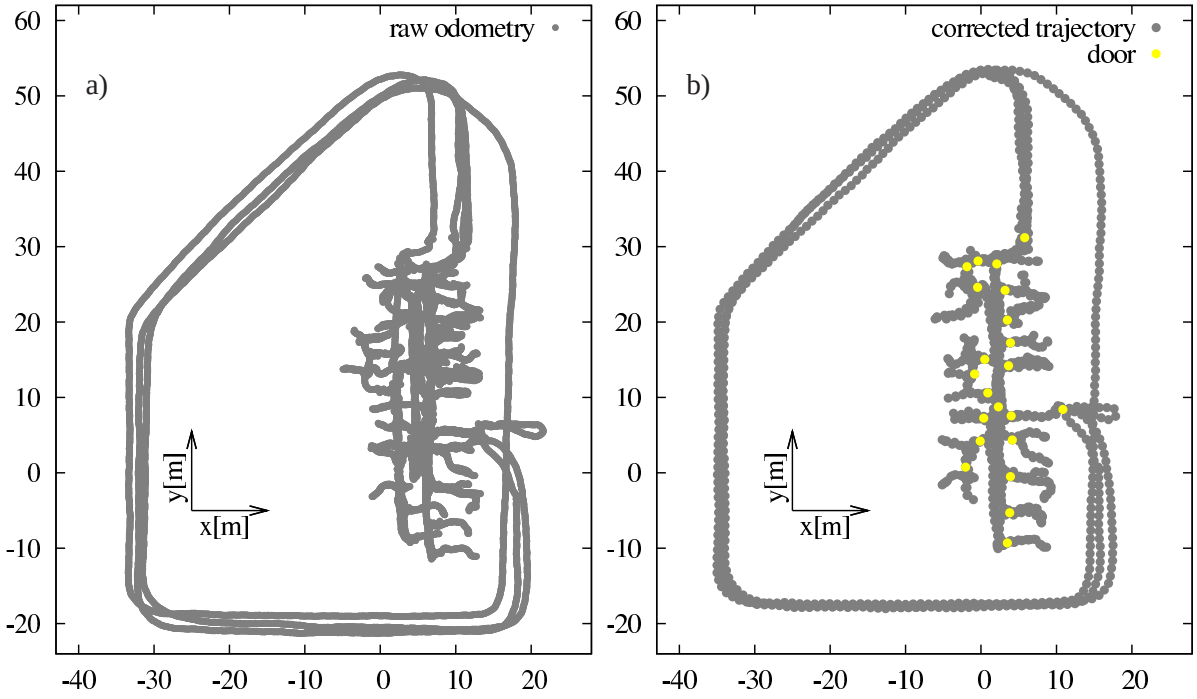

Figure 6.16: Outcome of the fourth experiment. The raw odometry trajectory is shown in (a) and the corrected one is visualized in (b). The respective parts of the trajectory inside the building are shown in Figure 6.17.

We also performed an extensive set of two-dimensional experiments (i.e., covering only a single floor level) with different subjects than in the first three experiments.

The fourth experiment was recorded in the same building as the first one. This time, the trajectory was approximately 1.6 km long including 133 door handling events. Our approach reliably detected 125 out of the 133 events with an average error of 0.5 m± 0.24 m using the same parameters as in the first experiment. The raw odometry trajectory is shown in Figure 6.16 (a) and the corrected one is visualized in Figure 6.16 (b). This experiment also contains several loops around the building. The parts of the trajectory which were recorded inside the building are shown separately in Figure 6.17 (a) and (b). The latter includes the result of our approximate mapping algorithm and therefore also contains the estimated locations of walls.

The fifth experiment contains a trajectory of approximately 1.3 km and was obtained by walking inside the same university building as in the second experiment. Again, we intentionally walked closely around rows of tables and chairs. The magnetic disturbances led to a high pose error, as can be seen in the raw odometry trajectory (see Figure 6.18 (a)). As in the second experiment, we used a variance of 0.1 m per meter along all dimensions. Although the initial odometry differs up to 30 m for the same place, we were able to correct it as shown in Figure 6.18 (b). In this experiment, we detected all 63 door handling events with an error of 0.61 m± 0.17 m.

The final experiment was recorded in a typical office environment. For this experiment we used a dataset recorded by Xsens. The raw odometry trajectory is shown in Figure 6.19 (a) and the corrected trajectory using our approach is shown in Figure 6.19 (b). The trajectory is approximately 0.4 km long and we detected 24 out of 27 door handling events by using the same parameters as in the previous one. However, this experiment was recorded by a different team and we do not have ground truth data of the door locations but a floor plan of the building (see Figure 6.19 (d) or Figure 6.9 (b) on page 130).

The outcome of all experiments together with the parameters used are also summarized in Table 6.2. Note that the 2D experiments were performed without the step detection algorithm.

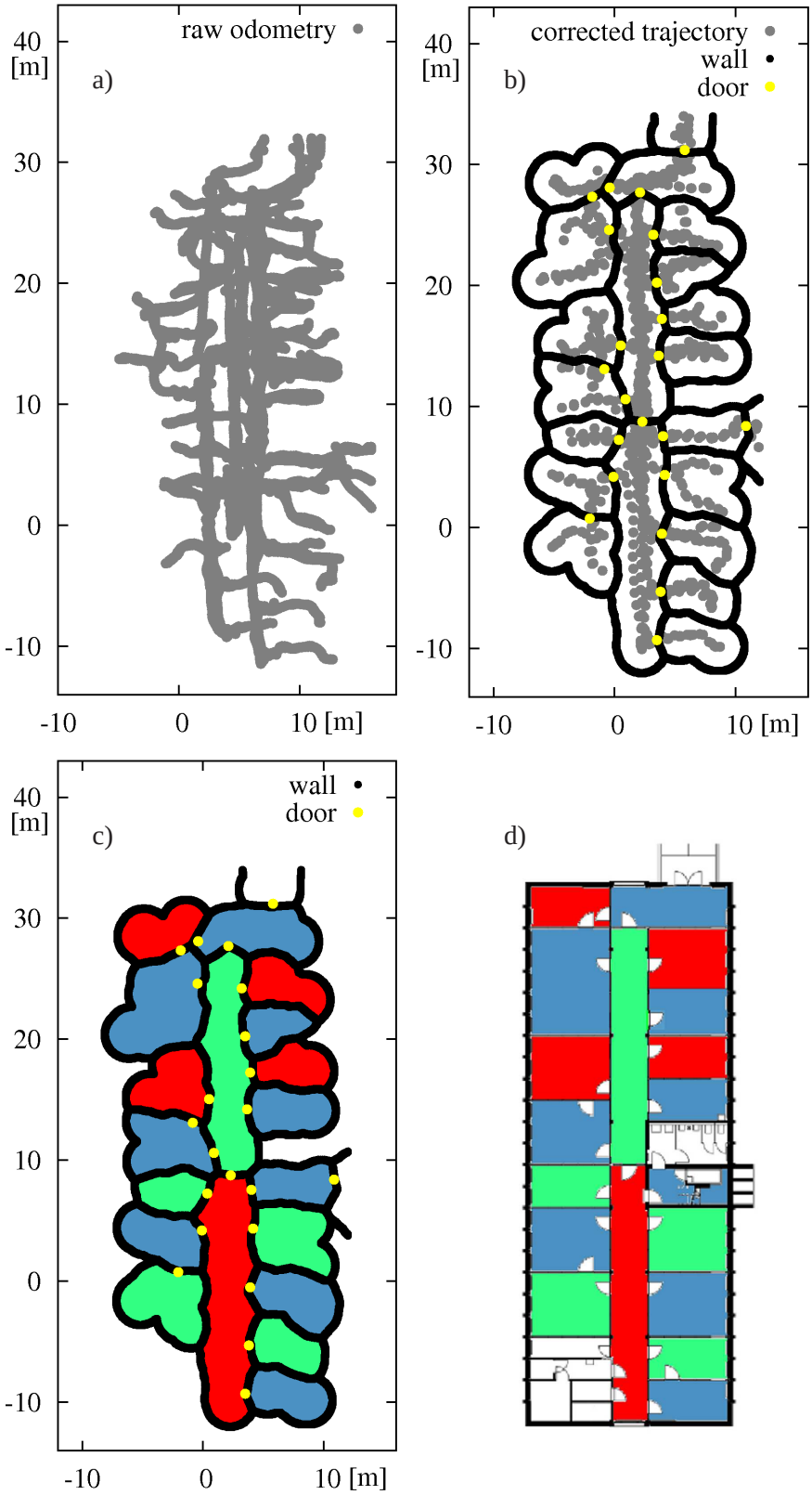

Figure 6.17: The part of the fourth experiment which was recorded inside the building. The raw odometry trajectory is shown in (a) and the corrected one using our approach is visualized in (b). The latter also shows the result of our approximate mapping algorithm. Since we segment the trajectory according to different rooms, we also get the topological representation of the building. This is visualized in (c) using three colors in total. The corresponding floor plan (d) has been colored accordingly for better readability.

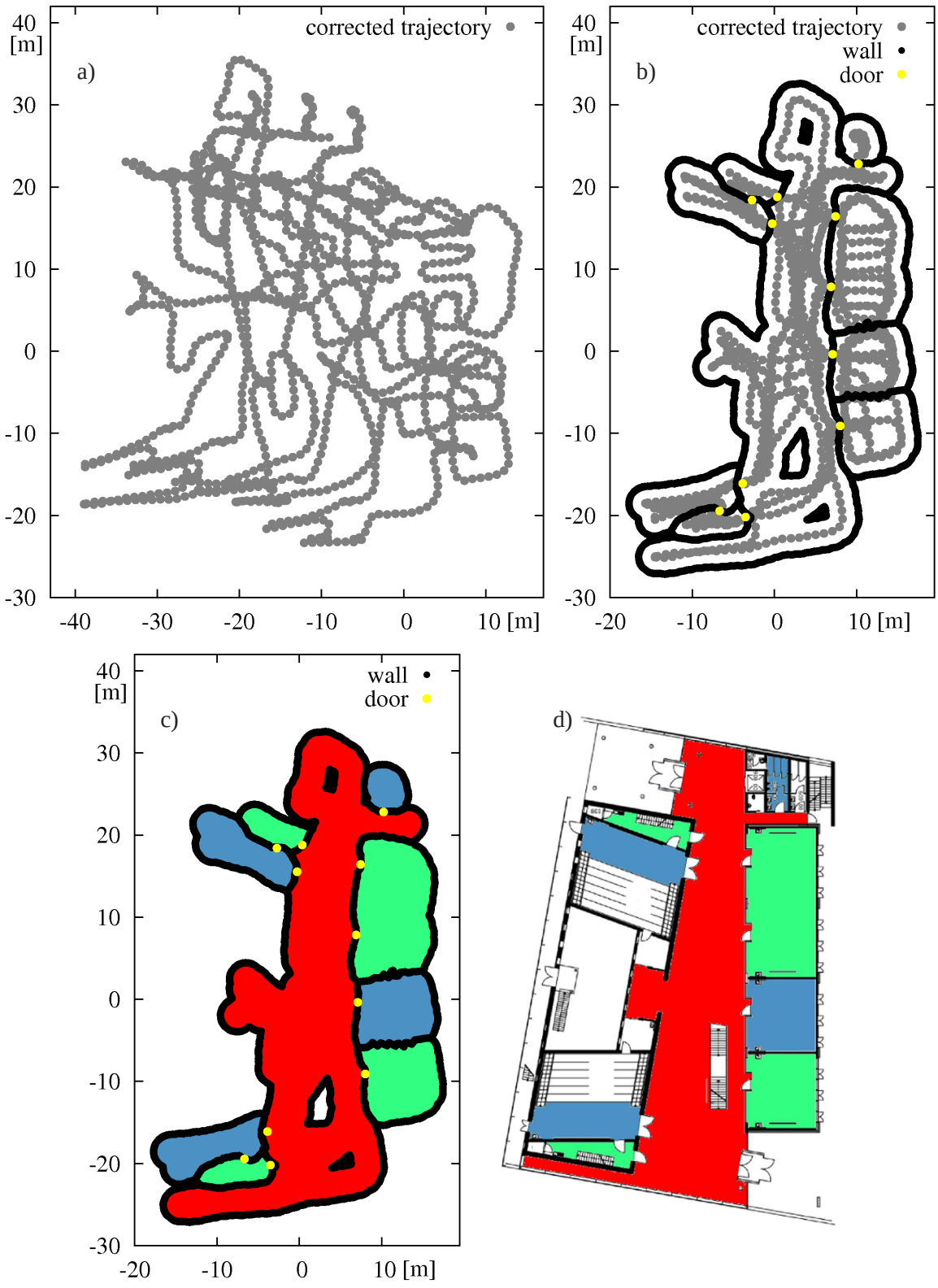

Figure 6.18: The raw odometry trajectory of the fifth experiment is shown in (a). The corrected trajectory including the approximate locations of walls using our approach is visualized in (b). Again, we also obtain a topological representation of the environment given our segmentation approach. The topological representation using three colors in total is shown in (c). The floor plan of the same building is shown in (d) and has been colored respectively for better visualization.

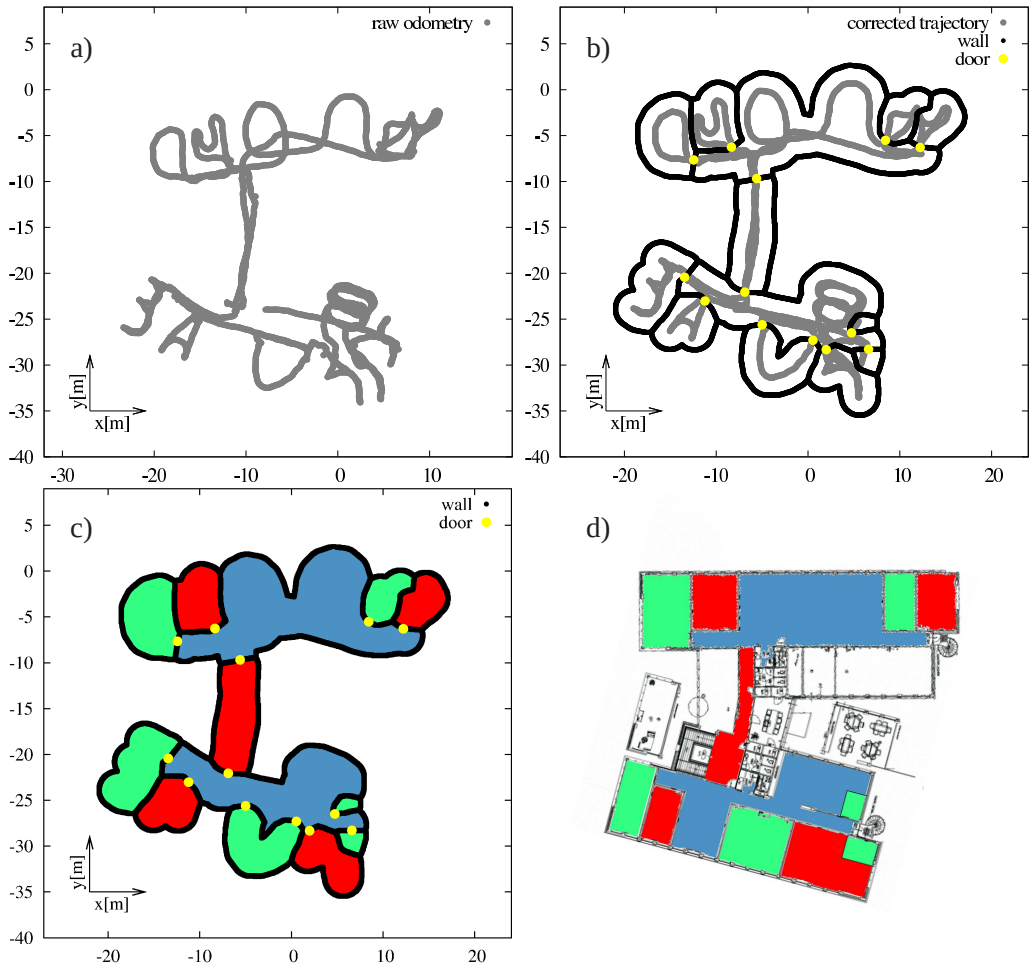

Figure 6.19: The sixth experiment. The raw odometry trajectory and the corrected one are shown in (a) and (b), respectively. The topological representation is shown in (c) and the corresponding floor plan is shown in (d).

Experiment No.	Trajectory length	No. of floors	Parameters				Building id
			λ_{new}	N	σ_{xy}^2	σ_z^2	
1	2.2 km	2	0.03	3	0.03	0.1	079
2	2.85 km	3	0.03	3	0.1	0.1	101
3	1.46 km	5	0.03	3	0.1	0.1	106
4	1.6 km	1	0.03	3	0.03	0.1	079
5	1.3 km	1	0.03	3	0.1	0.1	101
6	0.4 km	1	0.03	3	0.1	0.1	Xsens

Experiment No.	Door detection		Step detection		Error of estimated door locations	Subject id
	Recall rate	FP	Recall rate	FP		
1	0.968	1	0.914	1	$0.31\,\text{m}\pm0.17\,\text{m}$	A
2	0.983	1	0.869	0	$1\,\text{m}\pm0.41\,\text{m}$	A
3	0.933	1	0.968	0	$0.67\,\text{m}\pm0.40\,\text{m}$	A
4	0.94	0	n/a	n/a	$0.5\,\text{m}\pm0.24\,\text{m}$	B
5	1	0	n/a	n/a	$0.61\,\text{m}\pm0.17\,\text{m}$	B
6	0.889	0	n/a	n/a	n/a	C

Table 6.2: Summary of all experiments. The recall rate is calculated as the ratio of true positives versus the actual number of events. Here, FP is short for false positives and N is short for N-Scan-back.

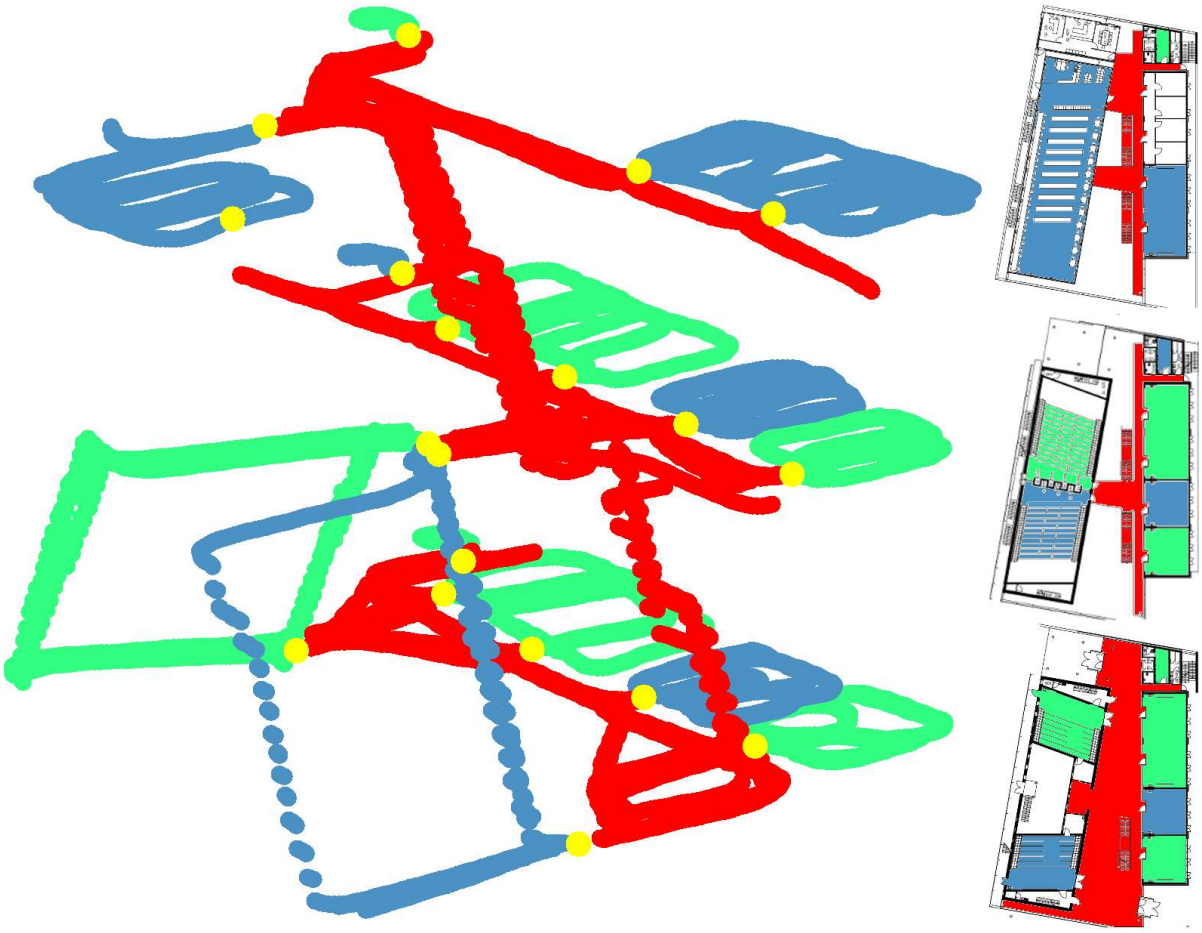

Figure 6.20: Outcome of our approximate and topological mapping algorithm for the second experiment. We omit the plotting of the rooms extends, as the perspective view of the 3D structure in combination with outer walls would render the image black. The floor plans on the right hand side show the individual floors of the building (see also Figure 6.12 and Figure 6.13) and are colored with respect to the outcome of our approach for better visibility.

6.7.2 Room Segmentation and Approximate Mapping

In this section we show our results of our room segmentation and approximate mapping algorithm for floors of different buildings. Figures 6.17, 6.18, and 6.19 show typical outcomes of our approach and the building's floor plans respectively. Note that our mapping technique segments the trajectory into different rooms. We therefore can calculate both, a geometrical and a topological map. The topological maps colored with respect to different rooms (using three different colors in total) are shown in (c) of Figures 6.17, 6.18, and 6.19. The corresponding floor plans have been manually colored and are shown in (d) of the same figure. As can be seen, there exists a high visual correlation between the estimated floor plans and the real ones. Errors mainly arise from rotational errors as can be seen in the bottom left part of Figure 6.18 (b)+(c). These rotational errors, however, can be corrected by including an additional loop around the building from the outside or inside as demonstrated in Figure 6.12. The walls within the map of Figure 6.18 (b)+(c) are present since all experiments were performed using a maximum distance of $d = 1.5\,\mathrm{m}$ as described in Section 6.5. Figure 6.20 shows the outcome of our segmentation approach for the second experiment. Here, we omit to plot the walls since the perspective view of the 3D structure in combination with the outer walls would render the figure completely black. However, the outcome of our segmentation algorithm also provides the topological structure of the building which accurately resembles the true layout as can be

seen by comparing it with the corresponding floor plans of the same floor. These experimental results demonstrate, that our approach is robust and can be applied in different environments providing accurate results.

6.8 Related Work

The problem of tracking the correct data association [75] as well as human indoor navigation and localization has recently become an active research field [96, 133, 99, 35]. A number of different sensors have been employed and different kinds of localization techniques have been used. One of the first approaches in this area has been proposed by Lee and Mase [96], who employ wearable accelerometers and other sensors, i.e., a digital compass and a velocity sensor, to recognize when humans perform specific activities and change their locations in indoor environments. They integrate the accelerometer data over time and estimate the position of humans in a known environment based on higher level descriptors such as *standing*, *2 steps north*, or *40 steps east* etc. The field of human indoor navigation and localization is therefore closely related to activity recognition using accelerometer data. Bao and Intille, [14] as well as Ravi *et al.* [126] have presented approaches to predict certain low level activities like *walking*, *standing*, *running*, *sit-ups*, and others using features from raw accelerometer data and a variety of different learning algorithms. However, they do not employ this information for indoor positioning. Schindler *et al.* [133] utilize an accelerometer together with an infrared proximity sensor mounted on a pair of headphones to detect when a human is passing through a doorway. In this work, the authors are able to construct topological maps, where rooms are represented by single nodes and edges represent the path in steps between doorways. For building these maps and for detecting loop closures, the human user has to indicate by gesture which door was passed, i.e., giving each door a unique identifier via the infrared proximity sensor. They furthermore apply a Bayesian filtering scheme to localize the person within the resulting map.

In the last years, low-cost inertial measurements units (IMU) based on MEMS have become available and many researchers use such sensors for pedestrian localization, either alone or in combination with other sensors. Foxlin *et al.* [51] incorporate a zero velocity update allowing to estimate the users trajectory using an extended Kalman filter. Borenstein *et al.* [20] use a highly precise IMU also combined with zero velocity updates and obtain an accurate dead reckoning odometry. Woodman *et al.* [163, 164] as well as Wang *et al.* [161] include additional information using WiFi. Both research groups employ a particle filter to track possible trajectories and calculate the weights of the particles based on the WiFi signal strength. Fischer *et al.* [48] discuss the possibility of using ultrasound sensors to reduce the error introduced by the MEMS sensors and present simulation results. Feliz *et al.* [46] utilize a neural network to estimate the step size using a single IMU and thus estimate the odometry. Coley *et al.* [37] use wavelets to detect steps using gyroscopes only. In the work of Toth *et al.* [154], a prototype for pedestrian dead-reckoning and their general concept of sensor fusion is discussed. The HeadSLAM approach by Cinaz and Kenn [35] employs a laser scanner together with an IMU mounted on a helmet. They use the IMU sensor to project the laser scans into a horizontal plane in a global coordinate system and employ a variant of GMapping [61] for mapping. In particular, they incorporate a simplified motion model with two modes. Whereas the first mode corresponds to the activity walking and assumes constant velocity, the second mode represents the situation that the person is standing still and assumes zero speed. An overview over existing techniques can also be found in [47].

6.9 Conclusion

We presented a novel approach to accurately estimate the 3D trajectories of humans based on data gathered with a motion capture suit. Our approach extracts two different activities from the motion data, namely door handling and stair climbing events. We consider the trajectory of the person and the height estimates of our step detection algorithm as motion constraints. The door handling events detected using specific motion templates are used as landmarks within a graph-based SLAM approach. To cope with the high data association uncertainty, we employ a multi-hypothesis tracking approach. Additionally, our method can create approximate geometrical as well as topological maps of the environment based on the estimated trajectory and activities. Our system has been implemented and successfully tested on real data recorded with different subjects in several buildings on a university campus as well as in a typical office environment. The experimental results demonstrate that our approach is able to robustly keep track of the true data association and accurately estimates the trajectory taken by the person. Furthermore, the resulting geometrical as well as topological maps accurately resemble the corresponding environments.

Conclusions and Outlook

Chapter 7

Conclusions and Outlook

A fundamental prerequisite for a sensor system, whether it is mounted on a robot or integrated into the garment of the human, is knowledge about the current state. In state-of-the-art robotics, the current state includes also the information about the current location. This information allows robots to perform navigation tasks autonomously or general sensor systems (not necessarily mounted on a robot) to assist humans in their mission. However, robots as well as other non-robotic embedded sensor systems (i.e., based on the data suit) are only envisioned as useful when the overall system is robust, small, and is able to operate autonomously over an extended period of time without the need of human interference. Specially, flying robots must be equipped with a high level of autonomy. Here, the complexity of the robot makes is hard to remotely steer it, in particular in confined indoor locations where a good quality of a radio link cannot be guaranteed. Additionally, given the limited payload and high time constraints, this imposes several challenges for the underlying algorithms.

To obtain a estimate of the current state, it is inevitable to either use an existing map of the environment or to build a map during the mission of the agent (i.e., robot or human). Especially in indoor environments, a map of the building is not (readily) available in most of the cases. Since the map has to be build on-line, efficient mapping techniques are needed in order to correct for sensor noise. Simultaneous localization and mapping (SLAM) aims to estimate the current state of the agent and simultaneously build a map of the environment. Using a graph-based representation allows us to divide such systems into two parts, namely the front-end and the back-end. The main gaols of the front-end SLAM system include determining the incremental motion and detecting loop closures. In other words, the front-end calculates the nodes and edges of a graph. Optimizing techniques, namely, algorithms estimating the configuration of the nodes which minimizes the overall error are called the back-end of a SLAM system, respectively.

In this thesis, we developed an efficient graph-based optimization technique which allows a system to efficiently correct for odometry errors after detecting loop closures. We have demonstrated that this back-end system calculates a minimum-error configuration orders of magnitude faster than other state-of-the-art systems without any loss in accuracy. We achieved this by introducing a novel tree parametrization which divides the overall optimization problem into smaller sub-problems. We furthermore presented an extension to our approach which allows for efficient graph optimization in 3D. We also presented an approach for node reduction. This yields an optimization technique with an computational complexity being dependent on the space the robot explored and not the time the robot spent in the environment. Our proposed algorithm is a general framework which can be used with a variety of SLAM front-ends.

Autonomous flying robots are envisioned as one of most important robotic systems during these days. Such robots can assist humans in search and rescue missions as well be used as a remote eye where wheeled robots cannot operate. We developed a navigation system for autonomous indoor flying using a quadrotor robot. Using our graph-based optimization back-end, we built a SLAM front-end which meets the demanding requirements on high accuracy and low computational complexity which is needed for such an embedded system. We furthermore developed techniques which allows the flying robot to autonomously reach desired locations and avoid obstacles entering the robot's field of view. We have shown in an extensive set of experiments, that our developed platform is able to robustly operate in structured indoor environments and build accurate maps (2D as well as 3D) of the area the robot is operating in.

However, flying platforms are only one possibility on how to assist humans. Especially in the context of search and rescue missions, it is envisioned that the knowledge about the pose of the rescue team (e.g., firefighter) will help to save lives. Since scenarios where such teams are operating in, prevent the usage of light dependent sensors like cameras or laser scanners (e.g., due to smoke) we developed a mapping technique based on human activity and motion only. Here, we used human activities, like opening or closing doors, and treated them as landmarks in a feature-based SLAM system. Again, our graph-based optimization was used as the back-end of the overall system. We have demonstrated in several experiments, that we are able to accurately recover the 3D trajectory of the agent and that we can build approximate geometrical as well as topological maps of the environment which resemble the floor plans of the building with a remarkable accuracy. The detected activities are passive, i.e., the agent does not need to learn specific gestures of commands and can concentrate himself on his mission. Therefore, we believe that our proposed algorithm will be helpful during daily life of rescue workers.

In summary, all developed algorithms were extensively tested using real world data. We demonstrated that our graph optimization technique is a robust and fast error minimization algorithm intended to be used as the SLAM back-end. We furthermore developed a navigation system enabling fully autonomous indoor flying using a quadrotor robot. Finally, we developed an efficient and robust approach for simultaneously localize a human and map the indoor environment employing the motions of the human as the only sensory input. We believe, that the presented techniques will allow to build systems that can be used in everyday work improving the work quality of humans and that are helpful for a variety of applications, including search and rescue, potentially helping saving lives.

In spite of our promising results presented in this thesis there are many possibilities on how to extend the presented solutions. Within our quadrotor navigation system, we use a laser scanner for perceiving the environment only. One possibility would be to add additional sensors, like cameras or radar and fuse the overall information. This would improve the robustness of the vehicle and extend it's autonomous capabilities to complex and highly cluttered environments. Another possibility is to learn the dynamic model of the flying robot. This would allow the quadrotor to perform impressive maneuvers using on-board sensors only, which currently are only possible with an external, fast, and highly accurate camera tracking device. Another possibility is to extend the quadrotor's operational environment from air to air and water. Although diving with a quadrotor would be restricted to a few meters only it opens a highly interesting research question. How should a navigation system be designed and which sensors must be used in order to allow robust and fast autonomous flights as well as underwater missions?

Our approach to map indoor environments based on human activity uses multi-hypothesis tracking to deal with the high data association. Given any N-scan-back with a finite N, a human could walk a sufficiently long trajectory, which would lead to a suboptimal data association. Here, one could further investigate this problem in order to find a solution which is more robust than the multi hypothesis tracker and still computational feasible. Clearly, one could imagine including more activities. One possibility would be to extend the set of activities from passive to active ones. In other words, the subject would explicitly perform an activity, like touching a wall, in order to improve the results. Although motion templates have been shown to be a good framework to learn door handling events they are computationally very intensive. Thus the question arises if there exists another technique for learning motion patterns which takes much less calculation time. Another possibility to extend the current work is the usage of more raw data from the inertial measurement units, especially earth-magnetic field measurements. We believe that this information could be employed as landmarks, improving the raw odometry estimate.

Finally, one could extend the work by combining the navigation system of the quadrotor with the map estimate of the subject. A possible scenario would be that both agents are exploring the same environment, or that the quadrotor is estimating a map of the building by flying outdoors around it whereas the human is building an approximate map indoors. An interesting question here arises for the path planning. In more detail, one could address the question on how to guide the quadrotor to maximize the probability that the flying robot would perceive the human with his sensors, e.g., through a window. This would generate additional loop closures and improve the overall map build by the two agents.

List of Figures

1.1 One goal of this thesis: a fully autonomous indoor quadrotor 2
1.2 One goal of this thesis: mapping indoor environments based on human activity 2
1.3 One goal of this thesis: a robust and efficient solution to graph-based optimization 3

2.1 The coordinate system used in this thesis. 9
2.2 Example for motion composition 10

3.1 Introductory example for a graph consisting of robot poses and landmarks . . . 18
3.2 Introductory example of the concept *graph optimization* 18
3.3 Terminology for describing a graph 19
3.4 Effect of the configuration space on graph optimization: global poses 29
3.5 Effect of the configuration space on graph optimization: relative poses 30

4.1 Two examples of a robot trajectory and the corresponding tree structures 40
4.2 Processing order of the constraints given a tree structure 45
4.3 Evaluation of our approach: schematic view of the data set 47
4.4 Evaluation of our approach and the PPO algorithm for the a data set 47
4.5 Evolution of the network for both approaches at iteration $1, 5, 10$, and 30 48
4.6 Evolution of nodes depending on their connectivity in the graph 49
4.7 Example illustrating the problem of distributing an rotational error in 3D 50
4.8 Graph obtained from a car driving multiple times through a parking lot covering three floors 55
4.9 Experimental evaluation for a big simulated data 59
4.10 Experimental results using Frese's multi-level-relaxation algorithm 60
4.11 Evolution of the χ^2 error per constraint 60
4.12 Convergence comparisions between our approach and PPO 60
4.13 Experiment using the data set recorded at the Intel Research Lab in Seattle . . . 61
4.14 Experiment using the Bovisa data set from the Rawseeds project 62
4.15 Experiment using the CSail data set 62
4.16 Snapshots of our 3D graph optimization using the sphere dataset 64
4.17 Snapshots of our 3D graph optimization using the box dataset 65
4.18 Evolution of the χ^2 error per constraint for the sphere and the box data sets . . 66
4.19 Experiment using a data set recorded with an instrumented car at the EPFL campus in Lausanne 68
4.20 Experiment using the data set of the Intel Research Lab in Seattle 69
4.21 Corrected network from a car driving in a parking lot containing three floors multiple times 70
4.22 Experiment using a blimp equipped with a single camera and an IMU 70

4.23 Experiment using data recorded by a human carrying a stick with two cheap cameras and an IMU 71

5.1 Snapshot of an autonomous flight of our quadrotor in a cluttered office room . . 78
5.2 The quadrotor platform used in our experiments 80
5.3 Grid maps for a laser reading given different grid resolutions 83
5.4 A grid map and the corresponding likelihood field 84
5.5 Loop closing and graph-based map optimization 89
5.6 Example for joining different levels using our multi-level SLAM algorithm . . 93
5.7 Our test bench for learning a mapping between a command and the angle . . . 94
5.8 A grid map and the corresponding cost map used for path planning 96
5.9 Global localization of our quadrotor in a map previously acquired by a ground-based platform 98
5.10 Map of an office building built with our approach using the quadrotor 99
5.11 Object detection using 3D data acquired by our quadrotor platform 101
5.12 3D map of an office environments 102
5.13 Individual 3D scan taken at positions 1 to 12 (see Figure 5.12) 103
5.14 Individual 3D scan taken at positions 13 to 23 (see Figure 5.12) 104
5.15 Ground truth confusion matrix and examples of matches 105
5.16 Computed confusion matrix and recall rates for the place recognition experiment 105
5.17 Experiment for estimating the global height of the vehicle and the underneath floor level 107
5.18 Experiments for autonomous pose and yaw stabilization 107
5.19 Experiment for path planning and dynamic obstacle avoidance 108
5.20 Fuel-cell prototype 109
5.21 Evaluation of the fuel-cell under autonomous flight conditions 110
5.22 Snapshot from an autonomous indoor flight using the fuel-cell 111

6.1 The Xsens MVN data suit and typical data obtained when opening a door. . . . 116
6.2 Raw odometry obtained by the data suit and the corrected trajectory given our approach 117
6.3 Location of individual IMUs on the Xsens MVN data suit 118
6.4 A synthetic example for motion templates 119
6.5 Motion Template for a door opening sequence 121
6.6 Data obtained from the data suit while climbing up a stair 122
6.7 Explanatory example for multi-hypothesis tracking and N-scan-back pruning . 127
6.8 Room Segmentation 129
6.9 Approximate map generation 130
6.10 Two dimensional views of the outcome of the first experiment 134
6.11 Perspective view of the outcome for the first experiment 135
6.12 Two dimensional views of the outcome of the second experiment 136
6.13 Perspective view of the outcome for the second experiment 137
6.14 Two dimensional views of the outcome of the third experiment 138
6.15 Perspective view of the outcome for the third experiment 139
6.16 Outcome of the fourth experiment 140
6.17 Outcome of the fourth experiment: approximate and topological mapping . . . 141
6.18 Outcome of the fifth experiment 142
6.19 Outcome of the sixth experiment 143
6.20 Outcome of our approximate mapping algorithm for the second experiment . . 144

List of Tables

4.1 Mathematical comparison between PPO and our approach 46
4.2 Comparison to SAM for the sphere datasets with different noise realizations . . 66

5.1 Effect of the scan-matching algorithm on the pose stability of the flying robot . 86
5.2 Evaluation of our SLAM approach developed for the quadrotor 100
5.3 Comparison of our incremental SLAM without map optimization 100

6.1 Features employed in the neural network for step detection 123
6.2 Summary of all experiments on trajectory reconstruction 143

List of Algorithms

1 General Error Minimization Technique 23
2 Multi-Resolution Correlative Scan-Matcher 85
3 Monte-Carlo Localization 87
4 Multilevel-SLAM 91
5 Learn Class Template 120
6 Human Indoor Mapping 131

Bibliography

[1] Ascending Technologies. http://www.asctec.de/, last visited on 02/17/2011.

[2] P. Abbeel, A. Coates, M. Quigley, and A.Y. Ng. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Proc. of the Conf. on Advances in Neural Information Processing Systems (NIPS)*, page 1, 2007.

[3] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Autonomous Navigation and Exploration of a Quadrotor Helicopter in GPS-denied Indoor Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2008.

[4] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Stereo Vision and Laser Odometry for Autonomous Helicopters in GPS-denied Indoor Environments. *Unmanned Systems Technology XI*, 2009.

[5] S. Ahrens, D. Levine, G. Andrews, and J.P. How. Vision-based Guidance and Control of a Hovering Vehicle in Unknown, GPS-denied Environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2643–2648, 2009.

[6] M. Al-Baali. Descent Property and Global Convergence of the Fletcher-Reeves Method with Inexact Line Search. In *IMA Journal of Numerical Analysis*, volume 5, page 121, 1985.

[7] E. Altug, J.P. Ostrowski, and R. Mahony. Control of a Quadrotor Helicopter using Visual Feedback. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2002.

[8] E. Altug, J.P. Ostrowski, and C.J. Taylor. Control of a Quadrotor Helicopter Using Dual Camera Visual Feedback. In *Int. Journal of Robotics Research (IJRR)*, volume 24, page 329, 2005.

[9] K.O. Arras, S. Grzonka, M. Luber, and W. Burgard. Efficient People Tracking in Laser Range Data using a Multi-Hypothesis Leg-Tracker with Adaptive Occlusion Probabilities. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

[10] K.O. Arras, B. Lau, S. Grzonka, M. Luber, O. Martinez-Mozos, D. Meyer-Delius, and Burgard W. *Towards Service Robots for Everyday Environments*, chapter Range-Based People Detection and Tracking for Socially Aware Service Robots, pages 235–281. Springer STAR series, 2012.

[11] N. Ayache and O.D. Faugeras. Maintaining Representations of the Environment of a Mobile Robot. In *IEEE Transactions on Robotics and Automation (T-RA)*, volume 5, pages 804–819, 1989.

[12] S. Azrad, F. Kendoul, D. Perbrianti, and K. Nonami. Visual Servoing of an Autonomous Micro Air Vehicle for Ground Object Tracking. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5321–5326, 2009.

[13] A. Bachrach, R. He, and N. Roy. Autonomous Flight in Unknown Indoor Environments. In *Proc. of the Int. Journal of Micro Air Vehicles (IMAV)*, volume 1, pages 217–228, 2009.

[14] L. Bao and S.S. Intille. Activity Recognition from User-annotated Acceleration Data. In *IEEE Pervasive Computing Magazine*, pages 1–17, 2004.

[15] T. Barrera, A. Hast, and E. Bengtsson. Incremental Spherical Linear Interpolation. In *Proc. of the Swedish Chapter of Eurographics (SIGRAD)*, volume 13, pages 7–13, 2004.

[16] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2006.

[17] D.P. Bertsekas, A. Nedic, and A.E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.

[18] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based MAV Navigation in Unknown and Unstructured Environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 21–28, 2010.

[19] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D.G. Sorrenti, and J.D. Tardos. Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets. In *Proc. of the Workshop on Benchmarks in Robotics Research of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[20] J. Borenstein, L. Ojeda, and S. Kwanmuang. Heuristic Reduction of Gyro Drift for Personnel Tracking Systems. In *Journal of Navigation*, volume 62, pages 41–58, 2009.

[21] M. Bosse, P.M. Newman, J.J. Leonard, and S. Teller. An ALTAS framework for scalable mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 1899–1906, 2003.

[22] L. Bottou. Stochastic Learning. In *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, 2004.

[23] S. Bouabdallah, C. Bermes, S. Grzonka, C. Gimkiewicz, A. Brenzikofer, R. Hahn, D. Schafroth, G. Grisetti, W. Burgard, and R. Siegwart. Towards Palm-Size Autonomous Helicopters. In *International Conference and Exhibition on Unmanned Aerial Vehicles (UAV)*, 2010.

[24] S. Bouabdallah, C. Bermes, S. Grzonka, C. Gimkiewicz, A. Brenzikofer, R. Hahn, D. Schafroth, G. Grisetti, W. Burgard, and R. Siegwart. Towards Palm-Size Autonomous Helicopters. In *Journal of Intelligent & Robotic Systems*, volume 61, pages 1–27, 2011.

[25] S. Bouabdallah and R. Siegwart. Full Control of a Quadrotor. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[26] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck. Image-based Visual Servo Control of the Translation Kinematics of a Quadrotor Aerial Vehicle. In *IEEE Transactions on Robotics (T-RO)*, volume 25, pages 743–749, 2009.

[27] D. Braid, A. Broggi, and G. Schmiedel. The TerraMax Autonomous Vehicle. In *Journal of Field Robotics (JFR)*, volume 23, pages 693–708, 2006.

[28] C.G. Broyden, J.E. Dennis, and J.J. Moré. On the local and superlinear convergence of quasi-Newton methods. In *IMA Journal of Applied Mathematics*, volume 12, page 223, 1973.

[29] H. Buss and I. Busker. Mikrokopter, http://www.mikrokopter.de/, last visited on 02/17/2011.

[30] P. Castillo, A. Dzul, and R. Lozano. Real-time Stabilization and Tracking of a Four-rotor Mini Rotorcraft. In *IEEE Transactions on Control Systems Technology (T-CST)*, volume 12, pages 510–516, 2004.

[31] K. Celik, S.J. Chung, M. Clausman, and A.K. Somani. Monocular Vision SLAM for Indoor Aerial Vehicles. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1566–1573, 2009.

[32] H.S. Chen and M.A. Stadtherr. A modification of Powell's dogleg method for solving systems of nonlinear equations. In *Journal of Computers & Chemical Engineering*, volume 5, pages 143–150, 1981.

[33] T. Cheviron, T. Hamel, R. Mahony, and G. Baldwin. Robust Nonlinear Fusion of Inertial and Visual Data for Position, Velocity and Attitude Estimation of UAV. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2010–2016, 2007.

[34] O. Chum and J. Matas. Matching with PROSAC - Progressive Sample Consensus. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[35] B. Cinaz and H. Kenn. HeadSLAM - Simultaneous Localization and Mapping with Head-Mounted Inertial and Laser Range Sensors. In *IEEE Int. Symposium on Wearable Computers*, volume 0, pages 3–10, 2008.

[36] A. Coates, P. Abbeel, and A.Y. Ng. Learning for Control from Multiple Demonstrations. In *Proc. of the Int. Conference on Machine Learning (ICML)*, 2008.

[37] B. Coley, B. Najafi, A. Paraschiv-Ionescu, and K. Aminian. Stair Climbing Detection during Daily Physical Activity using a Miniature Gyroscope. In *Journal of Gait & Posture*, volume 22, pages 287–294, 2005.

[38] I.J. Cox and S.L. Hingorani. An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and its Evaluation for the Purpose of Visual Tracking. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, volume 18, pages 138–150, 1996.

[39] F. Dellaert. Square Root SAM. In *Proc. of Robotics: Science and Systems (RSS)*, pages 177–184, 2005.

[40] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for Mobile Robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

[41] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. In *Int. Journal of Robotics Research (IJRR)*, volume 25, page 1181, 2006.

[42] T. Duckett, S. Marsland, and J. Shapiro. Fast, On-line Learning of Globally Consistent Maps. In *Journal of Autonomous Robots*, volume 12, pages 287 – 300, 2002.

[43] B. Erginer and E. Altug. Modeling and PD Control of a Quadrotor VTOL Vehicle. In *Proc. of IEEE Intelligent Vehicles Symposium*, pages 894–899, 2007.

[44] C. Estrada, J. Neira, and J.D. Tardós. Hierachical SLAM: Real-Time Accurate Mapping of Large Environments. *IEEE Transactions on Robotics (T-RO)*, 21(4):588–596, 2005.

[45] R. Eustice, H. Singh, and J.J. Leonard. Exactly Sparse Delayed-State Filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2428–2435, 2005.

[46] R. Feliz, E. Zalama, and J.G. García-Bermejo. Pedestrian Tracking using Inertial Sensors. In *Journal of Physical Agents*, volume 3, pages 35–42, 2009.

[47] C. Fischer and H. Gellersen. Location and Navigation Support for Emergency Responders: A Survey. In *IEEE Pervasive Computing Magazine*, volume 9, pages 38–47, 2010.

[48] C. Fischer, K. Muthukrishnan, M. Hazas, and H. Gellersen. Ultrasound-aided Pedestrian Dead Reckoning for Indoor Navigation. In *Proc. of the first ACM Int. Workshop on Mobile Entity Localization and Tracking in GPS-less Environments*, pages 31–36, 2008.

[49] R. Fletcher and CM Reeves. Function minimization by conjugate gradients. In *The Computer Journal*, volume 7, page 149, 1964.

[50] D. Fox. Adapting the Sample Size in Particle Filters through KLD-sampling. In *Int. Journal of Robotics Research (IJRR)*, volume 22, page 985, 2003.

[51] E. Foxlin. Pedestrian Tracking with Shoe-mounted Inertial Sensors. In *IEEE Computer Graphics and Applications*, pages 38–46, 2005.

[52] P.E. Frandsen, K. Jonasson, H.B. Nielsen, and O. Tingleff. Unconstrained Optimization. Technical report, Technical Univ. of Denmark, Dep. of Informatics and Mathematical Modelling, 2011. http://www.maths.bris.ac.uk/∼maxmr/opt/nielsen_unconstrained.pdf, last visited on 02/17/2011.

[53] U. Frese. A proof for the Approximate Aparsity of SLAM Information Matrices. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 329–335, 2005.

[54] U. Frese. Treemap: An $O(\log n)$ Algorithm for Indoor Simultaneous Localization and Mapping. In *Journal of Autonomous Robots*, volume 21, pages 103–122, 2006.

[55] U. Frese and G. Hirzinger. Simultaneous Localization and Mapping - A Discussion. In *Proc. of the Workshop on Reasoning with Uncertainty in Robotics at the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 17–26, 2001.

[56] U. Frese, P. Larsson, and T. Duckett. A Multilevel Relaxation Algorithm for Simultaneous Localisation and Mapping. In *IEEE Transactions on Robotics (T-RO)*, volume 21, pages 1–12, 2005.

[57] C. Geyer, T. Templeton, M. Meingast, and S.S. Sastry. The Recursive Multi-frame Planar Parallax Algorithm. In *Proc. of the Third International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 17–24, 2007.

[58] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient Estimation of Accurate Maximum Likelihood Maps in 3D. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[59] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A Tutorial on Graph-based SLAM. In *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 2011.

[60] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[61] G. Grisetti, C. Stachniss, and W. Burgard. Improving Grid-based Slam with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2432–2437, 2006.

[62] G. Grisetti, C. Stachniss, and W. Burgard. Nonlinear Constraint Network Optimization for Efficient Map Learning. In *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, volume 10, pages 428–439, 2009.

[63] S. Grzonka, S. Bouabdallah, G. Grisetti, W. Burgard, and R. Siegwart. Towards a Fully Autonomous Indoor Helicopter. In *Workshop Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[64] S. Grzonka, F. Dijoux, A. Karwath, and W. Burgard. Learning Maps of Indoor Environments Based on Human Activity. In *Spring Symposium Series of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2010.

[65] S. Grzonka, F. Dijoux, A. Karwath, and W. Burgard. Mapping indoor environments based on human activity. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[66] S. Grzonka, G. Grisetti, and W. Burgard. Autonomous Indoors Navigation using a Small-Size Quadrotor. In *Workshop Proc. of the Intl. Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2008.

[67] S. Grzonka, G. Grisetti, and W. Burgard. Towards a Navigation System for Autonomous Indoor Flying. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[68] S. Grzonka, G. Grisetti, and W. Burgard. A Fully Autonomous Indoor Quadrotor. *IEEE Transactions on Robotics (T-RO)*, 8(1):90–100, 2 2012.

[69] S. Grzonka, A. Karwath, F. Dijoux, and W. Burgard. Activity-based Indoor Mapping and Estimation of Human Trajectories. *IEEE Transactions on Robotics (T-RO)*, 8(1):234–245, 2 2012.

[70] S. Grzonka, C. Plagemann, G. Grisetti, and W. Burgard. Look-ahead Proposals for Robust Grid-based SLAM. In *Proc. of the Int. Conference on Field and Service Robotics (FSR)*, July 2007.

[71] S. Grzonka, C. Plagemann, G. Grisetti, and W. Burgard. Look-ahead Proposals for Robust Grid-based SLAM with Rao-Blackwellized Particle Filters. In *Int. Journal of Robotics Research (IJRR)*, pages 191–200, Feb 2009.

[72] S. Grzonka, B. Steder, and W. Burgard. 3D Place Recognition and Object Detection using a Small-sized Quadrotor. In *Workshop on 3D Exploration, Mapping, and Surveillance with Aerial Robots at Robotics: Science and Systems (RSS)*, 2011.

[73] N. Guenard, T. Hamel, and R. Mahony. A practical Visual Servo Control for an Unmanned Aerial Vehicle. In *IEEE Transactions on Robotics (T-RO)*, volume 24, pages 331–340, 2008.

[74] J.S. Gutmann and K. Konolige. Incremental Mapping of Large Cyclic Environments. In *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, 1999.

[75] D. Hähnel, W. Burgard, B. Wegbreit, and S. Thrun. Towards Lazy Data Association in SLAM. In *Proc. of the Int. Symposium of Robotics Research (ISRR)*, 2003.

[76] T. Hamel, R. Mahony, and A. Chriette. Visual Servo Trajectory Tracking for a Four Rotor VTOL Aerial Vehicle. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, volume 3, pages 2781–2786, 2002.

[77] R. He, S. Prentice, and N. Roy. Planning in Information Space for a Quadrotor Helicopter in a GPS-denied Environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

[78] J. Hermosillo, C. Pradalier, S. Sekhavat, C. Laugier, and G. Baille. Towards Motion Autonomy of a Bi-steerable Car: Experimental Issues from Map-building to Trajectory Execution. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

[79] G. Hoffmann, D.G. Rajnarayan, S.L. Waslander, D. Dostal, J.S. Jang, and C.J. Tomlin. The Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC). In *The 23rd Digital Avionics Systems Conference (DASC)*, volume 2, 2004.

[80] G.M. Hoffmann, H. Huang, S.L. Waslander, and C.J. Tomlin. Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2007.

[81] A. Howard, M.J. Matarić, and G. Sukhatme. Relaxation on a Mesh: A Formalism for Generalized Localization. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1055–1060, 2001.

[82] A. Howard, D.F. Wolf, and G.S. Sukhatme. Towards 3D Mapping in Large Urban Environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 419–424, 2004.

[83] H. Huang, G.M. Hoffmann, S.L. Waslander, and C.J. Tomlin. Aerodynamics and Control of Autonomous Quadrotor Helicopters in Aggressive Maneuvering. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 3277–3282, 2009.

[84] IEEE. Top 11 Technologies of the Decade. In *IEEE Specturm, The Magazin of Technology Insiders*, volume 1, 2011. http://spectrum.ieee.org/static/special-report-top-11-technologies-of-the-decade, last visited on 02/17/2011.

[85] N.G. Johnson. Vision-assisted Control of a Hovering Air Vehicle in an Indoor Setting. Master's thesis, Brigham Young University, 2008.

[86] M.I. Jordan. *Learning in Graphical Models*. Kluwer Academic Publishers, 1998.

[87] S. Julier, J. Uhlmann, and H.F. Durrant-Whyte. A new Approach for Filtering Nonlinear Systems. In *Proc. of the American Control Conference (ACC)*, pages 1628–1632, 1995.

[88] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast Incremental Smoothing and Mapping with Efficient Data Association. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[89] C. Kemp. *Visual Control of a Miniature Quad-rotor Helicopter*. PhD thesis, Churchill College University of Cambridge, 2005.

[90] S. Klose, J. Wang, M. Achtelik, G. Panin, F. Holzapfel, and A. Knoll. Markerless, Vision-assisted Flight Control of a Quadrocopter. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5712–5717, 2010.

[91] S. Koenig and M. Likhachev. Fast Replanning for Navigation in Unknown Terrain. In *IEEE Transactions on Robotics and Automation (T-RA)*, volume 21, pages 354–363, 2005.

[92] K. Konolige. Large-scale Map-making. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 457–463, 2004.

[93] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent. Sparse Pose Adjustment for 2D Mapping. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[94] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[95] R. Kümmerle, D. Hähnel, D. Dolgov, S. Thrun, and W. Burgard. Autonomous Driving in a Multi-level Parking Structure. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 3395–3400, May 2009.

[96] S.W. Lee and K. Mase. Activity and location recognition using wearable sensors. In *IEEE Pervasive Computing Magazine*, volume 1, pages 24–32, 2002.

[97] J.J. Leonard and H.F. Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. In *IEEE Transactions on Robotics and Automation (T-RA)*, volume 7, pages 376–382, 1991.

[98] J.J. Leonard, H.F. Durrant-Whyte, and I.J. Cox. Dynamic Map Building for an Autonomous Mobile Robot. In *Int. Journal of Robotics Research (IJRR)*, volume 11, page 286, 1992.

[99] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of Wireless Indoor Positioning Techniques and Systems. In *IEEE Transactions on Systems, Man and Cybernetics (T-SMC), Part C: Applications and Reviews*, volume 37, pages 1067–1080, 2007.

[100] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. In *Journal of Autonomous Robots*, volume 4, pages 333–349, 1997.

[101] D.W. Marquardt. An Algorithm for Least-squares Estimation of Nonlinear Parameters. In *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, volume 11, pages 431–441, 1963.

[102] D. Mellinger, N. Michael, and V. Kumar. Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors. In *Proc. of the Int. Symposium on Experimental Robotics (ISER)*, 2010.

[103] M. Montemerlo and S. Thrun. Simultaneous Localization and Mapping with Unknown Data Association using FastSLAM. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 1985–1991, 2003.

[104] M. Montemerlo and S. Thrun. Large-scale Robotic 3-D Mapping of Urban Structures. In *Experimental Robotics IX*, pages 141–150, 2006.

[105] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to Simultaneous Localization and Mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 593–598, 2002.

[106] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1151–1156, 2003.

[107] P. Moutarlier and R. Chatila. An Experimental System for Incremental Environment Modelling by an Autonomous Mobile Robot. In *Experimental Robotics I*, pages 327–346, 1990.

[108] muFly. A Fully Autonomous Micro-Helicopter, http://www.mufly.org/, last visited on 02/18/2011.

[109] M. Müller and T. Röder. Motion Templates for Automatic Classification and Retrieval of Motion Capture Data. In *Proc. of the ACM Eurographics Symposium on Computer Animation (SIGGRAPH)*, pages 137–146, 2006.

[110] MVN-Videos. http://ais.informatik.uni-freiburg.de/projects/mvn/, last visited on 04/20/2011.

[111] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. In *Experimental Robotics IX*, pages 363–372. Springer Verlag, 2006.

[112] NG-UAVP. Next Generation Universal Aerial Video Platform, http://ng.uavp.ch, last visited on 02/17/2011.

[113] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM with Approximate Data Association. In *Proc. of the 12th Int. Conference on Advanced Robotics (ICAR)*, pages 242–249, 2005.

[114] E. Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2008.

[115] E. Olson. Real-Time Correlative Scan Matching. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 4387–4393, 2009.

[116] E. Olson, J. Leonard, and S. Teller. Fast Iterative Optimization of Pose Graphs with Poor Initial Estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2262–2269, 2006.

[117] M.A. Paskin. Thin Junction Tree Filters for Simultaneous Localization and Mapping. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1157–1164, 2003.

[118] P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart. Towards Mapping of Cities. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[119] E. Polak. *Optimization: Algorithms and Consistent Approximations*. Springer Verlag, 1997.

[120] D. Poole. *Linear Algebra: A Modern Introduction*. Brooks/Cole Pub Co, 2005.

[121] P. Pounds, R. Mahony, and P. Corke. Modelling and Control of a Quad-Rotor Robot. In *Proc. of the Australasian Conference on Robotics and Automation (ACRA)*, 2006.

[122] O. Purwin and R. D'Andrea. Performing Aggressive Maneuvers using Iterative Learning Control. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 1731–1736, 2009.

[123] Quadrotor Videos. http://ais.informatik.uni-freiburg.de/projects/quadrotor/, last visited on 02/17/2011.

[124] L. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[125] A. Ranganathan, M. Kaess, and F. Dellaert. Loopy SAM. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007.

[126] N. Ravi, N. Dandekar, P. Mysore, and M.L. Littman. Activity recognition from accelerometer data. In *Proc. of the Seventeenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 1541–1546, 2005.

[127] D. Reid. An algorithm for tracking multiple targets. In *IEEE Transactions on Automatic Control (T-AC)*, volume 24, pages 843–854, Dec 1979.

[128] M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm. In *Proc. of the IEEE Int. Conference on Neural Networks (ICNN)*, pages 586–591, 2002.

[129] J.F. Roberts, T. Stirling, J.C. Zufferey, and D. Floreano. Quadrotor Using Minimal Sensing For Autonomous Indoor Flight. In *Proc. of the European Micro Air Vehicle Conference and Flight Competition (EMAV)*, 2007.

[130] N. Roy, M. Montemerlo, and S. Thrun. Perspectives on Standardization in Mobile Robot Programming. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[131] A.P. Ruszczyński. *Nonlinear Optimization*. Princeton University Press, 2006.

[132] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma. Flying Fast and Low Among Obstacles: Methodology and Experiments. In *Int. Journal of Robotics Research (IJRR)*, volume 27, page 549, 2008.

[133] G. Schindler, C. Metzger, and T. Starner. A Wearable Interface for Topological Mapping and Localization in Indoor Environments. In *Proc. of the Second Int. Workshop on Location- and Context-Awareness (LoCA)*, volume 3987, pages 64–73, 2006.

[134] B. Schumitsch, S. Thrun, G. Bradski, and K. Olukotun. The Information-form Data Association Filter. In *Proc. of the Conf. on Advances in Neural Information Processing Systems (NIPS)*, volume 18, page 1193, 2006.

[135] K. Shoemake. Quaternions. Technical report, Department of Computer and Information Science University of Pennsylvania Philadelphia, PA, 1991.

[136] R. Smith, M. Self, and P. Cheeseman. A Stochastic Map for Uncertain Spatial Relationships. In *Proc. of the 4th Int. Symposium on Robotics Research*, pages 467–474, 1988.

[137] R. Smith, M. Self, and P. Cheeseman. Estimating Uncertain Spatial Realtionships in Robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[138] R.C. Smith and P. Cheeseman. On the Representation and Estimation of Spatial Uncertainty. In *Int. Journal of Robotics Research (IJRR)*, volume 5, page 56, 1986.

[139] J.A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-based Algorithms*. Springer Verlag, 2005.

[140] C. Stachniss, G. Grisetti, and W. Burgard. Recovering Particle Diversity in a Rao-Blackwellized Particle Filter for SLAM after Actively Closing Loops. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 655–660, 2006.

[141] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, and W. Burgard. Estimating Consistent Elevation Maps using Down-Looking Cameras and Inertial Sensors. In *Proc. of the Workshop on Robotic Perception on the International Conference on Computer Vision Theory and Applications*, 2008.

[142] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard. Learning Maps in 3D Using Attitude and Noisy Vision Sensors. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 644–649, 2007.

[143] B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual SLAM for Flying Vehicles. *IEEE Transactions on Robotics (T-RO)*, 24(5):1088–1093, 10 2008.

[144] B. Steder, G. Grisetti, M. Van Loock, and W. Burgard. Robust On-line Model-based Object Detection from Range Images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[145] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place Recognition in 3D Scans Using a Combination of Bag of Words and Point Feature based Relative Pose Estimation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[146] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D Range Image Features for Object Recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[147] A. Tayebi and S. McGilvray. Attitude stabilization of a VTOL Quadrotor Aircraft. *IEEE Transactions on Control Systems Technology (T-CST)*, 14(3):562–571, 2006.

[148] T. Templeton, D.H. Shim, C. Geyer, and S.S. Sastry. Autonomous Vision-based Landing and Terrain Mapping using an MPC-controlled Unmanned Rotorcraft. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 1349–1356, 2007.

[149] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[150] S. Thrun, M. Diel, and D. Hähnel. Scan Alignment and 3-D Surface Modeling with a Helicopter Platform. In *Field and Service Robotics (STAR Springer Tracts in Advanced Robotics)*, volume 24, pages 287–297, 2006.

[151] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H.F. Durrant-Whyte. Simultaneous Localization and Mapping With Sparse Extended Information Filters. In *Int. Journal of Robotics Research (IJRR)*, volume 23, pages 693–716, 2004.

[152] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the DARPA Grand Challenge. In *Journal of Field Robotics (JFR)*, volume 23, pages 661–692, 2006.

[153] G.D. Tipaldi, G. Grisetti, and W. Burgard. Approximate Covariance Estimation in Graphical Approaches to SLAM. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[154] C. Toth, D.A. Grejner-Brzezinska, and S. Moafipoor. Pedestrian Tracking and Navigation using Neural Networks and Fuzzy Logic. In *Proc. of the IEEE International Symposium on Intelligent Signal Processing (WISP)*, pages 1–6, 2008.

[155] G.P. Tournier, M. Valenti, J.P. How, and E. Feron. Estimation and Control of a Quadrotor Vehicle Using Monocular Vision and Moire Patterns. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, pages 21–24, 2006.

[156] R. Triebel, P. Pfaff, and W. Burgard. Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[157] J. Uhlmann. *Dynamic Map Building and Localization: New Theoretical Foundations*. PhD thesis, University of Oxford, 1995.

[158] C. Urmson. *Navigation Regimes for Off-Road Autonomy*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2005.

[159] Vicon Company. Vicon Motion Capture Systems, http://www.vicon.com/, last visited on 02/17/2011.

[160] M.R. Walter, R.M. Eustice, and J.J. Leonard. Exactly Sparse Extended Information Filters for Feature-based SLAM. In *Int. Journal of Robotics Research (IJRR)*, volume 26:4, page 335, 2007.

[161] H. Wang, H. Lenz, A. Szabo, J. Bamberger, and U.D. Hanebeck. WLAN-based Pedestrian Tracking using Particle Filters and Low-cost MEMS Sensors. In *4th Workshop on Positioning, Navigation and Communication (WPNC)*, pages 1–7, 2007.

[162] S.L. Waslander, G.M. Hoffmann, J.S. Jang, and C.J. Tomlin. Multi-Agent Quadrotor Testbed Control Design: Integral Sliding Mode vs. Reinforcement Learning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005.

[163] O. Woodman. *Pedestrian Localisation for Indoor Environments*. PhD thesis, University of Cambridge, 2010.

[164] O. Woodman and R. Harle. Pedestrian Localisation for Indoor Environments. In *Proc. of the 10th Int. Conference on Ubiquitous Computing*, pages 114–123, 2008.

[165] Xsens. MVN Suit http://www.xsens.com/en/general/mvn, last visited on 02/17/2011.

[166] M. Yguel, C.T.M. Keat, C. Braillon, C. Laugier, and O. Aycard. Dense Mapping for Range Sensors: Efficient Algorithms and Sparse Representations. In *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[167] A. Zell, G. Mamier, M. Vogt, N. Mache, R. Huebner, K.-U. Herrmann, T. Soyez, M. Schmalzl, T. Sommer, A. Hatzigeorgiou, S. Doering, D. Posselt, and T. Schreiner. SNNS, Stuttgart Neural Network Simulator, User manual, Version 4.1. Technical Report 6/95, University of Stuttgart, 1995.